'Attention is all you need': Transformer para modelado de secuencias en NLP e Imágenes

Evangelina Garza Elizondo
Victor Manuel Gómez Espinosa
Centro de Investigación en Matemáticas. Unidad Monterrey
Email: evangelina.garza@cimat.mx/victor.gomez@cimat.mx

***Resumen*—En el presente reporte se exponen los detalles de una arquitectura basada únicamente en mecanismos de atención conocida como Transformer. Esta arquitectura se presentó por primera vez en 2017 por Vaswani et al. como una alternativa a las desventajas de los modelos utilizados hasta ese entonces, encontrando ventajas significativas. Dichas ventajas, así como algunas desventajas de esta arquitectura, se incluyen también. Por último, se presentan sus aplicaciones en tareas de procesamiento de lenguaje natural, específicamente en análisis de sentimientos, y clasificación de imágenes demostrando resultados satisfactorios.**

I. Introducción

Hasta hace unos años atrás, las arquitecturas de redes neuronales que utilizaban recurrencia eran el enfoque estándar para resolver tareas de modelado de secuencias y problemas de traducción.

En los modelos recurrentes, se generan estados ocultos h a un tiempo t como función del estado oculto previo a tiempo $t-1$, lo que permite alinear las posiciones de dicha secuencia. Esta manera de tratar el problema no permite su paralelización lo cual limita el entrenamiento de los modelos a la memoria disponible. Además se presenta en ellas el problema de pérdida de información para secuencias largas y el correspondiente problema con el desvanecimiento del gradiente.

A pesar de que se han explorado métodos para contrarrestar estas limitantes, como el empleo de redes convolucionales, el problema aún persiste. En el caso de las redes convolucionales, la desventaja que se presenta está relacionada a encontrar dependencias entre posiciones distantes de la secuencia, lo que requiere crecer la red en profundidad (agregar más capas) haciendola más compleja y por lo tanto más difícil de aprender.

Se ha visto que los mecanismos de atención son eficaces para modelar dependencias sin importar su distancia o posición en la secuencia. Esto ha hecho que se conviertan en el estado del arte para atacar problemas de traducción y modelado de secuencias. En el 2017, Vaswani et al. propusieron una nueva arquitectura de red que utiliza únicamente mecanismos de atención, dejando afuera tanto la recurrencia como la convolución. Los experimentos y resultados que presentaron en su artículo: "*Attention is all you need*"demostraron muy buenos resultados de traducción, además de que se permitió la paralelización del entrenamiento con lo cual se redujo el tiempo de procesamiento vs modelos normalmente utilizados.

Entre algunas de sus desventajas más importantes (y que aún es un área de investigación), es el incremento en el costo computacional que se presenta al manejar secuencias muy largas. Entre los modelos basados en transformers que se encuentran en el estado del arte para diversas aplicaciones en NLP se encuentran GPT-2 (Radford et al. 2018) para generar textos, BERT (Devlin et al. 2019) para embeddings, T5 (Raffel et al. 2020) para distintas tareas como Q&A, clasificación, traducción y resumen automático.

II. Arquitectura del Transformer

En esta sección se describe a profundidad la arquitectura del Transformer, que consiste en un codificador y decodificador que utilizan mecanismos de auto atención (Figura 1).

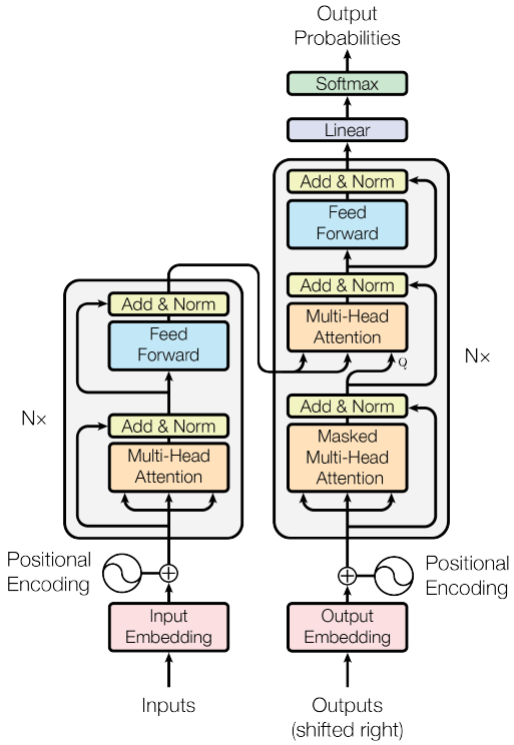

Figura 1. Representación del Transformer. Tomado y modificado de [12].

