Universitat Oberta de Catalunya (UOC)

Máster Universitario en Ciencia de Datos (*Data Science*)

TRABAJO FINAL DE MÁSTER

Área: Ciencias de datos aplicadas a la salud

Clasificación de imágenes radiológicas aplicando modelos Transformers

---

Autor: Francisco Javier Corrales Estrella

Tutor: Carlos Luis Sanchez Bocanegra

Profesor: Albert Solé Ribalta

Madrid, 15 de enero de 2023

Créditos/Copyright

FICHA DEL TRABAJO FINAL

Título del trabajo:	Clasificación de imágenes radiológicas con Transformers
Nombre del autor:	Francisco Javier Corrales Estrella
Nombre del colaborador/a docente:	Carlos Luis Sanchez Bocanegra
Nombre del PRA:	Albert Solé Ribalta
Fecha de entrega (mm/aaaa):	01/2023
Titulación o programa:	Master de Ciencia de Datos
Área del Trabajo Final:	Ciencia de Datos aplicadas a la Salud
Idioma del trabajo:	Español
Palabras clave:	Autoatención, Transformers, Radiodiagnóstico

Dedicatoria/Cita

A mi tía Sagra, por hacerme amar las matemáticas, y animarme a empezar este master, y aunque no haya podido ver el final y ahora este con Gauss y Taylor, esta dedicatoria debe ser para ella.

Agradecimientos

A Cristina, Alba, Lucia y Diego por ser mis cuatro patas, sin ellos esto no tendría sentido, perdonadme por haberos robado tantas tardes. Agradecimientos a Carlos, mi tutor, por animarme y espabilarme cuando todo lo veía negro.

Resumen

La detección temprana de enfermedades mediante el estudio de imágenes es clave para aumentar la esperanza de vida del paciente. Existen diferentes tipos de pruebas como radiografías, tomografía, resonancias magnéticas, mamografía, ecografías [23]. La aplicación de técnicas de IA es una nueva forma de abordar este problema y que permite a los radiólogos detectar enfermedades. Mediante el presente trabajo se pretende hacer una revisión del estado del arte de las redes neuronales basadas en Transformers para el diagnostico por imágenes y realizar una aplicación práctica sobre un conjunto de imágenes para evaluar sus resultados.

Palabras clave: Autoatención, Transformers, Radio-diagnóstico

Abstract

The early detection of diseases through imaging studies is key to increasing the life expectancy of the patient. There are different types of tests such as x-rays, tomography, magnetic resonance imaging, mammography, ultrasound [23]. The application of AI techniques is a new way to approach this problem and allows radiologists to detect diseases. Through this work, we intend to review the state of the art of neural networks based on Transformers for image diagnosis and carry out a practical application on a set of images to evaluate their results.

keywords: Auto attention, Transformers, radiodiagnosis

Índice general

Índice de figuras

Índice de cuadros

Capítulo 1

Introducción

1.1. Contexto y justificación del trabajo

Según el informe del SEOM Las cifras del cancer en España en el 2022 [14] se estima que el número de tumores diagnosticados durante el 2020 fue de 18.094.716 casos en todo el mundo. Solo en España la incidencia durante el 2020 fue de 280.100 casos y se estima que para el 2040 llegue a los 341.000 casos. En la figura 1.1 puede observarse la proyección que se hace de la incidencia hasta el 2040.

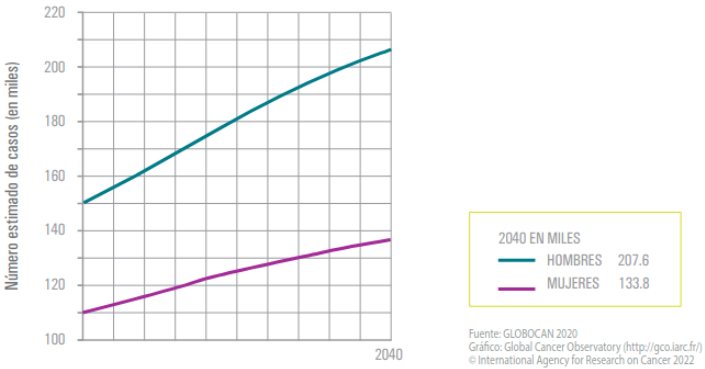

Figura 1.1: Incidencia estimada de tumores en España para los años 2020 y 2040, por sexos. Fuente: Globocan 2020

Los tipos de cancer con mayor incidencia que se prevee diagnosticar durante el 2022 serán el de colon y recto (43.370 casos), mama (34.750 casos), pulmón (30.948 casos), próstata (30.884 casos) y vejiga (22.295).

En número de casos absolutos diagnosticados ha aumentado debido a varios factores, como el aumento de la población, un mayor envejecimiento o mejoras en los diagnósticos. En los últimos años ha existido una variación en la incidencia de los diferentes tipos de cancer debido a los cambios de hábitos, por ejemplo una reducción del tabaquismo en los hombres con una disminución en los casos de cancer de pulmon, por contra, las mujeres han aumentado ese hábito y ha aumentando la incidencia de ese cancer.

El cancer es una de las principales causas de mortalidad, los datos que proporciona la International Agency for Research on Cancer (IARC) estima que aproximadamente 9,9 millones de personas fallecieron por cancer durante el 2020. Estos datos son estimaciones que se realizarón previamente al Covid-19 por lo que hay que entenderlo como una estimación si no hubiese existido la pandemia. En este contexto, se refleja la importancia que cobrá el diagnostico de estas enfermedades, y en particular el diagnóstico por imagenes. La evolución de la Inteligencia Artificial en los ultimos años será clave para el diagnostico temprano en particual del cancer. Según un estudio publicado en The Lancet Digital Health [10] la inteligencia artificial detecta diferentes tipos de tumores con la misma precisión con la que la harían los profesionales sanitarios, aunque ahora mismo los estudios realizados no tienen el suficiente número de casos de estudio para poder hacer está afirmación con rotundidad como expresan los autores del artículo.

Según el artículo Artificial intelligence in healthcare: past, present and future [9] donde se hace una revisión de la literatura medica y la aplicación de la IA, los algoritmos más utilizados en la detección de enfermedades en imagentes son las SVM y las redes neuronales. En la figura 1.2 se pueden ver la proporción de uso de cada algoritmo.

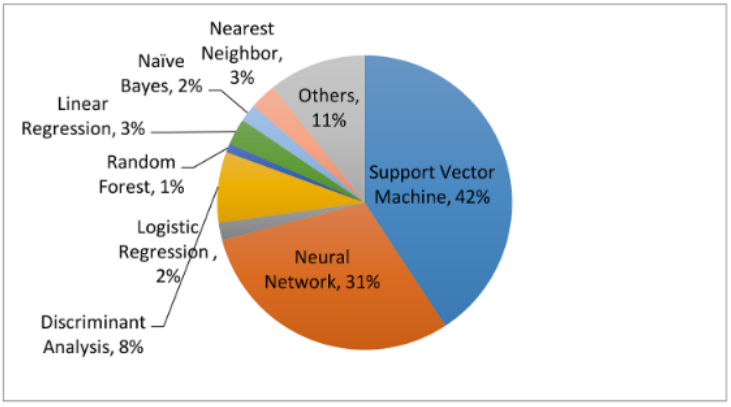

Figura 1.2: Algoritmos de machine learning usados para diagnostico de imagenes.

Aunque hasta hace relativamente poco el uso de algoritmos basados en SVM y Redes neu-

ronales era semejantes, desde el año 2013 se observa una tendencia a que las redes neuronales (Deep Learning) se conviertan en la técnica más usada. 1.3.

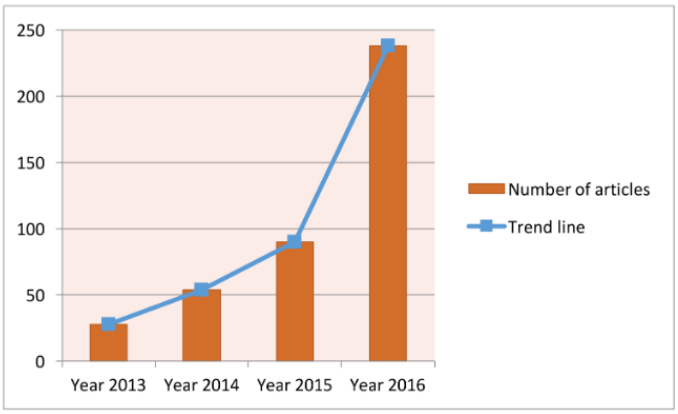

Figura 1.3: Tendencia actual para el aprendizaje profundo.

En el año 2017 se publicó el paper Attention is All you Need [22], que supuso un nuevo rumbo en el uso de las redes neuronales consiguiendo hitos como AlphaFold2 [1] poder analizar secuencias de datos genómicos, un problema hasta ahora incompleto con las tecnologías anteriores.

1.2. Motivación

En este contexto en el que las redes neuronales convolucionales (CNN) o recurrentes (RNN) representaban el estado del arte, la aparición de mecanismos basados en la atención hacen posible una nueva ola en al avance del Deep Learning.

Aprovechando que durante el master no profundice lo suficiente en el uso de redes neuronales, está es una gran oportunidad para poder hacerlo. Este trabajo me va a permitir hacer una revisión del estado del arte de las redes neuronales y poder realizar un caso completo con una red neuronal basada en transformers. De esta forma podré comprobar las ventajas de este nuevo mecanismo, esperando que los resultados pueden aportar mejoras significativas en la detección de enfermedades.

1.3. Objetivos del Trabajo

El objetivo principal (OP) es aplicar un mecanismo basado en transformers sobre un conjunto de imágenes reales, para resolver un problema de clasificación de diagnósticos.

Son varios los objetivos adicionales que se pretenden conseguir con este trabajo:

- (OS1) Comprobar si este modelo mejora otros resultados anteriores de clasificación sobre algún conjunto de imagenes público del que ya se tengan medidas de precisión.
- (OS2) Medir los tiempos de aprendizaje en este modelo.
- (OS3) Evaluar si se necesitan menos datos de aprendizaje con este modelo.
- (OS4) Secundariamente, y si se disponen de los recursos suficientes, poder especificar los requisitos mínimos de hardware necesarios.

1.4. Enfoque y método seguido

En primer lugar se inicializarán tareas para conseguir un conjunto de datos anonimizado de un servicio público de salud. Existen ahora mismo posiblidades con el Servicio Andaluz de Salud (SAS) y con Servicio de Salud de Castilla la Mancha (SESCAM). En caso de que esto no sea posible, se usará un dataset publico de una plataforma habitual como kaagle o un dataset más especifico como los proporcionados por el National Institute of Cancer.

Paralelamente se empezará haciendo una revisión del estado del arte y una preparación del entorno de trabajo, para esto ultimo se han inicilizado vias para ver que facilidades puede proporcionar la uoc, ya que se prevee que se necesite una buena GPU para el procesamiento de datos.

A continuación se realizarán las tareas propias de limpieza de datos, que en este caso no estimamos que sea una tarea complicada, al menos que las imagenes tengan datos del paciente y se necesiten anonimizar.

Una vez listo el entorno de trabajo y el conjunto de datos, se comenzará con la tarea de implementar el modelo de transformers, probablemente en lenguaje python. Si el conjunto de datos es extenso, quizas sea necesario trabajar en algún entorno de big data como cloudera, y usaría pyspark.

Hasta que tengamos el método totalmente implementado, trabajaría inicialmente con un conjunto de datos limitado para que los tiempos de procesamiento no penalicen. Y finalmente aplicaríamos el modelo sobre el conjunto de datos completo.

Por ultimo, sería el momento de medir la calidad del modelo, tiempos de ejecución y comparar con otros trabajos anteriores de clasificación de imagenes.

1.5. Planificación del trabajo

Para organizar el trabajo voy a utilizar la herramientas dos herramientas, por un lado haré uso de la versión gratuita de Team Gantt en la que ya he elaborado una planificación inicial.

