UNIVERSIDAD DE CHILE
FACULTAD DE CIENCIAS FÍSICAS Y MATEMÁTICAS
DEPARTAMENTO DE CIENCIAS DE LA COMPUTACIÓN

CLASIFICADORES DE MEMES CON DEEP LEARNING

MEMORIA PARA OPTAR AL TÍTULO DE
INGENIERO CIVIL EN COMPUTACIÓN

CRISTÓBAL IGNACIO JARAMILLO ANDRADE

PROFESOR GUÍA:
BENJAMÍN BUSTOS CARDENAS

PROFESORA CO-GUÍA:
MAGDALENA SALDAÑA VILLA

MIEMBROS DE LA COMISIÓN:
JUAN MANUEL BARRIOS
FELIPE BRAVO MÁRQUEZ

SANTIAGO DE CHILE
2022

Resumen

Los memes son un tipo de imagen muy particular, la cual se ha ganado gran popularidad con el pasar de los últimos años. Esto gracias principalmente a las redes sociales y otras fuentes de Internet, así como también a la rápida forma que tienen de propagarse y llegar a distintas personas. Es por esto por lo que generan un gran impacto en la comunidad. Este tipo de imágenes presenta una continua evolución por lo que su estudio es muy difícil de realizar.

Dado que los memes presentan una estructura muy variada y una evolución constante, se requiere de una herramienta que los permita clasificar automáticamente y así lograr analizarlos en mayor profundidad. Por otro lado, este tipo de imágenes presentan temáticas que indican el tipo de humor o contenido que se quiere expresar. Es por esto por lo que también se desea lograr categorizarlos automáticamente bajo ciertas clases y así lograr entenderlos de mejor manera.

Hoy en día, el estado del arte para las clasificaciones de imágenes y datos en general son las redes neuronales. Existen distintos modelos tanto para texto como imagen, las cuales llegan a ser útiles para distintas tareas que se requieran. También existen distintas técnicas para adaptar redes ya entrenadas y modificarlas para resolver un problema distinto.

Se realizaron distintos experimentos que involucraban variadas arquitecturas. Para esto se utilizaron dos dataset. El primero categorizaba imágenes entre *Meme*, *No Meme/Sticker*, *Sticker* y *Dudoso* descartando esta última. El segundo conjunto de datos categorizaba las imágenes en 17 categorías distintas de memes, las cuales fueron definidas por un grupo de expertos.

Dentro de la investigación se pudo observar que, para resolver los distintos problemas de clasificación de memes, se requiere analizar tanto la imagen como el texto que esta posee. Esto logró mejorar las métricas en gran medida, aunque se debe considerar que existen tipos de imágenes que se logran clasificar de mejor manera bajo ciertos modelos, dado que los memes no siguen una estructura fija.

Los resultados experimentales muestran que el mejor modelo probado hasta el momento para clasificar memes es el modelo mixto de ResNet con BERT con un 90,7 % de $F1$ *Score*. Por otro lado, en la clasificación por tópicos, la mejor red neuronal es BERT con un 70,6 %, la cual trabaja solamente con los textos de las imágenes.

Peron por siempre

Agradecimientos

Primero me gustaría agradecer al Instituto Milenio Fundamentos de los Datos por darme la oportunidad de poder aportar en la investigación de una de sus áreas. También por brindar los datos que fueron utilizados en esta memoria como así también el apoyo que recibí durante este proceso.

Agradecer al profesor Benjamín Bustos y a la profesora Magdalena Saldaña por brindar un gran apoyo durante todo este proceso como así también todas las enseñanzas brindadas. Esta fue tanto una etapa de investigación como de aprendizaje por lo que sus constantes aportaciones fueron vitales para lograr los resultados obtenidos. También agradecer al profesor Felipe Bravo por brindar el computador donde se corrieron la mayoría de los experimentos, los cuales gracias a este el tiempo fue muchísimo menor.

Un pilar fundamental de apoyo durante todo el proceso de la universidad fue mi familia. El apoyo brindado por mi madre y padre fue vital a la hora de seguir adelante ante todos los problemas que fueron surgiendo.

Finalmente agradecer a mi grupo de amigos y compañeros que me acompañaron en la universidad en especial al grupo Peron, con el cual estudiábamos y comíamos juntos la mayor parte del día. Mención especial al Foton quien me ayudo a seleccionar los memes presentes en la memoria y la Luni quien me ayudo con inglés y me dio muchísimo apoyo en el final de la carrera.

Tabla de Contenido

Capítulo 1

Introducción

En los años actuales, el Internet es una de las herramientas más utilizadas a nivel global lo que trae muchas ventajas. Una de estas es el nivel de información que está disponible para cualquier persona por medio de búsquedas en Google, redes sociales o incluso en los medios tradicionales como la televisión hacen uso del Internet para mantenerse al día con las últimas noticias que han sucedido en el mundo, aunque por otro lado estos medios traen distintos problemas que se deben abarcar para verificar la veracidad de la fuente o de las imágenes que se comparten por Internet. La difusión de información se ha vuelto muy fácil con la diversificación y conexión entre las redes sociales, pues basta con obtener unas pocas visualizaciones en Twitter o Facebook para aparecer en la pantalla de millones de usuarios. Se han conocido muchos casos de transmisión de noticias falsas y discursos de odio que han dañado la credibilidad y reputación de ciertas personas o instituciones. Lo anterior es mucho peor cuando el usuario con un solo click puede compartir estas noticias a sus amigos o seguidores. Una de las formas de difusión de información más utilizada el día de hoy son las imágenes. Estas permiten captar la atención de las personas de una forma rápida y eficaz en comparación con solo texto. Uno de los recursos relacionados con esto son los memes.

En el último tiempo, el Instituto Milenio Fundamentos de los Datos (IMFD) se ha enfocado en estudiar ciertos problemas que se ven afectados por la falta de información robusta, por medio de su proyecto Generación de Estructuras de Información Robusta. Es por esto que se requieren crear sistemas automáticos que permitan estudiar distintas fuentes de datos no estructuradas. Uno de estos datos son los memes que se entienden como un objeto que intenta transmitir una idea con tono humorístico. Estos se están convirtiendo en una fuente útil de datos para analizar el comportamiento en las redes sociales. Sin embargo, un problema es cómo identificar correctamente un meme. Esto dado que la cantidad de memes publicados en las redes sociales es enorme, por lo que se quiere poder clasificar automáticamente para poder analizarlos en profundidad [37].

Hoy en día una de las herramientas más utilizadas para clasificar imágenes son las redes neuronales. Algunas de las más conocidas permiten clasificar a animales, objetos cotidianos, imágenes de números, etc. Sería muy útil ampliar estos conocimientos y crear herramientas que permitan filtrar y reconocer de manera automática el contenido de las imágenes de memes y así estudiar en mayor profundidad, el comportamiento de las redes sociales. Esto no

es una tarea fácil, pues con el pasar del tiempo los memes han cambiado y modificado ciertas características importantes, como la utilización de distintos templates o incluso dimensiones (imágenes en vertical u horizontal) pero existen ciertas peculiaridades que la red podría utilizar para obtener un mejor conocimiento como lo puede ser el texto que el propio meme posee y mismas imágenes para distintos memes. Dado lo anterior se generan ciertas preguntas ¿Existirán redes que permitan clasificar mejor ciertos tipos de memes? ¿El uso de un template permitirá a la red mejorar su precisión sobre otros memes con el mismo template? ¿La red presentará problemas con aquellas imágenes que poseen texto y no son memes?

Este trabajo ya ha tenido avances en memorias anteriores [39] [33], usando técnicas clásicas de Machine Learning y pequeños modelos de redes neuronales, se clasifican las imágenes en memes, no memes y stickers. En este trabajo se quiere llevar a más profundidad en el área de deep learning y además poder llegar a clasificar estos memes según su tipo, es decir entender que tipo de idea intenta transmitir, la cual puede ser político, motivacional, etc siendo esta la que definirá su categoría. Por otro lado, se explorará el recurso del texto propio de las imágenes, técnica que no fue considerada en trabajos anteriores.

1.1. Problema

Dado el gran crecimiento que están teniendo las redes sociales, se desea poder analizar en profundidad su comportamiento. Los memes se están convirtiendo en una fuente muy útil para llevar a cabo esta tarea. Un problema que existe es que la cantidad de imágenes publicadas en las redes sociales es enorme y no todas cumplen con la categoría de meme. Es por esto que existe la necesidad de clasificar y buscar en grandes conjuntos de datos imágenes del tipo meme.

Por otro lado, estos memes poseen distintas categorías, es decir, sobre qué temática están haciendo referencia. Un ejemplo puede ser analizar el comportamiento de las redes sociales por medio de memes en temas relacionados con la política. Es por esto que se desea categorizar automáticamente grandes conjuntos de memes y así estudiar, en profundidad, áreas más especificas.

1.2. Metodología

Este es un problema de investigación e experimentación por lo que la metodología a seguir debiese centrarse en explorar y probar distintas formas de resolver el problema. Para esto se debe seguir una serie de pasos e ir repitiendo ciertas etapas de ser necesario. Al tratarse de una investigación, lo importante debe ser en entender cuál es el problema y qué se necesita para mejorarlo.

- Investigar cómo abordar el problema con redes neuronales
- Programar ciertos modelos o algoritmos para la experimentación

- Experimentar y analizar resultados
- Ver posibles mejores en los modelos

El problema de los memes es un tema aun en exploración, por lo que cualquier resultado obtenido es importante para seguir avanzando.

1.3. Objetivos

Objetivo principal

El objetivo principal es categorizar memes usando distintas técnicas de Deep Learning, estudiando los diferentes comportamientos que pueden existir para las diversas imágenes de memes que se presentan.

Objetivos específicos

1. Aplicar distintas técnicas para mejorar el dataset disponible.
2. Diseñar e implementar redes neuronales que permitan verificar si una imagen es un meme o no.
3. Implementar técnicas de redes neuronales que permitan clasificar los tipos de memes.
4. Diseñar metodologías de experimentación capaces de medir los resultados obtenidos.
5. Aplicar la técnica de ablation study para identificar sectores de la red que aportan en la clasificación de las imágenes.
6. Analizar los resultados obtenidos.

1.4. Evaluación

Para comprobar el cumplimiento de los objetivos se verificarán que los modelos de redes cumplan el poder clasificar los tipos de memes. Para esto se emplearán métricas de accuracy propias de las redes neuronales para los distintos modelos planteados. Por otro lado, se documentarán todos los pasos de experimentación para analizar las distintas técnicas ocupadas. Los análisis que se realizarán con ablation study serán graficados y explicados en el informe final, para así argumentar todas las decisiones tomadas durante el proceso de experimentación.

Capítulo 2

Conceptos básicos

En esta sección se explicarán algunos conceptos básicos para la comprensión de la memoria. Primero se abarcará el concepto de meme, pasando por sus características y aspectos importantes que se deben tomar en cuenta a la hora de trabajar con ellos. También se revisarán ciertos problemas que pueden llegar a tener.

Luego se profundizará en el concepto de redes neuronales. Se explicará cómo funcionan en gran profundidad, mostrando la matemática que está involucrada como así también distintas técnicas para mejorar los resultados que obtienen a la hora de clasificar. Por otro lado, se abordarán las métricas que existen para evaluar los resultados de los distintos modelos. Estas se utilizarán más a adelante para concluir lo obtenido.

Finalmente se abordarán los distintos tipos de redes neuronales que se utilizarán en los experimentos. Las redes para imágenes y para texto, explicando el funcionamiento de distintos modelos.

2.1. Memes

En los tiempos actuales la palabra meme es una de las más utilizadas por las nuevas generaciones, pues basta con revisar las redes sociales actuales (utilizadas por una gran parte de la población), en donde los memes abundan en gran cantidad. Estos son intercambiados a gran escala, todos los días y a todas horas, convirtiéndolos en una gran fuente de información. Los memes consisten en un objeto que intenta transmitir o expresar una idea con un enfoque humorístico [37]. Son difundidos principalmente por redes sociales, aunque existen páginas que se especializan en la creación y publicación de estos. Existen en distinto formato como lo son por vídeo, imágenes e incluso por texto.

La creación y difusión de los memes se basa en la participación cultural que existe dentro de Internet [25]. En esta son los usuarios quienes eligen qué contenido desean consumir [26]. Esto genera que los memes difundan información a grupos de personas donde no necesariamente llegan los medios de comunicación. Sin embargo, no todos los usuarios interactúan de la misma manera con este tipo de imágenes [36]. Esto pues en ocasiones se requiere de un

contexto previo que permita dar un mejor entendimiento a la imagen.

2.1.1. Orígenes

En general, la palabra meme comenzó a ser popular gracias a los medios digitales de hoy en día, pero su origen se remonta en la biología. Richard Dawkins, inspirado en el funcionamiento del ADN, acuñó el concepto de meme a una unidad de transmisión cultural. Esto dado a que, como los genes, los memes tienen la capacidad de replicarse, posibilitando el desarrollo de la cultura [37].

2.1.2. Características de los memes

Comúnmente los memes presentan una imagen con un texto, aunque existen otro tipo donde solo se muestra la imagen en sí, sin ningún texto encima. La imagen presenta una situación o una idea enfocada al humor. Esto para causar gracia o intentar llamar la atención de la persona que está viendo la imagen. El texto acompañante ayuda a darle aún más contexto o explicar el meme. Lo anterior se hace siempre intentando expresar una idea humorísticamente.

En la Figura 2.1 se puede observar una característica muy común en los memes durante mucho tiempo y es que poseen un texto sobre una imagen, intentando expresar una idea, generalmente de un tono humorístico. Esto nos abre las puertas a dos puntos importantes, uno es el intentar captar este texto y analizarlo para ver si se logra clasificar dentro de un meme y también un patrón para la red que le permita clasificar con mayor eficacia.

(a) (b) (c)

Figura 2.1: Diferentes tipos de memes

Otra característica muy importante para tomar en cuenta se centra en que los memes son creados por cualquier persona. Esto quiere decir que un individuo podría hacer un meme para su familia, por lo que, si no se posee el contexto, el meme no tendrá ningún sentido. Este contexto es muy importante para entender el meme. Por ejemplo, pueden existir memes enfocados en un país o en un grupo especifico, haciendo vital tener entendimiento del tema para que el meme tenga algún sentido [37].

2.1.3. Tipos de memes

Existen muchos tipos de memes, algunos enfocados a la política, cosas típicas de un país o incluso relacionarse con algún personaje. Es por esto por lo que los memes van evolucionando día tras día, creándose nuevas categorías de memes. Esto dificulta su estudio pues se convierte en un trabajo muy dinámico.

En la Figura 2.2 se pueden observar tres tipos de memes totalmente distintos. El primero trata sobre política, más específicamente en cómo se ha comportado la dieta parlamentaria en Chile durante los últimos años. El segundo presenta una imagen meme enfocada en el deporte, en donde se muestra a un jugador del futbol chileno muy conocido. Finalmente se muestra una imagen del tipo chistosa, pues presenta sobre algo típico con lo que se molesta a Chile, su forma de hablar.

(a) (b) (c)

Figura 2.2: Memes

2.1.4. Stickers

Los stickers son otro tipo de dato que sigue una estructura semejante a la de un meme. Estas pueden tener texto o no y en comparación a los memes, no intentan transmitir o expresar un idea en tono humorístico, si no que pretende dar una respuesta de manera rápido y concisa. En la Figura 2.3 se pueden ver algunos ejemplos de estos stickers.

(a) (b)

Figura 2.3: Stickers

2.1.5. Problemas a la hora de analizar memes

No toda imagen con texto puede considerarse meme pues depende del enfoque y de la intención que esta posee, pues para que sea considerada meme, debe tener un tono humorístico. Es por esto por lo que este trabajo presenta un alto nivel de desafío, dado que se requiere de mucha experimentación y estudio sobre las distintas técnicas en el área de Deep Learning para lograr un mejor resultado.

Otra tarea importante dentro de este mundo se basa en intentar darles un contexto en el que la imagen es presentada, es decir, poder encontrar cuál es el tipo del meme entregado, los cuales pueden ser del tipo político, jocoso, de propaganda, motivacionales, ideológicos, etc. Esto es una tarea difícil, pues son dos ramas las que se tienen que analizar, primero con la imagen en sí y por otro lado el texto que posee. Esto abre grandes puertas dentro del campo ¿Podrá de alguna forma una red neuronal entender el contexto en el que es presentado el meme?

2.2. Redes neuronales

El concepto de red neuronal artificial viene con la idea de imitar el funcionamiento de las redes neuronales de los organismos vivos [10]. Estas redes biológicas consisten principalmente en un conjunto de neuronas conectadas entre sí, las cuales trabajan en conjunto. Algo importante sobre este tipo de redes es que, con la experiencia, las neuronas van creando y reforzando ciertas conexiones para de cierta manera aprender.

Las redes neuronales artificiales intentan seguir el modelo biológico mencionado anteriormente. Sus unidades básicas son las neuronas que generalmente se organizan en capas, como se muestra en la Figura 2.4. Estas neuronas se pueden entender como unidades de procesamiento que se encuentran interconectadas.

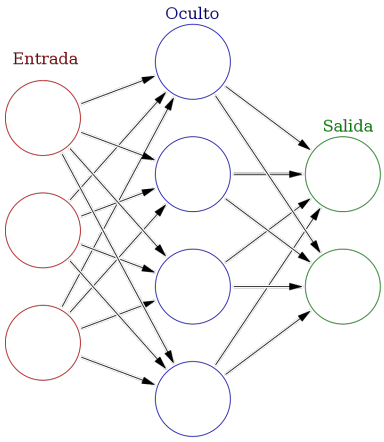


Figura 2.4: Red neuronal

Estas unidades de procesamiento se organizan en distintas capas. Por lo general existen tres: La capa de entrada la cual se encarga de recibir y computar los valores iniciales; las

capas ocultas que se encuentran en medio de la red; y una capa de salida con una o varias unidades que representan el resultado. El proceso dentro de estas capas es simple, se entregan los datos de entrada a la primera capa y estos se van propagando por cada neurona hasta llegar a la capa final, en donde se entrega el resultado final.

2.2.1. Proceso de cómputo

Cada neurona está conectada con otras por medio de enlaces con cierto peso w el cual se va modificando en el proceso de aprendizaje [10]. Por otro lado, cada unidad de cómputo presenta un valor y el cual se va a computando a medida que el proceso se va propagando. También cada capa posee un *bias* b propio que se utilizará a la hora de realizar el cómputo. Finalmente existe una función de activación f presente en cada una de las distintas capas de la red. Lo descrito anteriormente se puede ver en la Figura 2.5.

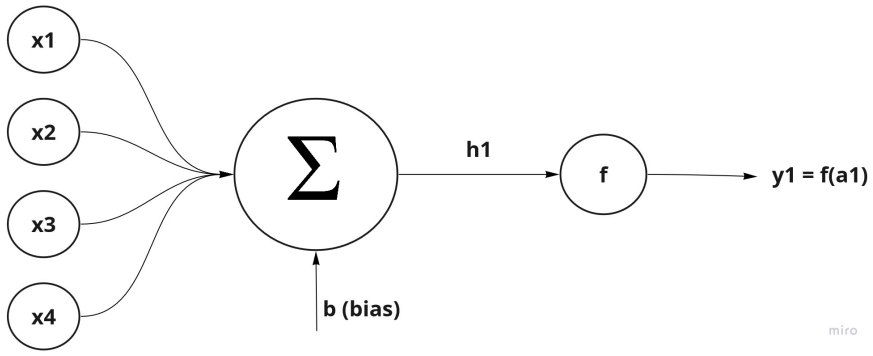


Figura 2.5: Proceso de cómputo

Para realizar el proceso propagación se tiene una unidad de cómputo la cual sigue la siguiente fórmula:

$$y_i = \sum_j w_{ij} y_j \tag{2.1}$$

Esta nos indica que el valor de la unidad de cómputo i es igual a la suma de todos los pesos multiplicados por el valor de su respectiva unidad de cómputo, es decir, de donde salen sus enlaces. También es importante notar que a esta suma de pesos ponderados se le agrega el sesgo *bias* y se le aplica la función de activación f.

$$\begin{aligned} z_i &= b + \sum_j w_{ij} y_j \\ y_i &= f(z_i) \end{aligned} \tag{2.2}$$

Este proceso se hace para cada capa presente en la red neuronal. Para la capa de entrada, los valores respectivos de y_i corresponde a los valores iniciales que se les entrega, pues no existen enlaces.

2.2.2. Funciones de activación

Una unidad de cómputo no solo transmite la entrada que recibe. Existe un paso adicional relacionado con la función de activación [15]. Esta utiliza la suma ponderada calculada en el paso anterior $z = b + \sum_i w_i x_i$ y la transforma una vez más como salida. Cada una de las capas que conforman la red neuronal tienen una función de activación que permitirá reconstruir o predecir.

Función lineal:

Esta función conocida como identidad, permite que la entrada sea igual a la salida. Por lo general se utiliza en regresiones lineales.

$$f(x) = x \tag{2.3}$$

Función umbral:

También conocida como función escalón, indica que cuando el valor que es menor que cero, la neurona entregara un cero mientras que si es mayor o igual a cero dará como salida un 1. Esta se utiliza por lo general para clasificar.

$$f(x) = \begin{Bmatrix} 0 & x < 0 \\ 1 & x \geqslant 0 \end{Bmatrix} \tag{2.4}$$

Función sigmoide:

Esta función está en un rango de valores de salida entre cero y uno por lo que la salida es interpretada como una probabilidad. Si se evalúa la función con valores de entrada muy negativos, la función será igual a 0 mientras que con valores altos su valor se aproxima a 1. Por lo general se utiliza para clasificar en dos categorías.

$$f(x) = \frac{1}{1 - e^{-x}} \tag{2.5}$$

Función tangente hiperbólica:

Esta función tiene un rango de valores de salida entre -1 y 1. Esta se encuentra centrada en 0.

$$f(x) = \frac{2}{1 + e^{-2x}} - 1 \tag{2.6}$$

Función ReLu:

Esta es una de las funciones más utilizadas debido a que permite el aprendizaje muy rápido en redes neuronales. Si esta función se le da valores de entrada muy negativos el resultado

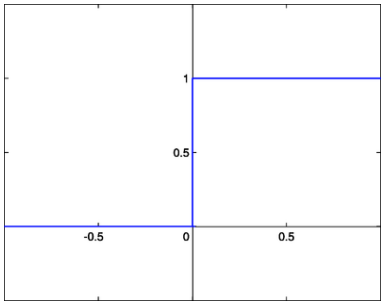
será cero, pero si se le da valores positivos queda igual. Existe una variante denominada Leaky ReLu que va a prevenir que existan neuronas muertas debido a la pendiente que existe cuando $x < 0$.

$$f(x) = max(0, x) \tag{2.7}$$

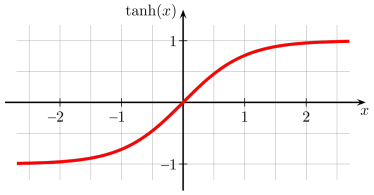
Función softmax:

Esta función se usa para clasificar información. Esta es utilizada en la capa final de la red neuronal. Básicamente lo que hace es transformar los resultados calculados por las neuronas en una distribución de probabilidades. Softmax se usa para clases múltiples y cuando se va a asignar probabilidades a cada clase que pertenezca a clases múltiples.

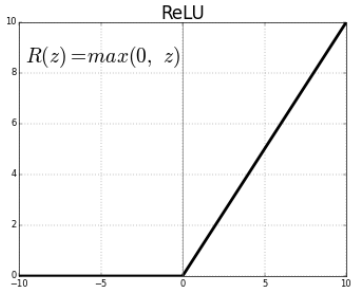
$$f(x)_j = \frac{e^{x_j}}{\sum_{k=1}^{K} e^{x_k}} \tag{2.8}$$



(a)



(b)



(c)

2.2.3. Clasificación

Una vez que el proceso de cálculo para cada capa intermedia es realizado, la propagación llega a la última capa de la red. Esta capa por lo general presenta un número de neuronas igual a el número de categorías. Por ejemplo, si se quiere clasificar una imagen entre conejo, perro o gato, existirán tres neuronas en la capa final (cada una definiendo una clase). A este vector de valores de salida se le aplica la función softmax (Eq. 2.8), de esta manera se asigna una probabilidad a cada valor y la red puede discriminar cual categoría es más probable que la imagen pertenezca.

Veamos que en la Figura 2.6 existe el valor z_1 z_2 y z_3 correspondiente a la clase *conejo* *perro* y *gato* respectivamente. Cada una de estas obtuvo valores finales obtenidos como resultado del proceso de cómputo en la propagación. Si se aplica la función softmax obtenemos lo siguiente:

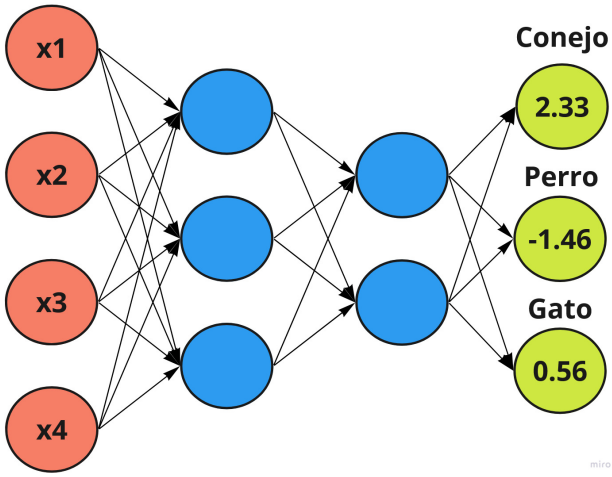


Figura 2.6: Clasificación de una red neuronal

$$\begin{aligned} softmax(z_1) &= softmax(2{,}33) = 0{,}8382 \\ softmax(z_2) &= softmax(-1{,}46) = 0{,}018 \\ softmax(z_3) &= softmax(0{,}56) = 0{,}1427 \end{aligned} \tag{2.9}$$

La clase de z_1 obtuvo un 83 % aproximadamente. Esto quiere decir que la red estimó que la imagen entregada es más probable que sea de un *conejo*, luego con un 14 % de un *gato* y un 0,1 % de un perro.