II-A. *Mencanismos de auto-atención*

Los mecanismos de auto-atención, en inglés *self-attention* o *intra-attention* obtienen una representación de las dependencias que existen dentro de una misma secuencia. Como se mencionó anteriormente, se ha demostrado en trabajos previos que este tipo de mecanismos tienen buen desempeño en tareas como comprensión lectora, resumen abstractivo, vinculación textual, entre otras. Algunos otros aspectos, además de los mencionados, que se logran mejorar utilizando este tipo de mecanismos son la complejidad computacional por capa y la cantidad de trabajo computacional paralelizable.

En las tareas de procesamiento de lenguaje natural, los mecanismos de auto-atención nos permiten obtener representaciones de cada uno de los tokens (palabras) de entradas, en las que se incluye información acerca del contexto de la misma. Esto se logra a través del producto punto de cada uno de los tokens con respecto a los otros. De esta multiplicación podemos obtener un peso con la importancia relativa, que se puede interpretar como el qué tanto aportan los demás a la traducción de cada uno de los tokens. Estos pesos se normalizan para obtener una densidad de probabilidad que es utilizada para crear una salida.

Esto se ve mejor ilustrado en la siguiente imagen, tomada de [12], en donde se muestra de manera general el mecanismo de auto atención.

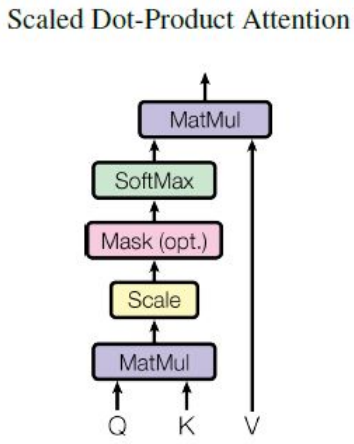

Figura 2. Representación del mecanismo de atención utilizado en el Transformer. Tomado de [12].

En esta imagen y en el artículo original de donde se obtuvo, se usa la notación **Q**,**K** y **V** para referirse a las Queries, Keys y Values como una analogía a las bases de datos. En este caso las queries son los parámetros que introducimos en el primero paso, los cuales se obtienen multiplicando los vectores de los tokens para los cuales queremos obtener las representaciones de importancia por la matriz de parámetros de Queries. Las keys serían el resto de los tokens multiplicados por la matriz de parámetros keys y los values, los vectores de representación (embeddings) originales multiplicados por la matriz de parámetros values. Esto se puede representar con productos puntos o con notación matricial como lo hace el artículo original en la ecuación:

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

Esto es, en una primera instancia tomamos los vectores de queries, keys y values de cada token. Estas se pueden acomodar como renglones de manera ordenada en las matrices Q,K y V. De esta manera, el producto de QK^T nos da el producto punto del vector q_i con respecto al vector k_j en la posición i, j de la matriz resultante. Este producto se escala por un factor $\sqrt{d_k}$, para evitar problemas de gradientes pequeños en la función softmax cuando el producto punto es muy grande debido a la alta dimensionalidad de las keys. Esto nos da un puntaje de la importancia relativa de cada token con respecto a las demás. La pasamos por una función softmax para obtener una distribución de probabilidad de estos pesos y por último la multiplicamos de nuevo por la matriz de valores (Values) para obtener una representación de salida.

II-B. *Auto atención múltiple*

Un solo mecanismo de auto atención permite obtener todas las diferentes relaciones entre los componentes de las secuencias, pero no te permite capturar por si solo la jerarquía, por lo cual haciendo uso de múltiples cabezas de atención es que se puede lograr lo anterior.

Esto se puede entender mejor viéndolo como una forma similar a como lo hacen las redes convolucionales, múltiples filtros o kernels que se aprenden durante el entrenamiento y que obtienen las características de las imágenes (Figura 3). Para realizar la auto atención múltiple, los values (V), keys (K) y queries (Q) se proyectan utilizando diferentes matrices $\left(W_i^Q, W_i^K, W_i^V\right)$ que se aprenden durante el entrenamiento de forma paralela, de igual forma las salidas de la auto atención se concatenan y nuevamente se proyectan utilizando otra matriz $\left(W^O\right)$ que igual se aprende, esto es lo que permite obtener información de diferentes sub espacios a diferentes posiciones (Figura 4).

$$MultiHead\,(Q, K, V) = Concat\,(head_i, ...head_h)\,W^O$$

Donde $W_i^Q, W_i^K \in R^{d_m \times d_k}, W_i^V \in R^{d_m \times d_v}, W^O \in R^{hd_v \times d_m}$. son las matrices de proyección y la auto atención es $head_i = Attention\left(QW_i^Q, KW_i^K, VW_i^V\right)$

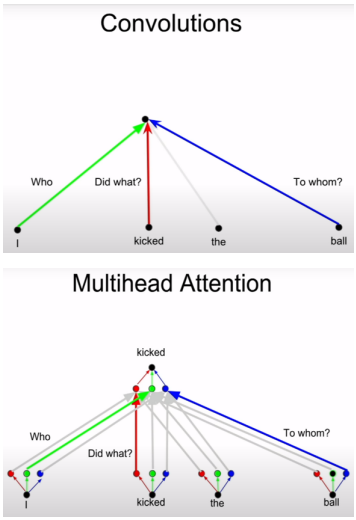

Figura 3. Representación de la convolución y del mecanismo de atención múltiple. Tomado y modificado de [11].