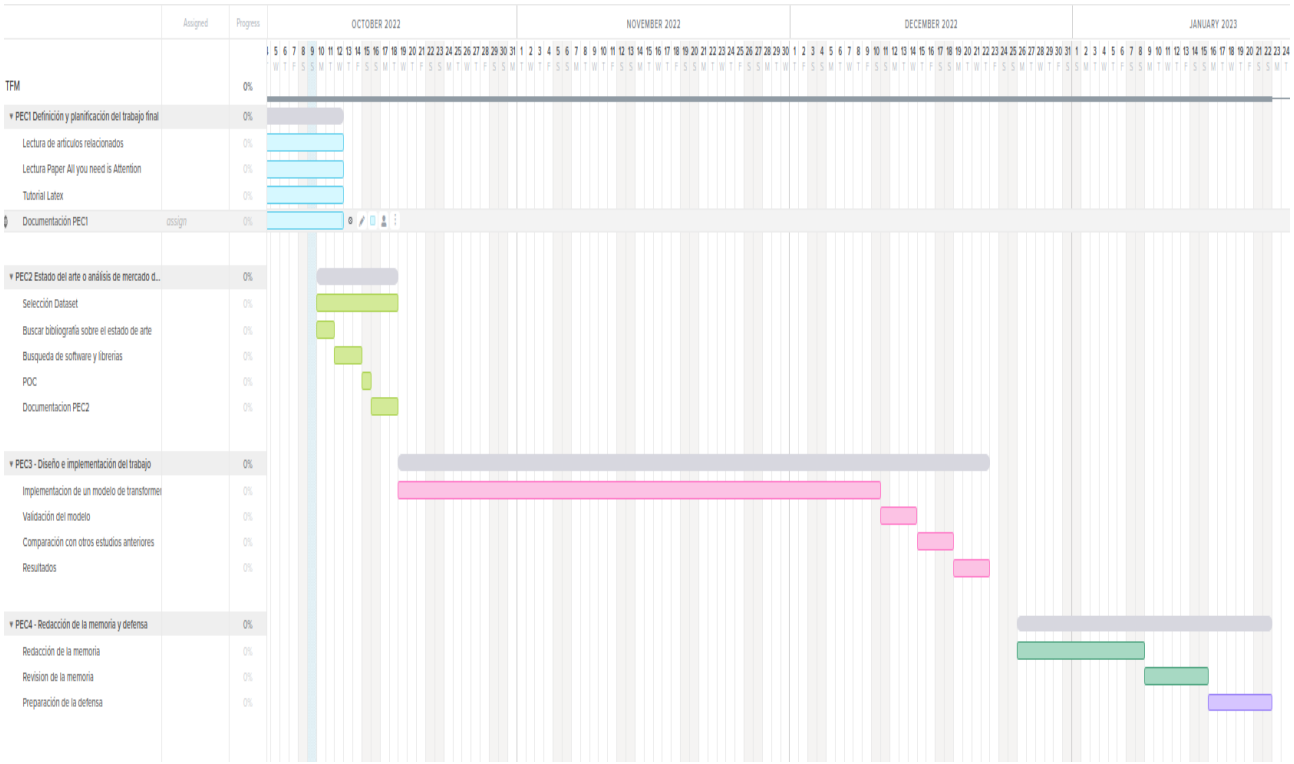

Figura 1.4: Diagrama Gant.

Y paralelamente para la comunicación con mi tutor y que me pueda ayudar a identificar o definir tareas, usaré Trello, con el dashboard habitual de Kanban con Lista de tareas, En Progreso y Hecho.

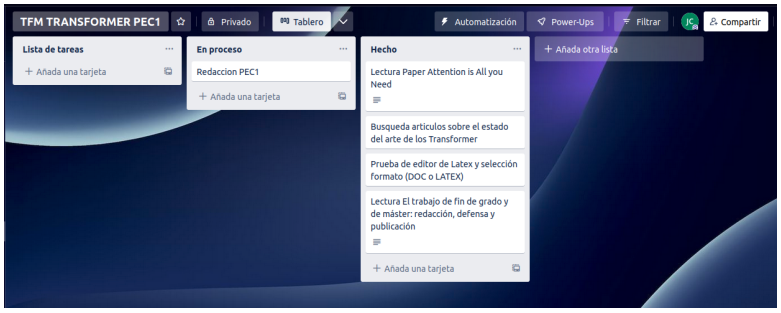

Figura 1.5: Proyecto Trello.

FASE 1 Cubre la PEC1, y terminará el 9 de octubre del 2022

1. Lectura de articulos para la definición del alcance
2. Lectura paper All you need is attention
3. Tutorial Latex
4. Elaboracion PEC1

FASE 2 Cubre la PEC2, y Terminará el 23 de octubre del 2022

1. Selección del Dataset
2. Bibliografía sobre el estado de arte
3. Puesta en marcha del entorno de trabajo (Pyspark, Github, BD,...)
4. Investigación sobre librerias
5. Primera prueba de concepto
6. Documentacion PEC2

FASE 3 Cubre la PEC3, y terminará el 25 de diciembre del 2022, Diseño e implementación del trabajo

1. Implementación de un mecanismo de atención basado en transformers
2. Validación del modelo
3. Comparación con otros estudios anteriores con redes RNN o CNN
4. Validación del modelo
5. Resultados

FASE 4 Terminará el 22 de enero

1. Hasta el 8 de enero preparación de la memoria
2. Hasta el 15 de enero revisión de la memoria
3. Preparación de la defensa

Capítulo 2

Estado del Arte

2.1. Historia

Las redes neuronales o Deep Learning las podemos englobar como un subconjunto de técnicas dentro del aprendizaje automático que a su vez se engloba dentro del conjunto de técnicas de la Inteligencia Artificial.

Según la clasificación de Pedro Domingos en su libro The Master Algorithm, existen cinco grandes escuelas de pensamiento sobre el aprendizaje automático, donde cada una de ellas ha desarrollado sus propios algoritmos.

- Los simbólicos: Sus técnicas se centran en la representación simbólica del conocimiento adquirido por inducción, es decir, determinan lo que es el conocimiento por un procedimiento de deducción y luego generalizan el conocimiento. Dentro de ellos podemos destacar los **árboles de decisión** como el algoritmo mas conocido aunque existen otros modelos de clasificación basados en reglas de tipo simbólico, como es la **inducción de reglas o las listas de decisión**.

- Las técnicas analógicas: En este caso, el conocimiento se extrapola en base a los datos ya conocidos, estableciendo similitudes entre ellos y detectando patrones. Dentro de estas técnicas se englobarían por ejemplo, los clasificadores ***k*-NN** o las maquinas de soporte vectorial **SVM**.

- Los Bayesianos. Esta familia de técnicas están basadas en modelos probabilísticos, cuya centro es el teorema de Bayes, que nos permite realizar inferencias probalísticas pasando de la probabilidad a priori de un suceso a la prioridad a posteriori. Permiten resolver problemas de tipo no supervisado como los algoritmos de ***k*-medias o *k*-means**; supervisados como el uso del algoritmo **Naïve Bayes** o métodos probabilísticos mas generales como las redes bayesianas.

- Los evolutivos. Su piedra angular es la evolución natural y sus técnicas mas destacadas serían los algoritmos genéticos.

- Los conexionistas. Basadas en las conexiones neuronales en la función cerabral. Hasta ahora es la familia de técnicas que mas se ha acercado a conseguir máquinas capaces de programarse a si mismas. Es dentro de esta escuela del pensamiento donde se engloban las **Redes Neuronales Artificales**.

Historicamente, tal como introduce el famoso libro de Ian Goodfellow, Yoshua Bengio y Aaron Courvillese [5] se han producido tres olas de desarrollo en el aprendizaje profundo; en la primera todavia no existía el término Deep Learning, y se la conoce como la ola cibernética, surge en las decadas de 1940 y 1960 con los desarrollos basados en las teorías de aprendizaje en la biología de McCulloch y Pitts [13] y con la presentación del perceptrón en el National Phsycal Laboratory por parte de Frank Rosenblatt [19]. La segunda ola, se inicia en los años 80 con la aparición del algoritmo de propagación de errores denominado *backpropagation* que se popularizó en el año 1986 con la publicación del artículo de David Rumenhart, Geoffrey Hinton y Ronald Williams ,"Learning representations by back-propagating errors"[20]. La tercera ola comienza durante el año 2006 con trabajos de autores como Geofrey Hinton [6], Yoshua Bengio [3] y Yann LeCun, en las que se publicaron trabajos en los que se proponían nuevos modelos de entrenamiento para redes neuronales mas complejas de las que se habían podido utilizar anteriormente. La publicación en el año 2017 del artículo ."All you need is attention"de Ashish Vaswani y su equipo de Google [22], ha supuesto una nueva ola con la incorporación del mecanismo de Transformers ea la arquitectura de redes neuronales.

2.2. Mecanismos de atención

Una función indispensable para el ser humano es mecanismo de atención [18]. Como no podemos procesar todos los estimulos y lo que percibimos de una sola vez, somos capaces de seleccionar la parte de la información que necesitamos, podemos concentrarnos solo en lo que nos interesa. Los mecanismos de atención se pueden dividir en dos categorías según su propósito.

- **Bottom-up** Atención inconsciente, es dirigida por un estímulo externo.

- **Top-down** Atención enfocada, la atención se redirige de forma consciente y activa hacia un cierto objeto.

La mayoria de los mecanismos de atención en Deep Learning son de esta segunda categoría y están diseñados para tareas concretas y se basan en atención enfocada.

En el área de la vision computerizada, los mecanismos de atención supusieron una solución a los casos en los que la capacidad de computación es limitada, pudiendo extraer características locales para obtener regiones salientes potenciales [8]. Dentro de las redes neuronales, se empezarón a usar mecanismos de atención en RNN para clasificar imágenes [15]. Bahdanau, Cho, y Bengio [2],propusieron el usó de mecanismos de atención para realizar simultáneamente traducción y alineación en tareas de traducción automática.

Los mecanismos de atención fueron gananado importancia en su uso dentro de las arquitecturas neuronales [16] y se fueron aplicado a diversas tareas, como la generación de leyendas de imágenes ,clasificación de texto , traducción automática, reconocimiento de acciones , análisis basado en imágenes, reconocimiento de voz y sistemas de recomendación. Adicionalmente, los mecanismo de atención podían ser usados como una herramienta para darnos una visión intuitiva de la arquitectura que subyace dentro de una red neuronal y el peso que la red le otorga a cada variable, como se puede ver en la siguiente figura.

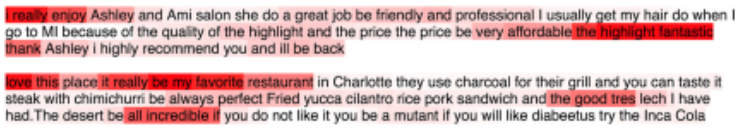

Figura 2.1: Ejemplo de mecanismo de atención. [16]

2.3. Arquitecturas

A continuacion se hará un breve repaso a las diferentes arquitecturas que han ido evolucionado y aplicando mecanismos de atención hasta llegar a los Transformers.

2.3.1. RNN

Tal y como explica el artículo de Sequence to Sequence Learning with Neural Networks, [21] los problemas de modelado de secuencias son aquellos en los que el input, o el output o ambos son una secuencia de datos, ya sean palabras o letras por ejemplo. De esta necesidad, surgen las Redes Neuronales Recurrentes (RNN), donde dada una secuencia de inputs $x_1, ..., x_T$, una RNN standard generará una secuencia de ouputs $y_1, ..., y_T$ iterando la siguiente ecuacion:

$$h_t = sigm(W^{hx}x_t + W^{hh}h_{t-1})$$

$$y_t = W^{yh}h_t$$

Las redes neuronales recurrentes son la arquitectura natural a los problemas íntimamente ligados con secuencias y listas. En los últimos años, ha habido un éxito increíble al aplicar RNN a una variedad de problemas Illustrated guide to recurrent neural networks : reconocimiento de voz, modelado de lenguaje, traducción, subtítulos de imágenes...