Es importante notar que la red trabaja con probabilidades. Esto quiere decir que a cada categoría le asigna una probabilidad. Esta dependerá de los valores de peso que tengan los enlaces de las redes. Este peso w tendrá que ser entrenado para que la red pueda discriminar de mejor manera.

2.2.4. Fase de Entrenamiento

La idea del entrenamiento de una red neuronal es descubrir cual es el conjunto de pesos w que producen el menor error posible. Para esto se utiliza la fórmula de la entropía cruzada (Eq. 2.10). Dado el resultado final de la red neuronal y el valor correcto, se calcula el error de entrenamiento. Con este error se realiza lo que se conoce como *backpropagation* [10].

$$CE(p, q) = \sum_x p(x) \log \left(\frac{1}{q(x)} \right) = -\sum_x p(x) \log q(x) \tag{2.10}$$

Consideremos que Q es el valor obtenido y P es al que nos queremos acercar.

$$CELoss(Q, P) = \frac{1}{N} \sum_i CE(p_i, q_i) \tag{2.11}$$

Para *backpropagation* se utiliza el método del gradiente con el fin de optimizar los parámetros de la red y encontrar el menor error posible. Para esto debemos calcular la derivada del error respecto a cada parámetro. Esta derivada nos permitirá movernos en la dirección de la pendiente en bajada y así disminuir el error.

Ahora tomemos y como los valores correctos de clasificación e $\hat{y}$ como el resultado obtenido por la red

$$\mathcal{L} = CELoss(\hat{y}, y) \tag{2.12}$$

Debemos calcular la derivada de este error para cada uno de los parámetros y de esta manera realizar el paso del gradiente.

$$\frac{\partial \mathcal{L}}{\partial \theta} \tag{2.13}$$

Como podemos ver esta derivada depende de $\mathcal{L}$ lo que a su vez depende de todos los pesos w de los enlaces, los *bias* y cada valor de las unidades de cómputo. Esto nos indica que debemos hacer un proceso de propagación hacia atrás, partiendo del resultado de la capa final, derivando hacia atrás y acumulando gradiente en cada unidad.

Cuando este proceso termina se ajusta el parámetro dependiendo el gradiente obtenido multiplicado por un factor conocido como *aprendizaje*.

$$\theta = \theta - \frac{\partial \mathcal{L}}{\partial \theta} \tag{2.14}$$

2.2.5. Regularización

Uno de los objetivos principales de la fase de entrenamiento se traduce en que nuestra red pueda generalizar los datos. Esto quiere decir que logre clasificar correctamente un dato que nunca ha visto (es decir que no fue dato de entrenamiento). Para lograr esto hay que evitar que la red neuronal se aprenda los datos de entrenamiento de memoria y así disminuir el error de generalización. Para hacer esto existen métodos de regularización [24], los cuales consisten en modificaciones a los algoritmos de aprendizaje.

Por lo general estos métodos actúan sobre los parámetros de la red, aunque también existen técnicas sobre la experimentación y los datos con los que se entrenan.

Weight decay

Este método [17] consiste en penalizar los valores que se les asignarán a los pesos y de esta manera no posean valores muy elevados. De esta forma se evita que ciertas neuronas tomen el dominio de la red en ciertas capas y lleva a que las características poco informativas (ruido que no aporta información) tengo pesos cercanos a 0.

Para aplicar este regulador se modifica la fórmula de actualización de pesos 2.14. Se introduce un nuevo termino conocido como *beta*. La idea es que en cada paso de actualización, antes de actualizar los parámetros, se multiplique por $(1 - beta)$, es decir la actualización de cada parámetro θ sea:

$$\theta = (1 - beta)\theta - \frac{\partial \mathcal{L}}{\partial \theta} \tag{2.15}$$

Dropout

Dropout [41] es un método que desactiva un número de neuronas de una red neuronal de forma aleatoria. En cada iteración de la red neuronal dropout desactivará diferentes neuronas, las neuronas desactivadas no se toman en cuenta para la propagación ni para el backpropagation lo que obliga a las neuronas cercanas a no depender tanto de las neuronas desactivadas.

Este método ayuda a reducir el overfitting ya que las neuronas cercanas suelen aprender patrones que se relacionan y estas relaciones pueden llegar a formar un patrón muy específico con los datos de entrenamiento, con dropout esta dependencia entre neuronas es menor en toda la red neuronal, de esta manera las neuronas necesitan trabajar mejor de forma solitaria y no depender tanto de las relaciones con las neuronas vecinas.

Para aplicar este método se asigna una probabilidad a cada neurona de ser desactivada en la fase de entrenamiento. Esto quiere decir que las conexiones que tenía esa neurona desaparecerán momentáneamente.

Es importante ver que esta desactivación solo se hace en fase de entrenamiento, pues en fase de prueba todas las conexiones deben estar activas.

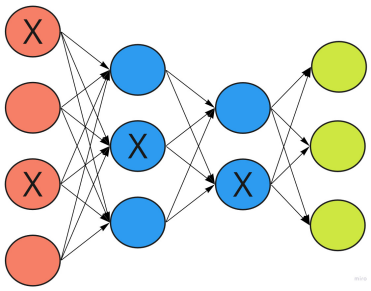


Figura 2.7: Proceso de desactivación de ciertas neuronas

Agregar más datos

La manera más efectiva de disminuir el error de generalización es agregar más datos de entrenamiento. Esto permitirá a la red neuronal tener más referencias del mundo total que se está clasificando. El problema de esto es que generar más datos no es necesariamente fácil.

Uno de los métodos más utilizados se basa en *data augmentation* [35]. Este proceso consiste en generar datos sintéticos por medio de agregar ruido al dato, rotar en caso de que sea una imagen, o agregar cambios al texto.

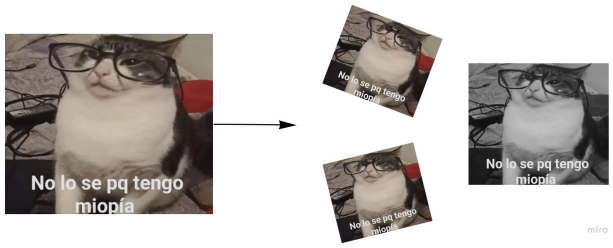


Figura 2.8: Proceso para obtener más datos a partir de uno

Early Stopping

El early stopping [9] consiste en hacer un estudio sobre la experimentación. Uno de estos estudios se basa en comprar el error de entrenamiento y prueba en cada época. Luego se elige la época donde este error sea menor y se le asigna a la red los pesos obtenidos en ese momento.

Al hacer este método se obtiene un nuevo hiper parámetro, la época. También se pueden ir modificando otros hiper parámetros como el tamaño del batch de entrenamiento, el grado de aprendizaje, etc.

Batch Normalization

La normalización en lotes [23] consiste en añadir un paso extra entre las neuronas y la función de activación. El objetivo principal de este método es normalizar las activaciones de salidas. Por lo general esta normalización se hace utilizando la media y varianza de todo el conjunto de entrenamiento, aunque a veces se usan estas medidas de cada batch de entrada.

El primer paso para hacer este método es calcular el valor promedio y la varianza de la muestra, donde m es el tamaño del batch.

$$\mu_B = \frac{1}{m}\sum_{i=1}^{m} x_i \tag{2.16}$$

$$\sigma_B^2 = \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_B)^2 \tag{2.17}$$

Luego los datos de la muestra se normalizan para llevar el batch a una media de cero y una varianza de 1. Hay que considerar una constante ε, esto es para evitar la división por cero.

$$\widehat{x_1} = \frac{x_i - \mu_B}{\sqrt{\sigma_B^2 + \varepsilon}} \tag{2.18}$$

La normalización ayuda a la red neuronal a trabajar mejor, reduciendo las oscilaciones de las funciones de coste. Esto permite aumentar el *learning rate* o tasa de aprendizaje, pues

se reduce el riesgo de terminar en un mínimo local y así la convergencia hacia el mínimo se produce más rápido, acelerando el proceso de aprendizaje.

2.2.6. Optimizadores

Como se explicó anteriormente, el objetivo de la fase de entrenamiento consiste en reducir el error de acierto que posee la red. Este error va a depender de los pesos que posean las conexiones entre las distintas neuronas, por lo que se necesitan optimizar de mejor manera estos valores. Para lograr esta optimización existen distintas formas, algunas son variaciones de ciertos algoritmos mientras que otras modifican completamente la forma de hacerlo.

Es importante notar que ningún algoritmo es mejor que otro, en general cada uno se adapta de forma distinta y se debe elegir el que más se adecue a lo que se quiere realizar (o el que mejor funcione en la experimentación).

Descenso del gradiente

Fijemos la atención en un peso w de una conexión, si solo variamos este peso dejando fijo los demás, se puede obtener una función de error para esa conexión en específico. Lo ideal sería que este error fuera el mínimo de la función y eso aplicarlo para todos los pesos. Como las redes neuronales poseen tantas conexiones, la función en cuestión tendría muchos parámetros.

La idea sería encontrar el mínimo de esta función y así minimizar el error. Para lograr esto se puede derivar la función que componen todos los pesos e ir bajando por la dirección negativa del gradiente.

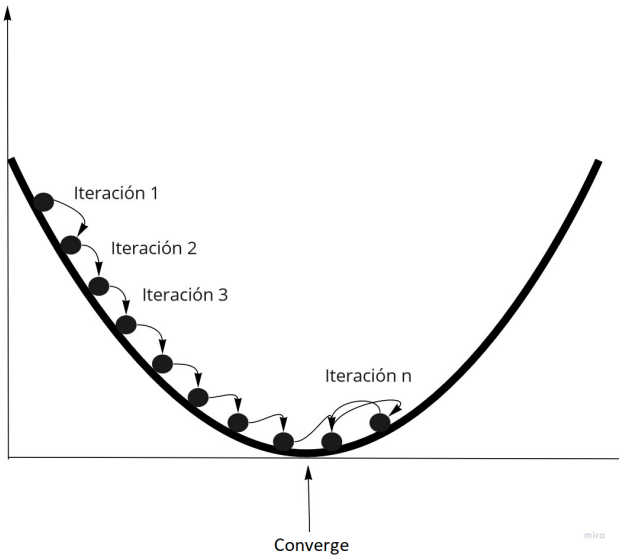


Figura 2.9: Representación gráfica descenso gradiente

Este es el descenso de gradiente [10]. Formalmente, si comenzamos en un punto x_0 y nos movemos una distancia positiva α en la dirección del gradiente negativo, entonces nuestro nuevo y mejorado x_1 se verá así:

$$w_1 = w_0 - \alpha \nabla f(w_0) \tag{2.19}$$

Más general, podemos decir que en cada paso del gradiente x_n va mejorando a un x_{n+1} que disminuye el error.

$$w_n = w_{n-1} - \alpha \nabla f(w_0) \tag{2.20}$$

A partir de una conjetura inicial x_0, la seguimos mejorando poco a poco hasta encontrar un mínimo local. Este es un proceso muy lento dándole así la complejidad computacional que poseen los entrenamientos de las redes neuronales.

Este algoritmo posee distintos problemas que se han intentado mejorar con nuevas versiones. Uno de estos problemas es que encuentra mínimos locales y no generales. Puede pasar que, al descender, se estanque en un mínimo local no deseado y no se pueda seguir iterando para encontrar el global, lo que se traduce en una mala optimización de la red.

Otro punto importante para tomar en cuenta es el avance del gradiente. Si se elige un tamaño muy pequeño puede no llegar a converger al mínimo mientras que si se elige un tamaño muy grande es posible que en cada pase se exceda.

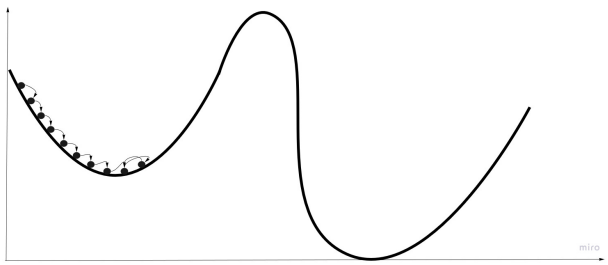


Figura 2.10: Problema del minimo local

En la Figura 2.10 se puede observar que a pesar de que se convergió a un mínimo, este es local pues más adelante existía otro mucho menor que el encontrado. Este es un gran problema a la hora de entrenar redes neuronales, pues no hay forma eficiente de saber si el mínimo encontrado corresponde al global. Lo anterior limita el aprendizaje que puede obtener el modelo.

Descenso del gradiente con momentum

Para dar soluciones a los problemas que existen en el descenso del gradiente se agrega un nuevo termino, el momentum [43].

Este nuevo término adicional tendrá en cuenta los valores de los gradientes anteriores. El gradiente actual se multiplicará por la tasa de aprendizaje α y el valor anterior de los pesos x_{n-1} por una constante conocida como coeficiente de momentum μ. Típicamente este tiene un valor de 0.9.

$$\triangle w_t = \mu \cdot \triangle w_{t-1} - \alpha \cdot \nabla f(w_{t-1}) \tag{2.21}$$
$$w_{t+1} = w_t + \delta w_t \tag{2.22}$$

Lo que obtendremos será una media ponderada del gradiente, de tal forma que el avance sea más rápido cuando nos movemos en la dirección correcta y las posibles oscilaciones se atenúen cuando se rebote en la función explicada en el optimizador anterior. Todo esto para permitir que la convergencia se produzca más rápida.

RMSProp

La motivación de este optimizador [48] se basa en que la magnitud de los gradientes puede diferir para distintos pesos y pueden cambiar durante el aprendizaje, haciendo complicado escoger un learning rate global. En este método la tasa de aprendizaje se adapta para cada parámetro. RMS incluye la media móvil exponencial del gradiente al cuadrado. También aparece una constante ρ, que se conoce como factor de olvido.

$$v_t = \rho \cdot v_{t-1} + (1-\rho) \cdot \nabla^2 f(w_t) \tag{2.23}$$
$$\triangle w_t = -(n/\sqrt{v_t}) \cdot \nabla f(w_t) \tag{2.24}$$
$$w_{t+1} = w_t + \triangle w_t \tag{2.25}$$

Los valores comúnmente utilizados para ρ y n son 0,9 y 0,001 respectivamente.

Adam

Se trata de una combinación de RMSProp con el Momentum [48]. Por un lado, tendremos la media móvil exponencial del gradiente al cuadrado, y por el otro la media móvil exponencial de los pasos anteriores.

$$v_t = \beta_1 \cdot v_{t-1} - (1-\beta_1) \cdot \nabla f(w_t) \tag{2.26}$$
$$s_t = \beta_2 \cdot s_{t-1} - (1-\beta_2) \cdot \nabla^2 f(w_t) \tag{2.27}$$
$$\triangle w_t = -(n \cdot v_t/\sqrt{s_t + \varepsilon}) \cdot \nabla f(w_t) \tag{2.28}$$
$$w_{t+1} = w_t + \triangle w_t \tag{2.29}$$

Fijarse que existe un término ε, esto es para evitar la división por cero. Esto pues en los primeros pasos no existe registro del gradiente, por lo que podría llegar a ser cero el término s_t.

Es importante notar que ningún optimizador es mejor que otros, algunos convergen más rápido que otros, pero es importante experimentar con ellos pues todo dependerá de cómo este conformada la red y el objetivo que se le quiera dar.

2.2.7. Fase de Prueba

En esta fase los parámetros como los pesos w de los enlaces no serán modificados. Esto dado que la red no está en fase de aprendizaje, sino que solamente clasificará el parámetro de entrada. En este proceso la red realizará una propagación hacia delante partiendo en la capa de entrada con el input y calculará las probabilidades finales de clasificación.

Esta debe tener un modo de evaluar la clasificación. Para esto existen distintas métricas que se pueden calcular y así obtener un indicador de qué tan efectiva es nuestra red. Para esto debemos hacernos la siguiente pregunta. ¿Cómo sabemos si nuestro modelo generaliza bien? ¿Nuestro modelo está generando overfitting?

Para revisar si nuestro modelo está generando overfitting, se puede medir el desempeño de la arquitectura con los datos de entrenamiento y compararlos con los de prueba y compararlos. Si esta brecha es muy alta, quiere decir que nuestro modelo está memorizando y no aprendiendo.

Una vez se ha entrenado el modelo, podemos obtener predicciones sobre unos datos. Para cada clase que queremos clasificar tenemos cuatro opciones posibles. Esto es discutido en la Sección 2.5 del documento.

2.2.8. Transfer Learning

Esta es una técnica muy utilizada en el mundo de las redes neuronales [49]. Consiste en seleccionar una arquitectura ya entrenada (pesos ya calculados y asignados), la cual sabemos que es buena en su propia tarea y adaptarla para nuestro problema. La idea es simple, primero seleccionamos dicha arquitectura, quitamos las ultimas capas (de clasificación) y ponemos las nuestras.

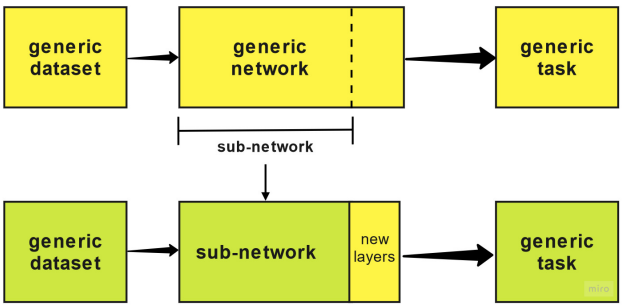


Figura 2.11: Proceso de Transfer learning

Es importante mencionar que, en el proceso de entrenamiento del modelo modificado, solo se calculan los pesos de la capa puesta por nosotros. El resto del modelo queda estático a cambios en sus pesos.

Existe otro proceso un poco más complejo denominado **Fine Tuning** [47]. En este se calculará el gradiente para parte de la arquitectura ya entrenada (o incluso el modelo entero). Esto quiere decir que los pesos del modelo elegido serán modificados y adaptados para resolver

nuestro problema. El entrenamiento de este modelo adaptado puede demorar más tiempo. Hay que recordar que la mayoría de los modelos pre-entrenados que existen tienen un alto grado de complejidad y fueron entrenados con súper computadores.

Cuál de estas técnicas escoger dependerá exclusivamente de nuestros datos. Si estos son parecidos a los datos originales del modelo escogido, solo nos bastará adaptar la última capa agregada. En cambio, si los datos difieren de los originales, será mucho mejor pulir el modelo completo para obtener mejores resultados.

Entre algunos modelos ya entrenados se encuentra:

- AlexNet [30]
- VGG [31]
- ResNet [20]
- DenseNet [22]
- BERT (para texto) [13]

2.3. Redes neuronales para imágenes

2.3.1. CNN

Las redes convolucionales [6] en imágenes permiten obtener ciertas características de estas. Su funcionamiento consta en el elegir una cantidad de kernels dependiendo la cantidad de características que se quieran extraer. Estos kerneles son matrices cuadradas del mismo tamaño. Una vez seleccionadas se aplica la operación de convolución sobre la imagen con estas matrices. El resultado de esta operación se pone en una nueva matriz, la cual tendrá tantas dimensiones de características como cantidad de kernels elegidos.

Es importante ver que para esta operación se tiene que elegir un número de padding y stride. El padding indica cuanto borde se le coloca a la imagen. Esto se hace para permitirle al kernel pasar por todos los píxeles de la imagen. Por otra parte, el stride indica cuanto mover el kernel entremedio de cada convolución. La operación de convolución es $kernel * input = i1k1 + i2k2 + i3k3 + i4k4 + i5k5 + i6k6 + i7k7 + i82k8 + i9k9 = J$

A este tipo de redes también se le aplica el proceso de Pooling [6]. Este se encarga de reducir las dimensiones de las imágenes, reduciendo el número de píxeles de la matriz final. Es importante notar que esto no disminuirá la cantidad de dimensiones que tenga. Existen dos tipos de Pooling.

- **Max Pooling:** Este toma el sector de la matriz que le corresponde y toma el valor del píxel máximo

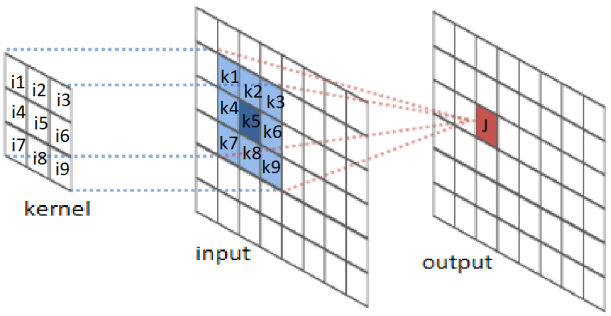


Figura 2.12: Proceso de convolución

- **Average Pooling:** Este toma el sector de la matriz y saca el promedio entre todos los píxeles

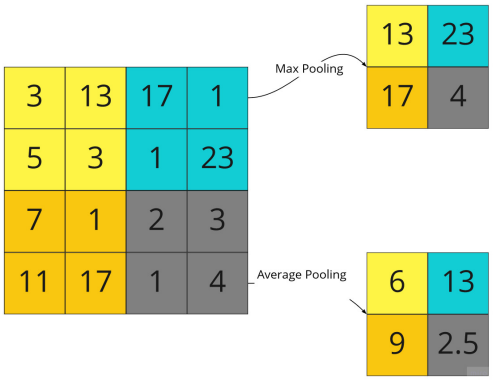


Figura 2.13: Proceso de pooling

Las redes CNN se componen de muchas capas de estos procesos. Se van modificando parámetros como la cantidad de kernels y el stride o padding de estos. Al final de las redes siempre se llega a una cada de 1×1 con varias dimensiones (representando las características). Esta es una capa fully connected y con ella se hace la predicción aplicando la función softmax.

2.3.2. Resnet

Residual Network (ResNet) [20] es uno de los modelos más famosos de redes neuronales para el reconocimiento de imágenes. Este introduce distintas técnicas muy útiles a la hora de entrenar y modelar redes neuronales.

Esta red fue entrenada con el dataset CIFAR-10 [50]. Este dataset es muy utilizado para entrenar y probar las distintas redes del tipo CNN.

Bloques residuales

Uno de los grandes problemas que existe a la hora de entrenar redes tan grandes es que no todas las capas logran recibir el gradiente correspondiente. Esto genera que su aprendizaje

se vea afectado en gran manera. Los bloques residuales [20] fueron creados para aliviar el problema de entrenamiento con redes tan grandes.

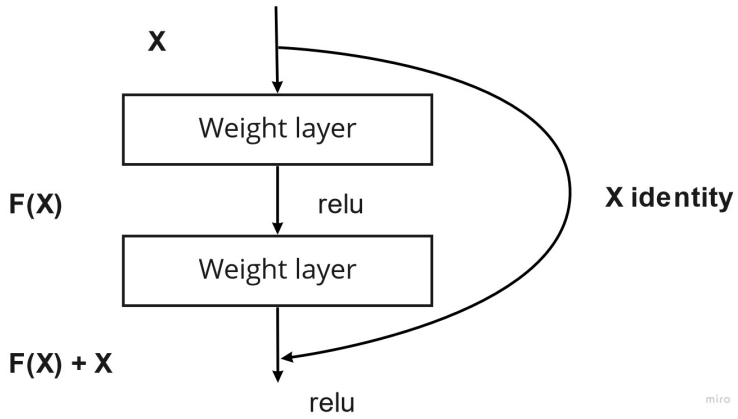


Figura 2.14: Bloque Residual

En la Figura 2.14 se puede ver que existe una conexión directa que salta algunas capas del modelo. Esta conexión se denomina conexión de salto [20] y es el corazón de los bloques residuales. Esto permite que la red decida saltarse algunas neuronas.

Esta idea de conexión de salto deja las salidas como.

$$H(x) = f(w \cdot x) + b \tag{2.30}$$
$$H(x) = f(x) \tag{2.31}$$

Esto quiere decir que la red puede decidir entre usar la función que usa los pesos de la capa de la red o una función identidad que hace saltarse la capa.

Esta técnica de conexión de salto en ResNet permite resolver el problema de la desaparición del gradiente en las CNN profundas al permitir un camino alternativo para que fluya el gradiente. Esta idea del gradiente permitió a las redes del tipo CNN ser mucho más profundas de lo que eran previamente, haciéndolas mucho más efectivas a la hora de tratar la clasificación de imágenes. También la red aprenderá a ver si alguna capa daña el rendimiento, omitiéndola mediante la regularización.