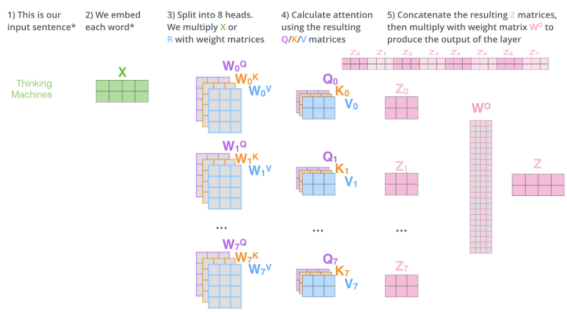

Figura 4. Representación del mecanismo de atención múltiple. Tomado y modificado de [1].

II-C. Entradas y salidas del Transformer

Las entradas al codificador y decodificador consisten en una suma de dos vectores: el embedding pre entrenado del token correspondiente (por ejemplo, word2vec) y el vector de codificación de posición (Figura 5).

Para el vector de posición se eligen funciones seno y coseno en base a la posición de la palabra (par o impar) y la dimensión del embedding, para crear un patrón específico para cada token según su dimensión y de esta forma también tomar en cuenta en el modelo la posición (Figura 6).

$$PE_{(pos,2i)} = \sin\left(pos \Big/ 10000^{2i/d_m}\right)$$

$$PPE_{(pos,2i+1)} = \cos\left(pos \Big/ 10000^{2i/d_m}\right)$$

En cuanto a la salida del transformer, se utiliza una capa lineal profunda que proyecta la salida del decodificador en un vector del tamaño del vocabulario, que representa pesos de cada palabra, seguido de una capa con función softmax sobre para convertirlas en probabilidades para predecir el token siguiente (Figura 7).

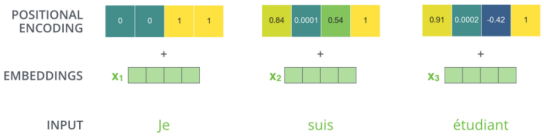

Figura 5. Representación de las entradas al Transformer. Tomado de [1].

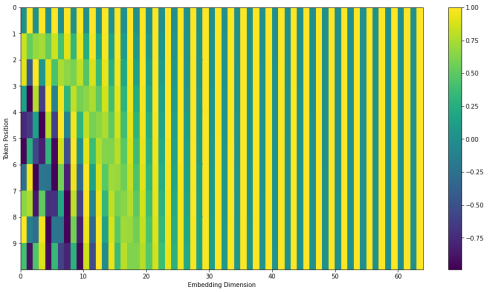

Figura 6. Representación del patrón de posición. Tomado de [1].

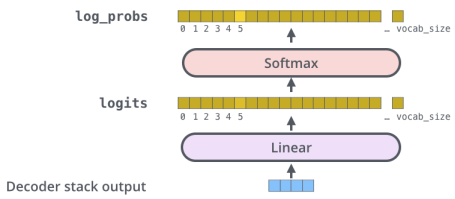

Figura 7. Representación de la salida del Transformer. Tomado de [1].

II-D. Codificador

El codificador del transformer consiste en un apilado de 6 bloques codificadores, donde cada bloque contiene dos subcapas, una de auto atención múltiple con 8 cabezas y otra subcapa profunda con función de activación ReLU.

$$FFN\left(x\right) = \max\left(0, xW_1 + b_1\right)W_2 + b_2$$

Al final de cada una de estas subcapas se agrega una capa que suma la entrada y salida de la capa previa y las normaliza (Figura 8).

$$LayerNorm\left(x + Sublayer\left(x\right)\right)$$

En el codificador, los keys, queries y values provienen del bloque previo de codificadores (o de la entrada), con lo cual se obtienen todas las relaciones con las salidas de los bloques anteriores (Figura 9).

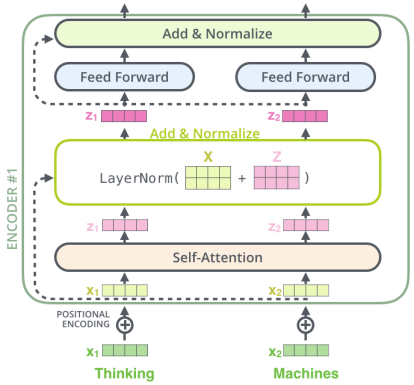

Figura 8. Representación de las subcapas del codificador. Tomado de [1].