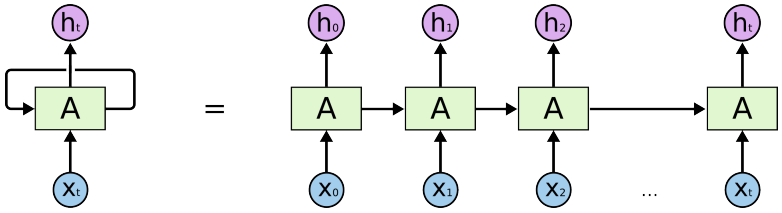

Figura 2.2: Diagrama general de una RNN. Fuente: https://colah.github.io/posts/2015-08-Understanding-LSTMs/

Pero las RNN tienen dos problemas fundamentales, la longitud de la entrada y la salida pueden no ser iguales, tipicamante cualquier tarea de traducción, y si las cadenas son muy largas existe lo que se denomina "The Problem of Long-Term Dependencies", donde las dependencias entre las primeras entradas y las ultimas disminuye, perdiendo contexto que puede ser importante.

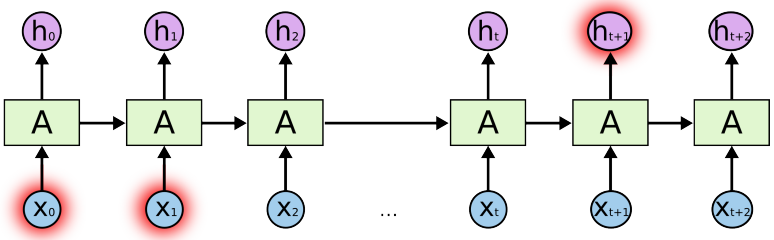

Figura 2.3: The Problem of Long-Term Dependencies. Fuente: https://colah.github.io/posts/2015-08-Understanding-LSTMs/

2.3.2. LSTM

Este problema se resuelve gracias a las Long Short Term Memory networks (LSTM), introducidas en el año 1997 por Hochreiter y J. Schmidhuber [7], un tipo especial de RNN, que son capaces de aprender dependencias de terminos a largo plazo. El objetivo principal de una LSTM [21] es estimar la probabilidad condicionada $p(y_1, ..., y_{T'} \mid x_1, ..., x_T)$ donde $(x_1, ..., x_T)$ es la entrada y $(y_1, ..., y_{T'})$ es la salida con una longitud T' diferente de T. La LSTM primero procesa esta probabilidad condicionada para obtener una representación dimensional v de la secuencia de entrada $(x_1, ..., x_T)$ procedente del ultimo estado oculto de LSTM, y posteriormente

se calcula la probabilidad $y_1, ..., y_{T'}$ con formulación de la LSTM cuyo estado inicial oculto está establecido por el vector v de $x_1, ..., x_T$:

$$p(y_1, ..., y_{T'} \mid x_1, ..., x_T) == \prod_{t=1}^{T'} p(y_t \mid v, y_1, ..., y_{t-1})$$

2.3.3. Seq2Seq

En un modelo **Seq2Seq**, se utilizan LSTM; un codificador (encoder) que recibe la entrada en forma de vector de estado oculto y un descodificador(decoder) que produce la salida. En el modelo tradicional de secuencia a secuencia, la LSTM del enconder combinan la entrada actual con la salida anterior del encoder y comprimen todo la infomrmación de la secuencia de entrada en el estado inicial $h(t)$, donde cada palabra tiene el mismo peso.

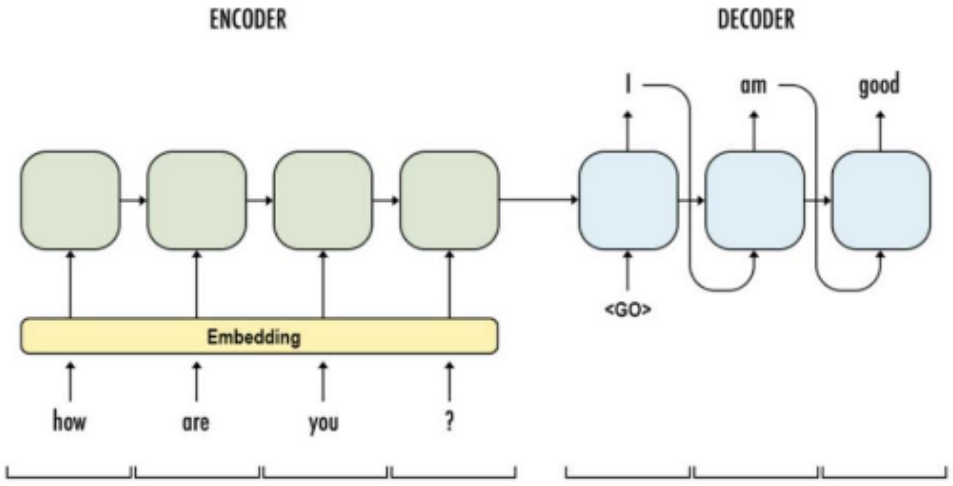

Figura 2.4: Modelo Seq2Seq. Fuente:
https://mlearninglab.com/2019/05/26/introduccion-a-los-mecanismos-de-atencion/

La primera celda del codificador recibe una palabra de entrada, cada celda siguiente recibe la salida de la celda anterior y su correpondiente palabra de entrada, la última celda entrega al decoder la información completa de todo el contexto de la frase. El decodificador recibe de entrada la salida del encoder mas un token de inicio. Cada celda del descodificador tiene dos salidas, una que es la salida de esa celda y otra interna, ambas son entradas de la celda siguiente. El conjunto de salidas individuales asociadas a cada celda del decode constituyen la salida completa del modelo.

Aunque en el parrafo anterior estamos hablando de *celda*, en realidad cada celda sería una red neuronal recurrente, en concreto, una LSTM.

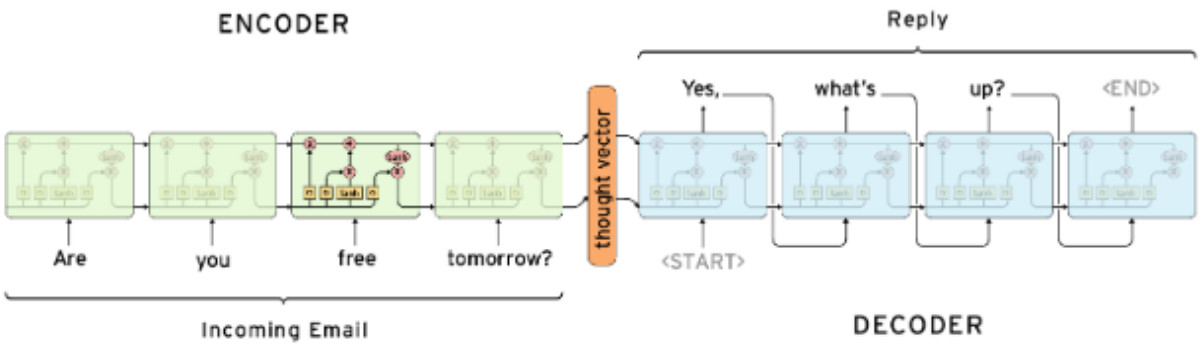

Figura 2.5: Modelo Seq2Seq Utilizando LSTM. Fuente: https://mlearninglab.com/2019/05/26/introduccion-a-los-mecanismos-de-atencion/

2.3.4. Mecanismo de atención en Seq2Seq

El principal problema derivado de los modelos Seq2Seq es que la secuencia de salida depende principalmente de la salida final del codificador, y en los casos de sentencias largas, es bastante probable que se haya perdido contexto del comienzo de la frase.

Para poder resolver este problema, Bahdanau [2] propuso su modelo RnnSearch en el que se añade un mecanismo de atención sobre un modelo encoder-decoder, con el objetivo es que cada parte de la salida del modelo dependa de todas las entradas.

En un modelo Seq2Seq, el codificador pasa todos los estados ocultos al decodificador, no solo el ultimo. El decodificador asigna un peso a cada estado oculto procedente de palabra de entrada $h_1, ..., h_T$ en lugar de tener en cuenta solo h_T con el propósito de amplificar la importancia de aquellos estados ocultos con mayor relevancia.

El decodificador en cada instante t tendrá una función de salida

$$s(t) = f(s(t-1, y(t-1), c(t))$$

donde c(t) es un vector que representa el contexto de los estados ocultos, y se calcula como la suma ponderada de estos:

$$c(t) = \sum_{i=1}^{T} \alpha_{ti} h(i)$$

Y α es el coeficiente de ponderación de cada entrada i en el instante t, y se calcula de la siguiente forma:

$$\alpha_{ti} = \frac{exp(score(s(t-1), h(i)))}{\sum_{i\prime=1}^{T} exp(score(s(t-1), h(i\prime)))}$$

El coeficiente α será un parametro de entrada del modelo, y suele ser entrenado por una red neuronal, aunque se pueden usar otras funciones como el producto escalar, coseno,etc. En la siguiente figura puede verse una arquitectura de seq2seq con mecanismo de atención aplicado:

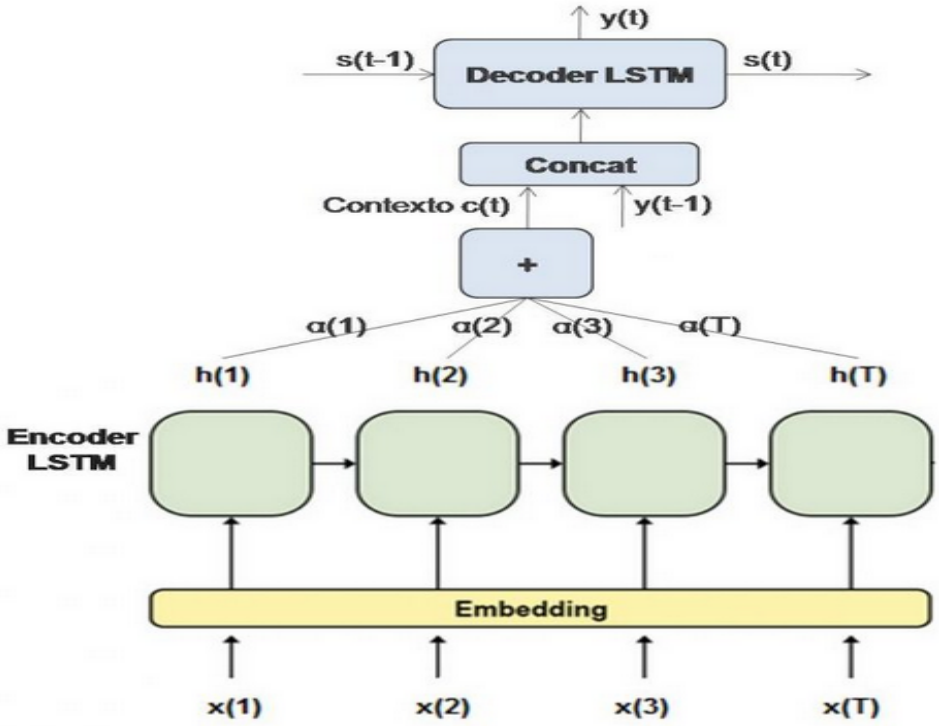

Figura 2.6: Modelo Seq2Seq con mecanismo de atención. Fuente: https://mlearninglab.com/2019/05/26/introduccion-a-los-mecanismos-de-atencion/

Este modelo todavia tiene una naturaleza secuencial, lo que imposibilita el procesamiento en paralelo y poder sacar el máximo rendimiento al uso de GPU y su poder computacional.

2.3.5. Transformers

En la proxima sección (proxima PEC...) se verá de forma mas extensa la arquitectura de los modelos Transformer aplicados en el campo de la visión, pero ahora se incluye una breve descripción de esta arquitectura sobre la que se basa este trabajo. El modelo Transformers se introdujo por primera ver en el paper Attention is All you Need [22] y al igual que los modelos Seq2Seq también se basa en un codificador y un decodificador, pero en este caso se eliminan las redes recurrentes.
El codificador recibe toda la secuencia a la vez, dicha entrada sufre un proceso llamado **Input Embedding** por el que cada palabra es convertida a un vector, palabras parecidas tendrán vectores mas cercanos. Tambien se aplica lo que denomina **positional Encoding** para poder proporcionar un valor a la posición de la palabra dentro de la frase. Este encoding se procesa por las capas Multi-Head Attention que captan las diferentes relaciones que tiene cada palabra con el resto de palabras de la entrada. El decodificador tambien tiene un proceso de entrada llamado **Output Embedding** y su correspondiente capa de Multi-Head Attention.