Existen distintas versiones de ResNet [20] en donde varían la cantidad de capas. Estas son:

- ResNet18
- ResNet34
- ResNet50
- ResNet101
- ResNet152

2.4. Redes neuronales para texto

2.4.1. Word Embedding

Word Embeddings [7] es una técnica del Procesamiento de Lenguaje Natural. Consiste básicamente, en asignar un vector a cada palabra. Este vector guarda información semántica, lo que permite ser asociado o disociado a otros vectores según el contexto gramatical en el que se encuentre. La idea consta en que el vector de una palabra tendrá una cierta dimensión. Cada valor en este vector corresponde a una característica en particular. El valor será mayor o menor dependiendo el contexto que se encuentre la palabra.

Es importante notar que existe lo que se conoce como *EmbeddingDim*, es decir la dimensión que tendrán los vectores el embedding. Esto proporcionará cuantas características se quiere que tenga cada vector de las palabras. Por otro lado, estos valores que poseen los vectores serán ajustados en el entrenamiento de la red, por lo que en un inicio tienen valores al azar.

Existen modelos ya pre-entrenados que poseen su propio vocabulario con los embeddings para cada palabra y sus valores correspondientes.

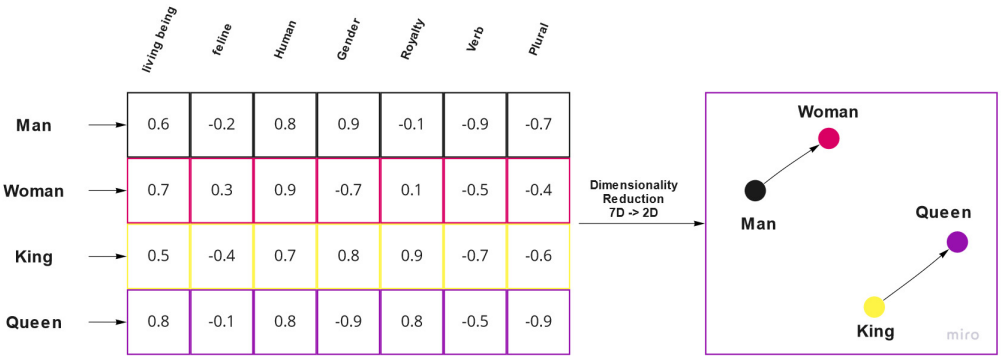


Figura 2.15: Word Embeddings

En la Figura 2.15 se puede observar que existen los vectores para las palabras *man*, *woman*, *king* y *queen*. Cada valor de su respectivo vector corresponde a una cierta característica, que pueden ser $feline$, *human*, *gender*, etc. La idea es poder llevarlo a un espacio de dos dimensiones para ver qué tan cercanas o lejanas son las palabras entre sí. Por otro lado, al restar o sumar estos vectores podría llegar a obtenerse otro tipo de palabras. Por ejemplo, si al vector de *king* le restamos el de *man* y sumamos el de *woman* debiésemos obtener un vector cercano al de *queen*.

2.4.2. Lineal

Este modelo se basa en la técnica de EmbeddingBag [27]. La característica principal es que calcula los embeddings de las palabras, pero no trabaja con todo el vector obtenido, sino

que realiza operaciones para disminuirlo la mayor cantidad posible. Las operaciones pueden ser promediar el vector, obtener el máximo, etc. Esto le permite a la red poder ser entrenada de una manera mucho más rápida, pero a su vez se pierde información en el camino. Por ejemplo en la Figura 2.16 se puede ver que la palabra *man* fue reducida por medio del atributo max a solo ocupar el atributo *gender*

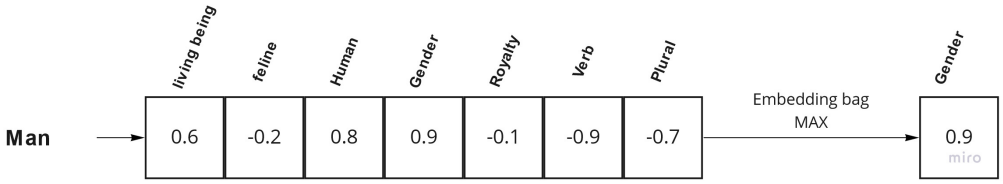


Figura 2.16: Operación max para embedding +

Este proceso se realiza en una capa de la red neuronal conocida como EmbeddingBag. Durante el entrenamiento de esta se van ajustando los valores de los embedding para así mejorar la precisión que poseen con cada palabra que esté presente en el vocabulario. También se pueden agregar embedding ya entrenados que por lo general se logran mejores resultados.

La capa de EmbeddingBag va unida a una red fully connected, es decir, una celda está conectada con todas las celdas de la siguiente capa. De manera que los resultados de las operaciones al embedding sean utilizados en el cálculo de la red y así realizar una predicción acorde al problema.

2.4.3. RNN

Este tipo de redes [38] tienen la peculiaridad de que en cada instante de tiempo se calcula una predicción. Por otro lado, cada valor calculado en un instante de tiempo se propaga al siguiente. Esto quiere decir que tiene una especie de memoria.

En la Figura 2.17 se puede observar que en cada espacio de tiempo existe una entrada X, y cada estado H calculado se propaga hacia las capas más profundas. Fijarse que en cada instante de tiempo t se hace una predicción. Todas las predicciones se unen en una *Loss* final.

2.4.4. LSTM Bidireccional

Las redes LSTM [21] son un tipo especial de redes recurrentes. La característica principal es que este tipo de modelos aprende a recordar ciertas entradas en un plazo más largo en la profundidad de la red. Se podría decir que tiene una mayor memoria que una red recurrente estándar.

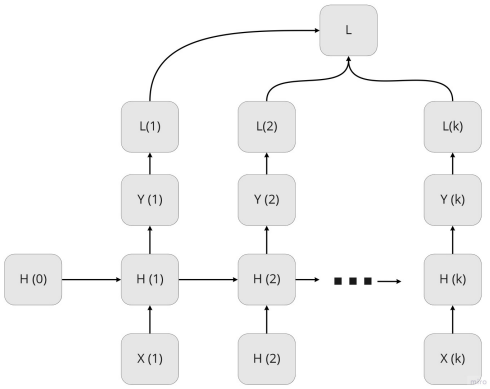


Figura 2.17: Modelo Recurrente

Consideremos la Figura 2.18, en ella se puede ver un nodo H, que corresponde al valor actual de la red recurrente. Un nodo C corresponde a la celda de memoria. También hay un nodo $C \sim$ que es la celda de memoria candidata y finalmente X que es el input en el tiempo t. Es importante recordar que como se trata de una red recurrente, esta tiene capas para cada instante de tiempo. Las flechas entrecortadas señalan los parámetros que indicarán cuanto de la celda de memoria pasa al estado final. Fijarse que pasará un porcentaje de la celda en el tiempo $t-1$, de la celda candidata y la celda propia del tiempo t. Lo interesante de este modelo, es que la red aprenderá cuanto porcentaje pasará de cada celda dependiendo los inputs que se entreguen.

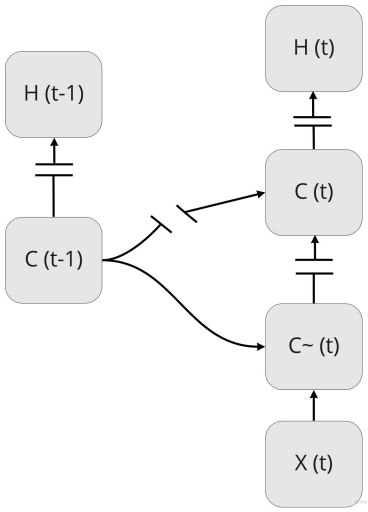


Figura 2.18: Red LSTM

2.4.5. Convolucional

Estas [28] redes funcionan de igual manera que las convolucionales para las imágenes, pero en una dimensión, es decir que el kernel actuara en solo una dimensión sobre el vector de embeddings. Esto le permitirá ir sacando ciertas características sobre palabras que se encuentran juntas en una frase.

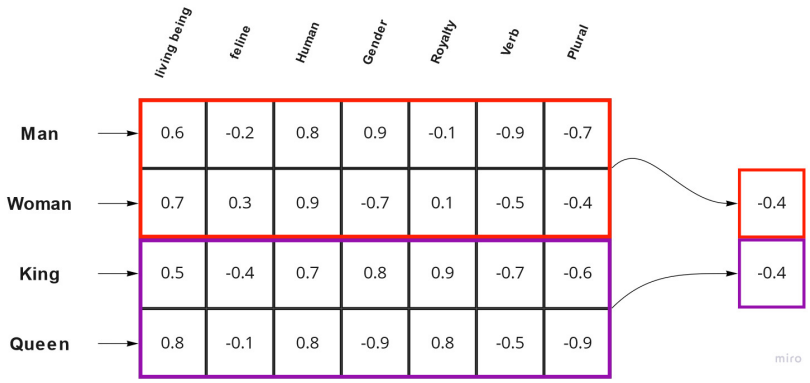


Figura 2.19: Red convolucional

Como se puede ver en la Figura 2.19, el kernel trabaja sobre las dos primeras palabras y las dos últimas, obteniendo un vector con las características de la relación de estas palabras correspondientemente. Esto se va haciendo sobre todas las palabras que están en la frase que se quiere clasificar. Es importante ver que los valores del kernel se van adaptando en el entrenamiento del modelo

2.4.6. BERT

BERT [13] (Bidirectional Encoder Representations from Transformers) o Representación de codificador Bidireccional de Transformadores es una técnica basada en redes neuronales para el pre-entrenamiento del procesamiento de lenguaje natural desarrollada por google. La innovación que presenta BERT es el aplicar el entrenamiento bidireccional de un Transformer, un modelo de atención del modelado de lenguaje.

BERT hace uso de Transformer [44], un mecanismo de atención que aprende relaciones contextuales entre palabras o subpalabras en un texto. Estos incluyen dos mecanismos, un codificador que lee la entrada del texto y un decodificador que produce una predicción.

- **Codificador:** Son varias unidades recurrentes juntas. Estas pueden ser del tipo LSTM, GRU, etc. Cada una de estas acepta un solo elemento de la secuencia de entrada, recopila información para ese elemento y propaga hacia adelante.
- **Vector codificador:** Tiene el objetivo de encapsular la información de todos los elementos de entrada para pasárselos al decodificador
- **Decodificador:** Varias unidades recurrentes donde cada una predice una salida en el tiempo.

Un transformador básico consta de un codificador para leer la entrada de texto y un decodificador para producir la predicción.

Para el proceso de aprendizaje se enmascaran al azar alrededor del 15 % de las palabras en una oración con un símbolo o token (MASK) y luego trata de predecirlas basándose en las palabras que rodean a la palabra enmascarada. Así es como BERT puede ver las palabras de izquierda a derecha y de derecha a izquierda.

A los datos de entrada se le agrega cierta información útil para el modelo. Primero se ponen token embeddings para marcar el principio y final de las oraciones. Luego se ponen segment embeddings para distinguir entre las distintas oraciones y por ultimo los positional embeddings para indicar la posición de las palabras en una oración.

Una vez termina de predecir las palabras, el modelo aprovecha la predicción de la siguiente oración. Esto le permite a BERT analizar la relación entre dos oraciones haciendo comprender mejor el contexto de todo el conjunto de datos.

2.4.7. RoBERTa

RoBERTa [32] presentado por Facebook, es un enfoque de BERT sólidamente optimizado. A esta se le mejora el algoritmo con el que se entrena y además fue entrenada con 10 veces más cantidad de datos, lo que aumenta su potencia de cómputo.

Para mejorar la fase de entrenamiento, RoBERTa elimina la predicción de la próxima oración como lo hace BERT. Por otro lado, introduce un enmascaramiento dinámico para que los tokens cambien durante el entrenamiento.

Una de las grandes mejoras de RoBERTa es que usa 160GB de texto para su entrenamiento, incluidos los 16Gb que utiliza BERT. Esto permitió superar a BERT en los resultados de referencia de GLUE [45].

2.5. Métricas de evaluación

La Búsqueda y Recuperación de Información [11], llamada en ingles Information Search and Retrieval, es la ciencia de búsqueda de información en cualquier tipo de documento electrónico. La idea principal consta de extraer metadatos que describan a estos documentos para así realizar distinto tipos de tareas.

Existen distinto tipos de medidas que han sido propuestas para evaluar el rendimientos de estos sistemas de recuperación de la información. Las más utilizadas son Accuraccy, Precision, Recall y F1 Score.

Es importante mencionar que con positivo y negativo nos referimos a cada clase por separado.

- **TP (True Positive):** Son los valores que la red clasifica como positivos y que realmente son positivos.

- **TN (True Negative):** Son valores que la red clasifica como negativos y que realmente son negativos.
- **FP (False Positive):** Son valores que la red clasifica como positivo cuando realmente son negativos.
- **FN (False Negative):** Son valores que la red clasifica como negativo cuando realmente son positivos

A partir de estos datos podemos crear una matriz de confusión para cada clase que se requiere clasificar. De esta forma podemos medir que tan efectiva es nuestra red a la hora de realizar una clasificación. En la Figura 2.20 se puede observar que en la diagonal se encuentran los aciertos del modelo, mientras que en los otros lugares los desaciertos.

	Positive	**Negative**
Positive	TP	FP
Negative	FN	TN

Figura 2.20: Matriz de confusión

Existen distintas métricas para medir un modelo en general (tomando todas las clases). Cada una tiene una distinta función de medición, como por ejemplo cuando hay desbalance de clases en el dataset.

Accuracy

Esta métrica representa el porcentaje total de valores correctamente clasificados, tanto positivos como negativos. Esta no funciona bien cuando las clases están desbalanceadas. Es una de las métricas más utilizadas, pero no muy recomendable de utilizar.

$$accuracy = \frac{TP + TN}{TP + TN + FP + Fn} \tag{2.32}$$

Precision

Esta métrica de precisión es utilizada para poder saber qué porcentaje de valores que se han clasificado como positivos son realmente positivos.

$$Precision = \frac{TP}{TP + FP} \tag{2.33}$$

Recall

Esta métrica se utiliza para saber cuántos valores positivos fueron correctamente clasificados, es decir nos indica la muestra clasificada correctamente

$$Recall = \frac{VP}{VP + FN} \tag{2.34}$$

F1 Score

Esta es una métrica utilizada comúnmente cuando el conjunto de datos que se va a analizar se encuentra desbalanceado. Esta combina Precision y Recall para obtener un valor mucho más objetivo.

$$F1 = 2 \cdot \frac{Recall \cdot Precision}{Recall + Precision} \tag{2.35}$$

Capítulo 3

Procesamiento de Datos

En esta sección se abordará el tema de cómo fueron procesados los datos antes de ser utilizados en las redes neuronales. Uno de los grandes problemas de la memoria es que la información se encuentra desbalanceada. Esto trae muchos problemas a la hora de experimentar con redes neuronales pues el modelo podría hacer overifitting sobre alguna clase. Para esto se realizó un proceso de balanceo por pesos. Otra forma de dar una solución a este problema es generar más datos a partir de los existentes con una técnica llamada data augmentation. Finalmente se explicará el procedimiento que se tuvo que hacer al texto para que los modelos pudieran utilizarlo sin problemas.

3.1. Balance de clases

Uno de los grandes problemas que existe a la hora de trabajar con datos y clases es el posible desbalanceo que estas posean. Este desbalanceo para una red neuronal puede implicar que el modelo aprenda demasiado de una clase en específico ignorando al resto. Es por esto que se debe realizar un preprocesamiento a la hora de crear los batches que se le entregaran a la red a la hora de realizar el proceso de entrenamiento.

Existen distintas formas de tratar este problema. Una de estas consiste en calcular los pesos de cada clase [46] presente en el dataset de entrenamiento y a partir de este, asignar probabilidades a la hora de elegir datos en la creación de los batches.

El balanceo consiste en contar cuantos elementos de cada clase hay. Luego para cada categoría se divide el total de elementos por este conteo anterior, obteniendo el peso. Finalmente, se asigna una probabilidad a cada clase dependiendo el peso obtenido.

La probabilidad se usa para elegir un elemento de la clase para el batch, siendo más alta para clases con un mayor peso. Esto permite que la red entrene con elementos de manera equilibrada.

Consideremos N como la cantidad total de elementos con el que se generara un batch y t_{meme}, $t_{no-meme}$, $t_{sticker}$ la cantidad de cada clase respectivamente, V como el universo de

elementos que aún no son seleccionados y S los que si para un cierto Batch. Es importante notar que a cada elemento se le asigna el peso de su clase.

Algorithm 1 Balance classes

$w_{meme} \leftarrow N/t_{meme}$
$w_{no-meme} \leftarrow N/t_{no-meme}$
$w_{sticker} \leftarrow N/t_{sticker}$
$M \leftarrow$ total elements
for k = 1 to M **do**
    $p_i \leftarrow w_i / \sum_{s_j \in V-S} w_j$be the probability of item i to be selected in round k and include in S.
end for

Es importante notar que este balanceo se hace solamente para la clase de entrenamiento. Cuando se está probando la red neuronal, los batches se generan a partir de la proporción en la que vienen los datos, es decir no pasan por ningún proceso de balanceo.

3.2. Aumento de datos

Cuando se entrena un modelo de red neuronal, lo que se está haciendo es afinar ciertos parámetros como los pesos, etc. En particular se le entrega un input (imagen o texto) y esta entrega un *output* o clasificación. Lo ideal sería entregarles todos los datos dentro del universo que se está explorando para que así la red logre generalizar. El problema de esto es que obtener una gran cantidad de datos es muy caro y a veces imposible. Es por esto por lo que existen técnicas para aumentar la cantidad de data disponible sin necesidad de ir en búsqueda de más.

Data Augmentation en imágenes

Para una red, una imagen es un tensor con cierta cantidad de dimensiones (dependiendo el formato de colores, etc). Si por ejemplo tenemos una imagen de un gato y la rotamos, esta seguirá siendo un gato, pero para la red habrán cambiado muchos valores de posición por lo que para el modelo será una nueva foto.

Algunas de estas técnicas van desde rotar la imagen cierta cantidad de grados, modificar la tonalidad de colores (generalmente ir hacia los grises), hacer zoom a ciertas partes de la imagen, voltear la imagen, etc.

Es importante mencionar que cuando se hace alguna de estas técnicas a una imagen, su categoría debe permanecer constante. Es decir, si hacemos data augmentation [35] a una foto de un perro, los resultados seguirán siendo fotos de un perro, no se puede cambiar el label.

Data Augmentation en texto

Aplicar técnicas de data augmentation a texto es un caso muy distinto al explicado anteriormente. En este debemos tener cuidado con la estructura gramatical del texto. Esto dado que los modelos de redes neuronales utilizan el contexto para realizar clasificaciones. Si modificamos este, se obtendrá un dato con un *label* totalmente distinto.

Un método utilizado llamado *back translation* [14] consiste en utilizar una API de Google para traducir el texto a un idioma y luego volver al idioma original. Esto hace que la estructura del texto se modifique conservando el contexto.

También se pueden utilizar los sinónimos. Es decir, escoger ciertas palabras clave dentro del texto (no stop words) y modificarlas por su sinónimo. Esto permitirá que el texto conserve su contexto.

Existen algunas técnicas que utilizan la aleatoriedad para cambiar ciertas palabras de orden, eliminarlas o incluso añadir palabras nuevas. Esto se debe hacer manteniendo el sentido de la frase. Una de estas realiza un cambio en el orden de ciertas frases de la oración. Esto quiere decir que, si tenemos un texto, podemos tomar una frase y ponerla antes de otra. Esta es utilizada cuando se está trabajando con textos largos.

3.3. Procesamiento de texto

Cuando trabajamos con texto, debemos encontrar una forma de poder entregárselo al computador y que este lo entienda y pueda trabajar con él. Es por esto por lo que el texto debe pasar por un pre-procesamiento antes de entrar a la red neuronal. Para esto vamos a tener un vocabulario con todas las palabras disponibles en los textos y una enumeración que corresponderá a su posición en el vocabulario. Luego de esto se recorrerá palabra por palabra en cada texto y se irá creando un vector con la posición de la palabra y su valor en el vocabulario.

Para esto se debe programar un tokenizador. Este se encarga de realizar el procesamiento antes explicado para cada texto en el dataset. También tiene la tarea de crear el vocabulario inicial (pues por lo general se crea a medida que se van leyendo los textos).

Este tokenizador hace algunas modificaciones para hacer más fácil el procesamiento. Realiza un proceso de *lower* a los textos, es decir pasa todo a minúsculas y de esta manera se ahorra problemas con diferenciar palabras iguales pero que difieren en las mayúsculas (para un computador $casa \mathrel{!}= Casa$). También se encarga de eliminar las stop words. Estas son palabras comunes dentro del lenguaje que no aportan nada al contexto, por ejemplo, “el”, “las”, “los”, “que”, etc.

Dentro del vocabulario existen palabras especiales que sirven para ayudar al proceso de aprendizaje. Una de estas es el $< pad >$. Algunos modelos de redes neuronales requieren que las entradas sean del mismo tamaño, cosa que con los textos no suceden, es por esto que se agrega el *padding*. Para hacer este proceso se toma el texto más largo y todos se llevan a ese largo, agregando el $padding < pad >$ correspondiente.

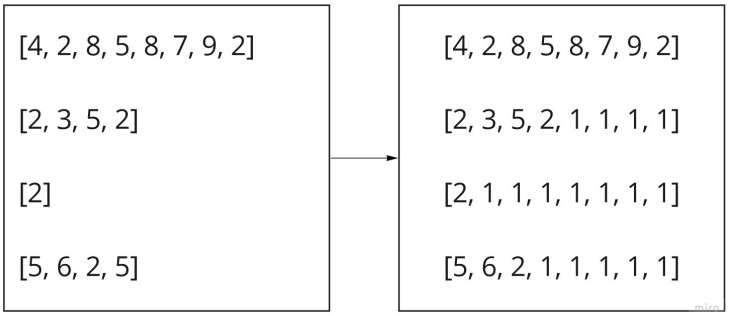


Figura 3.1: Proceso de padding

3.4. Extracción de texto automatico

Al trabajar con texto de una imagen es importante tener un método eficaz de extracción de texto. Para esto se utilizan técnicas de OCR (Optical Character Recognition) [34] junto a redes neuronales capacitadas en la detección de este. Por lo general existe un preprocesamiento a la imagen que permita aislar el texto para así facilitar la tarea de detección a el modelo. Existen muchos modelos en la actualidad disponibles para su utilización. Uno de ellos es EasyOCR.

EasyOCR [1] es una librería de OCR basada en Python que extrae el texto de una imagen. Esta realiza ciertos pasos de preprocesamiento (pasando la imagen a escala de grises) de manera de preparar la imagen de mejor manera antes de pasársela al modelo. Luego aplica el algoritmo de CRAFT (Character Region Awareness for Text Detection) [8]. CRAFT se encarga de determinar el área de una imagen donde existe texto y viendo la afinidad entre los distintos caracteres, creando delimitadores mínimos en un mapa binario. Un ejemplo de esto se puede apreciar en la Figura 3.2.

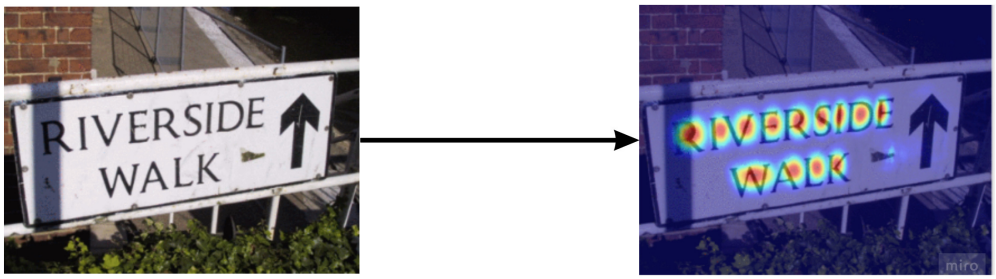


Figura 3.2: Algoritmo de CRAFT

El resultado es entregado a distintos modelos. Para el reconocimiento se utilizan redes de CRNN (Convolutional recurrent neural network) principalmente ResNet y el etiquetado de secuencia se realiza mediante modelos de LSTM y CTC [18] de manera de etiquetar los datos de secuencia con RNN. El modelo general se puede resumir en la Figura 3.3.

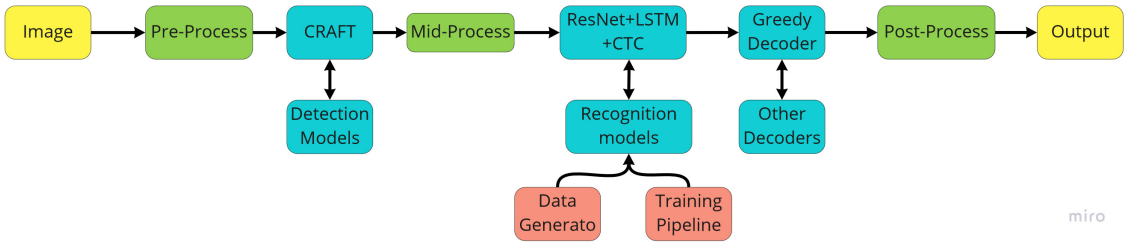


Figura 3.3: Modelo EasyOCR

Existen otras librerías que emplean técnicas de OCR como tesseract OCR [5] de Google, pero que emplean métodos muy semejantes entre sí, variando algunos modelos o preprocesamientos.