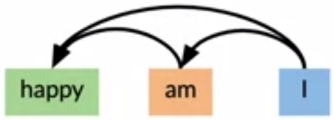

Figura 9. Representación de la auto atención en el codificador. Tomado de [7].

II-E. Decodificador

El decodificar es similar al codificador en su mayor parte, ya que también consiste en un apilado de 6 bloques de decodificadores, donde cada uno cuenta con subcapas de auto atención múltiple con 8 cabezas de atención y subcapas profundas con sus respectivas capas normalizadoras. Sin embargo, la diferencia aquí es que se tienen dos subcapas de auto atención múltiple, seguidas de la capa profunda.

En este caso, las subcapas de auto atención tienen algunas diferencias, la primera capa sus keys, values, queries provienen de la capa anterior de decodificadores (o de la entrada), con lo cual se obtienen todas las relaciones con las salidas de los bloques anteriores. Adicional a esto, se imponen valores infinitos negativos para todos los valores de entrada a la función softmax que corresponden a aquellas partes futuras en la secuencia de entrada del decodificador, para de esta forma obtener valores 0 en las probabilidades de estos tokens (Figura 10).

En la segunda capa de auto atención múltiple, sólo los queries provienen de la salida de la subcapa, ya que los keys y values provienen del codificador, con lo cual se busca obtener todas las relaciones existentes con todas las salidas del codificador (Figura 11).

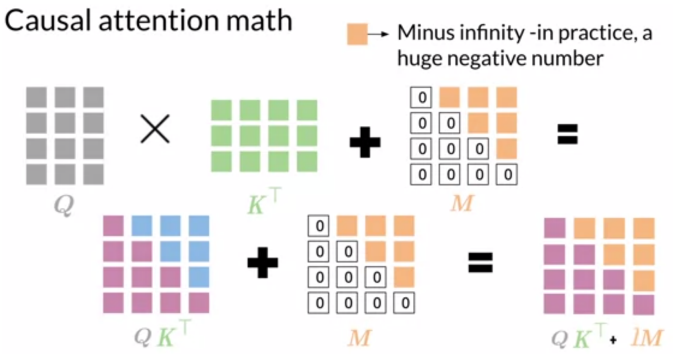

Figura 10. Representación del producto punto con la mascara (valores infinitos negativos) en la primera subcapa de auto atención múltiple del decodificador. Tomado de [7].

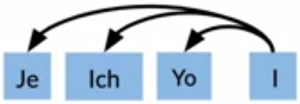

Figura 11. Representación de la auto atención en la segunda subcapa del decodificador. Tomado de [7].

II-F. Regularización

Al embedding que entra al transformer se le añade regularización dropout con tasa del 0.1. De la misma forma se agrega la misma tasa de regularización a la salida de cada bloque codificador y decodificador.

III. Aplicaciones del Transformer

En esta sección se discute la metodología y resultados en aplicaciones específicas en NLP e imágenes.

III-A. Análisis de sentimientos en textos

Cómo se mencionó en secciones anteriores, el transformer es útil para los modelos de tipo secuencia a secuencia, en los que es común emplear redes recurrentes. Algunas de las estructuras posibles son de muchos a uno o de muchos a muchos, dónde para el caso particular en NLP, para el primer tipo tenemos aplicaciones como análisis de sentimientos o clasificación de textos, y para el segundo tipo, generación de texto o traducción.

En este trabajo se utilizó el primer tipo de modelo, de muchos a uno, para la tarea de análisis de sentimientos en textos en inglés y en español, y dada la naturaleza de este tipo de modelos, sólo se requiere utilizar del transformer la parte del codificador.

Para los textos en inglés se utilizó el conjunto de datos IMDB disponible en la librería de Keras, que consiste en 25,000 observaciones de entrenamiento y 25,000 de validación, dónde cada una de estas observaciones corresponde a críticas de películas etiquetadas con el sentimiento positivo o negativo,

es decir es un problema de clasificación binaria balanceada. Para las secuencias se utiliza un máximo de 10,000 palabras y una longitud máxima de la secuencia de 500 con padding al inicio (ceros al inicio de las secuencias).

Para los textos en español se utilizó un conjunto de tweets en español de México del TASS, estos se encontraban desbalanceados inicialmente, pero se balancearon con un conjunto de tweets en español de España (Navas-Loro et al. 2017). El conjunto de datos balanceado consiste en 3,103 observaciones de entrenamiento y 776 de validación, dónde cada una de estas observaciones está etiquetada con el sentimiento positivo, negativo o neutro, es decir es un problema de clasificación multi clase balanceado. Para las secuencias se utiliza un máximo de 40,000 palabras y una longitud máxima de la secuencia de 35 con padding al inicio.

