Descriptor de imágenes en español utilizando técnicas de Deep Learning

Lina María Sierra Vidal

Asesora: Haydemar María Núñez Castro

Departamento de Ingeniería de Sistemas y Computación

Facultad de Ingeniería

Universidad de los Andes

Julio, 2023

Resumen

Este proyecto de grado se centra en el desarrollo de un modelo basado en técnicas de Deep Learning para generar automáticamente descripciones de imágenes en español. La arquitectura Encoder-Decoder se explora para capturar relaciones contextuales entre el texto y las características visuales de las imágenes. El modelo se entrena utilizando un amplio conjunto de datos de imágenes con sus descripciones en español. Se realizaron diversas comparaciones entre arquitecturas, modelos e hiperparámetros para obtener resultados coherentes y precisos. La evaluación del modelo se enfocó en métricas cuantitativas como la exactitud y la pérdida, así como en métricas específicas para tareas de generación de lenguaje natural, como BLEU y METEOR. Estas métricas permitieron medir la calidad y similitud de las descripciones generadas con respecto a las descripciones de referencia en el dataset.

Este trabajo representa una contribución en el campo de la generación automática de descripciones de imágenes en español, con potenciales aplicaciones en diversos ámbitos, incluyendo la accesibilidad tecnológica para personas con discapacidad visual o neurológica. Una vez ajustado y validado, el modelo se implementó en una aplicación accesible que permite a los usuarios cargar imágenes y recibir descripciones de manera rápida en español. El enfoque de accesibilidad fue esencial durante todo el proceso de desarrollo, con el objetivo de brindar una solución tecnológica inclusiva para personas con diferentes niveles de discapacidad visual o neurológica.

Abstract

This project focuses on the development of a model based on Deep Learning techniques to automatically generate image descriptions in Spanish. The Encoder-Decoder architecture is explored to capture contextual relationships between text and visual features of images. The model is trained using a large dataset of images with their descriptions in Spanish. Various comparisons were made between architectures, models and hyperparameters to obtain consistent and accurate results. Model evaluation focused on quantitative metrics such as accuracy and loss, as well as specific metrics for natural language generation tasks, such as BLEU and METEOR. These metrics allowed us to measure the quality and similarity of the generated descriptions with respect to the reference descriptions in the dataset.

This work represents a contribution in the field of automatic generation of image descriptions in Spanish, with potential applications in various fields, including technological accessibility for people with visual or neurological disabilities. Once adjusted and validated, the model was implemented in an accessible application that allows users to quickly upload images and receive descriptions in Spanish. The accessibility approach was essential throughout the development process, with the aim of providing an inclusive technological solution for people with different levels of visual or neurological impairment.

Contenido

1. Introducción 5
1.1. Objetivos 5
1.2. Antecedentes 6
2. Diseño y especificaciones 7
2.1. Definición del problema 7
2.2. Especificaciones 7
2.3. Restricciones 8
3. Desarrollo del diseño 8
3.1. Diseño elegido 8
3.2. Alternativas de diseño 11
4. **Implementación 11**
4.1. Entendimiento del negocio 12
4.2. Entendimiento de los datos 13
4.3. Preprocesamiento de los datos 15
4.3.1. Carga de datos 15
4.3.2. Vectorización y limpieza de datos 15
4.3.3. Ajuste de imágenes 17
4.4. Construcción del modelo 17
4.5. Desplegar el modelo 23
5. Validación 24
5.1. Métodos 24
5.2. Validación resultados 25
6. Conclusiones 26
6.1. Discusión 26
6.2. Trabajo futuro 26
7. Referencias 28

1. Introducción

En el contexto actual, los sistemas de titulación de imágenes que generan descripciones automáticas se han vuelto ampliamente utilizados, sin embargo, una barrera significativa en este ámbito es la falta de modelos pre-entrenados en el idioma español, lo que limita el acceso a esta tecnología para personas con discapacidad en países de habla hispana.

El acceso a la tecnología es una necesidad imperante, especialmente considerando que en la población mundial existen aproximadamente 253 millones de personas con discapacidad visual, y en América Latina se estima que hay alrededor de 15 millones de personas que presentan esta condición (Dabian & Peña Moyano, 2020). Además, discapacidades neurológicas, como la agnosia visual, afectan a un porcentaje significativo de la población global (3%), lo que complica aún más el reconocimiento de imágenes (Dabian & Peña Moyano, 2020).