Capítulo 4

Clasificación de memes

La clasificación de memes consiste en intentar discriminar cuales imágenes corresponden a memes y cuáles no. Dentro de estas categorías existen las de stickers por lo que esta tarea consta de tres tipos de clases distintas.

En esta sección se abordará todo el proceso realizado con la clasificación de memes. Se explicará el problema a intentar resolver como también los datos que se tienen para realizar experimentos. Por otro lado, se hablará de los distintos modelos que se utilizaran para los distintos experimentos, explicando cada decisión tomada a la hora de crear las redes neuronales.

Finalmente se mostrarán y analizarán los resultados obtenidos, concluyendo y explicando que se podría mejorar para lograr mejores clasificaciones.

4.1. Problema

En el Internet de nuestros días existe una gran difusión de información. Las imágenes son una estructura muy utilizada para divulgar información, siendo los memes un tipo de estas. Es por esto que se necesita poder distinguir si una imagen es un meme o es simplemente una imagen cualquiera. Esto para poder analizarlas en mayor profundidad y entender ciertos patrones que se presentan en redes sociales de una manera mucho más sencilla.

El problema a investigar a continuación se centra principalmente en dar un clasificador de imágenes que logre comprender cuando una imagen es del tipo meme y cuando no. Para esto se explorarán técnicas de deep learning tanto para la imagen como para el texto.

4.2. Dataset a utilizar

El dataset utilizado fue proporcionado por el IMFD y fue creado por un grupo de expertos en el tema. Este se basó en un estudio sobre el uso de memes en la plataforma twitter.

Contiene datos desde mayo de 2019 hasta mayo de 2020. El procesamiento de la información tuvo dos etapas. La primera constó en clasificar el dato en meme y no meme, en donde a los codificadores se le entregaron instrucciones para identificar estas imágenes, según la presencia de humor, intertextualidad, etc. En una segunda etapa se identificaron los distintos tópicos presentes. Para verificar la calidad de la codificación se usaron los valores de agreement y Krippendorff's alpha. El coeficiente alfa de Krippendorff es una medida estadística del acuerdo alcanzado al codificar un conjunto de unidades de análisis. Estos análisis deben tener un mínimo de rigor además de que participen más de dos investigadores [2]. El dataset usado en esta parte contiene alrededor de 52 mil imágenes divididas en:

- Meme: 1.194
- Sticker: 1.443
- No meme: 49.347
- Dudoso: 16

Es importante notar que la clase no meme no incluye a las imágenes del tipo Sticker. Por otro lado, la clase *Dudoso* fue descartada para la realización de los experimentos, pues no se llego a un acuerdo sobre a que clase pertenecía.

El dataset contiene los siguientes atributos:

- descriptions: Una pequeña descripción de la imagen
- interpretations: Una interpretación de la imagen
- urls: url original de donde fue obtenida
- localurls: url local donde fue almacenada
- targets: Tipo de imagen (meme, no meme, sticker o dudosa)
- imgids: id de la imagen
- texts: Texto de la imagen si contiene

El dataset también contiene un vocabulario en forma de diccionario, donde cada palabra posee un índice. Esto pues los textos, interpretaciones y descripciones vienen en forma de vector con estos índices. Los datos fueron divididos en un 70 % de entrenamiento y un 30 % para pruebas.

4.3. Modelos

Para abordar este problema se utilizaron dos modelos específicos. Uno utiliza solo una red neuronal para clasificar imágenes, en donde se utilizaron redes del tipo CNN y las ResNet.

Por otro lado, se programó un modelo mixto que mezcla redes para analizar imágenes y redes de NLP para clasificar textos. Para textos se usaron redes de LSTM, lineales, RNN y BERT.

Modelo de imagen

Este es el modelo más básico. Las imágenes de entrada son redimensionadas a un tamaño de 56×56 y entregadas al modelo y este realiza las predicciones. Es importante notar que para el caso de la red CNN se entrenó desde 0 mientras que en el caso de las ResNet se realizó un trabajo de fine tuning. Esto quiere decir que venía con pesos precargados y fueron adaptados por medio de entrenamiento a la clasificación deseada.

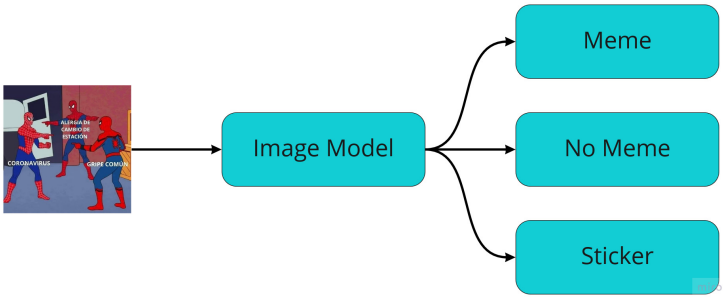


Figura 4.1: Modelo de imagen

Se opto por experimentar con este modelo dada la naturaleza del problema. Al tratar se analizar imágenes, uno de los modelos más básicos para esto se basa en hacer redes neuronales que clasifiquen este tipo de datos.

Modelo mixto

El modelo diseñado consiste en generar dos redes independientes. Una de esta trabaja solamente con el texto y la otra con la imagen. Una vez cada una termine el procesado, se unen en un modelo lineal, el cual realizará la predicción de la clase de la imagen. Para el caso de la imagen se utilizó solo un modelo convolucional, mientras que para el texto se usaron cuatro modelos por separado, los cuales son una red Recurrente, Lineal, Convolucional y LSTM Bi-direccional. Es importante notar que las dimensiones de entrada tanto para el modelo de imagen como el de texto es un vector de 256 elementos. Para el modelo de la imagen, la entrada es redimensionada a un tamaño de 56×56 mientras que el tamaño máximo del texto es de 128 caracteres.

Se decidió experimentar con este tipo de modelo pues las imágenes de los memes contienen un texto que le dan mucho valor como imagen. Es importante entender el contexto del mensaje de la imagen para que la red pueda discriminar entre un meme o una imagen normal. Por otro lado, tanto stickers como memes son una imagen con texto por lo que, para diferenciarlos, se necesita entender el mensaje que intenta entregar.

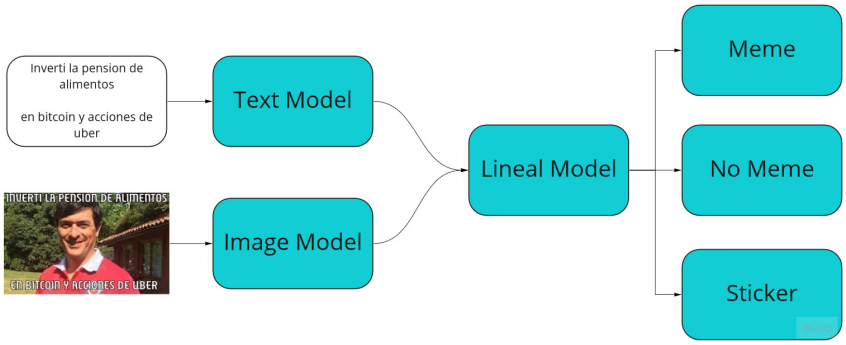


Figura 4.2: Modelo General

4.4. Experimentos

4.4.1. Experimento 1: Imagen

En estos experimentos solo se utilizarán modelos CNN creados desde cero para analizar las imágenes. Se modificarán ciertos parámetros como lo son la cantidad de kernels, el stride o el padding, además de redimensionar la imagen a ciertos tamaños iniciales.

Uno de los modelos aplicados consiste en dos capas convolucionales con un stride y padding de 1. El kernel utilizado fue de tamaño 3×3. Luego de una de estas capas, existe un pooling que reduce la imagen a la mitad del tamaño. El resultado es un vector de 256 elementos, el cual está listo para entrar a la parte lineal del modelo general.

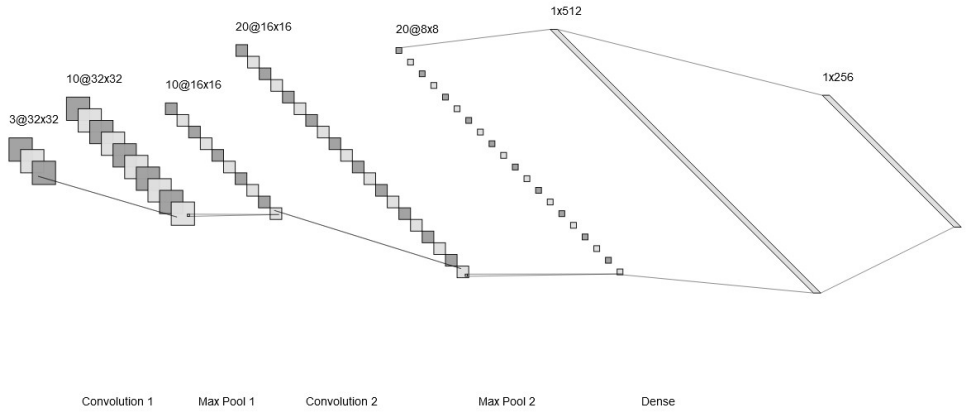


Figura 4.3: Modelo CNN para imagen

4.4.2. Experimento 2: Imagen con Fine tuning

Para esta sección de experimentos se probaron distintos modelos de CNN ya entrenados. Estos pertenecen a las redes neuronales ResNet a los cuales se le aplicaron técnicas de fi-

ne tuning y transfer learning de manera de aprovechar los parámetros ya calculados para ajustarlos a la tarea que se desea realizar.

Para los experimentos se probaron los siguientes tipos de ResNet.

- ResNet18
- ResNet34
- ResNet152

A estos se les extrajo la última capa de clasificación que tenían y se les puso una nueva capa fully connected con tres neuronas que corresponden a la capa de clasificación. Luego el entrenamiento se dividió en dos formas de hacerlo. La primera consistía en solo entrenar los pesos de la nueva capa final incluida. Esto significaba congelar el gradiente de la red ResNet de tal forma de no modificar los valores de sus pesos. La segunda forma consistía en entrenar todos los parámetros de la red. Es importante mencionar que este último proceso es mucho más lento que el primero.

4.4.3. Experimento 3: Modelo mixto

Para este modelo se decidió incluir el texto. Para esto se trabajan por separado tanto la imagen como el texto con sus respectivos modelos. Una vez cada red entregue su resultado en la última capa, se unen en una capa fully connected para realizar la clasificación. Es importante notar que el output debe tener las mismas dimensiones, pues se realiza una concatenación de los dos resultados. Este modelo se representa en la Figura 4.2.

Para el modelo de imagen se utilizaron las redes explicadas en los experimentos anteriores, las CNN y las del tipo ResNet. Mientras que en el modelo de texto se utilizaron distintos tipos de redes enfocadas a la clasificación de este. Es importante mencionar que, dependiendo el tipo de red, el texto se debió procesar de una manera distinta adaptándolo al modelo a utilizar.

Una red que se implementó es la que utiliza el método de EmbeddingBag. Esta posee el atributo max, es decir que una vez creados los embeddings para cada frase, se tomara el máximo (en la dimensión 0) y lo entregara como output a una red Lineal. Es importante que esta posea la dimensión del embedding, la cual fue 200 para los experimentos.

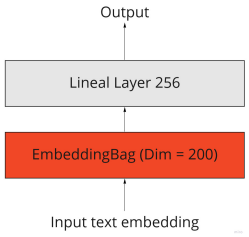


Figura 4.4: Modelo Lineal para el texto

Otro modelo que se utilizo fue la red LSTM. Esta fue una de las más complejas. Consta de una capa de embedding junto a dos capas de Long short-term memory, ambas del tipo bidireccional. Luego de pasar por las dos capas de LSTM, se hace un pooling de average y max, esto para disminuir las dimensiones de los outputs. Además, posee 3 capas fully connected. Este modelo tiene una salida de un vector de tamaño 256.

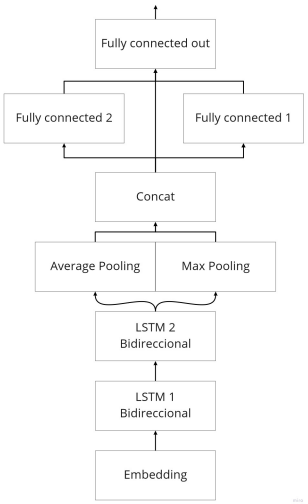


Figura 4.5: Red LSTM para texto

La red recurrente programada está compuesta de una capa de embedding, una recurrente bidireccional y finalmente una capa fully connected. Este modelo entrega un vector de 256 elementos. Es importante notar que las dimensiones de la salida de la capa recurrente se le deben ajustar las dimensiones.

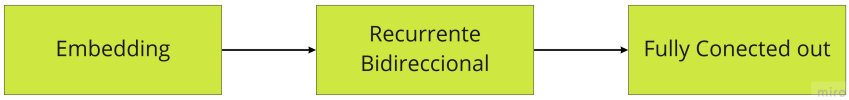


Figura 4.6: Red recurrente para texto

La red convolucional implementada está compuesta principalmente por una capa de Embedding. Además, posee una capa convolucional de una dimensión, con 256 canales de salidas y con un kernel del tamaño del embedding multiplicado por 3. Finalmente, se le aplica una función de activación relu.

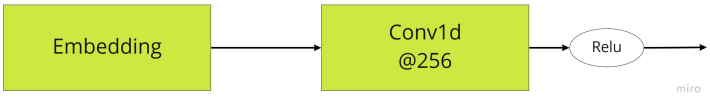


Figura 4.7: Modelo convolucional utilizado

4.4.4. Experimento 4: Modelo probado con texto automático

Los modelos del experimento pasado fueron entrenados con texto que había sido extraído manual, es decir presentaba un sentido y contexto correcto. Cuando se quiera usar las redes para clasificar nuevas imágenes, estas no contendrán el texto por lo que deberá ser extraído automáticamente usando librerías especiales.

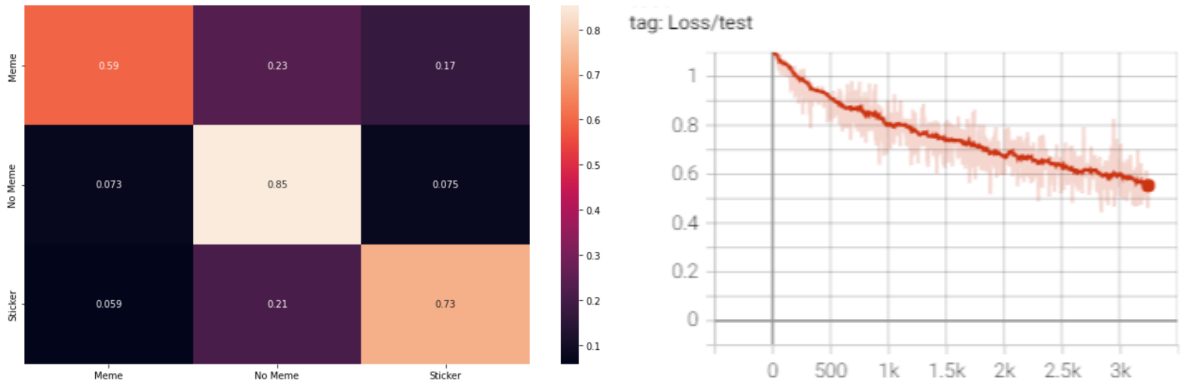
Este experimento consiste en probar con texto automático los modelos ya entrenados con texto manual. De esta forma se podrá ver cómo se comportan frente a situaciones de uso real. Para la extracción se utilizó una librería de Python *EasyOCR*.

Los resultados obtenidos aquí permitirán ver qué tan efectivo es la extracción de texto comparado a tenerlo manual. Además, que dará un acercamiento al uso que se le dará al modelo, pues antes de que la red clasifique necesitará el texto, el cual será extraído automáticamente.

4.5. Resultados experimentales

4.5.1. Experimento 1: Imagen

Modelo CNN



(a) Matriz de confusión

(b) Gráfico de Loss

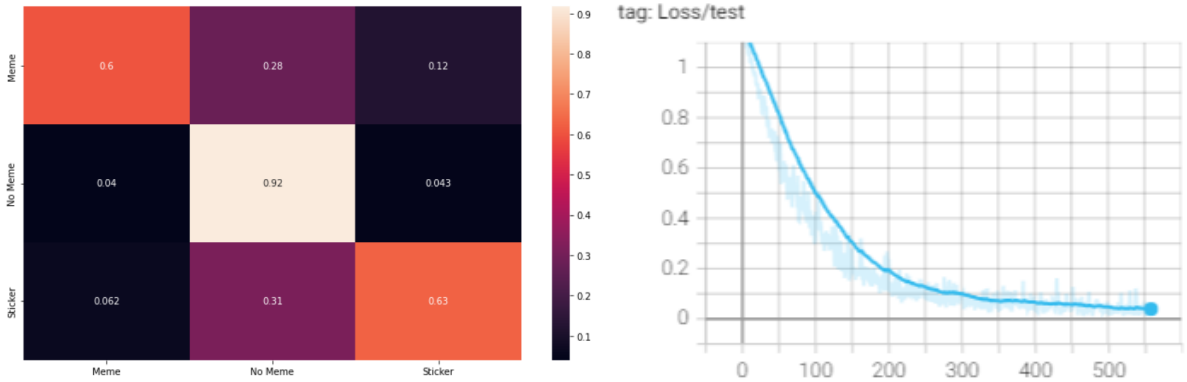
Figura 4.8: Modelo CNN

Las métricas obtenidas para este modelo fueron:

- **Accuracy**: 0.841
- **Recall**: 0.725
- **Precision**: 0.452
- **F1 Score**: 0.497

4.5.2. Experimento 2: Imágenes con fine Tuning

Modelo ResNet50



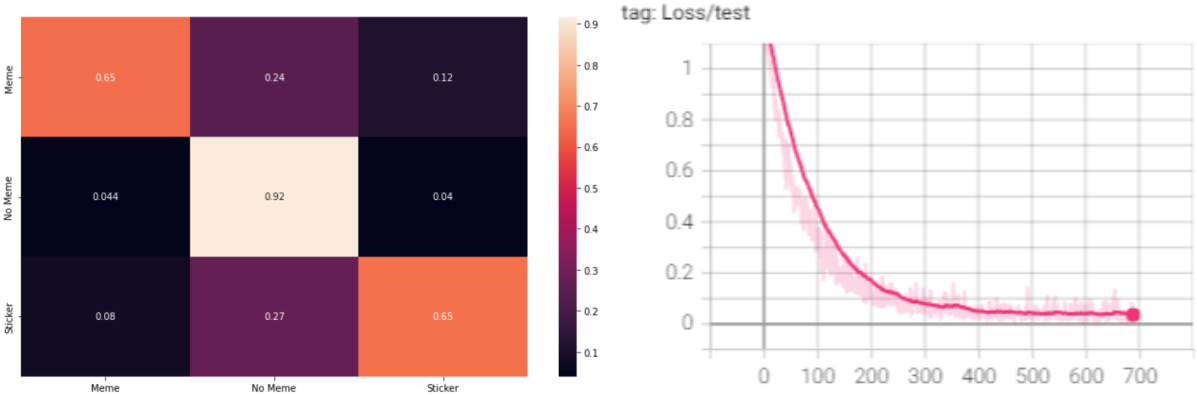
(a) Matriz de confusión (b) Gráfico de Loss

Figura 4.9: Modelo ResNet50

Las métricas obtenidas para este modelo fueron:

- **Accuracy**: 0.905
- **Recall**: 0.715
- **Precision**: 0.506
- **F1 Score**: 0.564

Modelo ResNet152



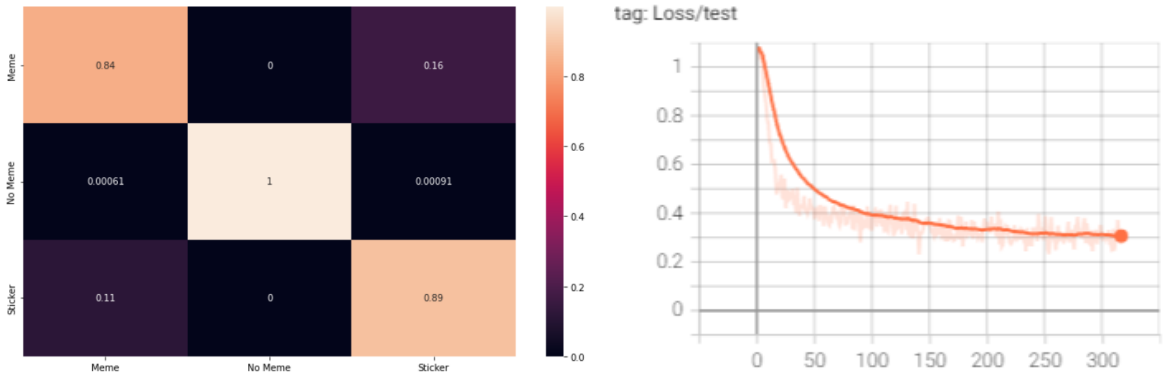
(a) Matriz de confusión (b) Gráfico de Loss

Figura 4.10: Modelo ResNet152

- **Accuracy**: 0.903
- **Recall**: 0.739
- **Precision**: 0.512
- **F1 Score**: 0.574

4.5.3. Experimento 3: Modelo mixto

Modelo CNN-Lineal



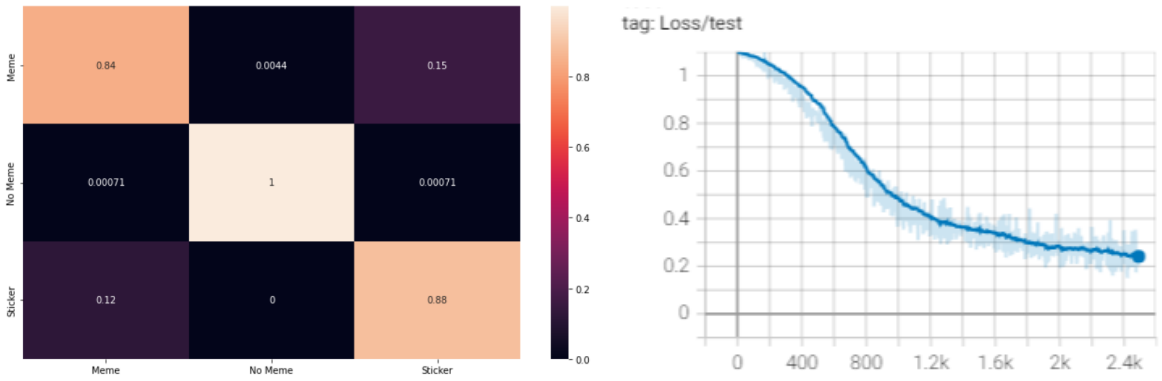
(a) Matriz de confusión (b) Gráfico de Loss

Figura 4.11: Modelo CNN-Lineal

Las métricas obtenidas para este modelo fueron:

- **Accuracy**: 0.992
- **Recall**: 0.908
- **Precision**: 0.893
- **F1 Score**: 0.901

Modelo CNN-LSTM



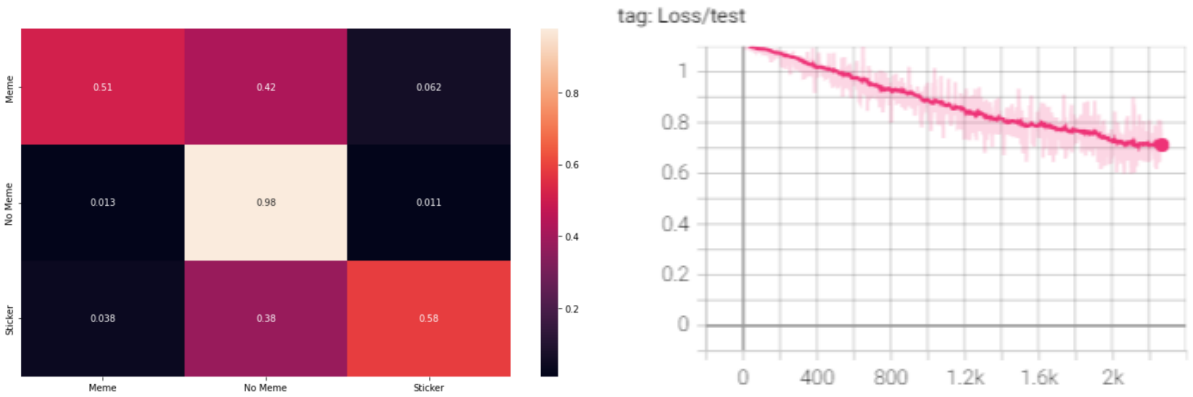
(a) Matriz de confusión (b) Gráfico de Loss

Figura 4.12: Modelo CNN-LSTM

Las métricas obtenidas para este modelo fueron:

- **Accuracy**: 0.992
- **Recall**: 0.907
- **Precision**: 0.894
- **F1 Score**: 0.900