Para el caso de los tweets en español, se aplicó un preprocesamiento a estos, donde se puso en minúsculas, se quitaron acentos y duplicados, y se reemplazan los hashtags, números, emojis, links, arrobas, por palabras.

Para todos los modelos en esta aplicación se utilizó la plataforma de GoogleColab con Tensorflow sobre GPU. Para el caso de textos en inglés, ya se contaba con trabajos previos sobre ese conjunto con modelos de redes recurrentes y convolucionales (Chollet 2017), por lo cual, sólo se aplicó un modelo basado en el codificador del transformer, cuya arquitectura se puede observar en la Figura 12, donde las tasas de regularización dropout son del 0.25 en la capa del embedding, 0.05 en el bloque de codificador de transformer (con 8 cabezas de multi atención y 32 unidades en la capa densa), 0.15 en la capa dropout 3, 0.15 en la capa droput 4 y una capa densa con función ReLU. Este modelo se entrenó con el optimizador Adam, con la función de costo cross entropy categórica, métrica de evaluación Accuracy y con tamaño de minibatch de 32.

En la Figura 13 se pueden observar los resultados de los diferentes modelos, observe que con un solo bloque codificador (encoder) es el modelo que tiene menos parámetros de entrenamiento, además de ser el más rápido en comparación con los otros y con un resultado muy cercano al basado en redes recurrentes bidireccionales. Además, note como el incremento de capas convolucionales aumenta considerablemente el número de parámetros de entrenamiento.

Para el caso de los tweets, se probaron varios modelos, entre ellos una máquina de soporte vectorial con kernel gaussiano y bolsa de palabras con TFIDF, otros con embeddings (sin preentrenamiento) uno de ellos con una capa bidireccional (LSTM), otro con una capa convolucional 1D, y otro con un bloque codificador de transfomer. Además de los anteriores se probaron también otro tipo de modelos ya disponibles en librerías de Python, como Vader (Hutto 2014) que utiliza puntuaciones de las palabras para dar el sentimiento, y un transformer pre entrenado para análisis de sentimientos (Huggingface).

En este trabajo sólo se explicará a detalle la arquitectura del modelo basado en el codificador del transformer, la cual se puede observar en la Figura 14, donde las tasas de regularización dropout son del 0.45 en la capa del embedding, 0.4 en el bloque de codificador de transformer (con 4 cabezas de multi atención y 16 unidades en la capa densa), 0.3 en la capa dropout 43, 0.3 en la capa droput 44 y una capa densa con función ReLU. Este modelo se entrenó con el optimizador RMSprop con learning rate de 1e-4, con la función de costo cross entropy categorica, métrica de evaluación Accuracy y con tamaño de minibatch de 32.

En la Figura 15 se pueden observar los resultados de los diferentes modelos, observe que con un solo bloque codificador (encoder), nuevamente es el más rápido en comparación con los otros y con resultados muy similares a los basados en redes recurrentes bidireccionales o convolucionales. Además, para este caso, se notó que los resultados de los modelos anteriormente mencionados son muy similares al de máquina de soporte vectorial, y todos estos están por encima de los modelos Vader y de transformer pre entrenado.

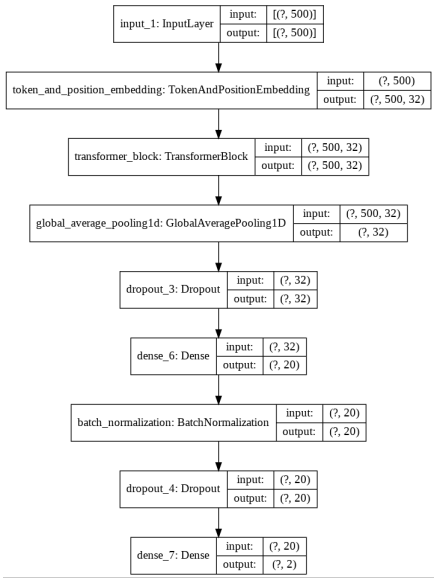

Figura 12. Arquitectura del modelo basado en el codificador del transformer para textos en inglés.