El objetivo de este trabajo es abordar esta problemática mediante el desarrollo de un modelo basado en técnicas de Deep Learning que sea capaz de generar textos descriptivos para imágenes en el idioma español. Incluso herramientas como los lectores de pantalla, una herramienta fundamental para el acceso a dispositivos electrónicos para personas discapacitadas, se ven obstaculizados al encontrar imágenes sin descripciones alternativas. La solución propuesta se basa en los textos alternativos, que describen los detalles básicos de una imagen y permiten a las personas con discapacidad visual comprender su contenido. Esta solución no se limita a ser usada para navegar en internet, como base, su potencial puede llegar hasta identificar objetos o contextos en la vida real, como buscador de imágenes, etiquetar y organizar imágenes, entre otros (Nain, 2021).

1.1. Objetivos

Objetivo general: Desarrollar un prototipo que permita generar texto descriptivo en español a partir de una imagen y que apoye en la construcción de soluciones que requieran este tipo de modelos.

Objetivo específico: Construir un modelo con técnicas de Deep learning para el procesamiento de imágenes basado en encoder-decoder que genere texto alternativo en español.

1.2. Antecedentes

La primera vez que se introdujo un modelo sequence to sequence fue en 2014 por Google, se usó para traducciones entre idiomas, “nuestro método utiliza una memoria a corto plazo (LSTM) multicapa para mapear la secuencia de entrada a un vector de una dimensionalidad fija, y luego otra LSTM profunda para decodificar la secuencia objetivo del vector.” (Sutskever, Vinyals, & V. Le, 2014)

Para enfrentar el problema actual de “traducir” entre imágenes y texto se han usado métodos similares a los que se usan para traducir entre idiomas. Lo que se hace en estos casos es convertir la imagen en una representación de esta que toma sus características relevantes y se usa como entrada, después es convertida en una secuencia de palabras que describen la imagen y esa es su salida (Ghandi, Pourreza, & Mahyar, 2023). Esta arquitectura se conoce como encoder-decoder.

Estos métodos han ido evolucionando y perfeccionándose y los métodos más usados actualmente se pueden resumir en la siguiente figura. (Ver fig. 1)

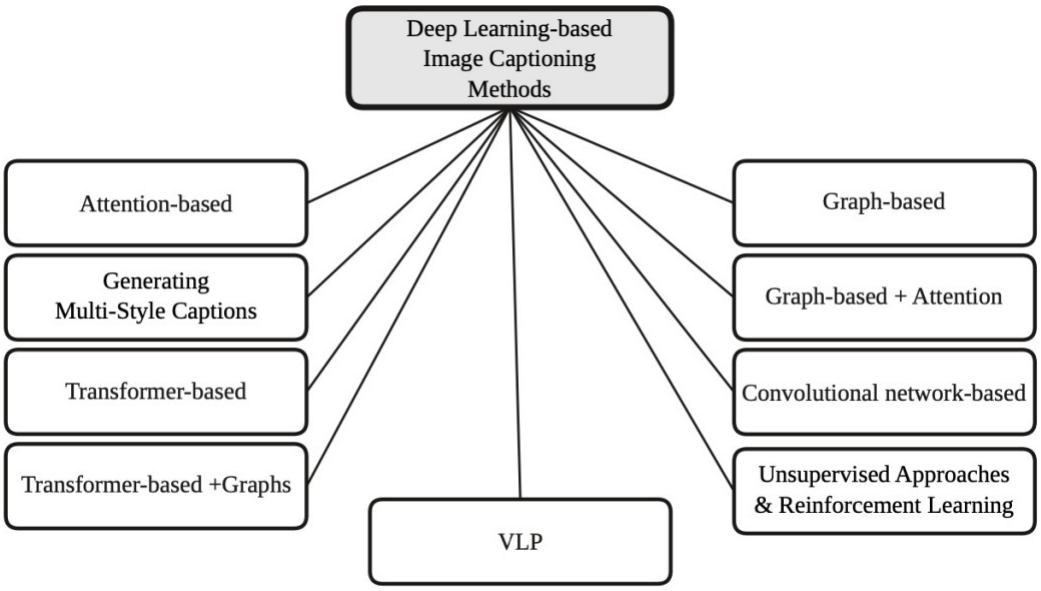

Figura 1: Métodos basados en Deep Learning para modelos de Image Captioning (Ghandi, Pourreza, & Mahyar, 2023)

Aunque se han probado todos los métodos mostrados y se han hecho experimentos, la mayoría de estos se hacen utilizando datasets en Ingles. La solución más común para esto ha sido traducir los resultados en ingles al español. Sin embargo, entrenar el modelo en español tiene ciertas ventajas como que se pueden capturar mejor las peculiaridades y matices específicos del lenguaje, ya que se entrena directamente en este idioma y, por ende, puede obtener mejores resultados.