Modelo CNN-RNN



(a) Matriz de confusión (b) Gráfico de Loss

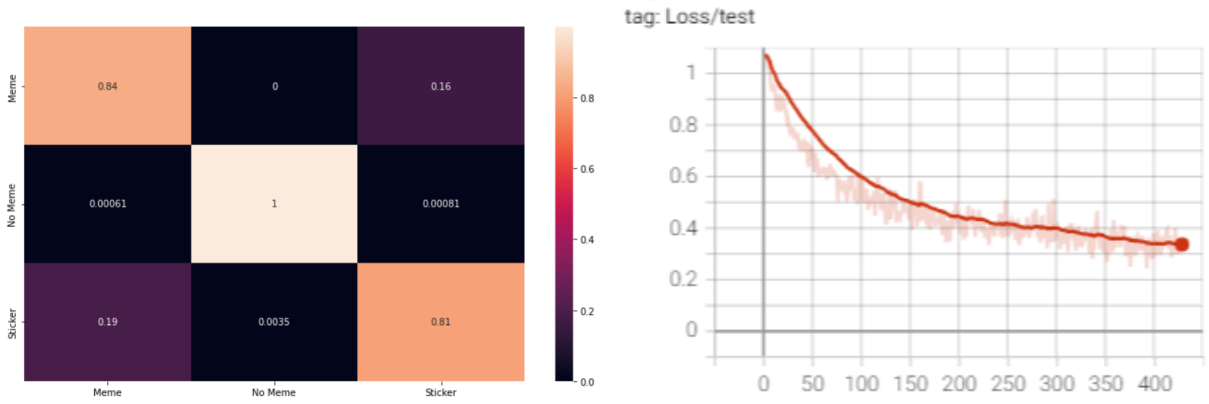
Figura 4.13: Modelo CNN-RNN

Las métricas obtenidas para este modelo fueron:

- **Accuracy**: 0.954

- **Recall**: 0.691
- **Precision**: 0.672
- **F1 Score**: 0.681

Modelo CNN-Convolucional



(a) Matriz de confusión

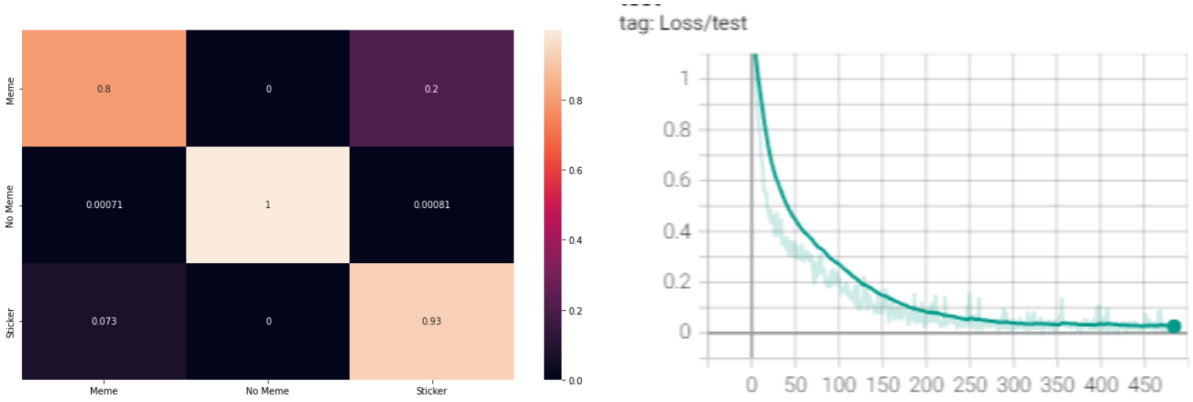
(b) Gráfico de Loss

Figura 4.14: Modelo CNN-Convolucional

Las métricas obtenidas para este modelo fueron:

- **Accuracy**: 0.989
- **Recall**: 0.881
- **Precision**: 0.866
- **F1 Score**: 0.873

Modelo ResNet50-Lineal



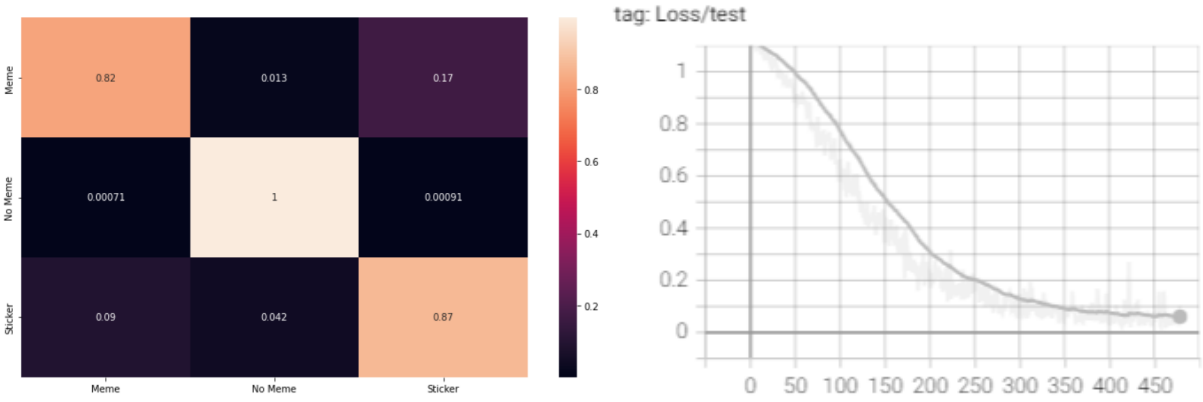
(a) Matriz de confusión

Figura 4.15: Modelo ResNet50-Lineal

Las métricas obtenidas para este modelo fueron:

- **Accuracy**: 0.992
- **Recall**: 0.907
- **Precision**: 0.899
- **F1 Score**: 0.902

Modelo ResNet50-LSTM



(a) Matriz de confusión

(b) Gráfico de Loss

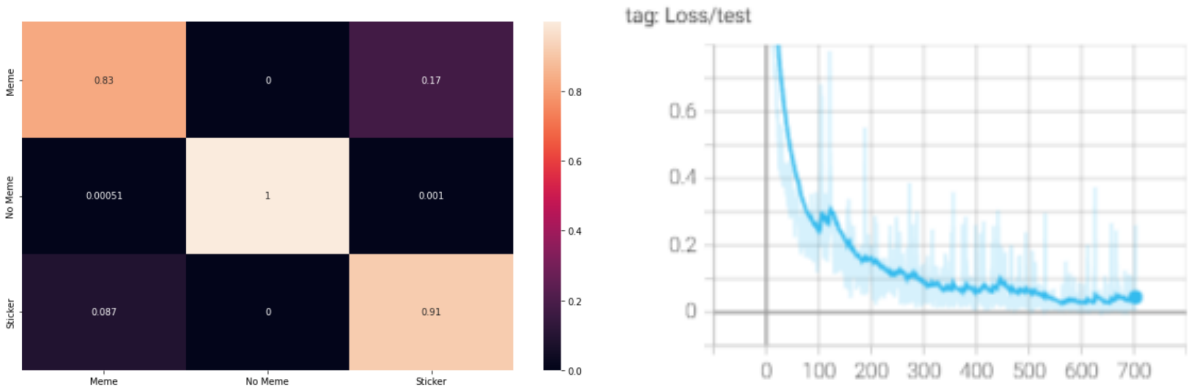
Figura 4.16: Modelo ResNet50-Lineal

Las métricas obtenidas para este modelo fueron:

- **Accuracy**: 0.991

- **Recall**: 0.895
- **Precision**: 0.896
- **F1 Score**: 0.896

Modelo ResNet152-BERT



(a) Matriz de confusión (b) Gráfico de Loss

Figura 4.17: Modelo ResNet152-BERTl

Las métricas obtenidas para este modelo fueron:

- **Accuracy**: 0.992
- **Recall**: 0.913
- **Precision**: 0.902
- **F1 Score**: 0.907

4.5.4. Experimento 4: Modelo probado con texto automático

Modelo ResNet152-BERT con texto automático

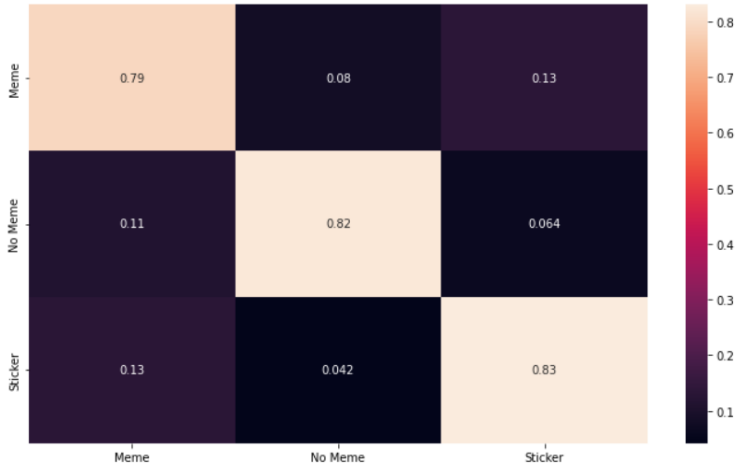


Figura 4.18: Matriz de confusión ResNet152-BERT probado con texto automático

Las métricas obtenidas para este modelo fueron:

- **Accuracy**: 0.819
- **Recall**: 0.814
- **Precision**: 0.466
- **F1 Score**: 0.512

Modelo CNN-BERT con texto automático

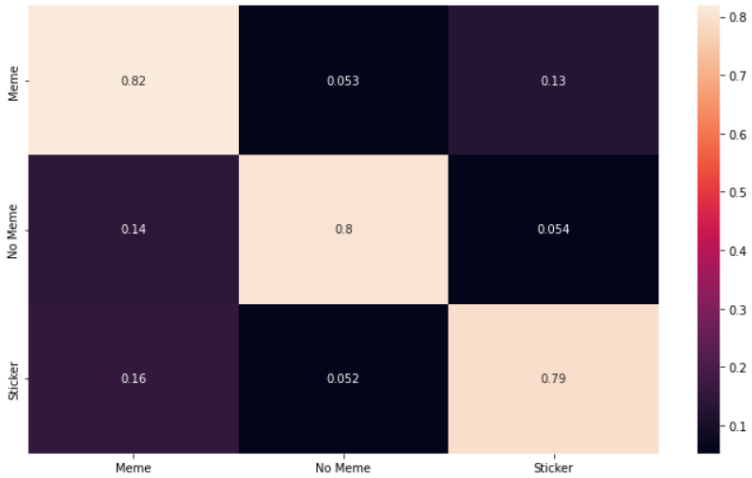


Figura 4.19: Matriz de confusión CNN-BERT probado con texto automático

Las métricas obtenidas para este modelo fueron:

- **Accuracy**: 0.780
- **Recall**: 0.805
- **Precision**: 0.466
- **F1 Score**: 0.503

4.5.5. Resultados Generales

Tabla resumen

Modelo	Dato analizado	Accuracy	Recall	Precision	F1 Score
CNN	Imagen	0.841	0.725	0.452	0.497
ResNet50	Imagen	0.905	0.715	0.506	0.564
ResNet152	Imagen	0.903	0.739	0.512	0.574
CNN-Lineal	Imagen-Texto	0.992	0.908	0.893	0.901
CNN-LSTM	Imagen-Texto	0.992	0.907	0.894	0.900
CNN-RNN	Imagen-Texto	0.954	0.691	0.672	0.681
CNN-Convolucional	Imagen-Texto	0.989	0.882	0.866	0.873
ResNet50-Lineal	Imagen-Texto	0.992	0.908	0.899	0.902
ResNet50-LSTM	Imagen-Texto	0.991	0.895	0.896	0.896
ResNet152-BERT	Imagen-Texto	0.992	**0.913**	**0.902**	**0.9068**

4.6. Análisis de resultados

4.6.1. Experimento 1: Imagen

Este experimento consistió en crear un modelo de redes convolucionales para realizar la clasificación de memes. En la Figura 4.8 se observa la matriz de confusión con los resultados y las distintas métricas obtenidas.

Con lo obtenido se puede ver que la clase $No - Meme$ obtiene un buen porcentaje de acierto 85 %, por lo que el modelo se comporta bien frente a discriminar si una imagen es un $No - Meme$ y $No - Sticker$. El problema se encuentra en clasificar un meme y un sticker pues alcanza un nivel de precisión muy bajo. Un 59 % para los memes y un 73 % para los stickers.

La métrica que más nos interesa es la de $F1$ *Score* pues estamos trabajando sobre clases desbalanceadas. El puntaje obtenido fue de un 49,6 %.

4.6.2. Experimento 2: Imágenes con Fine Tuning

En este caso se utilizaron modelos de clasificación de imágenes más complejos, como ResNet. Por u lado ResNet50 con los resultados mostrados en la Figura 4.9 y ResNet152 en la Figura 4.10. Se puede observar que sigue los mismos patrones encontrados en el modelo de CNN.

Las redes logran interpretar bien cuando una imagen es un $No - Meme$ pero tienen problemas para clasificar una imagen entre $Meme$ y $Sticker$. Las métricas que obtuvieron disminuyeron en comparación con el experimento anterior. ResNet50 obtuvo un 56,4 % en F1 y ResNet152 un 57,3 % en la misma. Esto da a entender que complejizar un modelo convolucional con una mayor cantidad de capas no mejora la clasificación, al contrario, la empeora.

Análisis del experimento

Estos resultados se deben principalmente a que el modelo entiende que por lo general un meme o sticker sigue un esquema. Es una imagen con un texto encima que intenta expresar una idea. Esto lo toma el modelo y predice correctamente sobre si una imagen presenta el esquema. El problema lo encuentra cuando la imagen sigue esta forma, pero no entiende el la idea del texto por lo que falla en la clasificación la mayoría de las veces. Es por esto por lo que se necesita poder analizar el texto.

Como se observa en la Figura 4.20, la mayor cantidad de memes sigue un esquema parecido. Este presenta una imagen con un texto encima el cual le intenta expresar una idea. El texto se puede encontrar en cualquier posición de la imagen.

Figura 4.20: Esquema típico de los memes

Es importante mencionar que el tener texto en una imagen no implica que sea un $Meme$ o $Sticker$, pues podría tratarse de una imagen común con texto encima. Como por ejemplo en la Figura 4.21. Esta se trata de un anuncio de mercado libre la cual sigue el mismo esquema explicado antes, pero claramente no trata de una imagen $Meme$ o $Sticker$.

Figura 4.21: Imagen de venta

4.6.3. Experimento 3: Modelo mixto

Modelo CNN-Lineal

Este modelo compuesto presenta una gran mejora respecto a los anteriores, como se observa en la Figura 4.11 la cantidad de aciertos para la clase meme y sticker aumentaron en gran consideración, alcanzando un 84 % para los memes y un 89 % para los stickers. Esta red alcanzo un 90,07 % en la métrica de $F1Score$.

Modelo CNN-LSTM

Este modelo presenta unos resultados muy parecidos al modelo anterior. Como se observa en la Figura 4.12 los memes tiene un 84 % de acierto, un 88 % en los Stickers y casi un total acierto en la clase No Meme. En las métricas generales hubo pequeñas variaciones respecto al modelo anterior. Alcanzo un 90,03 % de $F1$ $Score$.

Modelo CNN-RNN

Este fue el modelo que peor se comportó en esta fase de experimentos. Como se observa en la Figura 4.13 se comporta de manera parecida a los modelos de solo imagen. La clase meme alcanzo un 51 % de acierto, los Sticker un 58 % y los No meme un 98 %, por lo que tiene problemas para discriminar entre imágenes que presentan este esquema de texto. La red obtuvo un 68,12 % de $F1$ $Score$.

Modelo CNN-Convolucional

Este modelo se comportó semejante a los dos primeros, con una precisión un poco más baja. En la Figura 4.14 se observa que en la clase Meme obtuvo un 84 % de precisión, un 81 % en los stickers y casi una totalidad en la clase $No - Meme$. En la métrica $F1$ *Score* logro un 87,29 %.

Modelo ResNet50-Lineal

En este modelo se introdujo una red pre entrenada, ResNet50 junto a una red lineal de clasificación de texto. Los resultados se pueden ver en la Figura 4.15. Para la clase $Meme$ se obtuvo 80 % de precisión, un 93 % para los *Stickers* y casi una totalidad en los $No - Meme$. El modelo obtuvo un 90,19 % en $F1$ *Score*.

Modelo ResNet50-LSTM

En la Figura 4.16 se pueden observar los resultados de este modelo. Obtuvo un 82 % de acierto en memes, un 87 % en stickers y casi una totalidad en No-Meme. En la metrica de $F1$ *Score* logro un 89,56 %.

Modelo ResNet152-BERT

Este fue el modelo que mejor métricas obtuvo. En la Figura 4.17 se puede observar que para la clase $Meme$ logro un 83 % de precisión, un 91 % en los *Stickers* y casi una totalidad en la clase $No-Meme$. Para el caso de la métrica $F1$ *Score* se obtuvo un 90,68 % convirtiéndolo en la red con mayor porcentaje obtenido en esta métrica.

Análisis del experimento

Estos resultados obtenidos permiten ver que el texto es una información primordial para los distintos modelos a la hora de realizar la clasificación, sobre todo a la hora de discriminar entre la clase $Meme$ y *Sticker*. Para el caso de los $No - Memes$ se comporta de forma similar a los experimentos anteriores (aumentando un 3 % de precisión aproximadamente). Lo anterior nos permite concluir que el modelo mixto que analiza tanto la imagen como el texto entiende el esquema de una imagen meme o sticker y entiende la idea que intenta transmitir, permitiendo diferenciarlos.

En la Figura 4.22, se presenta un $Meme$ y un *Sticker*. El texto de la primera expresa una idea de lo que se quiere decir, el hijo pregunta a su padre sobre que es la historia y este le responde no saber pues es de derecha. Esto es un mensaje de humor político sobre el sector de derecha. Por otro lado, la segunda imagen de *Sticker* entrega un mensaje de saludo. Este no expresa o transmite alguna idea por lo que a priori no entrega ningún tipo de información.

(a) Meme

(b) Sticker

Figura 4.22: Imagenes con un mismo esquema

Como se explicó ambas imágenes tienen el mismo esquema de imagen-texto, pero es la intención y la idea del mensaje quien permite clasificarlo entre la clase *Sticker* y *Meme*.

4.6.4. Experimento 4: Modelo probado con texto automático

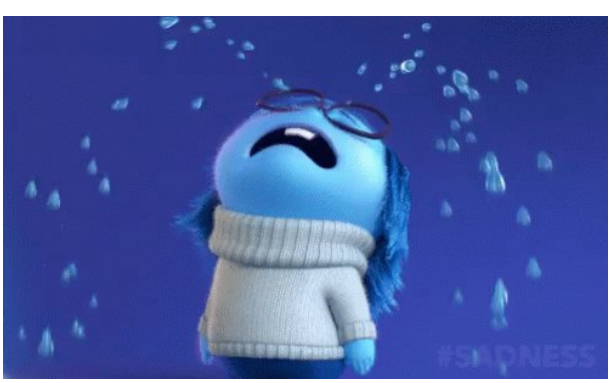
Los resultados de los modelos obtenidos son muy semejantes entre sí. Para el caso de ResNet152 con BERT se puede observar la Figura 4.18. Este logró un 51,1 % en la métrica $F1$ *Score*, viéndose disminuida casi un 40 % respecto a la clasificación con texto manual. Esto se debe a que la clase $No - Meme$ tuvo una gran disminución de su acierto (La clase *Sticker* y *Meme* siguen presentando valores semejantes a los experimentos pasados).

Para el caso de CNN BERT, los resultados se pueden ver en la Figura 4.19 obteniendo un 50,3 % en la métrica $F1$ *Score*.

En general el extractor presenta grandes fallas a la hora de obtener el texto. Puede distorsionar la palabra gracias a los colores o forma que presenta la letra dentro de la imagen. También puede obtener texto falso, es decir que no existe en la imagen o simplemente omitir texto.

Por ejemplo en la Figura 4.23 se presentan dos imágenes donde el extractor de texto hace que el modelo se confunda. En la primera se presenta una imagen $No - Meme$ la cual presenta una marca de agua en la parte inferior derecha con el texto *Sadness*, esto hace que la red la entienda como *Meme* pues le hace pensar que sigue el esquema de tal. Por otro lado, la segunda imagen es un *Meme* en donde el extractor no logra obtener el texto y le hace creer a la red que no presenta, haciendo que lo clasifique como $No - Meme$.

Por esto es muy importante tener un extractor de texto lo más eficiente posible, de manera que a la hora de utilizar el clasificador pueda funcionar de la mejor forma posible. Esto quiere decir que cualquier avance que se logre en esta área de OCR aportará de forma directa al clasificador.



(a) No Meme

(b) Meme

Figura 4.23: Imágenes con problema de extracción de texto

4.6.5. Análisis General

Los experimentos anteriores permiten observar que no basta solo analizar la imagen para lograr clasificar correctamente. El texto es una parte fundamental para que la red logre discriminar de mejor manera el esquema mencionado anteriormente. Esto dado que tanto la clase *Meme* como *Sticker* presentan un esquema parecido (imagen con texto sobrepuesto) por lo que es muy importante comprender la idea que se quiere expresar para mejorar la clasificación entre estas dos clases. Además, una imagen $No - Meme$ puede contener texto por lo que también aporta a la hora de clasificar en esta categoría. Esto se puede ver reflejado en los resultados obtenidos, principalmente en la métrica $F1$ *Score*, la cual aumento aproximadamente un 35 % con respecto a los modelos de CNN.

Un punto importante a la hora de trabajar con texto es poder obtenerlo de una manera sencilla. A la hora de entrenar los modelos se poseían los textos contenidos en la imagen lo que hacían más sencillo y eficaz el entrenamiento. Al clasificar nuevas imágenes, se requerirá extraer el texto para entregárselo al modelo por lo que es de vital importancia tener un método automático de extracción.

Todo lo anterior nos permite concluir que el mejor método hasta ahora probado es $ResNet152 - BERT$, pues alcanzó el puntaje más alto en las cuatro métricas calculadas.

4.6.6. Comparación de resultados

Como se comentó en las secciones anteriores, ya existían resultados en dos memorias anteriores, *Comparación de descriptores para clasificación de memes* [39] y *Combinador de clasificadores para identificar memes* [33]. Es por esto que es importante realizar comparaciones con las nuevas métricas obtenidas en este trabajo.

Metrica	Descriptores	Combinador	Deep Learning
Accuracy	$75\,\% \pm 4\,\%$	$76\,\% \pm 3\,\%$	$99.2\,\%$
Precisión	$75\,\% \pm 4\,\%$	$77\,\% \pm 3\,\%$	$91.3\,\%$
Recall	$75\,\% \pm 3\,\%$	$76\,\% \pm 2\,\%$	$90.1\,\%$
F1 Score	$74\,\% \pm 5\,\%$	$76\,\% \pm 3\,\%$	$90\,\%$

Tabla 4.1: Comparación de resultados.

En la Tabla 4.1 se pueden observar los resultados obtenidos en las tres memorias. El método de *Deep Learning* obtiene mejores metricas que las técnicas utilizadas anteriormente. Este aumenta un $15\,\%$ aproximadamente en $F1\ Score$, lo cual es una cifra considerable. Esto permite concluir que la utilización de redes neuronales es la mejor forma probada actualmente para resolver este tipo de problemas.

Capítulo 5

Detección de tópicos de memes

La detección de tópicos en memes consiste en intentar discriminar dentro de un grupo de memes, su respectiva temática. Existen muchas temáticas dentro de este tipo de imágenes y cada día se crean nuevas.

Dentro de esta sección se explicará todo el proceso realizado para la detección de tópicos y las distintas decisiones que fueron tomadas en el proceso de experimentación, como así también en la creación de los modelos.

Finalmente se mostrarán y explicarán los distintos resultados obtenidos, intentando darles alguna explicación como así también reflexionando que se podría mejorar para obtener mejores clasificaciones

5.1. Problema

En Internet existen una gran cantidad de memes sobre diferentes temas. Estas pueden ser sobre política, ciencia, cosas cotidianas, etc. También estas categorías van creciendo, dependiendo los sucesos que ocurran en un país. Por ejemplo, en Chile cuando ocurrió el estallido social del 18 de octubre, una gran cantidad de memes nuevos se vieron en redes sociales. Es por esto que su estudio se complejiza aún más, pues si se quiere estudiar un tema en específico como la política, se requiere filtrar todas las imágenes que se poseen y así analizar solo las deseadas.

El problema que se abordará será el de investigar sobre el tema de un clasificador en tópicos de memes. Este debe ser capaz de filtrar sobre todos los memes, cuales pertenecen a ciertas categorías. Para esto se aplicarán técnicas de deep learning sobre imágenes y texto.

5.2. Dataset a utilizar

Este dataset fue brindado por el IMFD y creado por un grupo de expertos en el tema, al igual que el anterior. El proceso consistió en analizar un aproximado de 800 imágenes y darles una categoría. En un inicio se utilizaron 22 categorizaciones, pero luego fueron reducidas a 17. Los datos fueron divididos en un 70 % de entrenamiento y un 30 % para pruebas.

Tabla con cantidad de memes por categoría

Categoria	Cantidad
Contenido Humano	139
Sebastian Piñera	120
Teconología y ciencia	5
Cultura	31
AFP	4
Deportes	29
Política	123
Economía	10
Misceláneos	18
Animales	5
Corrupción	5
Educación	11
Celebridades y entretención	11
Salud	5
Clima	38
Crimen	2
Estallido social	165

5.3. Modelos

Para abordar este problema se utilizaron los dos modelos mencionados en la Clasificación de memes solo que modificando la salida al número de clases correspondientes para este dataset. Además, se agregó un nuevo modelo enfocado solo a clasificar el texto.