	Train. Param	*Epochs*	*Avg t/epoch*	*Val-Accuracy*
1X RNN bid	338,291	5	35s	0.8906
2X CNN 1D	1,315,937	10	15s	0.8718
1X Encoder	327,206	1	32s	0.8872

Figura 13. Tabla comparativa de los resultados de los modelos para los textos en ingés

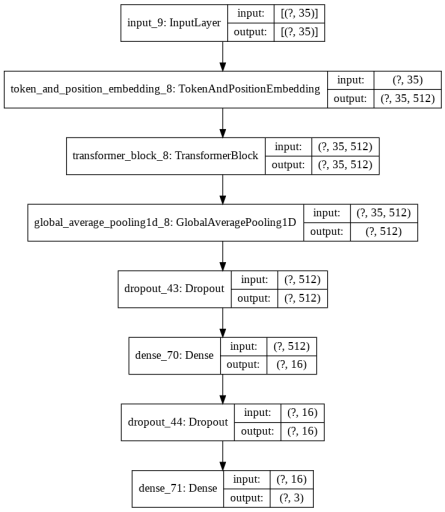

Figura 14. Arquitectura del modelo basado en el codificador del transformer para tweets en español

	Train. Param	*Epochs*	*Avg t/epoch*	*Val-Accuracy*
Vader	-	-	-	0.4
HuggingFace	-	-	-	0.36
SVM-TFIDF	-	-	-	0.64
1X RNN bid	20,620,611	8	22s	0.63
1X CNN 1D	20,628,067	24	20s	0.64
1X Encoder	21,557,843	12	11s	0.62

Figura 15. Tabla comparativa de los resultados de los modelos para los textos en español

III-B. Una imagen vale 16x16 palabras: Transformer para reconocimiento de imágenes

El transformer ha sido explotado ampliamente en tareas de procesamiento de lenguaje, sin embargo, su aplicación en otras tareas como visión computacional son muy limitadas. Debido a esto los modelos para reconocimiento y clasificación de imágenes que recurren a mecanismos de atención son normalmente utilizados con redes convolucionales, incluso el clásico modelo ResNet se encuentra aún en el estado del arte para estas tareas. Este año (2020), Dosovitskiy et al. presentaron el Vision Transformer **(ViT)**, una arquitectura que utiliza el Transformer original de Vaswani et al. (2017) con unas cuantas modificaciones para su aplicación en tareas de clasificación de imágenes.

En su artículo: "*An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale*"Dosovitskiy et al. explican cómo la partición de imágenes en "*parches*z la secuencia de embeddings lineares de estos parches pueden ser utilizados como entradas al codificador del Transformer, ya que se tratan a estos parches de la misma manera como se tratan a los tokens (palabras) en NLP. Cuando este modelo fue entrenado con conjuntos de datos grandes (14M-300M de imágenes) ViT se acerca o mejora los puntajes de los modelos en el estado del arte.

En la siguiente imagen se presenta un panorama general del modelo y de manera muy sencilla el cómo se obtienen los parches de una imagen:

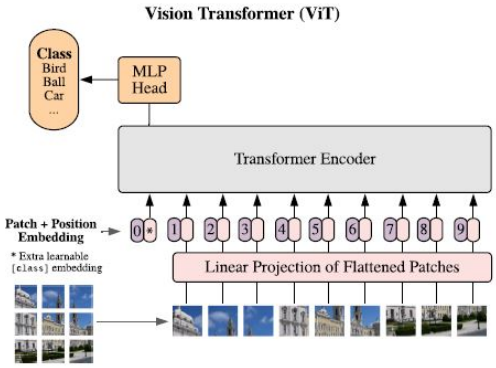

Figura 16. Vision Transformer. Tomado de [].

De esta manera se toma una imagen $(x\epsilon\mathbb{R}^{HxWxC}$ y se transforma en una secuencia de parches en dos dimensiones $(x_p\epsilon\mathbb{R}^{Nx(P^2xC)}$ en donde $H\ x\ W$ es la resolución de la imagen original, C son los canales, $P\ x\ P$ es la resolución de los parches y $N = HW/P^2$ el número de parches resultantes. Se .ªplanan"luego estos parches y se mapean a D dimensiones con una proyección linear entrenada, obteniendo así los embeddings de los parches. Se agregan los embeddings posicionales, que están también en una dimensión y el embedding resultante es el que se utiliza de entrada al codificador del transformer.

Comparado con Redes Convolucionales este modelo tiene menor sesgo inductivo debido a que se tiene menor información con respecto a los vecindarios 2-D en cada imagen. Esto se ve de manera clara debido al hecho de que tanto los parches como los embeddings posicionales guardan y acarrean información en la red solamente de los vecindarios en 1-D.

Para los experimentos realizados en el artículo original, se utilizaron 4 variantes del modelo base explicado anteriormente: En el modelo híbrido en lugar de los parches de la imagen original se utilizaron secuencias extraídas de mapas de características de una red convolusional. Después se utiliza la proyección linear descrita anteriormente para obtener los embeddings de los parches y por último se agregan los embeddings posicionales de igual manera que en el modelo base, el modelo "*Large*z "*Huge*"que se refieren al tamaño de capas que se agregaron y al tamaño de las base de datos en las que fueron entrenados, y por último el modelo base descrito anteriormente. En el cuadro 1 se incluyen los detalles de cada modelo.