2. Diseño y especificaciones

2.1. Definición del problema

La descripción automática de imágenes es una tarea compleja que requiere abordar varios desafíos interrelacionados. En primer lugar, el modelo debe ser capaz de comprender el contenido visual de una imagen, identificando objetos, personas, acciones y cualquier otro elemento relevante presente en la escena. Además, el modelo debe comprender el contexto y el significado de la imagen para generar una descripción coherente y relevante en español.

La tarea es un área de investigación activa, y el desarrollo de un modelo exitoso en español requerirá una amplia cantidad de datos anotados en este idioma para entrenar al modelo de manera efectiva. La disponibilidad de datos de entrenamiento en español es un factor crucial para el éxito del modelo, ya que cuantos más datos se tengan, más preciso y robusto será el modelo resultante.

Otro desafío es la complejidad de la representación semántica y lingüística del lenguaje español. Los idiomas tienen características únicas, y en español, hay aspectos como la flexión gramatical, las construcciones sintácticas y la diversidad de expresiones idiomáticas que deben ser capturados correctamente por el modelo para lograr una descripción coherente y natural.

Finalmente se debe crear una aplicación como prototipo para interactuar con el modelo, el modelo debe ser capaz de recibir la imagen entregada por el usuario en un formato común y devolver una cadena de texto que la describa.

2.2. Especificaciones

Requerimientos funcionales:

1. Entrada de imágenes: El prototipo debe poder aceptar imágenes en formato adecuado (JPEG y PNG) como entrada.
2. Salida de descripciones: El modelo debe producir descripciones en español de las imágenes de entrada.

Requerimientos no funcionales:

1. Calidad de las descripciones: Las descripciones generadas deben ser relevantes y capturar detalles importantes de la imagen, reflejando el contenido visual de manera adecuada y sin sesgos culturales o lingüísticos.
2. Coherencia: El modelo debe producir descripciones coherentes con el contenido visual observado.

3. Robustez: El modelo debe ser capaz de lidiar con diferentes estilos de imágenes y variaciones en la calidad visual, garantizando una descripción confiable independientemente de las condiciones de entrada.
4. Precisión: Debido a la generalidad de las imágenes se busca que el modelo ofrezca una comprensión suficiente del contenido visual para comunicar la idea principal.

2.3. Restricciones

Debido a que los modelos de Deep learning usualmente son costosos computacionalmente hay una limitación. Para la creación del modelo y entrenamiento del modelo se pueden tomar decisiones que se ajusten a esta limitación como el tamaño de lote, el número de épocas o la tasa de aprendizaje, se profundizara en esto más adelante.

Otra de las restricciones del proyecto es la falta de bases de datos con descripciones humanas, a pesar de que si existen no son muy grandes y esto es una limitación ya que la disponibilidad de datos de entrenamiento de alta calidad y tamaño adecuado es esencial para desarrollar un modelo preciso y robusto. Sin bases de datos adecuadas, el modelo puede tener dificultades para aprender patrones lingüísticos y contextuales necesarios para generar descripciones coherentes y relevantes para las imágenes en español.

3. Desarrollo del diseño

3.1. Diseño elegido

Para realizar el modelo se utilizó una arquitectura que se puede ver reflejado en la siguiente figura (Ver fig. 2)