Modelo de texto

Este modelo consiste en dejar la imagen del meme apartado y utilizar solo el texto que contiene. Este es procesado por el tokenizador correspondiente. Se utilizaron entrenamientos desde cero con modelos lineales y LSTM y además técnicas de fine tuning con modelos ya entrenados como BERT o RoBERTa.

Se optó por experimentar con este modelo pues se están trabajando con imágenes que poseen un texto. El contenido de este texto es fundamental para poder categorizarlo de manera correcta. Por ejemplo en la Figura 5.2 se presenta un meme sobre política. En este se observa una escena de Los Simpsons, la cual no nos da mucha información sobre que

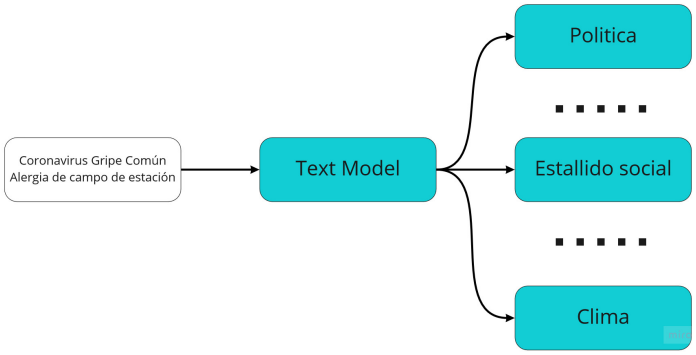


Figura 5.1: Modelo de texto

podría tratarse el meme. Por otro lado, el texto hace referencia a una ley, lo que indicaría fuertemente que trata sobre política.

Figura 5.2: Meme sobre política

5.4. Experimentos

5.4.1. Experimento 1: Imagen

Para estos experimentos se utilizó el modelo de CNN explicado en la clasificación de memes. Este fue adaptado para clasificar 17 categorías. La red convolucional consistía en dos capas convolucionales, se utilizó un kernel de tamaño 3 x 3. También existen capas de pooling que reducen el tamaño de la imagen a la mitad, la imagen tenia un tamaño de 53 en RGB. Por otro lado, se aplicó la técnica de transfer learning para los modelos de ResNet

5.4.2. Experimento 2: Modelo mixto

Para estos experimentos se aplicó el mismo modelo mixto utilizado en la clasificación de memes, adaptado a predecir sobre 17 categorías. Para el caso de las imágenes se utilizaron las redes convolucionales (CNN) y distintos tipos de ResNet aplicando la técnica de transfer learning. En el caso del texto se utilizaron modelos de clasificación lineal, LSTM, BERT y RoBERTa.

5.4.3. Experimento 3: Texto

Para estos experimentos se decidió clasificar las imágenes utilizando solo el texto, dejando de lado la imagen. Se utilizaron los modelos de BERT y RoBERTa y LSTM. Se decidió experimentar con este modelo por la naturaleza del problema. La mayor cantidad de imágenes memes contienen texto, por lo que aislarlo y analizarlo puede resultar útil en la investigación.

5.4.4. Experimento 4: Reduciendo categorías

Para este caso se redujeron el número de categorías, agrupando unas sobre otras. El resultado fueron dos nuevas categorizaciones, una con 6 clases y la otra con 7. Para esto se aplicaron los modelos de clasificación con solo texto.

Tabla de cantidad de memes con 7 categorías

Categoría	Cantidad
Human content	142
Politics	251
Other	62
Art and Culture	44
Sports	31
Weather	39
Political unrest	185

Tabla de cantidad de memes con 6 categorías

Categoría	Cantidad
Human content	142
Politics	436
Other	62
Art and Culture	44
Sports	31
Weather	39

5.4.5. Experimento 5: Modelo probado con texto automático

En este caso se probarán los distintos modelos entrenados en los experimentos anteriores pero con texto extraído de manera automática con las librerías de OCR y de esta manera probar la efectividad de la extracción del texto. Este experimento intenta acercar al escenario real, el uso que se le dará el modelo junto a la extracción de texto automática que requerirá el preprocesamiento de las imágenes.

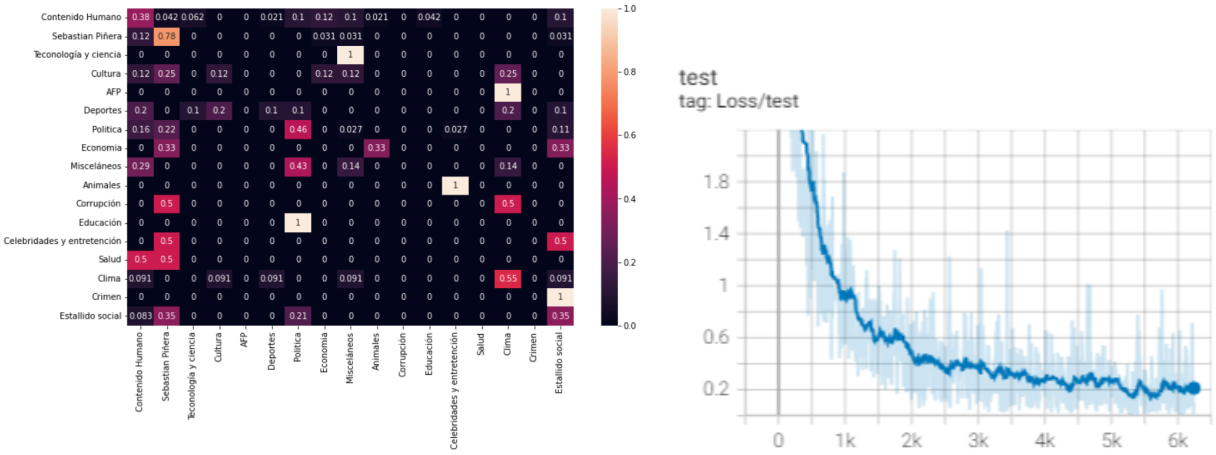
5.4.6. Experimento 6: Aplicando Data Augmentation

En este experimento se aplicarán técnicas de data augmentation a los mejores modelos obtenidos hasta el momento, es decir BERT para las 6 y 7 categorías. Se decidió realizar este proceso pues existen categorías que poseen una cantidad muy escasa de información, por lo que usar técnicas de data augmentation podría ayudar a aumentar las métricas del modelo.

5.5. Resultados experimentales

5.5.1. Experimento 1: Imagen

Modelo CNN



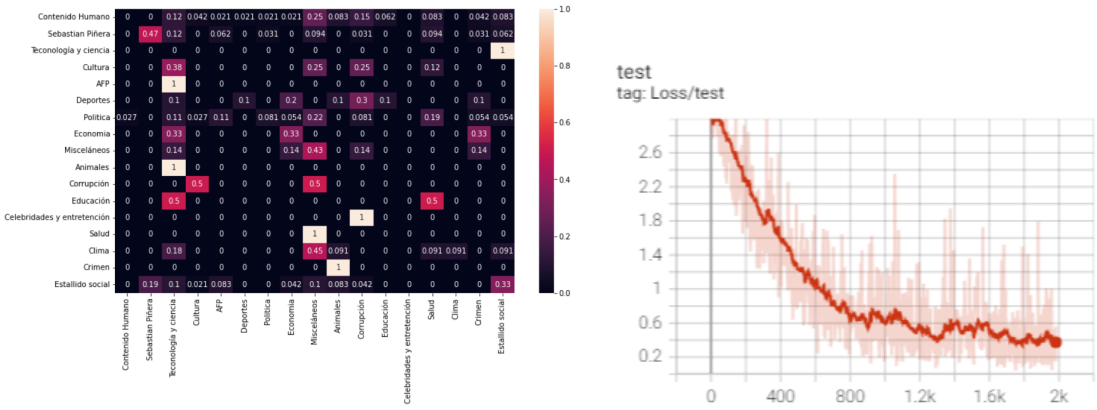
(a) Matriz de confusión (b) Gráfico de Loss

Figura 5.3: Modelo CNN

Las métricas obtenidas para este modelo fueron:

- **Accuracy**: 0.431
- **Recall**: 0.170
- **Precision**: 0.177
- **F1 Score**: 0.163

Modelo ResNet152



(a) Matriz de confusión (b) Gráfico de Loss

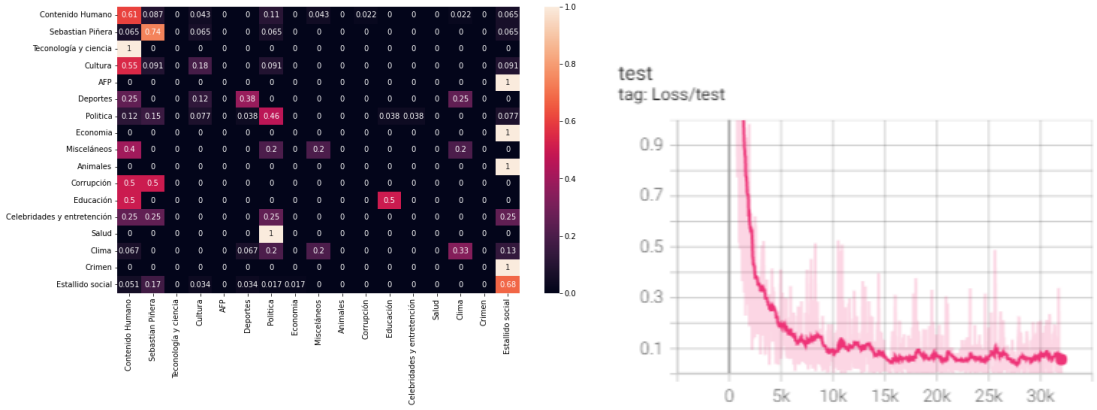
Figura 5.4: Modelo ResNet152

Las métricas obtenidas para este modelo fueron:

- **Accuracy**: 0.185
- **Recall**: 0.108
- **Precision**: 0.207
- **F1 Score**: 0.102

5.5.2. Experimento 2: Modelo mixto

Modelo CNN-Lineal



(a) Matriz de confusión (b) Gráfico de Loss

Figura 5.5: Modelo CNN-Lineal

Las métricas obtenidas para este modelo fueron:

- **Accuracy**: 0.528
- **Recall**: 0.240
- **Precision**: 0.239
- **F1 Score**: 0.236

Modelo CNN-LSTM

(a) Matriz de confusión

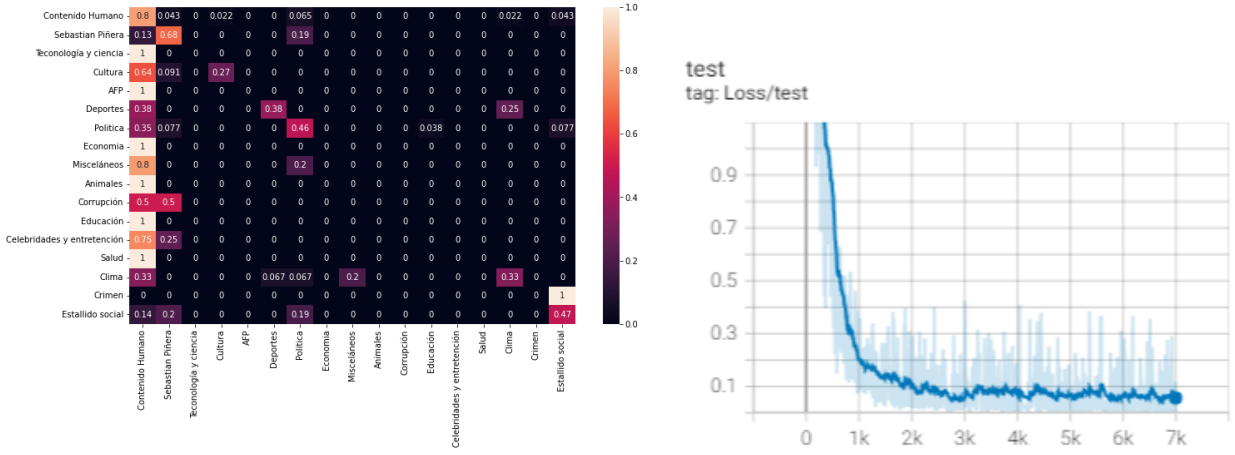
(b) Gráfico de Loss

Figura 5.6: Modelo CNN-LSTM

Las métricas obtenidas para este modelo fueron:

- **Accuracy**: 0.412
- **Recall**: 0.188
- **Precision**: 0.185
- **F1 Score**: 0.183

Modelo ResNet18-Lineal



(a) Matriz de confusión (b) Gráfico de Loss

Figura 5.7: Modelo ResNet18-Lineal

Las métricas obtenidas para este modelo fueron:

- **Accuracy**: 0.495
- **Recall**: 0.200
- **Precision**: 0.251
- **F1 Score**: 0.205

Modelo ResNet18-LSTM

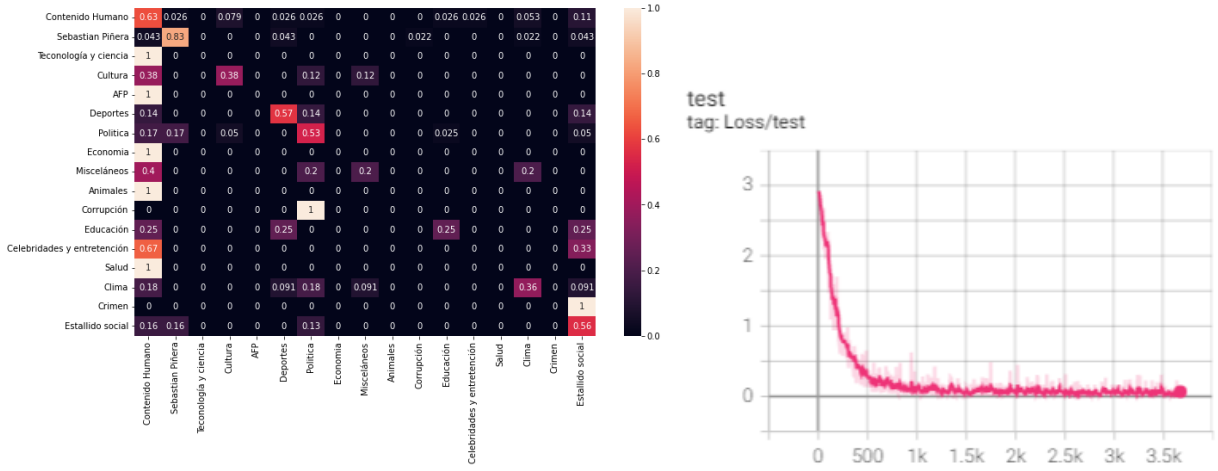
(a) Matriz de confusión (b) Gráfico de Loss

Figura 5.8: Modelo ResNet18-LSTM

Las métricas obtenidas para este modelo fueron:

- **Accuracy**: 0.500
- **Recall**: 0.196
- **Precision**: 0.222
- **F1 Score**: 0.197

Modelo ResNet152-Bert



(a) Matriz de confusión

(b) Gráfico de Loss

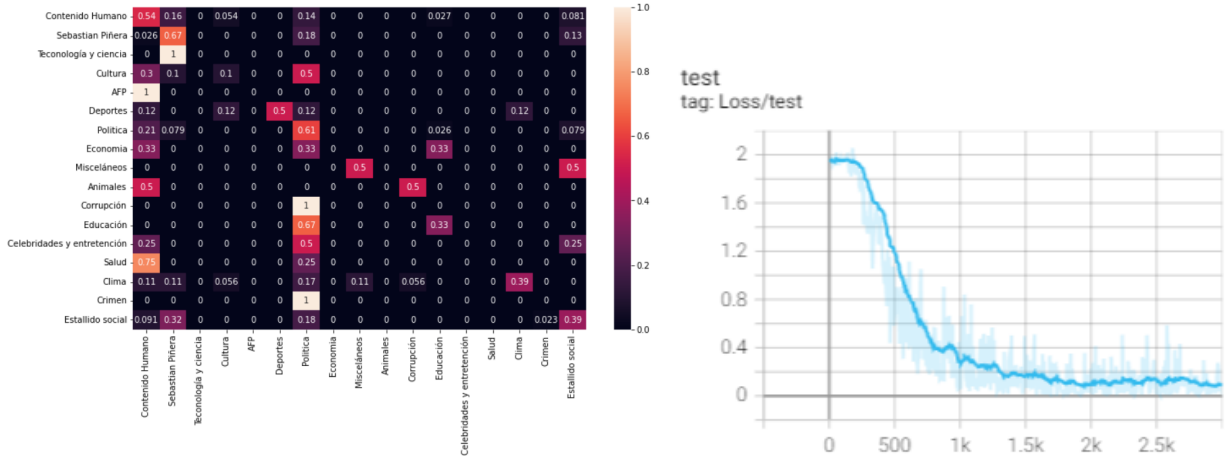
Figura 5.9: Modelo ResNet152-BERT

Las métricas obtenidas para este modelo fueron:

- **Accuracy**: 0.505
- **Recall**: 0.253
- **Precision**: 0.258
- **F1 Score**: 0.251

5.5.3. Experimento 3: Texto

Modelo LSTM



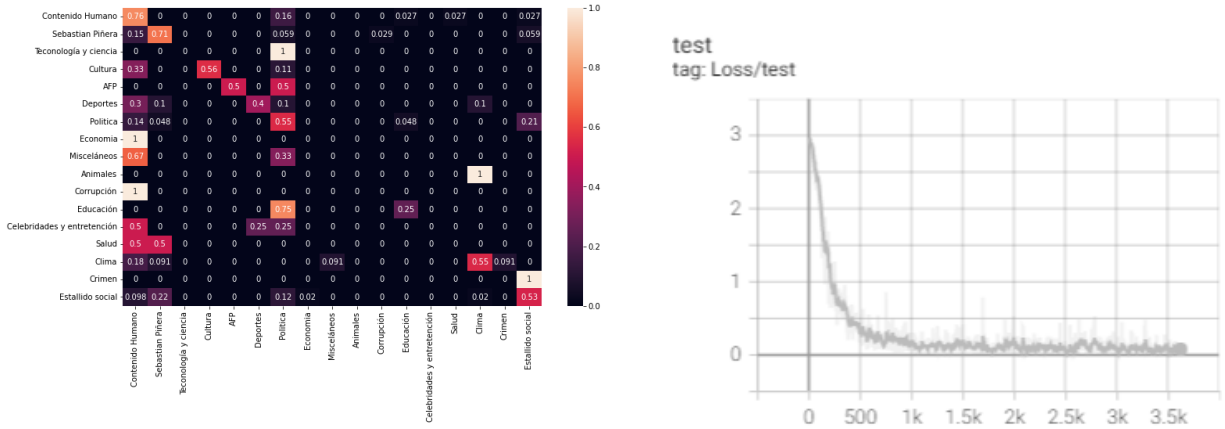
(a) Matriz de confusión (b) Gráfico de Loss

Figura 5.10: Modelo LSTM

Las métricas obtenidas para este modelo fueron:

- **Accuracy**: 0.463
- **Recall**: 0.240
- **Precision**: 0.286
- **F1 Score**: 0.243

Modelo BERT



(a) Matriz de confusión (b) Gráfico de Loss

Figura 5.11: Modelo BERT

Las métricas obtenidas para este modelo fueron:

- **Accuracy**: 0.560
- **Recall**: 0.282
- **Precision**: 0.350
- **F1 Score**: 0.300

Modelo RoBERTa

(a) Matriz de confusión (b) Gráfico de Loss

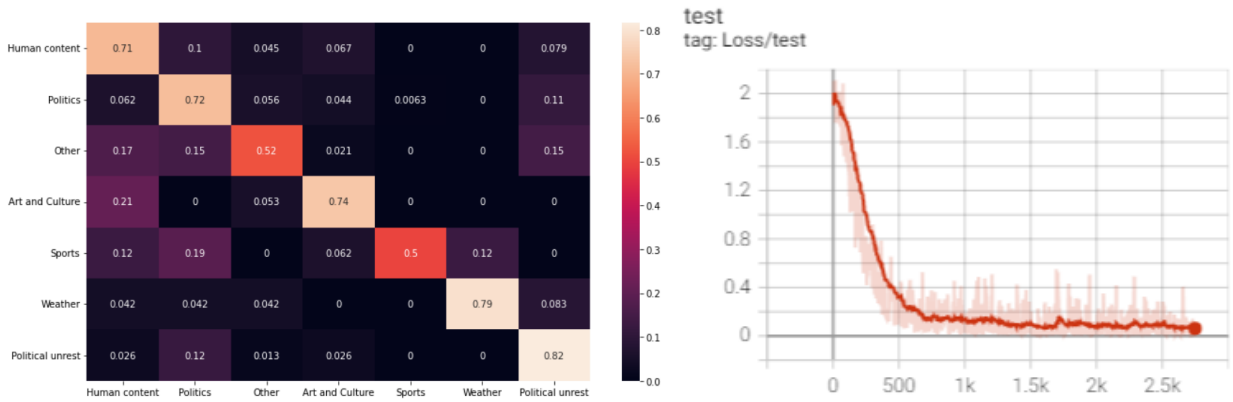
Figura 5.12: Modelo RoBERTa

Las métricas obtenidas para este modelo fueron:

- **Accuracy**: 0.528
- **Recall**: 0.333
- **Precision**: 0.411
- **F1 Score**: 0.354

5.5.4. Experimento 4: Reduciendo categorías

Modelo BERT 7 categorías



(a) Matriz de confusión

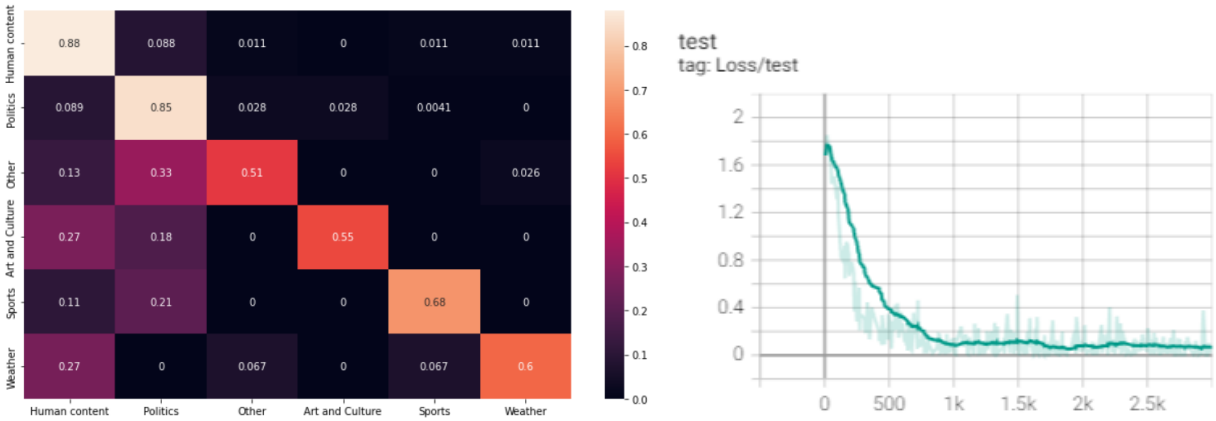
(b) Gráfico de Loss

Figura 5.13: Modelo BERT 7 categorías

Las métricas obtenidas para este modelo fueron:

- **Accuracy**: 0.674
- **Recall**: 0.685
- **Precision**: 0.714
- **F1 Score**: 0.684

Modelo BERT 6 categorías



(a) Matriz de confusión (b) Gráfico de Loss

Figura 5.14: Modelo BERT 6 categorías

Las métricas obtenidas para este modelo fueron:

- **Accuracy**: 0.755
- **Recall**: 0.679
- **Precision**: 0.750
- **F1 Score**: 0.706

5.5.5. Experimento 5: Modelo probado con texto automático

Modelo BERT 6 categorías

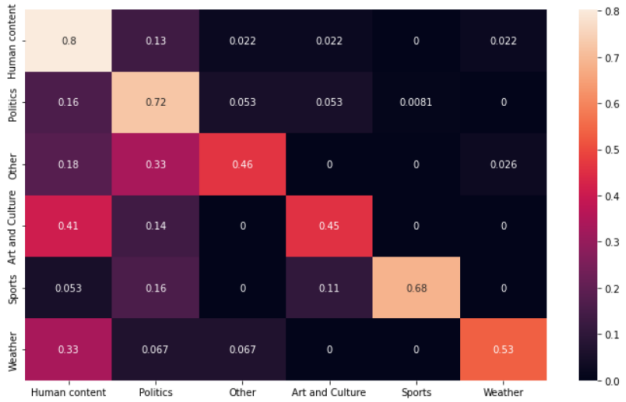


Figura 5.15: Matriz de confusión BERT 6 Categorías texto automático

Las métricas obtenidas para este modelo fueron:

- **Accuracy**: 0.694
- **Recall**: 0.610
- **Precision**: 0.647
- **F1 Score**: 0.618