Coches disponibles			
Modelo	Capas	Heads	Parametros
ViT-Base	12	12	86M
ViT-Large	24	16	307M
ViT-Huge	32	16	632M

Cuadro I
RESUMEN DE LAS VARIANTES DEL VIT

Los resultados que obtuvieron cuando se probaron en diferentes tareas y conjuntos de prueba se muestran resumidos en la siguiente tabla. En esta se utiliza la notación corta para el modelo base Huge con parches de 14x14 ViT-H/14, y de manera equivalente para los del modelo Large.

	Ours-JFT (ViT-H/14)	Ours-JFT (ViT-L/16)	Ours-I21K (ViT-L/16)	BiT-L (ResNet152x4)	Noisy Student (EfficientNet-L2)
ImageNet	**88.55** ± 0.04	87.76 ± 0.03	85.30 ± 0.02	87.54 ± 0.02	88.4/88.5*
ImageNet ReaL	**90.72** ± 0.05	90.54 ± 0.03	88.62 ± 0.05	90.54	90.55
CIFAR-10	**99.50** ± 0.06	99.42 ± 0.03	99.15 ± 0.03	99.37 ± 0.06	–
CIFAR-100	**94.55** ± 0.04	93.90 ± 0.05	93.25 ± 0.05	93.51 ± 0.08	–
Oxford-IIIT Pets	**97.56** ± 0.03	97.32 ± 0.11	94.67 ± 0.15	96.62 ± 0.23	–
Oxford Flowers-102	99.68 ± 0.02	**99.74** ± 0.00	99.61 ± 0.02	99.63 ± 0.03	–
VTAB (19 tasks)	**77.63** ± 0.23	76.28 ± 0.46	72.72 ± 0.21	76.29 ± 1.70	–
TPUv3-core-days	2.5k	0.68k	0.23k	9.9k	12.3k

Figura 17. ViT vs Modelos del estado del arte. Tomado de [4].

El código que utilizaron se encuentra disponible de manera abierta y en este se pueden utilizar ejemplos que se encuentran en alguna base de datos o utilizar ejemplos propios para hacer una clasificación. Por ejemplo cuando utilizamos la siguiente imagen obtenida del sitio picsum.photos para obtener las probabilidades de predicción con respecto a su categoría obtenemos lo siguiente:

```
2020-12-12 04:04:14 (156 KB/s) - 'picsum.jpg' saved [22173]
```

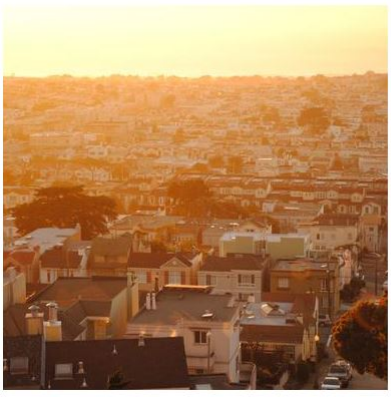

Figura 18. Imagen ejemplo. Tomado de [4].

```
0.19203 : alp
0.13293 : cliff, drop, drop-off
0.12670 : valley, vale
0.07687 : seashore, coast, seacoast, sea-coast
0.07473 : volcano
0.05703 : balloon
0.03229 : bell_cote, bell_cot
0.02437 : lakeside, lakeshore
0.02322 : beacon, lighthouse, beacon_light, pharos
0.02319 : promontory, headland, head, foreland
```

Figura 19. Probabilidades para la imagen ejemplo. Tomado de [4].

Para poder visualizar los mecanismos de atención se tomaron los pesos en todas las capas correspondientes del modelo, se promediaron y se proyectaron en la imagen original. El resultado de los mapas de atención se presentan a continuación con dos imágenes originales, su mapa de atención y su predicción correspondiente. En ambos casos el modelo permitió su correcta clasificación.

Figura 20. Mapas de atención para el modelo ViT. Tomado de [4].

IV. CONCLUSIONES

Algunas observaciones importantes sobre la aplicación de análisis de sentimientos en tweets en español son que, primero se evaluó qué pasaba si no se hacía un pre-proceso al texto, y se observó en este caso que el desempeño de los modelos disminuyó. Además, se probó con embeddings pre entrenados en español (word2vec) y nuevamente no se notó ninguna mejora. En el caso del Vision Transformer, el modelo obtenía resultados bastante malos cuando se entrenaba con conjuntos de datos de tamaño chico o mediano. Fue hasta que se entrenó en base de datos grandes (más de 13 millones de imágenes) que se pudo llegar a una precisión comparable o mejor con los modelos en el estado del arte (ResNet).