A diferencia de las redes neuronales recurrentes, los Transformers pueden paralelizar de forma muy eficiente, por lo que si tenemos el software adecuado pueden entrenarse modelos realmente grandes.

2.3.6. Casos de exito

Algunos casos de exito de los transformers son:

1. **BERT** (Bidirectional Encoder Representations from Transformers): Forma parte de los factores de clasificación que usa google para recomendarnos una pagina cuando utlizamos el buscador, haciendo que el lenguaje natural cobre importancia en los resultados de busqueda. Understanding searches better than ever before

2. **GPT-3** (Transformer, Generative Pre-trained): Es un modelo de generación de texto, que fue entrenado por unos 45TB de datos El lenguaje GPT-3 Transformer

3. **DALL-E** Generador de imagenes a partir de texto

4. **T5** Un tipo de transformer creado por Google AI y que se publico en el paper ” “Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformerz que con una arquitectura uniforme puede ser entrenado para multiples tareas [17].

5. **AlphaFold2** Un algoritmo desarrollado por la empresa DeepMind que ha proporcionado resultados espectaculares en la predicción de la estructura dimensional de las proteinas. AlphaFold 2 is here: what's behind the structure prediction miracle

Capítulo 3

Mecanismos de atención y Transformers

La arquitectura basada en transformers fue propuesta en año 2017 en el artículo .All you need is attention"de Ashish Vaswani y su equipo de Google [22] pero los mecanismos de atención son previos a este modelo, Bahdanau et al., 2014 [2] and Luong et al., 2015 [12] propusieron soluciones para implementar mecanismos de atención sobre redes recurrentes para resolver el problema de como el peso de los token de los primeros elementos de una secuencia se iba diluyendo en sentencias grandes.

Tal como explica el paper de Vaswni, el gran avance, es que en las tareas propias de NLP en las que se van leyendo datos de forma recurrente es suficiente con aplicar atención y que no es necesario procesar los datos recurrentemente y por tanto secuencialmenbte, es decir, podemos procesar los datos en paralelo, por lo que nuestra capacidad de procesamiento ahora puede ser escalable.

Para poder entender bien como funcionan las arquitecturas Transformers debemos primero profundizar en como se implementan los mecanismos de atención originalmente en problemas de NLP.

Los mecanismos de atención están basados en la focalización y en darle mayor valor a ciertos elementos cuando estamos procesando datos. Por tanto, la atención es un componente mas dentro de una red neuronal y en un problema de traducción se encargaría de establecer pesos que reflejen la importancia entre cada palabra del input y del output.

Los mecanismos de atención empezaron a aplicarse sobre modelos seq2seq dada su incapacidad para poder predecir en sentencias largas, creando una relación entre cada paso de tiempo de la salida de decodificador y cada estado oculto del decodificador, haciendo que para cada salida que hace el decodificador tenga acceso a toda la secuencia de entrada y pueda elegir elementos concretos para producir la salida.

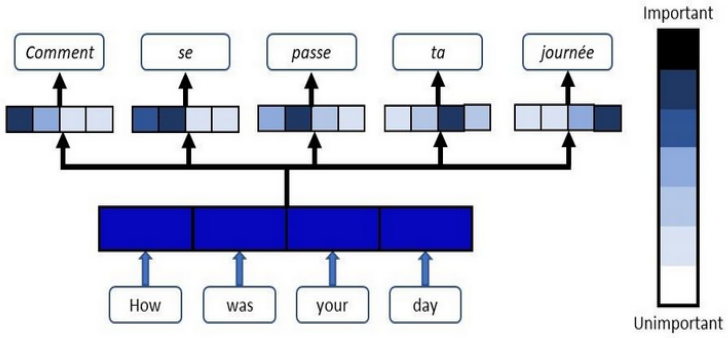

Figura 3.1: Ejemplo de atención en secuencia de palabras Fuente: https://blog.floydhub.com/attention-mechanism/

En la siguiente figura pueden verse las dos implementaciones propuestas por Bahdanau y Luong.

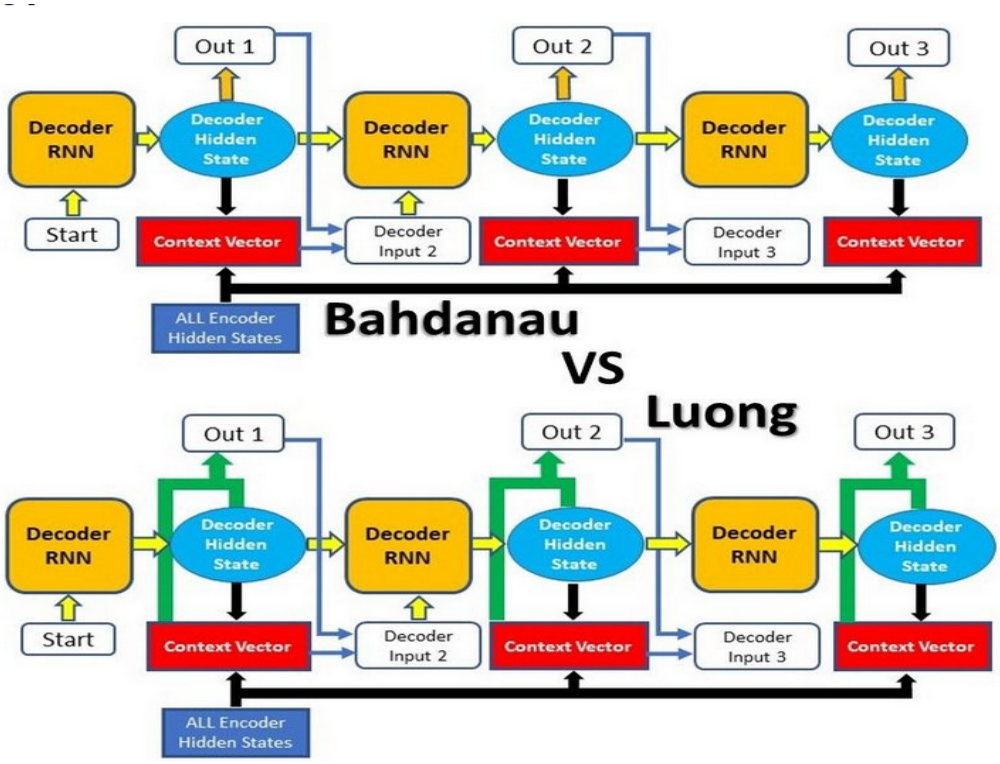

Figura 3.2: Bahdanau Attention y Luong Attention. Fuente: https://blog.floydhub.com/attention-mechanism/

A continuación profundicaremos en estas dos implementaciones.

El primer tipo de Atención, llamada comúnmente Additive Attention, y descrita por Bahdanau, alinea el decoder con las partes relevantes del input. Los pasos descritos en el paper son

los siguientes

1. El encoder produce estados ocultos para cada elemento de la secuencia de entrada.
2. Se calculan puntuaciones de alineamiento entre el estado oculto previo a un decoder y los estados ocultos de los encoder calculados en el paso anterior.
3. Las puntaciones de alineamiento de cada estado oculto del encoder se combinan y generán un único vector que se le aplica la función softmax.
4. Cada estado oculto del encoder y sus repectivas puntuaciones de alineamiento se multiplican para generar lo que llama Vector de contexto.
5. El vector de contexto se concatena con la salida del decoder anterior y sirve de entrada de la RNN del decodificador junto al estado oculto del decodificador anterior para producir una salida.
6. Los pasos de 2-5 se repiten para cada paso hasta que se llega a la longitud máxima especificada.

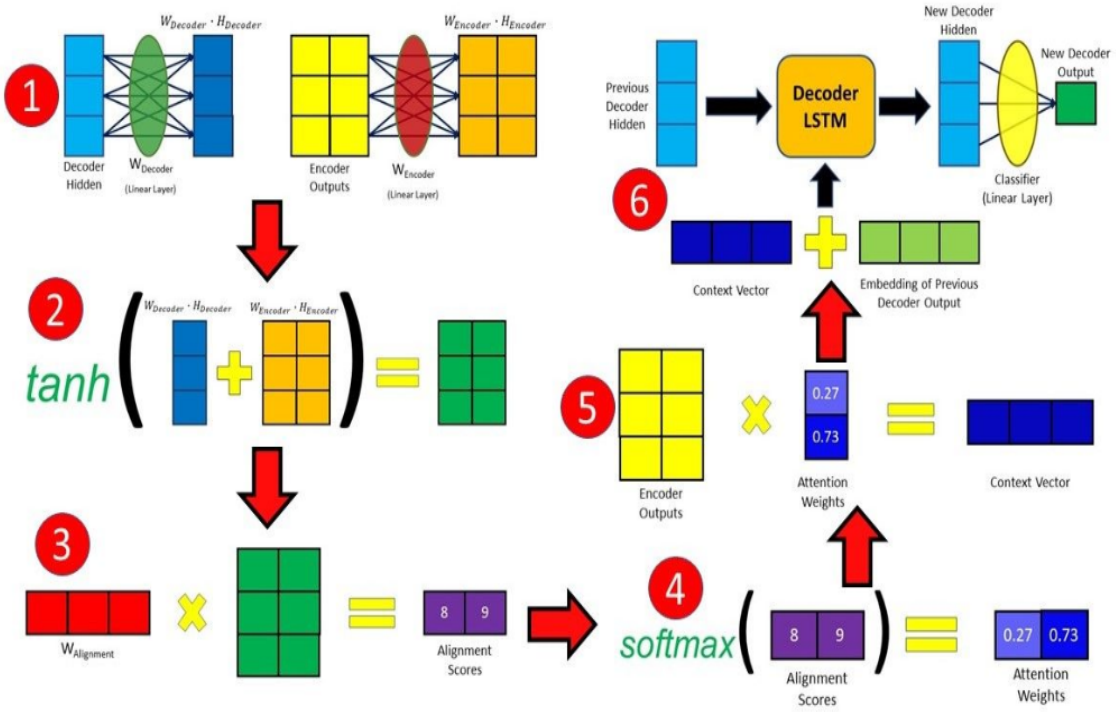

Figura 3.3: Flujo de mecanismo atencion de Bahdanau Fuente: https://blog.floydhub.com/attention-mechanism/

El segundo tipo de atención mencionado anteriormente es el propuesto por Luong y se suele referir como **Atención multiplicativa**. Las dos principales diferencias son que la forma de calcular la puntuacion de alineamiento es calculada y la posición donde se introduce el mecanismo de atención en el decoder. En esta arquitectura, el vector del contexto se utiliza solamente despues de la que RNN produzca el output correspondiente a ese paso. El proceso sería el siguiente:

1. El encoder produce estados ocultos para cada elemento de la secuencia de entrada.
2. El estado oculto previo del decoder y la salida previa del decoder se pasan a la **RNN del decoder** para generar un nuevo estado oculto
3. Se utiliza el estado oculto calculado en el paso anterior y los estados ocultos del encoder para calcular **las puntuaciones de alineamiento**.
4. Las puntuaciones de cada estado oculto del encoder se combinan en un único vector y se le aplica la función **softmax**.
5. Se multiplican los estados ocultos y sus correspondientes puntuaciones de alineación para generar el **vector de contexto**.
6. El vector de contexto se concatena con el estado oculto generado en el paso 2 y se pasa a una capa densa para producir una nueva salida
7. Los pasos 2-6 del procesos se repiten hasta que se produce un token o la salida supera la longitud máxima especificada.