Modelo BERT 7 categorías

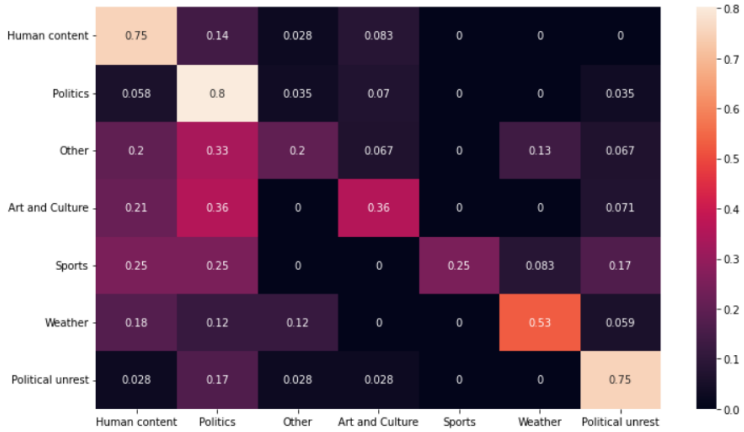


Figura 5.16: Matriz de confusión BERT 7 Categorias texto automático

Las métricas obtenidas para este modelo fueron:

- **Accuracy**: 0.630
- **Recall**: 0.520
- **Precision**: 0.637
- **F1 Score**: 0.541

5.5.6. Experimento 6: Aplicando Data Augmentation

Modelo BERT 7 categorías con data augmentation

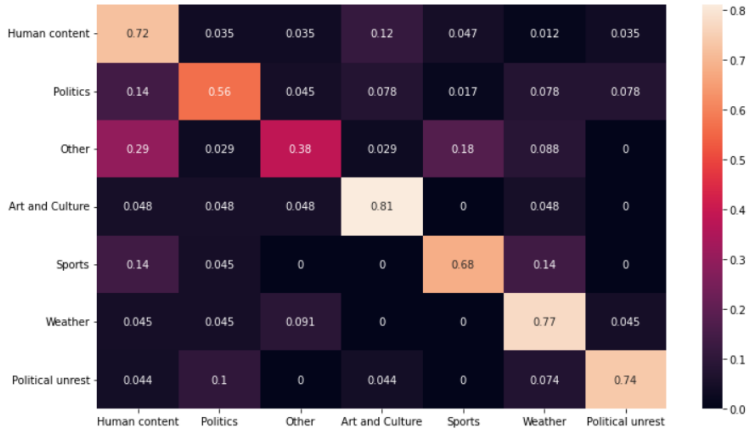


Figura 5.17: Matriz de confusión BERT 7 Categorías aplicando data augmentation

Las métricas obtenidas para este modelo fueron:

- **Accuracy**: 0.674
- **Recall**: 0.667
- **Precision**: 0.569
- **F1 Score**: 0.590

Modelo BERT 6 categorías con data augmentation

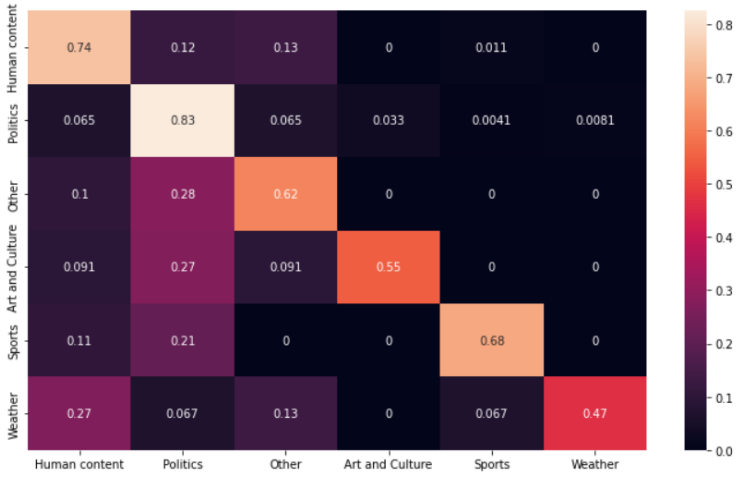


Figura 5.18: Matriz de confusión BERT 6 con data augmentation

Las métricas obtenidas para este modelo fueron:

- **Accuracy**: 0.694
- **Recall**: 0.646
- **Precision**: 0.697
- **F1 Score**: 0.661

5.5.7. Resultados Generales

Tabla resumen 17 categorías

Modelo	Dato analizado	Accuracy	Recall	Precision	F1 Score
CNN	Imagen	0.431	0.170	0.179	0.163
ResNet152	Imagen	0.185	0.108	0.207	0.102
CNN-Lineal	Imagen-Texto	0.185	0.108	0.207	0.102
CNN-LSTM	Imagen-Texto	0.412	0.188	0.185	0.183
ResNet18-Lineal	Imagen-Texto	0.495	0.200	0.251	0.200
ResNet18-LSTM	Imagen-Texto	0.500	0.196	0.222	0.197
ResNet152-BERT	Imagen-Texto	0.505	0.253	0.258	0.251
LSTM	Texto	0.463	0.240	0.286	0.243
BERT	Texto	**0.560**	0.282	0.350	0.300
RoBERTa	Texto	0.528	**0.333**	**0.411**	**0.354**

Tabla resumen 7 categorías

Modelo	Dato analizado	Accuracy	Recall	Precision	F1 Score
CNN	Imagen	0.505	0.348	0.365	0.350
ResNet152	Imagen	0.537	0.360	0.401	0.374
CNN-Lineal	Imagen-Texto	0.574	0.405	0.478	0.425
CNN-LSTM	Imagen-Texto	0.551	0.3903	0.484	0.412
ResNet18-Lineal	Imagen-Texto	0.523	0.416	0.460	0.432
ResNet18-LSTM	Imagen-Texto	0.519	0.329	0.382	0.345
ResNet152-BERT	Imagen-Texto	0.583	0.453	0.481	0.450
LSTM	Texto	0.551	0.448	0.472	0.431
BERT	Texto	**0.674**	0.685	**0.714**	0.684
RoBERTa	Texto	0.660	**0.702**	0.700	**0.689**

Tabla resumen 6 categorías

Modelo	Accuracy	Recall	Precision	F1 Score
CNN	0.597	0.399	0.413	0.403
ResNet152	0.616	0.3652	0.4002	0.377
CNN-Lineal	0.551	0.348	0.360	0.347
CNN-LSTM	0.546	0.355	0.375	0.354
ResNet18-Lineal	0.606	0.418	0.519	0.433
ResNet18-LSTM	0.634	0.393	0.515	0.424
ResNet152-BERT	0.634	0.382	0.541	0.415
LSTM	0.588	0.411	0.470	0.408
BERT	**0.755**	**0.679**	**0.750**	**0.706**
RoBERTa	0.743	0.673	0.644	0.652

5.6. Análisis de resultados

5.6.1. Experimento 1: Imagen

En este experimento, se aplicó una de las técnicas más básicas. Las redes neuronales se centraron en analizar solamente la imagen. Los resultados obtenidos no son buenos, pues las matrices no muestran ningún comportamiento esperado, lo que quiere decir que este no es un camino válido para resolver el problema.

Los resultados del modelo convolucional se pueden ver en la Figura 5.3. Esta no sigue ningún patrón destacable, solo se puede ver que la clase *Sebastian Piñera* obtuvo un 78 % de accuracy. Como se mencionó anteriormente la métrica importante es $F1$ *Score*. Esta red obtuvo un 16,3 %.

Para el caso de ResNet152 los resultados se pueden ver en la Figura 5.4. Al igual que la red anterior no existe ningún patrón. Para el caso de la clase *Sebastian Piñera* empeoro el accuracy con un 47 %. La métrica $F1$ *Score* obtenida fue 10,2 %.

Análisis del experimento

Como se observó, los resultados son mejores que el azar, pero no son lo suficientemente buenos como para concluir que es un buen modelo, por lo que se puede mencionar que no basta solamente con analizar la imagen para clasificar los memes por categorías. Esto se debe principalmente a que solo se está trabajando sobre imágenes meme, lo que quiere decir que todas siguen el mismo esquema explicado en el capítulo de clasificación de memes. Esto provoca que el modelo no pueda discriminar de manera correcta sobre el mismo esquema.

5.6.2. Experimento 2: Modelo mixto

Para este experimento se puede observar que los resultados ya logran tener cierto patrón que indicaría que ciertas clases están teniendo un porcentaje considerable de acierto. Existe una pequeña diagonal en la matriz.

Los resultados del modelo CNN-Lineal se pueden ver en la Figura 5.5. En esta se puede ver una ligera diagonal creada por ciertas clases. *Contenido Humano*, *Sebastian Piñera* y *Estallido Social* tienen sobre el 60 % de acierto, muy superior al experimento pasado. La métrica de $F1$ *Score* obtuvo un 23,6 %.

El modelo CNN-LSTM obtuvo resultados un poco inferiores al anterior. Estos se pueden ver en la Figura 4.12. La única clase que mantienen un buen porcentaje de aciertos es la de *Sebastian Piñera*. La métrica de $F1$ *Score* es 18,3 %.

Los modelos de ResNet18 con las redes de texto Lineal y LSTM les ocurre algo muy parecido. Clasifican la mayor parte de las clases como *Contenido Humano*. Para el caso de la Lineal se puede ver en la Figura 5.7 donde su $F1$ *Score* es 20,4 %. Por otro lado la LSTM se observa en la Figura 5.8 siendo su $F1$ *Score* de 19,6 %.

Para el caso del modelo de ResNet152-BERT la diagonal se puede ver un poco mejor. Esta se puede ver en la Figura 5.9. Algunas clases como *Deportes* y *Politica* vieron aumentado su porcentaje de aciertos. La métrica de $F1$ *Score* obtenida fue de un 25 %.

Análisis del experimento

Este experimento mejoro los resultados del anterior, aumentando las métricas, aunque no en gran medida como para concluir que es un buen modelo. Esto nos lleva a intuir que agregar una red que analice el texto aporta a la hora de realizar la clasificación.

Lo anterior podría indicar que la imagen por sí sola no está aportando información, pues todas siguen el esquema de imagen con texto. Como se muestra en la Figura 5.19 existen memes de distinta categoría que presentan la misma imagen de fondo pero distinto texto. En este caso la primera muestra un meme con temática de contenido humano mientras que la segunda de política, ambas utilizan la misma imagen de referencia. Esto podría indicar que una buena forma de analizar el problema es deshacerse de la imagen y utilizar solo el texto en cuestión.

5.6.3. Experimento 3: Texto

Este experimento mejora las métricas de los experimentos anteriores. Existen mayor cantidad de categorías sobre el 50 % de acierto. Por otro lado, se sigue manteniendo ese patrón de algunas categorías no presentan aciertos.

Para el caso del modelo LSTM, los resultados se pueden ver en la Figura 5.10. Se puede ver una ligera diagonal con aciertos. Por otro lado, la red clasifica varias imágenes de memes

(a) Vida Cotidiana (b) Estallido Social

Figura 5.19: Memes de distinto tópico con el mismo template

en *Politica* y *Contenido Humano*. La métrica de $F1$ *Score* obtenida fue de un 24,2 %.

Los resultados del modelo de BERT se pueden ver en la Figura 5.11. Existen categorías que tienen un porcentaje de aciertos considerable. Aún se mantiene, aunque en menor medida sobre algunas clases no presentan aciertos. *Contenido Humano* se lleva aciertos en otras categorías. La métrica de $F1$ *Score* es de un 30 %.

En el caso de RoBERTa se puede ver en la Figura 5.12. Este tiene una matriz muy semejante a la de BERT. En este es donde la diagonal se ve de mejor manera. Se mantiene que *Contenido Humano* se lleve aciertos de otras categorías. La métrica de $F1$ *Score* obtenida es de un 35,4 %.

Análisis del experimento

Este experimento deja ver que el texto es algo primordial a la hora de clasificar las imágenes memes por tópicos. Por otro lado, el sacar la imagen mejoro las métricas y cantidad de aciertos, por lo que se puede ver la imagen solo agrega ruido al resultado. Dado los resultados obtenidos se puede decir que la tarea de clasificar memes por tópicos de manera automática se reduce principalmente a categorizar texto.

Este modelo en general presenta problemas cuando el texto no le aporta información y es la imagen la que le brinda la categoría. Por ejemplo en la Figura 5.20 se puede ver un meme político en donde se requiere la imagen (cara del político) para poder categorizarlo. Además de eso se requiere un poco más de información externa a la imagen, como conocer que en el meme se muestra a una persona involucrada en el mundo de la política.

Figura 5.20: Meme de política

5.6.4. Experimento 4: Reduciendo categorías

Para este experimento se redujeron las categorías de diecisiete a siete y seis. Para la primera se utilizó el modelo de *BERT* y se pueden ver los resultados en la Figura 5.14. En este modelo si se puede apreciar una diagonal en la matriz de confusión. Las clases de *Human Content Politics* y *Policital Unrest* fueron las que mayor porcentaje de aciertos obtuvieron, aunque las demás están sobre un 50 %. Un punto para tomar en cuenta es que siguen existiendo imágenes que se clasifican de manera errónea en la clase *Politics* y *Human Content*. La métrica de *F*1 *Score* obtenido para esta red es de un 68,3 %.

Para el caso de la reducción a seis, los resultados se pueden ver en la Figura 5.13. Estos son similares a la anterior, solo que las categorías aumentaron un 5 % aproximadamente su nivel de aciertos. La categoría de *Political unrest* fue la eliminada. La métrica de *F*1 *Score* obtenida en este caso fue de un 70,5 %.

Análisis del experimento

Como se pudo ver en los resultados al agrupar las categorías el modelo se comporta mejor. Esto se debe a que, en los experimentos anteriores con 17 clasificaciones posibles, la red discriminaba varias como *Contenido Humano* o *Politica*. Ahora que esas categorías están unidas el modelo aumento el nivel de acierto en sus métricas. Aun así, siguen existiendo categorías que tienden a irse hacia *Humant Content* o *Politics* pero en menor medida. Están son *Others* y *Art and Culture*.

La categoría *Others* como se mencionó presenta un problema y es que acumula todas las imágenes memes que no caen dentro de una categoría. Es por esto por lo que el modelo presenta más dificultades para lograr clasificar correctamente dentro de esta clase. Como se observa en la Figura 5.21, la primera imagen presenta un meme que hace un juego de palabras mientras que el segundo presenta una situación en el ámbito de las mascotas. Esto es muy complicado para una red correlacionar las dos imágenes.

(a) Juego de palabras (b) Mascotas

Figura 5.21: Memes de la categoría otros

5.6.5. Experimento 5: Modelo probado con texto automático

En este se probaron los modelos entrenados en el experimento pasado, pero con el texto extraído con la librería *EasyOCR*. Existen clases que no presentan una gran variación como lo son *Human Content*, *Politics* y *Weather*, mientras que las otras presentan una disminución considerable, sobre todo en el modelo de siete categorías. La métrica $F1$ *Score* para la red de seis es de un 61,8 % lo que representa una disminución de un 8,7 %. Para el modelo de siete se obtuvo un 54,0 % en $F1$ *Score*, lo que es una disminución de un 14,3 %.

Análisis del experimento

Al igual que el problema anterior, es muy importante el proceso de extracción de texto para clasificar los memes. En este caso los modelos vieron disminuido aproximadamente un 10 % de sus aciertos, lo cual es una cifra significativa a la hora de realizar la categorización. En este contexto es aún más importante que el anterior, pues todas las imágenes poseen texto (caso que no ocurría en el problema anterior).

En la Figura 5.22 se puede observar una imagen meme en donde el extractor no logra reconstruir el texto de manera correcta. Esto dado que no respeta los espacios que existen en las palabras, lo cual es perjudicial para la red a la hora de clasificarlo. Por otro lado, existe una marca de agua que agrega un ruido al texto obtenido, dificultándole al modelo categorizar la imagen de manera correcta.

Figura 5.22: Meme con problemas de extracción de texto

Esto es un problema típico que se presenta a la hora de extraer texto. Esto se puede

deber a que las imágenes son muy variadas, con textos de distintos colores, tipos de letras y colores. Además, que los espaciados entre las letras son variados entre los distintos memes, lo que complejiza aún más la tarea.

5.6.6. Experimento 6: Aplicando Data Augmentation

Los resultados experimentales de los dos modelos son semejantes a los obtenidos sin aplicar la técnica. En el caso de BERT para 7 categorías, la red con data augmentation obtuvo un 58 % de $F1$ *Score* el cual es un 4 % mayor aproximadamente al resultado original. En el caso de BERT 6 la red obtuvo un 66 %, siendo un 5 % mayor al original.

En general, al observar las matrices de confusión se puede ver que cada una de las clases aumentan un poco la cantidad de aciertos que poseen. Esto es gracias a que al hacer el proceso de data augmentation al texto, el modelo logra tener un mejor contexto para cada una de las categorías.

5.6.7. Análisis General

Dado los resultados de los experimentos se puede observar que la imagen agrega ruido a los resultados de la clasificación por tópicos. Esto significa que se puede reducir la resolución de este problema a analizar el texto que contiene la imagen. El problema de esto es que hay ocasiones donde si o si se necesita de la imagen para poder categorizar las imágenes de memes como se mosto en la Figura 5.20.

Un punto muy importante a tomar en cuenta es que a veces no basta ni la imagen ni el texto para entender el meme. Esto dado que en ocasiones se necesita un contexto previo de conocimiento, haciendo que la imagen cause gracia solo a las personas que entienden sobre el tema. Por ejemplo en la Figura 5.23 se puede ver a una persona publica relacionada con la palabra temblor. El meme puede causar gracia si se sabe sobre que cada vez que ocurre un temblor en Chile, este personaje aparece en la televisión nacional explicando lo sucedido, por lo que ya se volvió costumbre.

Figura 5.23: Meme que requiere un contexto

Existen categorías donde el modelo comprende muy bien el contexto de la clase, como lo son los memes de *Politica* o *Contenido Humano* o *Sebastian Piñera*. Esto se debe a

que son las clases que más datos contiene, por lo que el modelo logra aprender lo suficiente para detectar correctamente estas categorías. Por otro lado, existen clases que no poseen la cantidad suficiente de información para que la red entienda sobre la categoría.

Para categorizar con 17 categorías, el mejor modelo obtenido fue el de RoBERTa, el cual obtuvo una métrica $F1$ *Score* de 35,4 %. Aun así, no es un modelo bueno para clasificar con esta cantidad de clases. Esto dado que en ciertas categorías no presenta una cantidad de aciertos razonable (en algunas tiene un 0 % de precisión).

La reducción de categorías permitió poder centralizar el modelo y así aumentar las métricas obtenidas. Existen temas importantes que resolver como por ejemplo el explicado con la categoría *Others*, además de la poca cantidad de datos que existen para el entrenamiento de los modelos. El mejor modelo para 7 categorías es RoBERTa, el cual obtuvo un 68,8 % de $F1$ *Score* mientras que, para 6 categorías, BERT se comportó mejor con un 70,5 % de $F1$ *Score*.

Capítulo 6

Conclusiones

Los memes son un tipo de dato muy variado, pues están en constante evolución y todos los días se crean nuevos. Esto hace que su análisis fuese más complejo que cualquier tipo de imagen por lo que se aplicaron técnicas que no habían sido probadas con anterioridad dentro del proyecto de investigación llevado a cabo en el IMFD. Una de estas se basó en mezclar un modelo de red neuronal con el análisis de la imagen y otro el texto y unirlos a la hora de realizar la clasificación.

A lo largo de los distintos experimentos se pudo observar que el texto es una fuente primordial a la hora de clasificar memes pues entrega un contexto importante el cual no se logra obtener solo analizando la imagen. Un punto importante de trabajar con él es que se requiere tener disponible tanto para el entrenamiento del modelo como a la hora de utilizarlos para clasificar nuevas imágenes. Esta es una tarea compleja de realizar, por lo que se tuvo que utilizar librerías externas.

El texto puede brindar un contexto, pero en ciertas imágenes no es suficiente para realizar la clasificación. En ocasiones los memes requieren un contexto externo a la imagen y es esta misma quien entrega el punto importante. Por esto fue muy importante probar distintos tipos de modelos para los problemas planteados.

Como se explicó en los resultados obtenidos de los distintos experimentos, cada imagen de meme es un problema distinto. Algunos requieren analizar solo la imagen y dejar el texto a un lado o viceversa o incluso se necesita incluir las dos fuentes de datos para clasificar de mejor manera. Es por esto por lo que sería muy beneficioso preprocesar antes la imagen para elegir el modelo adecuado dependiendo la necesidad de los datos, tarea la cual es muy complicada de realizar.

Una complejidad encontrada en este trabajo fue la de trabajar sobre un dataset desbalanceado. En el caso del primer problema existía una gran cantidad de datos para una clase, mientras que las otras dos presentaban una cantidad muy pequeña. Por otro lado, en el segundo problema existían cantidades muy distintas de imágenes para los distintos tópicos. Para abordar este problema se requirió un análisis en profundidad sobre las distintas técnicas de balanceo que existen. Finalmente se decidió aplicar la técnica por pesos, de esta manera se pudo aprovechar de mejor manera la poca información presente para algunas clases. Esto

es un punto para tomar en cuenta al futuro si se quiere mejorar el modelo, pues mientras más información pueda comprender la red neuronal, mejor funcionará su clasificación. Lo anterior sirve tanto los dos problemas.

En el segundo problema, se presentó la dificultad de tener categorías con una cantidad de datos muy escasa (alrededor de 3 o 5 imágenes). Claramente esto dificulta el proceso de entrenamiento para una red neuronal. Es por esto por lo que fue muy importante reducir en gran medida las categorías, agrupando unas sobre otras. El resultado obtenido fueron dos casos. El primero utilizar 7 categorías y el segundo 6. Con esto se obtuvo un modelo mucho más fiable que el anterior, aunque con ciertos problemas. El más notorio está presente en la categoría otros, dado que al ser una clase que agrupa las imágenes que no tienen cabida en las otras, no siguen una temática en común, por lo que el modelo presento dificultades para poder categorizarlas de manera correcta. Hay que tener presente que, aunque se hayan agrupado categorías, la cantidad de memes sigue siendo pequeña, por lo que, si se quiere seguir mejorando el modelo en un futuro, es necesario agrupar mucha más cantidad de información.

Otro problema importante encontrado en el camino fue la de extraer el texto de nuevas imágenes de memes. Esto dado que los distintos modelos requerían el texto presente en la imagen. En un comienzo el texto estaba en el dataset, pero si se quería dar una utilidad a las redes, se debía obtener el texto de las nuevas imágenes que llegaran. El mundo del OCR en imágenes es muy complejo y aun no existe una técnica clara para dar solución a este problema. En general existen modelos que logran extraer el texto de imágenes en donde éste está plasmado muy claramente, siguiente un orden, estilo, etc. En el caso de los memes es muy distinto pues existen distintas tipografías, colores, separaciones entre las letras e incluso hay memes con muy mala calidad en donde las distintas técnicas presentan grandes problemas de extracción.

Es importante mencionar que no se puede concluir a cabalidad que la imagen no aporta información para la detección de tópicos. Los mejores resultados en los modelos probados, indican que incluyendo solo el texto se pueden lograr mejores métricas, pero existen memes en los que es necesario analizar la imagen, como es el caso de los memes de política explicados en los resultados experimentales.

Luego de todo lo analizado con ablation study y obtenido con los distintos experimentos, se puede llegar a concluir que el mejor modelo obtenido hasta el momento para clasificar memes (primer problema) es una combinación de la red ResNet y BERT (con un 90,6 % $F1$ *Score*). Por otro lado, para el problema de la clasificación por tópicos, el modelo de BERT (con un 70,6 % $F1$ *Score* para 6 categorías y 67,4 % $F1$ *Score* para 7) es el que posee mejores métricas para la reducción de categorías.

6.1. Trabajo a Futuro

Dado que durante los experimentos se utilizaron modelos conocidos ya entrenados como ResNet, BERT y RoBERTa, un punto importante a tomar en cuenta seria seguir en esta búsqueda de modelos que puedan funcionar mejor para clasificar los memes. Esto dado que con el pasar del tiempo, se crear y mejorar ciertos tipos de redes neuronales, por lo que esto

podría llegar a ser beneficioso para seguir mejorando los resultados.

Por otro lado, trabajar con poca cantidad de información en redes neuronales es un problema muy importante para tener en cuenta. Es por esto por lo que para mejorar los resultados de esta investigación se requiere poder extraer mayor cantidad de información. Esto tanto para el clasificador de memes como para el de tópicos.

Un punto importante es mejorar el extractor de texto. Como fue mencionado en los análisis, la extracción no funciona de forma correcta en algunas situaciones. Es por esto por lo que buscar o experimentar una opción mejor es importante si se quiere mejorar el nivel de precisión del modelo.

Bibliografía

[1] EasyOCR, `https://github.com/jaidedai/easyocr`, Accessed: 2022-07-05.

[2] Krippendorff, k. (2011). computing krippendorff's alpha-reliability. retrieved from `https://repository.upenn.edu/asc_papers/43`.

[3] Pytorch, `https://pytorch.org/docs/stable/tensorboard.html`, Accessed: 2021-09-29.

[4] TensorBoard, `https://pytorch.org/docs/stable/tensorboard.html`, Accessed: 2021-09-29.

[5] Tesseract, `https://github.com/tesseract-ocr/tesseract`, Accessed: 2022-07-05.

[6] Saad Albawi, Tareq Abed Mohammed, and Saad Al-Zawi. Understanding of a convolutional neural network. In *2017 International Conference on Engineering and Technology (ICET)*, pages 1–6, 2017.