Adicional a lo anterior, se observa claramente que los resultados en análisis de sentimiento en español, están distantes a los encontrados en textos en inglés, esto puede ser principalmente debido a que se cuenta con muy pocos datos de entrenamiento y validación lo que causa que todos estos modelos sobre ajusten rápidamente, aún cuando se aplican

técnicas de regularización. Otro motivo, puede ser la calidad de los datos de entrenamiento, ya que con textos tomados de Twitter hasta para un humano pueden llegar a ser complejos de caracterizar, además de que la forma de balancearlos quizá pudo causar algo de sesgo.

También se encontró que los modelos de Transformers son muy sensibles al tamaño de la secuencia, para secuencias muy largas tienden a fallar, además de que aún son poco maduros y no se encuentran disponibles en todas las librerías y algunas de estas implementaciones pueden arrojar muchos errores de ejecución o compatibilidad. Esta limitación en particular se encontró con la librería tensor2tensor implementada por los investigadores de Google Brain y que, según lo que se investigó no cuenta con soporte para tensorflow 2.0, la versión actual de tensorflow. De manera similar cuando se trabajó con imágenes para su clasificación se encontraron muchos errores de compatibilidad, específicamente de las librerías en las que se ha implementado el Vision Transformer hasta ahora.

A pesar de lo anterior, para las tareas en NLP en análisis de sentimientos, los resultados compiten claramente con los obtenidos con los modelos de redes recurrentes y convolucionales, con la ventaja de que su entrenamiento es considerablemente más rápido. En las tareas de clasificación de imágenes, usando el Vision TranSformer, el poder computacional requerido para entrenar los modelos en grandes bases de datos es sólo una pequeña fracción del poder computacional con que se cuenta actualmente en los centros más grandes de investigación. Esto nos permite poner en perspectiva y considerar que este tipo de aspectos no son realmente desventajas significativas.

Como trabajo futuro en NLP, se pretende explorar otras implementaciones de Transformers diferentes a las ya disponibles en Tensorflow, como Pytorch, así como sus ventajas y complementarlo con las herramientas disponibles para visualizar la atención en textos, principalmente para tareas de detección de agresividad en tweets en español.

V. Referencias

[1]. Alammar, J. (2018), "The Illustrated Transformer – Jay Alammar – Visualizing machine learning one concept at a time.," Github, Available at http://jalammar.github.io/illustrated-transformer/.

[2]. Chollet, F. (2017), Deep Learning with Python, 2018 21st International Conference on Information Fusion, FUSION 2018, Manning Publications Co.3 Lewis Street Greenwich, CTUnited States.

[3]. Devlin, J., Chang, M.-W., Lee, K., Google, K. T., and Language, A. I. (2019), BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding.

[4]. Dosovitskiy, A., Beyer, L., Kolesnikov, A., Weissenborn, D., Zhai, X., Unterthiner, T. Dehghani, M., Minderer, M., Heigold, G., Gelly, S., Uszkoreit, J., and Houlsby, N. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929, 2020.*

[5]. Huggingface (n.d.). "Transformers — transformers 4.0.0 documentation," huggingface, Available at https://huggingface.co/transformers/.

[6]. Hutto, C. J. & G. (2014), "VADER: A Parsimonious Rule-based Model for Sentiment Analysis of Social Media Text.," Eighth International Conference on Weblogs and Social Media (ICWSM-14), Available at https://github.com/cjhutto/vaderSentiment#installation.

[7]. Mourri, Y. B., Kaiser, L., and Shyu, E. (n.d.). "Natural Language Processing with Attention Models," Coursera, Available at https://www.coursera.org/learn/attention-models-in-nlp.

[8]. Navas-Loro, M., Rodríguez-Doncel, V., Santana, I., and Sánchez, A. (2017), "Additional Information on the Spanish Corpus for Sentiment Analysis towards Brands," Springer, Cham. https://doi.org/https://doi.org/10.1007/978-3-319-66429-3_68.

[9]. Radford, A., Narasimhan, K., Salimans, T., and Sutskever, I. (2018), Improving Language Understanding by Generative Pre-Training.

[10]. Raffel, C., Shazeer, N., Roberts, A., Lee, K., Narang, S., Matena, M., Zhou, Y., Li, W., and Liu, P. J. (2020), Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer, Journal of Machine Learning Research.

[11]. Stanford (2019), "Stanford CS224N: NLP with Deep Learning — Winter 2019 — Lecture 14 – Transformers and Self-Attention - YouTube," standfordonline, Available at https://www.youtube.com/watch?v=5vcj8kSwBCY&list=PLakWuueTN59e

[12]. Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez A., Kaiser, L., and Polosukhin, I. Attention is all you need. *In NIPS, 2017.*