A continuación, veremos la arquitectura Transformers, y cuales son las diferencias sobre los modelos previos secuencia a secuencia con mecanismos de atención aplicados. Resumiendo, los mecanismos de atención extraen información de una secuencia, generando una suma ponderada de todos los estados anteriores del codificador y facilitando al decoder que en cada paso de tiempo pueda asignar mayor o menor importancia a un determinado elemento de la entrada, es decir, puede enfocar la atención sobre un elemento de la entrada para predecir el siguiente elemento de la salida. Pero el problema principal, es que es un proceso secuencial, y cada elemento de la secuencia se debe recorrer uno a uno, ya que cada paso depende del anterior y esto computacionalmente impide el paralelismo y no es eficiente. Esto es precisamente, lo que cambia la arquitectura transformers, ya que se extraen las características de cada item usando un mecanismo de self-attention para extraer como es de importante ese elemento en relación con el resto de elementos de la secuencia, y para hacer esto no es necesario recurrencias por lo que el método es paralelizable y eficiente.

Iremos desglosando cada parte de la arquitectura Transformes, en primer lugar observando el diagrama del paper .ªttention is all you need"hay dos bloques principales, la parte izquierda correspondiente al encoder y la parte derecha al decoder. Ambos, contienen bloques de atención y feed-forward network.

GRAFICO de arquitectura transformer

3.1. Self-Attention

Self-attention es una operación secuencia a secuencia, donde para una entrada de vectores $x_1, x_2, ..., x_t$ se genera otra salida de vectores $y_1, y_2, ..., y_t$.

Para producir la salida y_i, la operación de self-attention toma una media ponderada sobre todos los vectores de entrada.

$$y_i = \sum w_{ij} x_j$$

El peso w_{ij} no es un parámetro, se deriva de una función sobre x_i u x_j. La forma mas simple para esta función es el producto escalar.

$$w'_{ij} = x_i^\intercal x_j$$

El producto escalar nos da un valor entre menos infinito e infinito, por lo que se aplica softmax para mapear el valor entre $[0, 1]$ y asegurar que la suma de todos los pesos sea 1.

$$w_i j = \frac{\exp w'_i j}{\sum_j \exp w'_i j}$$

Es importante destacar dos propiedades sobre el mecanismo de autoatención.

- No hay parámetros, todo lo que hace la autoatención está totalmente definido por el mecanimos que genera la secuencia de entrada.
- La entrada se trata como un conjunto de datos y no como una secuencia, ya que las operaciones que se hacen como las sumas ponderadas el orden no se tiene en cuenta.

Todo vector de entrada x_i participa en el proceso de tres formas distintas, que el paper de Vaswani et al. denominan Key, Query y Value

3.1.1. Queries, keys y values

Por cada vector de entrada, podemos distinguir tres roles distintos: el Query, el Key y el Value.

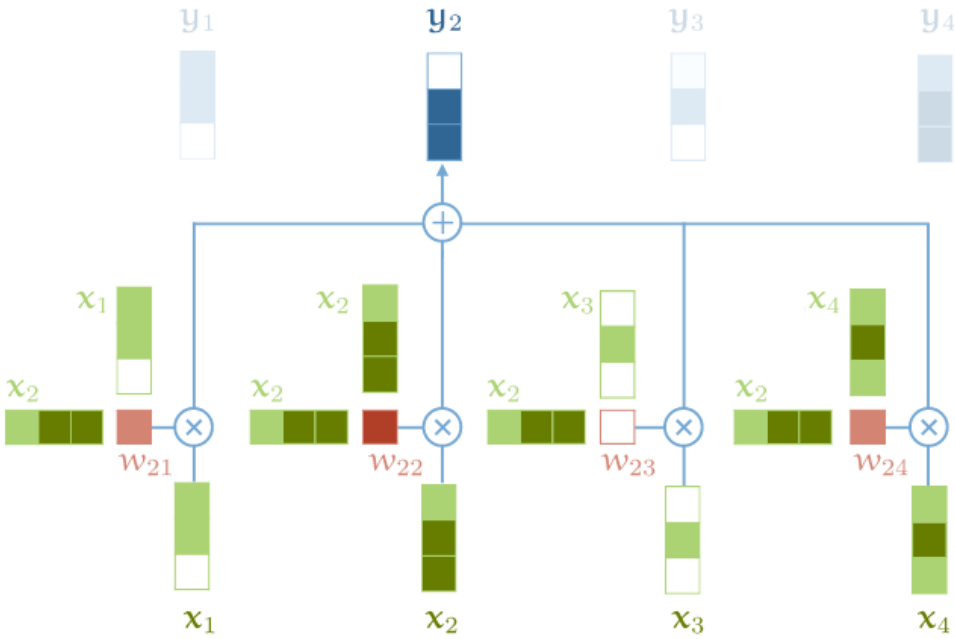

Figura 3.4: Mecanismo self attention Fuente: https://peterbloem.nl/blog/transformers/

- Se compara con el resto de vectores para establer su salida y_i (Query).
- Participa como Key para establecer los pesos para generar la salida y_j
- Se usa como parte de la suma ponderada para generar el vector Value de salida.

Podemos pensar en que tenemos 3 matrices de dimension k x k, W_q, W_k, W_v y que generan tres tranformaciones lineales.

$$q_i = W_q x_i \quad k_i = W_k x_i \quad v_i = W_v x_i$$
$$w'_{ij} = q_i^\intercal k_j$$
$$w_{ij} = softmax(w'_{ij})$$
$$y_i = \sum_j w_{ij} v_j$$

3.1.2. Dot Product

La función de softmax es sensible a secuencias muy largas de valores, para evitar que se vea afectado por la dimesion k de la secuencia, el producto escalar se escala difiriendo ligeramente del propuesto anteriormente.

$$w'_{ij} = \frac{q_i^\intercal k_j}{\sqrt{k}}$$

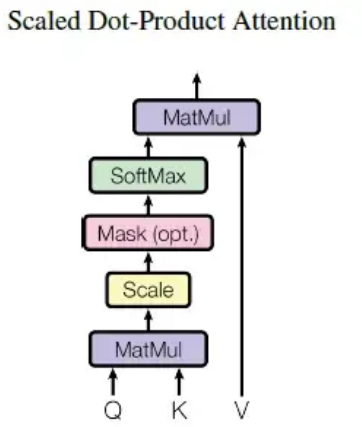

Figura 3.5: "Attention is all you need" paper by Vaswani, et al., 2017

3.1.3. Multi-head attention

El problema del mecanismo de autoatención visto hasta ahora es que no tenemos en cuenta el orden de las secuencia de entrada, ya que la información se trabaja de forma conjunta. El multi-head Attention divide los vectores de entrada en tantos r chunks como heads hayamos definido, por ejemplo, si el vector de entrada tiene dimensión 256, y tenemos 8 cabeceras, divide cada vector en subvectores de dimensión 32, esto quiere decir que se generan Matrices W_q^r, W_k^r, W_v^r de dimension 32x32, que posteriormente se concatenan para generar una única matriz que se pasa a la siguiente capa (Feed-Forward layer).

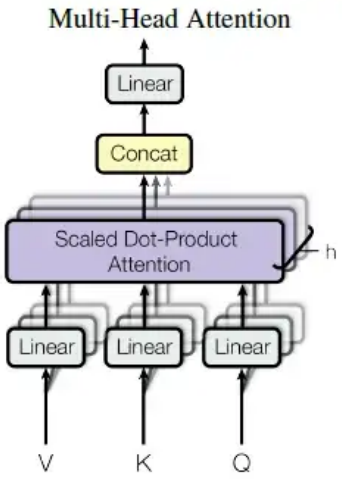

Figura 3.6: "Attention is all you need" paper by Vaswani, et al., 2017

3.1.4. Positional encoding

Los mecanismos de atención por si solos no tienen en cuenta el orden de entrada de las secuencias, por lo que se usa una función para mapear la posición en la oración a un vector de valor real. La red aprenderá a utilizar esta información, en el paper original lo hace usando la función sinusoidal.

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

Capítulo 4

Diseño e implementación

4.1. Introducción

Se ha seleccionado como dataset de pruebas el CBIS-DDSM, (Subconjunto curado de imágenes mamarias de DDSM), que incluye imágenes descomprimidas, seleccionadas y curadas de mamografías. El CBIS-DDSM (Subconjunto curado de imágenes mamarias de DDSM) es una versión actualizada y estandarizada de la base de datos digital para mamografías de detección (DDSM). El DDSM es una base de datos de 2620 estudios de mamografía de película escaneada. Contiene casos normales, benignos y malignos con información patológica verificada. El conjunto de datos proporciona un tamaño de conjunto de datos capaz de analizar los sistemas de apoyo a la toma de decisiones en mamografía. Otros de los motivos que nos han ayudado a seleccionar este conjunto de datos como veremos mas adelante es que ya vienen etiquetados los subconjuntos de entrenamiento y test balanceados correctamente en función de la dificultad del diagnóstico

4.2. Entorno de trabajo

Para el entorno de trabajo al no disponer de GPU en el equipo doméstico, se ha trabajado con Google Colab ya que es un entorno que dispone de GPU, no necesita ningún tipo de instalación y existen librerias compatibles con Drive, esto último es importante, pues nos ha permitido descargar el juego de pruebas una sola vez y dejarlo preparado en nuestro drive, por lo que en sucesivas ejecuciones solo teníamos que montar drive como una unidad de trabajo y ya tenemos acceso. El trabajo se ha estructurado todo en dos notebook:

- **LecturaCBIS_DDSM** Se descarga el juego de datos de kaagle, se hacen las tareas propias de limpieza y análisis, y por ultimo se implementa una sencilla red convulocional

para explorar unos resultados iniciales.

- **ViT** Se implementa un modelo basado en la arquitectura ViT (Vision transformers).

Todo el trabajo está disponible en el siguiente repositorio de github.

4.3. Carga de datos

El conjunto de datos original se puede acceder a través de The Cancer Image Archive en formato DICOM y ocupando 163.6 Gb, pero existe la posibilidad de descargarlo en kaagle ya en formato jpeg con tamaño de 6.3 Gb. A través de la libreria de TensorFlow, tambien permite utilizar este dataset, pero hay que realizar una descarga manual y convertirlo posteriormente a png. Por lo que finalmente se ha decidido descargarlo via API de kaggle.

El dataset descargado se compone de dos directorios.

- **jpeg** Se ubican todos los imagenes
- **csv** Se ubican 6 ficheros con los metadatos del dataset
  - calc_case_description_test_set.csv
  - calc_case_description_train_set.csv
  - dicom_info.csv
  - mass_case_description_test_set.csv
  - mass_case_description_train_set.csv
  - meta.csv

En el fichero dicom_info.csv se encuentran los metadatos sobre todo el conjunto de datos, es una tabla con 38 columnas, y nos interesan principalmente de este fichero la ubicación del fichero **image_path** , el tipo de imagen **SeriesDescription**, que puede ser cropped images o full mammogram images, y la columna **PatientID** que nos dice si la imagen corresponde a una *masa* o una *calcificación*.

	image_path	SeriesDescription	PatientID
0	/content/drive/My Drive/Colab Notebooks/db/cbi...	cropped images	Mass-Training_P_01265_RIGHT_MLO_1
1	/content/drive/My Drive/Colab Notebooks/db/cbi...	full mammogram images	Mass-Training_P_01754_RIGHT_CC
2	/content/drive/My Drive/Colab Notebooks/db/cbi...	full mammogram images	Calc-Training_P_00232_RIGHT_CC
3	/content/drive/My Drive/Colab Notebooks/db/cbi...	cropped images	Calc-Test_P_00562_LEFT_CC_2
4	/content/drive/My Drive/Colab Notebooks/db/cbi...	NaN	P_00562_LEFT_CC_2.dcm

Figura 4.1: Metadatos fichero dicom (Fuente propia)

4.4. Análisis de datos

En el conjunto de datos se pueden distinguir varios tipos de ficheros, mamografias completas, imagenes recortadas y imagenes ROI (Region of Interest). Para el proposito de este práctica, nos hemos quedado con las segundas. Dentro de ellas existen dos tipos de anomalias detectadas, de masa y calcificaciones. En el dataset vienen ya preparados conjuntos de entrenamiento y test para cada una de ellas. En este estudio, dado que el conjunto de datos es pequeño se han juntado ambos subconjuntos. Dentro de los diagnosticos, se definen tres patalogias:

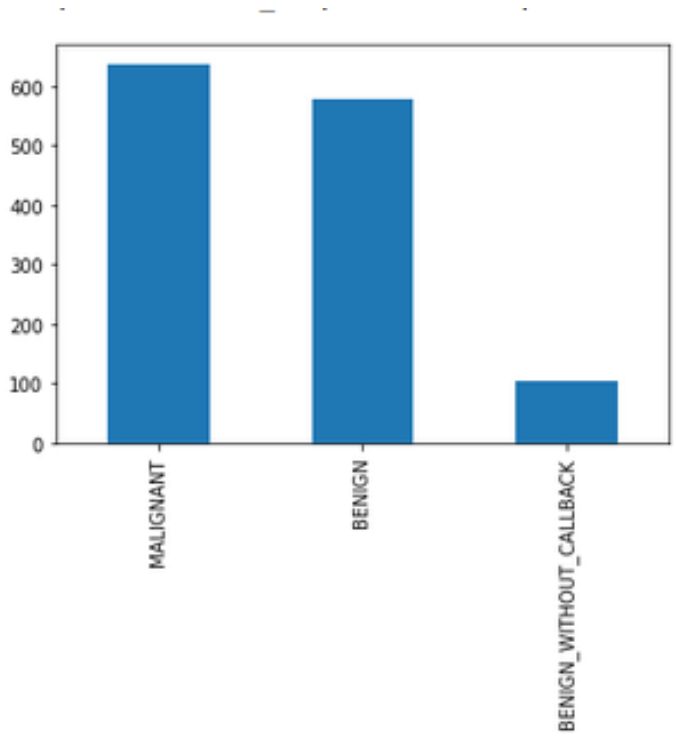

Figura 4.2: Distribución de Patologías (Fuente propia)

Se ha equiparado la categoría de BEGING_WITHOUT_CALLBACK con BENIGN, para al

final tener un problema de clasificación con dos clases.

4.5. Preparación de datos y data augmentation

El sobreajuste generalmente ocurre cuando hay una pequeña cantidad de ejemplos de entrenamiento, como es nuestro caso. Mediante la técnica de data augmentation se generan datos de entrenamiento adicionales a partir de los ejemplos existentes al aumentarlos mediante transformaciones aleatorias. Algunas de las transformaciones que se pueden hacer son:

- Aumento de desplazamiento horizontal y vertical, moviendo todos los píxeles de la imagen en una dirección, como horizontal o verticalmente, manteniendo las mismas dimensiones de la imagen.
- Aumento de volteo horizontal y vertical, pivotando todos los pixeles respecto a un eje imaginario.
- Rotacion aleatoria. Una transformación de rotación gira aleatoriamente la imagen en el sentido de las agujas del reloj un número determinado de grados de 0 a 360.
- Aumento de brillo aleatorio. Modificar aleatoriamente el brillo de una imagen hacia mas oscuro o mas claro.

En nuestro caso hemos utilizado una serie de capas secuenciales mediante la librería keras.

```
[ ] data_augmentation = keras.Sequential(
        [
            layers.Normalization(),
            layers.Resizing(image_size, image_size),
            layers.RandomFlip("horizontal"),
            layers.RandomBrightness(0.1, value_range=(0.0, 1.0))
            layers.RandomRotation(factor=0.02),
            layers.RandomZoom(
                height_factor=0.2, width_factor=0.2
            ),
        ],
        name="data_augmentation",
    )
```

Figura 4.3: Data Augmentation(Fuente propia)

4.6. Aplicacion red CNN

Se ha programado una sencilla red neuronal convulocional para tener alguna referencia para poder comparar los resultados con la ViT.

```
model = tf.keras.Sequential([
tf.keras.layers.Conv2D(32, (3, 3), padding = 'same', activation = 'relu', input_shape = (columnas,filas,  3)),
tf.keras.layers.MaxPooling2D(strides = 2),
tf.keras.layers.Conv2D(64, (3, 3), padding = 'same', activation = 'relu'),
tf.keras.layers.MaxPooling2D((3, 3),strides = 2),
tf.keras.layers.Conv2D(128, (3, 3), padding = 'same', activation = 'relu'),
tf.keras.layers.MaxPooling2D((3, 3),strides =2),
tf.keras.layers.Conv2D(128, (3, 3), padding = 'same', activation = 'relu'),
tf.keras.layers.MaxPooling2D((3, 3),strides =2),
tf.keras.layers.Flatten(),
tf.keras.layers.Dense(128, activation = 'relu'),
tf.keras.layers.Dense(2, activation = 'softmax')
])
model.summary()
```

Figura 4.4: Creación de una red convulocional (Fuente propia)

4.7. Implementación arquitectura Vision Transformers

Una vez introducidos en el capitulo anterior los componentes básicos de una arquitectura seq2seq basada en la atención, y viendo como el uso del positional encoding en una arquitectura basada en Transformers hace innecesario que los datos se procesen de forma secuencial, especificaremos como construir una arquitectura de Transformers orientada a la clasificación de imágenes, Vision Transformers (ViT), introducida por primera vez en el paper An image is worth 16x16 words [4].

En una arquitectura Transformers tradicional el codificador asigna una secuencia de entrada de representaciones de símbolos $(x_1, \ldots, x_n)$ a una secuencia de representaciones continuas $z = (z_1, \ldots, z_n)$. Dado z, el decodificador genera una secuencia de salida $(y_1, \ldots, y_m)$ de símbolos, un elemento a la vez. En cada paso, el modelo es autorregresivo, consumiendo los símbolos generados previamente como entrada adicional al generar el siguiente.

El codificador lee los datos de entrada y produce una representación intermedia de la entrada. El decodificador decodifica esta representación intermedia paso a paso y genera la salida.

El componente principales de ViT, es el enconder y el bloque de Multi-head self-attention dentro de él, pero previamente hay que generar parches a partir de imágenes y agregar incrustaciones posicionales.

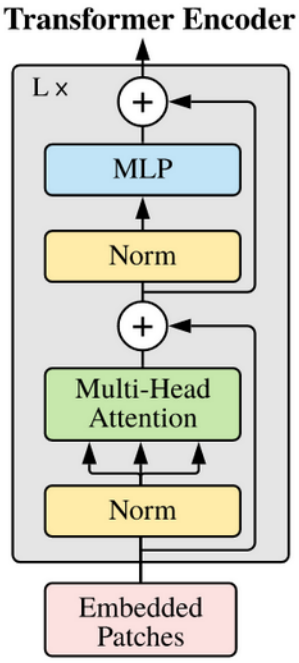

Figura 4.5: Fuente: https://www.kaggle.com/code/suvoooo/understand-implement-vision-transformer-tf2-0/

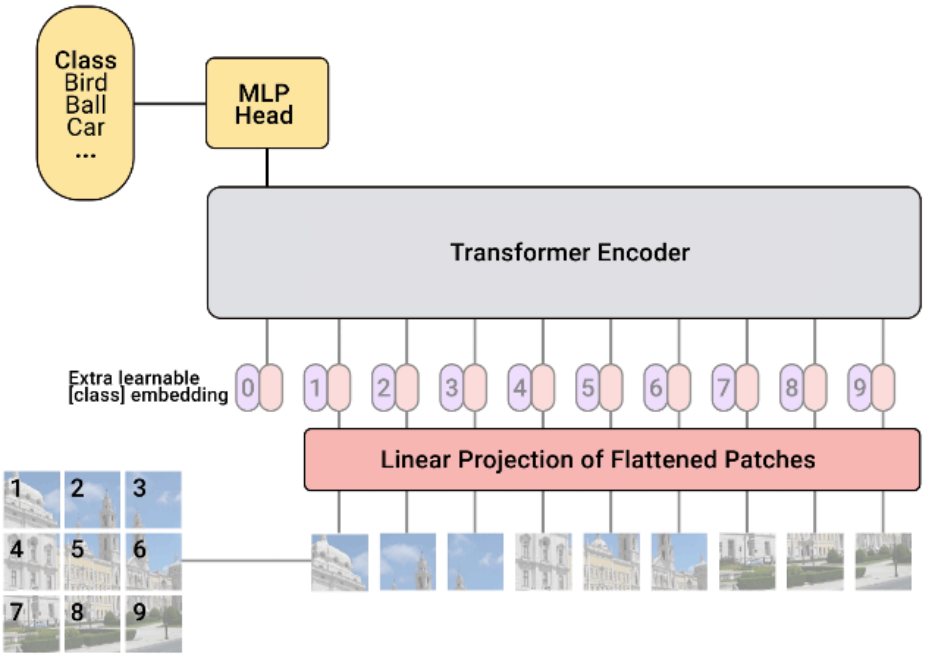

Figura 4.6: Fuente: https://www.kaggle.com/code/suvoooo/understand-implement-vision-transformer-tf2-0/

Todas las partes que se verán a continuación tienen su propia codificación en la implementación de la arquitectura.

4.7.1. Generacion patches

Este método dada una imagen, $x \in \mathbb{R}^{H \times W \times C}$, la convierte en una secuencia de parches $x_p \in \mathbb{R}^{N \times P \times P \times C}$, donde (H,W) es la altura y el ancho de la imagen original, C es el número de canales, (P,P) es la resolución de cada parche de imagen, y $N = HW/P^2$ es el número resultante de parches, que también sirve como longitud de secuencia de entrada efectiva para el Transformador.

A continuación puede verse un ejemplo sobre una imagen del conjunto de datos seleccionado

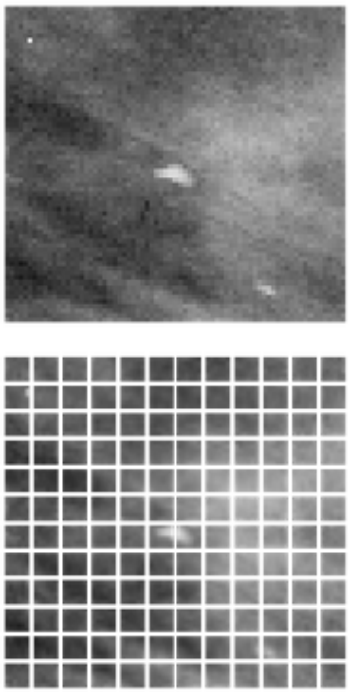

Figura 4.7: Fuente propia

La implementacion que se ha seleccionada basada en la documentación de Keras, primero debemos proyectar los parches (a través de una capa densa) a la misma dimensión que espera recibir la capa MultiHeadAttention.

4.7.2. Encoder

Los patches de imagen que recibe el encoder son básicamente tokens de una secuencia, de hecho este bloque es idéntico al propuesto por Vaswani et al. (2017), pero cambia el tipo de

entrada. El Encoder transformará linealmente cada patch al proyectarlo en un vector de tamaño proyección_dim. Además, agregando informacion en el vector proyectado sobre la posición de cada path, para hacer esto hemos desarrollado la clase PathEncoder, que se encarga de proyectar cada uno de los parches y crea el embedding de la posición que ocupa respecto a la secuencia completa.

```
: class PatchEncoder(layers.Layer):
    def __init__(self, num_patches, projection_dim):
        super(PatchEncoder, self).__init__()
        self.num_patches = num_patches
        self.projection = layers.Dense(units=projection_dim)
        self.position_embedding = layers.Embedding(
            input_dim=num_patches, output_dim=projection_dim
        )

    def call(self, patch):
        positions = tf.range(start=0, limit=self.num_patches, delta=1)
        encoded = self.projection(patch) + self.position_embedding(positions)
        return encoded
```

Figura 4.8: Fuente propia

4.7.3. Creacion del modelo

Hasta ahora los pasos que hemos dado, han sido aplicar una capa de data augmentation, parchear las imagenes y proyectarles linealmente agregando el embedding de la posición. Los siguientes pasos serían:

1. x1= Normalizar el resultado anterior
2. x2 = Procesarso y aplanarlo por la capa MultiHeadAttention y normalizar la salida
3. x3 = Aplicar la capa MLP
4. (x2,x3) Sumar las dos capas anteriores
5. Aplanar, aplicar dropout y una capa densa con el número de clases resultantes

Todo esto lo hacemos en la función createVitClassifier.