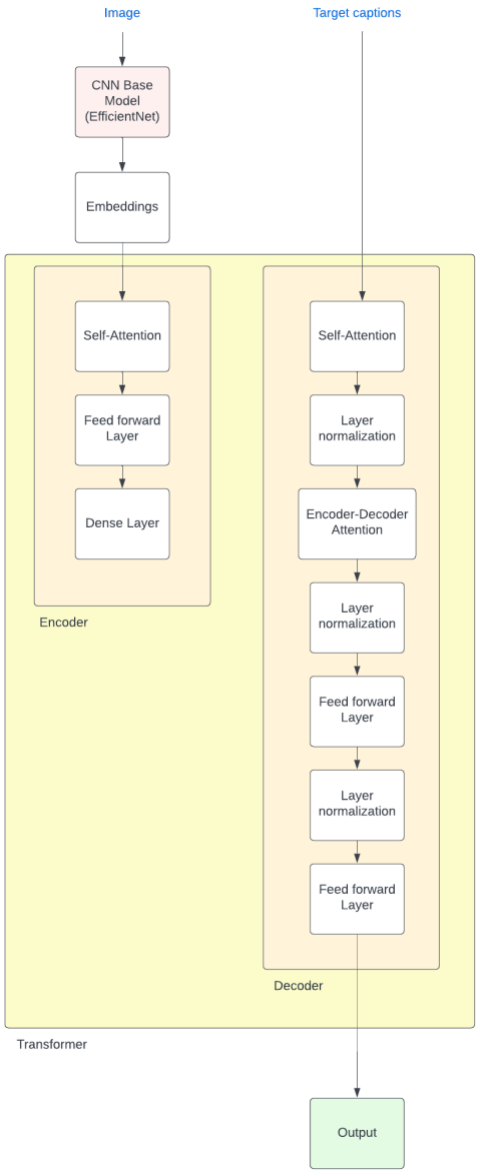

Figura 2: Arquitectura del modelo expuesto en este proyecto.

Consiste en una arquitectura encoder-decoder, la primera capa es un CNN (Convolutional Neural Network), después de pasar por un proceso de pre-procesamiento de la imagen esta es la primera capa que la recibe. Esta red funciona de forma que "cargaríamos la entrada, generalmente en forma de un vector multidimensional a la capa de entrada del cual la distribuirá a las capas ocultas. Las capas ocultas luego tomarán decisiones de la capa anterior y sopesarán cómo un cambio estocástico en sí mismo perjudica o mejora el resultado final, y esto se conoce como el proceso de aprendizaje." (Nash & O'Shea, 2015). El objetivo de esta capa es descubrir y aprender características de las imágenes, en este caso se busca que comience a identificar objetos, animales, situaciones, entre otros. En el diagrama se puede ver un ejemplo de su funcionamiento en un modelo de clasificación (Ver fig. 3).

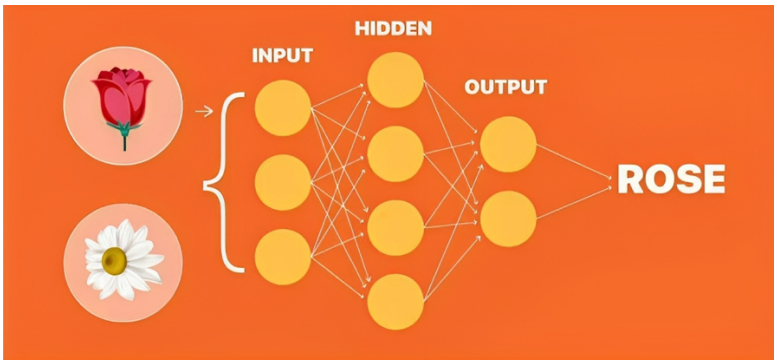

Figura 3: Ejemplo de funcionamiento de una CNN (Ambalina, 2023)

La siguiente capa es la del encoder que contiene varias capas, “son capas de unidades recurrentes donde, en cada paso de tiempo, se recibe un token de entrada, recopilando información relevante y produciendo un estado oculto.” (Ambalina, 2023)

Después el decoder es responsable de mapear la representación visual abstracta de la imagen en una secuencia de palabras o tokens que forman la descripción completa de la imagen. (Muñoz, 2020)

En este modelo también se usan capas de atención, estas capas se proponen como solución para “en lugar de codificar la secuencia de entrada en un solo vector de contexto fijo, el modelo de atención desarrolla un vector de contexto que se filtra específicamente para cada paso de la salida.” (Muñoz, 2020) se enfocan en partes específicas del "embedding" de la imagen mientras genera cada palabra de la descripción. Esto mejora la calidad de las descripciones generadas al permitir que el modelo preste más atención a las partes relevantes de la imagen en diferentes etapas de la generación de palabras. (Muñoz, 2020)

3.2. Alternativas de diseño

Aunque se escogió este diseño existen opciones que también han sido bastante exploradas como:

1. GANs: “La idea detrás de las GAN de subtítulos de imágenes es entrenar a un discriminador para detectar la desalineación entre una imagen y una oración generada y mejorar la capacidad del generador para alinear el subtítulo con la imagen correspondiente.” (Goodfellow, Pouget-Abadi, & Mirza, 2022). Sin embargo, uno de los problemas es que al momento de generar una respuesta es que el lenguaje es un problema discreto. Es decir, que proporcionan directamente estos tokens discretos como entradas al discriminador no permite que los gradientes se propaguen a través de ellos, ya que son discontinuos.