[7] Felipe Almeida and Geraldo Xexéo. Word embeddings: A survey. abs/1901.09069, 2019.

[8] Y. Baek, B. Lee, D. Han, S. Yun, and H. Lee. Character region awareness for text detection. In *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 9357–9366, Los Alamitos, CA, USA, jun 2019. IEEE Computer Society.

[9] Yingbin Bai, Erkun Yang, Bo Han, Yanhua Yang, Jiatong Li, Yinian Mao, Gang Niu, and Tongliang Liu. Understanding and improving early stopping for learning with noisy labels. In M. Ranzato, A. Beygelzimer, Y. Dauphin, P.S. Liang, and J. Wortman Vaughan, editors, *Advances in Neural Information Processing Systems*, volume 34, pages 24392–24403. Curran Associates, Inc., 2021.

[10] R Beale and T Jackson. Neural computing: An introduction, department of Computer Science, University of York. 1990.

[11] Hinrich Schütze Christopher D. Manning, Prabhakar Raghavan. An introduction to information retrieval. 2009.

[12] Mihai Gabriel Constantin, Dan-Stefan Pârvu, Cristian Stanciu, Denisa Ionascu, and Bogdan Ionescu. Hateful meme detection with multimodal deep neural networks. In *2021 International Symposium on Signals, Circuits and Systems (ISSCS)*, pages 1–4, 2021.

[13] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186, Minneapolis, Minnesota, June 2019. Association for Computational Linguistics.

[14] Sergey Edunov, Myle Ott, Michael Auli, and David Grangier. Understanding back-translation at scale. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 489–500, Brussels, Belgium, October-November 2018. Association for Computational Linguistics.

[15] Jianli Feng and Shengnan Lu. Performance analysis of various activation functions in artificial neural networks. *Journal of Physics: Conference Series*, 1237:022030, 06 2019.

[16] Roushan Kumar Giri, Subhash Chandra Gupta, and Umesh Kumar Gupta. An approach to detect offence in memes using natural language processing(nlp) and deep learning. In *2021 International Conference on Computer Communication and Informatics (ICCCI)*, pages 1–5, 2021.

[17] Giorgio Gnecco and Marcello Sanguineti. The weight-decay technique in learning from data: An optimization point of view. *Computational Management Science*, 6:53–79, 02 2009.

[18] Alex Graves, Santiago Fernández, Faustino Gomez, and Jürgen Schmidhuber. Connectionist temporal classification: Labelling unsegmented sequence data with recurrent neural networks. In *Proceedings of the 23rd International Conference on Machine Learning*, ICML '06, page 369–376, New York, NY, USA, 2006. Association for Computing Machinery.

[19] Jiuxiang Gu, Zhenhua Wang, Jason Kuen, Lianyang Ma, Amir Shahroudy, Bing Shuai, Ting Liu, Xingxing Wang, Gang Wang, Jianfei Cai, and Tsuhan Chen. Recent advances in convolutional neural networks. *Pattern Recognition*, 77:354–377, 2018.

[20] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, 2016.

[21] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9:1735–80, 12 1997.

[22] Gao Huang, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q. Weinberger. Densely connected convolutional networks. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2261–2269, 2017.

[23] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *Proceedings of the 32nd International Conference on International Conference on Machine Learning - Volume 37*, ICML'15, page 448–456. JMLR.org, 2015.

[24] Nusrat Ismoilov and Sung-Bong Jang. A comparison of regularization techniques in deep neural networks. *Symmetry*, 10:648, 11 2018.

[25] Ford S. Jenkins, H. and J Green. Cultura transmedia: la creación de contenido y valor en una cultura en red. Editorial Gedisa. 2013.

[26] Henry Jenkins. Convergence culture: la cultura de la convergencia de los medios de comunicación. Barcelona: Paidós. 2008.

[27] Peng Jin, Yue Zhang, Xingyuan Chen, and Yunqing Xia. Bag-of-embeddings for text classification. In *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence*, IJCAI'16, page 2824–2830. AAAI Press, 2016.

[28] Yoon Kim. Convolutional neural networks for sentence classification. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1746–1751, Doha, Qatar, October 2014. Association for Computational Linguistics.

[29] M krishna, M Neelima, Harshali Mane, and Venu Matcha. Image classification using deep learning. *International Journal of Engineering Technology*, 7:614, 03 2018.

[30] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C.J. Burges, L. Bottou, and K.Q. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 25. Curran Associates, Inc., 2012.

[31] Shuying Liu and Weihong Deng. Very deep convolutional neural network based image classification using small training sample size. In *2015 3rd IAPR Asian Conference on Pattern Recognition (ACPR)*, pages 730–734, 2015.

[32] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. Roberta: A robustly optimized bert pretraining approach, 2020.

[33] N. Machuca Guerra. Combinador de clasificadores para identificar memes. Memoria Universidad de Chile. 2021.

[34] Jamshed Memon, Maira Sami, Rizwan Ahmed Khan, and Mueen Uddin. Handwritten optical character recognition (ocr): A comprehensive systematic literature review (slr). *IEEE Access*, 8:142642–142668, 2020.

[35] Agnieszka Mikołajczyk and Michał Grochowski. Data augmentation for improving deep learning in image classification problem. In *2018 International Interdisciplinary PhD Workshop (IIPhDW)*, pages 117–122, 2018.

[36] G Pérez. Competencias digitales y memes en internet: Reflexiones en torno a la práctica docente. in book: Viralizar la educación: Red de experiencias didácticas en torno al meme de internet, eds. Nuria Rey and Melba cristina Marmolejo. 1st Edition. Pontificia Universidad Católica de Ecuador. 2013.

[37] Bustos B. Saldaña M Pérez-Martín, J. Semantic search of memes on twitter. paper presented to the ica annual conference; computational methods division. *ICA*, 2020.

[38] David E. Rumelhart and James L. McClelland. *Learning Internal Representations by Error Propagation*, pages 318–362. 1987.

[39] Alberto Sara. Comparación de descriptores para clasificación de memes. Memoria Universidad de Chile, 2020.

[40] S. Sriram, R. Vinayakumar, V. Sowmya, Moez Krichen, Dhouha Ben Noureddine, A. Shashank, and K.P. Soman. Deep Convolutional Neural Networks for Image Spam Classification. working paper or preprint, March 2020.

[41] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(56):1929–1958, 2014.

[42] Shardul Suryawanshi, Bharathi Raja Chakravarthi, Mihael Arcan, and Paul Buitelaar. Multimodal meme dataset (MultiOFF) for identifying offensive content in image and text. In *Proceedings of the Second Workshop on Trolling, Aggression and Cyberbullying*, pages 32–41, Marseille, France, May 2020. European Language Resources Association (ELRA).

[43] Ilya Sutskever, James Martens, George Dahl, and Geoffrey Hinton. On the importance of initialization and momentum in deep learning. In Sanjoy Dasgupta and David McAllester, editors, *Proceedings of the 30th International Conference on Machine Learning*, volume 28 of *Proceedings of Machine Learning Research*, pages 1139–1147, Atlanta, Georgia, USA, 17–19 Jun 2013. PMLR.

[44] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, L 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.

[45] Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel Bowman. GLUE: A multi-task benchmark and analysis platform for natural language understanding. In *Proceedings of the 2018 EMNLP Workshop BlackboxNLP: Analyzing and Interpreting Neural Networks for NLP*, pages 353–355, Brussels, Belgium, November 2018. Association for Computational Linguistics.

[46] Weronika Węgier and Paweł Ksieniewicz. Application of imbalanced data classification quality metrics as weighting methods of the ensemble data stream classification algorithms. *Entropy*, 22:849, 07 2020.

[47] Xiangnan Yin, Weihai Chen, Xingming Wu, and Haosong Yue. Fine-tuning and visualization of convolutional neural networks. In *2017 12th IEEE Conference on Industrial Electronics and Applications (ICIEA)*, pages 1310–1315, 2017.

[48] Raniah Zaheer and Humera Shaziya. A study of the optimization algorithms in deep learning. In *2019 Third International Conference on Inventive Systems and Control (ICISC)*, pages 536–539, 2019.

[49] Fuzhen Zhuang, Zhiyuan Qi, Keyu Duan, Dongbo Xi, Yongchun Zhu, Hengshu Zhu, Hui Xiong, and Qing He. A comprehensive survey on transfer learning. *Proceedings of the Institute of Radio Engineers*, 109(1):43–76, January 2021.

[50] Rasim Caner Çalik and M. Fatih Demirci. Cifar-10 image classification with convolutional neural networks for embedded systems. In *2018 IEEE/ACS 15th International Conference on Computer Systems and Applications (AICCSA)*, pages 1–2, 2018.

ANEXO

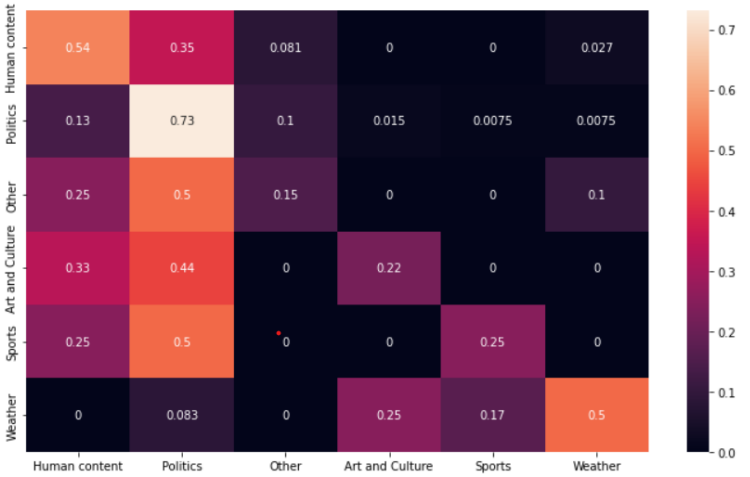


Figura 1: Matriz de confusión CNN 6 Categorías

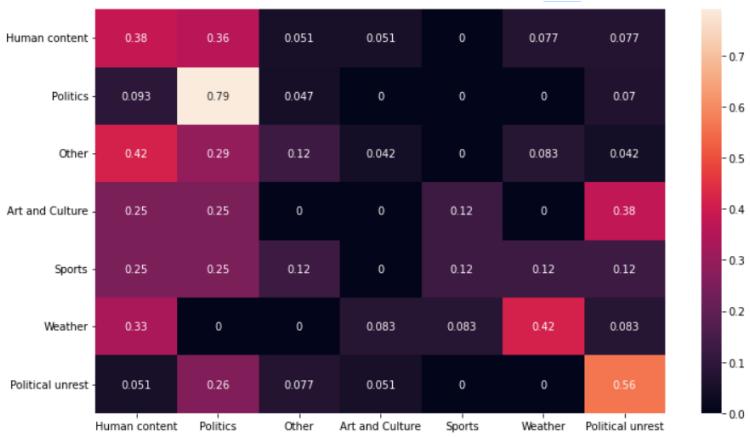


Figura 2: Matriz de confusión CNN 7 Categorías

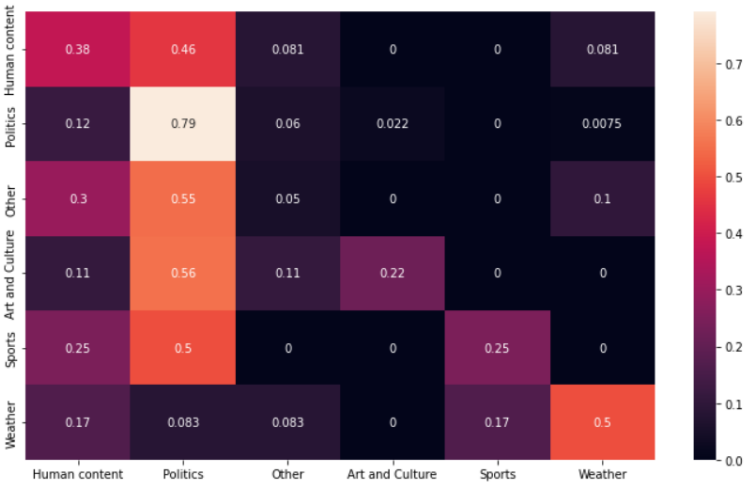


Figura A.3: Matriz de confusión ResNet50 6 Categorías

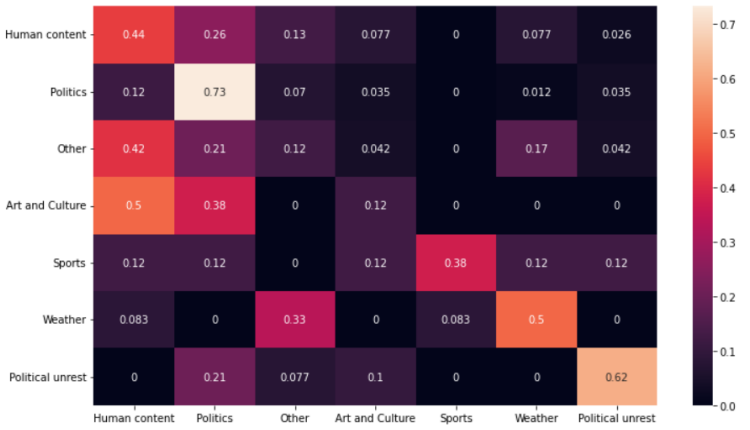


Figura A.4: Matriz de confusión ResNet50 7 Categorías

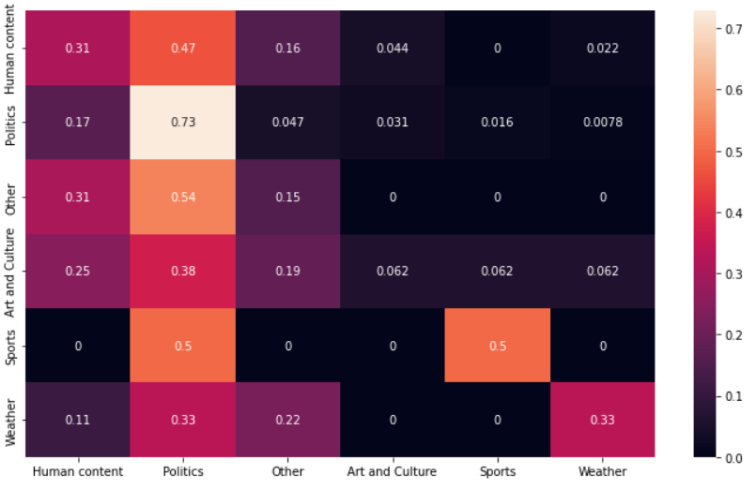


Figura A.5: Matriz de confusión CNN-Lineal 6 Categorías

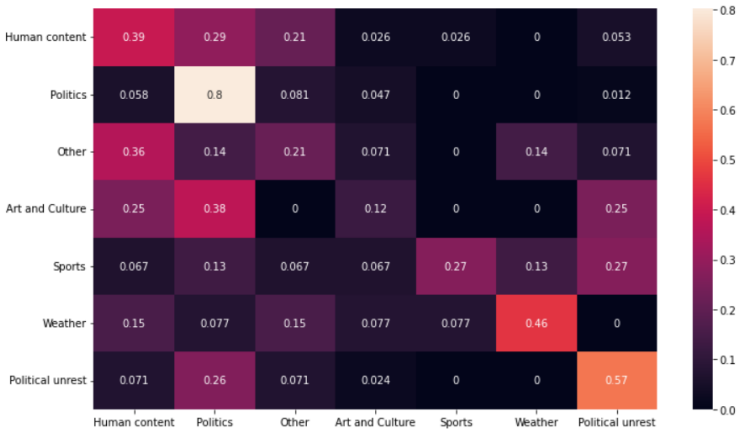


Figura A.6: Matriz de confusión CNN-Lineal 7 Categorías

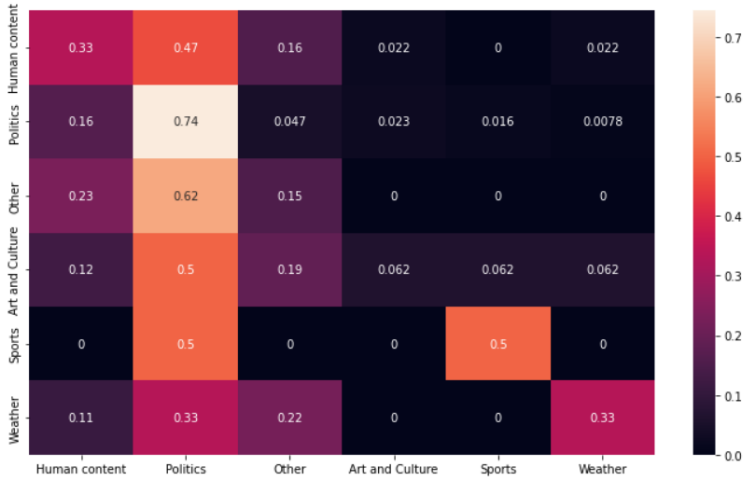


Figura A.7: Matriz de confusión CNN-LSTM 6 Categorías

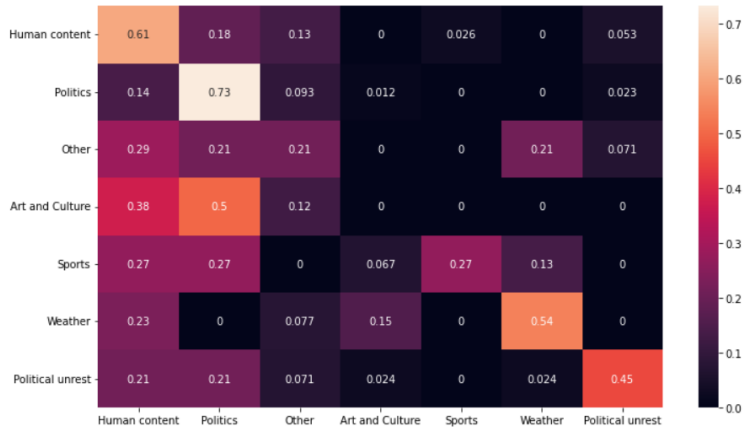


Figura A.8: Matriz de confusión CNN-LSTM 7 Categorías

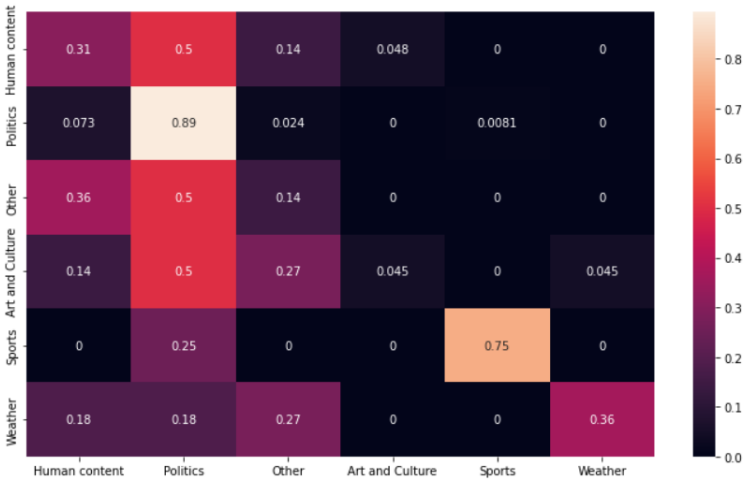


Figura A.9: Matriz de confusión ResNet50-Lineal 6 Categorías

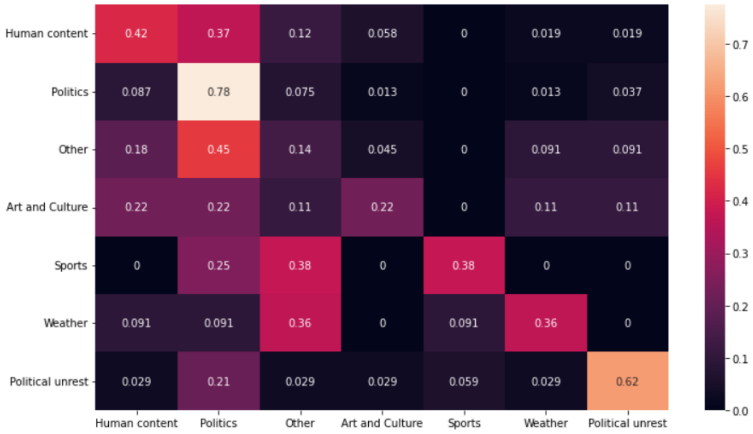


Figura A.10: Matriz de confusión RESNET50-Lineal 7 Categorías

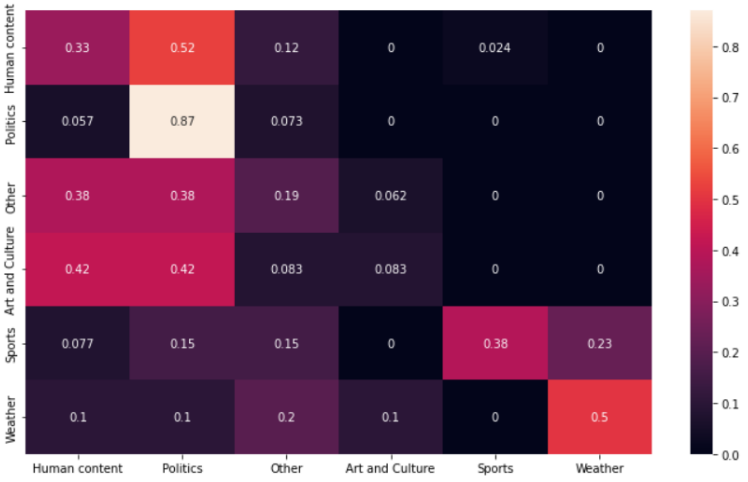


Figura A.11: Matriz de confusión ResNet50-LSTM 6 Categorías

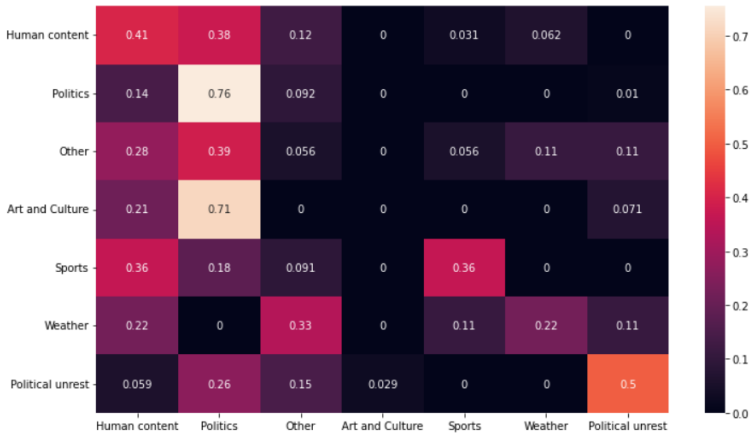


Figura A.12: Matriz de confusión ResNet50-LSTM 7 Categorías

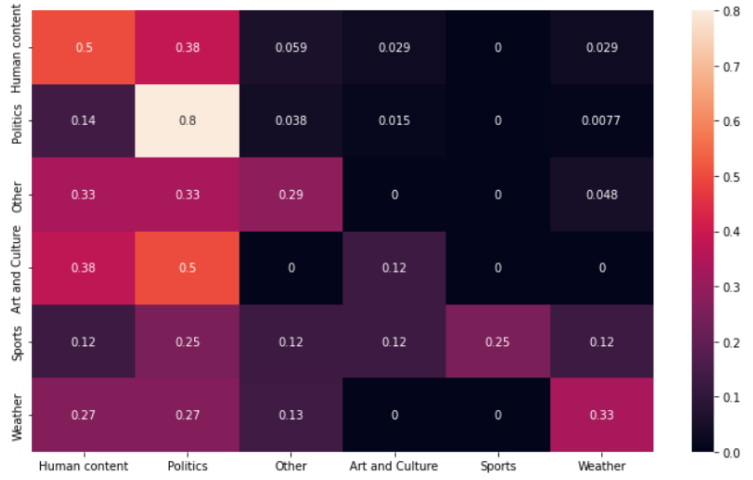


Figura A.13: Matriz de confusión ResNet152-BERT 6 Categorías

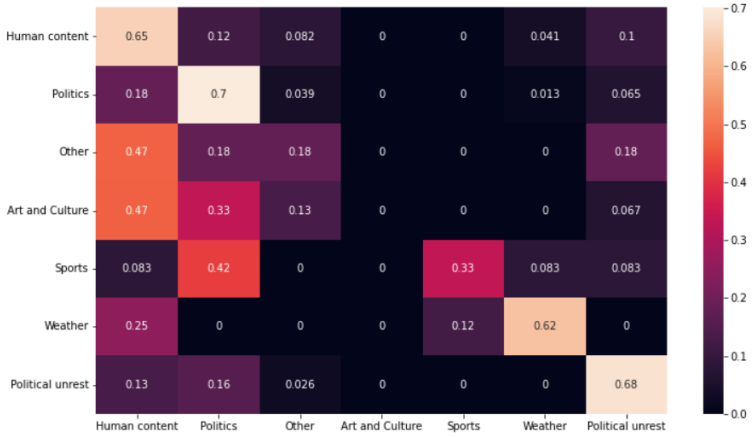


Figura A.14: Matriz de confusión ResNet152-BERT 7 Categorías