4.7.4. Entrenamiento del modelo

Por último solo queda entrenar el modelo y evaluar los resultados. El optimizador propuesto por defecto en la documentación de Keras que se ha seguido, es AdamW, que hereda del optimizador Adam y recibe dos parámetros:

```python
def create_vit_classifier():
    inputs = layers.Input(shape=input_shape)
    # Augment data.
    augmented = data_augmentation(inputs)
    # Create patches.
    patches = Patches(patch_size)(augmented)
    # Encode patches.
    encoded_patches = PatchEncoder(num_patches, projection_dim)(patches)

    # Create multiple layers of the Transformer block.
    for _ in range(transformer_layers):
        # Layer normalization 1.
        x1 = layers.LayerNormalization(epsilon=1e-6)(encoded_patches)
        # Create a multi-head attention layer.
        attention_output = layers.MultiHeadAttention(
            num_heads=num_heads, key_dim=projection_dim, dropout=0.1
        )(x1, x1)
        # Skip connection 1.
        x2 = layers.Add()([attention_output, encoded_patches])
        # Layer normalization 2.
        x3 = layers.LayerNormalization(epsilon=1e-6)(x2)
        # MLP.
        x3 = mlp(x3, hidden_units=transformer_units, dropout_rate=0.1)
        # Skip connection 2.
        encoded_patches = layers.Add()([x3, x2])

    # Create a [batch_size, projection_dim] tensor.
    representation = layers.LayerNormalization(epsilon=1e-6)(encoded_patches)
    representation = layers.Flatten()(representation)
    representation = layers.Dropout(0.5)(representation)
    # Add MLP.
    features = mlp(representation, hidden_units=mlp_head_units, dropout_rate=0.5)
    # Classify outputs.
    logits = layers.Dense(num_classes)(features)
    # Create the Keras model.
    model = keras.Model(inputs=inputs, outputs=logits)
    return model
```

Figura 4.9: Creación del modelo (Fuente propia)

1. Learning rate: Este parámetro es la tasa de aprendizaje, e indica cada cuanto son actualizados los pesos de nuestra arquitectura. En la documentación recomienda que para fotos muy parecidas como parece nuestro caso se seleccionen valores bajos, hemos seleccionado inicialmente un valor de 0.001.

2. Weight decay: Este parámetro se encarga de penalizar los cambios grandes en los pesos con un determinado valor (menor que uno) si estos superan el umbral. Con esto se consigue el entrenamiento sea más lineal y no tendrá grandes picos durante el entrenamiento evitando entre otras cosas el overfitting. El valor usado fue 0.001

4.7.5. Red convulocional

Para poder tener una referencia con otra arquitectura en el mismo entorno de trabajos y exactamente con el mismo conjunto de datos se ha desarrollado una red convulocional, este es el resumen de las capas de las que está compuesta.

4.7.6. Evaluación del módelo

Se han generado las gráficas habituales de la evolución de la precision y funciones de perdida a través de las sucesivas iteraciones para evaluar el modelo, que se comentarán en el siguiente capitulo

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_36 (Conv2D)           (None, 256, 256, 32)      896

max_pooling2d_32 (MaxPoolin  (None, 128, 128, 32)      0
g2D)

conv2d_37 (Conv2D)           (None, 128, 128, 64)      18496

max_pooling2d_33 (MaxPoolin  (None, 63, 63, 64)        0
g2D)

conv2d_38 (Conv2D)           (None, 63, 63, 128)       73856

max_pooling2d_34 (MaxPoolin  (None, 31, 31, 128)       0
g2D)

conv2d_39 (Conv2D)           (None, 31, 31, 128)       147584

max_pooling2d_35 (MaxPoolin  (None, 15, 15, 128)       0
g2D)

flatten_10 (Flatten)         (None, 28800)             0

dense_20 (Dense)             (None, 128)               3686528

dense_21 (Dense)             (None, 2)                 258

=================================================================
Total params: 3,927,618
Trainable params: 3,927,618
Non-trainable params: 0
```

Figura 4.10: Resumen de la red convulocional (Fuente propia)

Capítulo 5

Evaluación del modelo y resultados

5.1. Resultados

Para evaluar el modelo se han tenido en cuenta las siguientes métricas :

- **Accuracy**: Mide la precision del modelo, el total de aciertos entre el total de casos.

$$(TP + TN)/(TP + FP + TN + FN)$$

- **Funcion de perdida**: Nos proporciona una medida de la desviación entre las predicciones realizadas y los valores reales, nos informa de como de lejos o como de cerca estamos de que la probabilidad de acertar. En nuestro caso, al ser un problema de clasificación binaria, hemos usado como función de perdida una entropía cruzada binaria.

- **Matriz de confusión**: Permite visualizar de una forma muy intuitiva como estamos prediciendo los casos en función de la clase real a la que pertenecen.

Los datos están separados según el tipo de anomalía detectada, de masa o calcificación, se han evaluado los conjuntos de datos por separado y de forma conjunta para detectar si el tipo de anomalía es un factor determinante

Para poder comparar la arquitectura transformers con alguna referencia, se ha implementando una sencilla red convulocional que se ha probado en ambos conjuntos. En todas las arquitecturas con las que se ha trabajado se decidió que un tamaño óptimo era de 256x256 píxeles.

También se ha tomado como referencia el estudio A framework for breast cancer classification using Multi-DCNNs, en el que se aplican cinco arquitecturas distintas como AlexNet, GoogleNe o ResNet-18, estos datos hay que tomarlos con cautela pues el origen de datos de

imagenes con el que se trabaja varia ligeramente, ya que aparte del conjunto CBIS-DDSM, se ha añadido el conjunto de datos MIAS (mammographic image analysis society digital mammogram database). Estas 5 redes profundas se han entrenando utilizando el conjunto de datos de ImageNet, que tiene 1,2 millones de imágenes naturales en 1000 clases etiquetadas para posteriormente aplicar técnicas de transferencia de aprendizaje y que pueda usarse en cualquier problema de clasificación, reemplazando la última capa con una nueva capa para la clasificación de dos clases: benigno/maligno para el CBIS-DDSM, normal/anormal para el conjunto de datos MIAS.

El conjunto de datos del estudio referenciado se estructura de la siguiente manera:

			Training	**Testing**	**Total**
CBIS DDSM	**Benign**	2728	3691	1581	5272
	Malignant	2544			
MIAS	**Benign**	836	901	387	1288
	Malignant	452			

En este trabajo se ha analizado el conjunto de CBIS-DDSM, distinguiendo entre anomalías de masa y calcificaciones, los datos se distribuyen de la siguiente manera

			Training	**Testing**	**Total**
CBIS DDSM MASS	**Benign**	912	1318	378	1696
	Malignant	774			
CBIS DDSM CALC	**Benign**	1197	1544	326	1870
	Malignant	673			

En la siguiente tabla pueden verse los resultados obtenidos en el estudio objeto de comparación en el para el conjunto de muestras de masa de las dos vistas de mamografía: craneocaudal (CC) y mediolateral-oblicua (MLO).

DCNN Architecture	**DCNN Accuracy**	**Training Time**
AlexNet	74,68 %	6h, 30min
GoogleNet	76.01 %	12 h
ResNet-18	72,23 %	14h
ResNet-50	71,09 %	33h
ResNet-101	71,47 %	62h

Los resultados obtenidos con la red convulocional entrenada en nuestro estudio es bastante peor, solo del 62,17 %. Estos son los resultados medidos durante el entrenamiento:

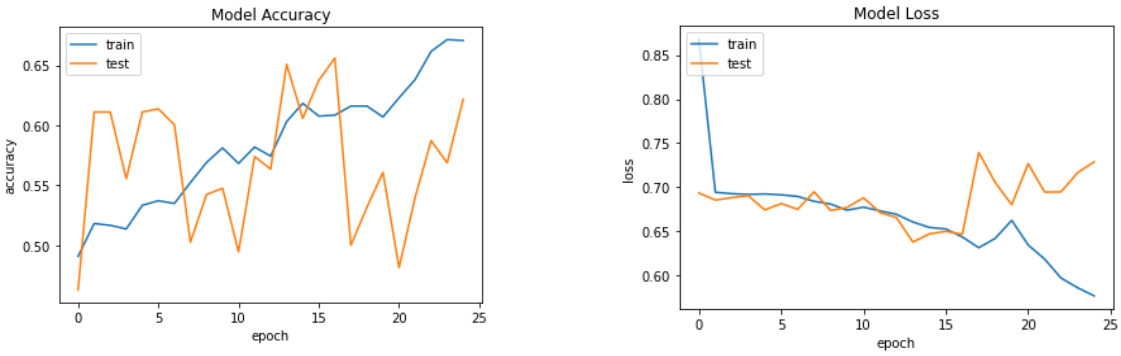

Figura 5.1: Métricas CNN sobre dataset de Masas

Observamos que el modelo (figura 5.1) si es capaz de ajustar los parámetros en el conjunto de entrenamiento, pero que en el conjunto de test el modelo oscila entre épocas sin tener una tendencia clara, parece claro que se está produciendo overfitting sin que el modelo aprenda, incluso la función de perdidas en las ultimas epochs tiene una tendencia a peor.

DCNN Architecture	DCNN Accuracy	Training Time
CNN local	62.17 %	30min

Observando la matriz de confusión, calculamos que la especificidad del modelo es del 78,35 %, y la sensibilidad es bastante peor, con un 36,73 %. Podemos concluir que esta red convulocional no está ofreciendo un buen resultado.

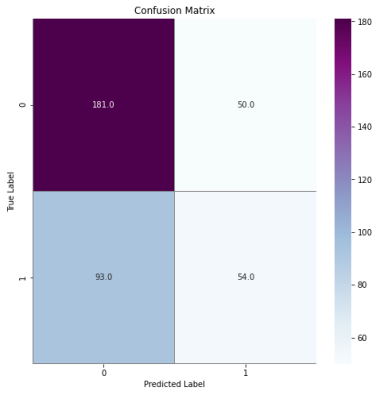

Figura 5.2: Matriz de confusion

Aplicando el estudio sobre el conjunto de datos de calcificaciones, los resultados son muy parecidos. Baja precisión, alta especificidad y baja sensibilidad.

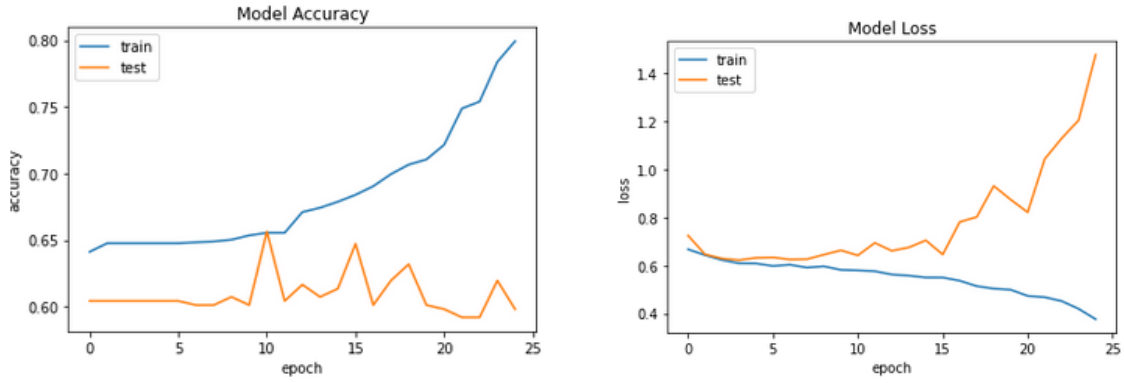

Figura 5.3: Métricas CNN sobre dataset de Calcificaciones

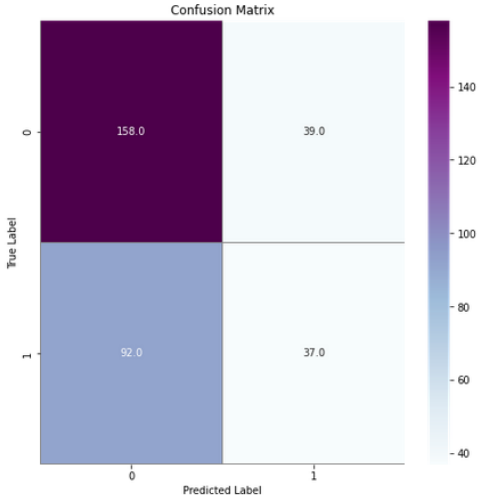

Figura 5.4: Matriz de confusión calcificaciones

5.1.1. Experimento 1 Vision Transformer, configuración por defecto

La configuración inicial con la que hemos ejecutado la arquitectura ViT ha sido la siguiente:

```
learning_rate = 0.001
weight_decay = 0.0001
batch_size = 256
num_epochs = 50
image_size = 256  # We'll resize input images to this size
patch_size = 32  # Size of the patches to be extract from the input images
num_patches = (image_size // patch_size) ** 2
projection_dim = 64
num_heads = 4
transformer_units = [
    projection_dim * 2,
    projection_dim,
]  # Size of the transformer layers
transformer_layers = 8
mlp_head_units = [2048, 1024]  # Size of the dense layers of the final classifier

num_classes = 2
input_shape = (256, 256, 3)
```

Figura 5.5: Configuración experimento 1

Observando la evolución de la precisión se observa que el modelo no está aprendiendo y que el conjunto de test oscila sin tener una tendencia. Esto en la red convulocional no pasaba, ya que aunque el modelo no era valido para el conjunto de test si que ajustaba los parámetros al menos para el conjunto de entrenamiento.

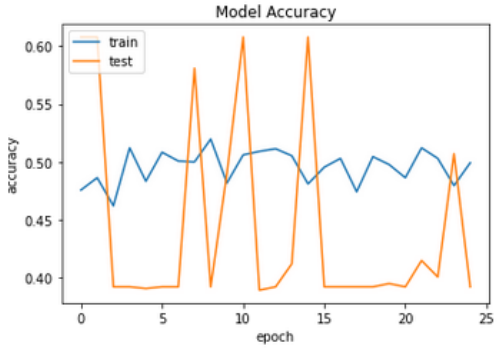

Figura 5.6: Precisión experimento 1

5.1.2. Experimento 2, aumentamos epochs a 50 y tamaños pequeño de patch

Cambiamos la configuración de epochs a 50, y el tamaño de los parches pasan de ser de 32x32 a 16x16, pero la GPU de la que disponemos en el entorno de google colab no puede procesarlo.

5.1.3. Experimento 3, bajamos la resolución de la imagen

Dejamos la configuración anterior, pero bajamos la resolución de la imagen a 128x128, en este caso si se puede procesar pero el resultado tampoco mejora.

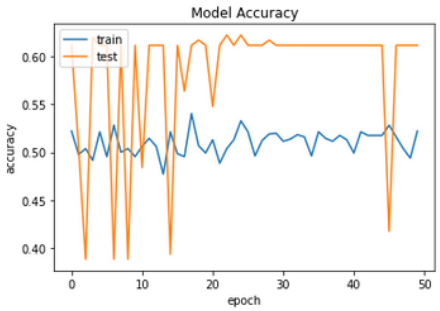

Figura 5.7: Precisión experimento 3

5.1.4. Experimento 5, data augmentation

Se incluye data augmentation para tener un conjunto de datos mejores y ver si mejora el módelo. En este caso se puede observar una mejoría sobre el conjunto de entrenamiento, aunque se este produciendo overfitting el modelo si es capaz de modificar los parámetros tal y como hacia en la CNN que desarrollamos.

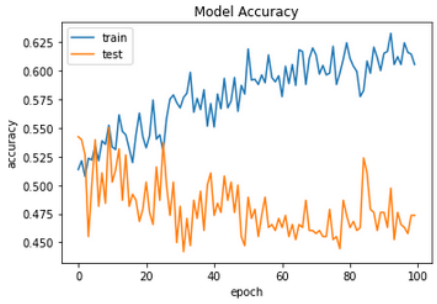

Figura 5.8: Precisión experimento Data Augmentation

5.1.5. Experimento 6, aumentamos capas del transformador

Se aumentan las capas del transformador de 8 a 12, y se consigue la mejor aproximación sobre el conjunto de test, pero la precisión sigue siendo pésima

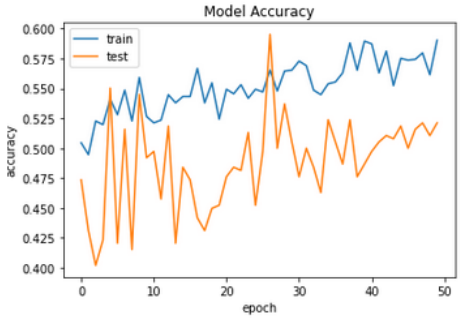

Figura 5.9: Precisión experimento aumentando capas del transformador

5.2. Conclusiones

En este estudio hemos podido comprobar como una arquitectura transformer necesita grandes conjuntos de datos para poder extraer propiedades locales mediantes los mecanismos de atención. Una red convulocional sencilla, sin aplicar tecnicas de data augmentation es capaz

de ajustar los parámetros de la red sobre el conjunto de entrenamiento, mientras que la arquitectura transformer incluso aplicando data augmentation y aplicando diversas configuraciones no ha podido aprender en varias de las pruebas o no alcanza el mismo nivel de precisión al trabajar con un conjunto de datos que no es lo suficientemente grande.

La conclusión de este estudio es que al menos para este tipo de problema de clasificación de imágenes radiológicas, en los que el conjunto de datos no es grande sigue siendo mas util aplicar arquitecturas basadas en redes convulocionales.

Tal y como dicen los autores del paper original .ªn Image is Worth 16x16 Words", [4], las arquitecturas ViT consumen mucha información, y la única forma de obtener mejores resultados que las redes neuronales convulocionales de ultima generacion es entrenar con conjuntos de datos de gran tamapo como JFT300M para luego aplicar Fine Tune a un número menor de clases.

5.3. Próximos estudios

La implementación inicial con la que he desarrollado la arquitectura transformers ha sido la propuesta en la documentación de Keras, y tal como hemos visto se queda muy corta, sería muy interesante en el futuro probar otras modelos pre-entrenados como los que se puede acceder a través de la librería TIMM, y aplicar directamente fine-tune.

En el estudio tomado como referencia del estado de arte además de usar DCNN, también usa SVM obteniendo unos resultados realmente buenos. A continuación pueden verse los resultados de combinar un modelo AlexNet con diferentes kernel de una SVM:

DCNN	Different Kernels	Accuracy (std)	AUC (std)	Sensitivity (std)	Specificity (std)	MCC (std)
AlexNet	Linear	91.3% (0.001)	0.97 (0)	0.918 (0.005)	0.911 (0.001)	0.829 (0.005)
	Quadratic	91.0% (0.002)	0.96 (0.004)	0.909 (0.006)	0.911 (0.001)	0.819 (0.007)
	Cubic	90.9% (0.002)	0.96 (0.001)	0.904 (0.004)	0.913 (0.005)	0.816 (0.007)
	Medium Gaussian	91.1% (0.001)	0.97 (0)	0.910 (0.001)	0.910 (0.001)	0.820 (0.001)
	Coarse Gaussian	89.2% (0.001)	0.96 (0.001)	0.884 (0.003)	0.899 (0.001)	0.782 (0.004)

Figura 5.10: Precisión AlexNet con SVM

En lineas futuras se podría analizar la posibilidad de utilizar fine tuning sobre modelos ViT preentrenados y combinarlos con SVM para medir si se obtienen resultados igual de buenos y en menor tiempo.

Otra linea de investigación muy interesante, es el uso de arquitecturas Vit De segunda generación como menciona el paper .[Ef]ficient Training of Visual Transformers with Small Datasets"[11], donde se combinan capas convulocionales con capas de atención.

Bibliografía

[1] Mohammed AlQuraishi. Machine learning in protein structure prediction. *Current Opinion in Chemical Biology*, 65:1–8, 2021. Mechanistic Biology * Machine Learning in Chemical Biology.

[2] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*, 2014.

[3] Yoshua Bengio, Pascal Lamblin, Dan Popovici, and Hugo Larochelle. Greedy layer-wise training of deep networks. In B. Schölkopf, J. Platt, and T. Hoffman, editors, *Advances in Neural Information Processing Systems*, volume 19. MIT Press, 2006.

[4] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, et al. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929*, 2020.

[5] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.

[6] Geoffrey E Hinton, Simon Osindero, and Yee-Whye Teh. A fast learning algorithm for deep belief nets. *Neural computation*, 18(7):1527–1554, 2006.

[7] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[8] L. Itti, C. Koch, and E. Niebur. A model of saliency-based visual attention for rapid scene analysis. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 20(11):1254–1259, 1998.

[9] Fei Jiang, Yong Jiang, Hui Zhi, Yi Dong, Hao Li, Sufeng Ma, Yilong Wang, Qiang Dong, Haipeng Shen, and Yongjun Wang. Artificial intelligence in healthcare: past, present and future. *Stroke and Vascular Neurology*, 2(4):230–243, 2017.

[10] Xiaoxuan Liu, Livia Faes, Aditya U Kale, Siegfried K Wagner, Dun Jack Fu, Alice Bruynseels, Thushika Mahendiran, Gabriella Moraes, Mohith Shamdas, Christoph Kern, Joseph R Ledsam, Martin K Schmid, Konstantinos Balaskas, Eric J Topol, Lucas M Bachmann, Pearse A Keane, and Alastair K Denniston. A comparison of deep learning performance against health-care professionals in detecting diseases from medical imaging: a systematic review and meta-analysis. *The Lancet Digital Health*, 1(6):e271–e297, 2019.

[11] Yahui Liu, Enver Sangineto, Wei Bi, Nicu Sebe, Bruno Lepri, and Marco Nadai. Efficient training of visual transformers with small datasets. *Advances in Neural Information Processing Systems*, 34:23818–23830, 2021.

[12] Minh-Thang Luong, Hieu Pham, and Christopher D Manning. Effective approaches to attention-based neural machine translation. *arXiv preprint arXiv:1508.04025*, 2015.

[13] Warren S. McCulloch and Walter Pitts. A logical calculus of the ideas immanent in nervous activity. *Bulletin of Mathematical Biology*, 52(1):99–115, 1990.

[14] SEOM Sociedad Española Oncología medica. Las cifras del cancer en españa. 2022. https://seom.org/images/LAS_CIFRAS_DEL_CANCER_EN_ESPANA_2022.pdf.

[15] Volodymyr Mnih, Nicolas Heess, Alex Graves, and Koray Kavukcuoglu. Recurrent models of visual attention. In *Proceedings of the 27th International Conference on Neural Information Processing Systems - Volume 2*, NIPS'14, page 2204–2212, Cambridge, MA, USA, 2014. MIT Press.

[16] Zhaoyang Niu, Guoqiang Zhong, and Hui Yu. A review on the attention mechanism of deep learning. *Neurocomputing*, 452:48–62, 2021.

[17] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, Peter J Liu, et al. Exploring the limits of transfer learning with a unified text-to-text transformer. *J. Mach. Learn. Res.*, 21(140):1–67, 2020.

[18] Ronald A. Rensink. The dynamic representation of scenes. *Visual Cognition*, 7(1-3):17–42, 2000.

[19] Franck Rosenblatt. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, (6):386–408, 1958.

[20] Hinton Geoffrey E y Williams-Ronald J. Rumelhart, David E. Learning representations by back-propagating errors. *Nature*, (6088):533–536.

[21] Ilya Sutskever, Oriol Vinyals, and Quoc V Le. Sequence to sequence learning with neural networks. *Advances in neural information processing systems*, 27, 2014.

[22] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. Attention is all you need. In I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017.

[23] José Rolando Castro-Pomaquiza y María Magaly Castro-Pomaquiza. Rol del diagnóstico por imagen para la detención temprana del cáncer. *Polo del Conocimiento*, 7(1), 2022.