2. Image-text alignment for captioning: En lugar de utilizar una arquitectura completamente integrada como el encoder-decoder, se pueden utilizar modelos independientes para el procesamiento de imágenes y el procesamiento de texto, y

luego “alinear” las representaciones para generar la descripción. Uno de los problemas más grandes es que es mucho más lento durante los tiempos de entrenamiento y prueba debido a la necesidad de calcular características distintas para cada par de imagen y texto. (Paischer, Adler, Hofmarcher, & Hochreiter, 2022)

3. Retrieval-based image captioning: Los métodos basados en recuperación describen imágenes recuperando subtítulos preexistentes de un repositorio. Los modelos actuales utilizan características como el color, la forma, etc. para encontrar imágenes similares. Sin debido a la necesidad de calcular características distintas para cada par de imagen y texto. (Vijayaraju, 2019)

4. Implementación

El enfoque adoptado para resolver el problema consistió en implementar una arquitectura de Transformer encoder-decoder combinada con una CNN (EfficientNetB0). La elección de esta combinación de modelos permitió aprovechar las capacidades de atención y procesamiento secuencial de Transformer para la generación de texto, junto con las capacidades de extracción de características visuales de la CNN para comprender el contenido visual de las imágenes.

Para el desarrollo y entrenamiento del modelo, se optó por utilizar Tensorflow de Keras como el marco de trabajo principal. Keras proporciona una interfaz de alto nivel para construir y entrenar redes neuronales, lo que facilitó la implementación del modelo y permitió concentrarse en el diseño y la experimentación. Se utilizaron como base para la implementación ejemplos de descripción de imágenes públicos de Keras [17].

Además, se utilizaron otras librerías como pandas y numpy para la manipulación y el procesamiento de datos, lo que contribuyó a la preparación del conjunto de entrenamiento y validación.

Para obtener los beneficios de una infraestructura de computación en la nube, se decidió utilizar Google Colab como plataforma de desarrollo y entrenamiento del modelo. Google Colab ofrece recursos de cómputo y acceso a GPU, lo que aceleró significativamente el proceso de entrenamiento del modelo y permitió manejar conjuntos de datos más grandes y complejos.

En cuanto a la implementación del modelo para el uso en tiempo real, se optó por crear un servidor REST utilizando Flask, una librería de Python para construir aplicaciones web. Además de los componentes mencionados anteriormente, también se incorporó el uso del API “Web Speech API” en la aplicación para leer en voz alta los resultados de las descripciones de imágenes generadas.

4.1. Entendimiento del negocio

El inicio del proceso de implementación del modelo, siguiendo la metodología establecida, implica el entendimiento del negocio. Durante esta fase, se llevó a cabo una investigación sobre la aplicación del Deep learning en la creación de textos alternativos.

La opción más común en las aplicaciones actuales es que el usuario al subir una foto o en su propia página web proporcione por sí mismo las descripciones de sus imágenes. También se motiva a los otros usuarios a colaborar con la creación de estos textos. Lo que conocemos como textos alternativos. No obstante, cada vez va creciendo más el reemplazo por una inteligencia artificial que pueda generar el texto alternativo.

Otro aspecto a tener en cuenta es que "el texto alternativo (Alt) está destinado a transmitir el "por qué" de la imagen en relación con el contenido de un documento o página web. Se lee en voz alta a los usuarios mediante un software lector de pantalla y se indexa en los motores de búsqueda. También se muestra en la página si la imagen no se carga." (*Write good Alt Text to describe images*, s/f). Lo que es busca al generar estos textos es explicar que hay en la imagen y no solo nombrar los objetos o el paisaje o el contexto si no una relación entre ellos y formar conexiones. Es por esto por lo que los modelos de Deep learning son una excelente opción para generarlos.

4.2. Entendimiento de los datos

El propósito del modelo es recibir imágenes de cualquier tipo y dar una descripción general de lo que sucede o se observa. Debido a esto, se buscaron bases de datos que contengan una gran cantidad de imágenes variadas, en diferentes ángulos, con diferentes objetos, en diferentes lugares, etc. Se encontraron datasets famosos como Flickr o MS-COCO.

La decisión final fue emplear el conjunto de datos MS-COCO (Garcia, 2020). Este conjunto de datos en español constituye una versión traducida del reconocido conjunto de datos MS-COCO, y está compuesto por imágenes emparejadas con sus respectivas descripciones. Cada imagen cuenta con 5 descripciones, dichas descripciones han sido generadas por anotadores humanos y proporcionan información detallada sobre el contenido de las imágenes.

La utilización de MS-COCO permitió el entrenamiento del modelo. Sin embargo, dado que MS-COCO se encuentra en inglés, el conjunto original de datos se tradujo a español para aprovechar el recurso en este idioma. Para lograr esta traducción, se utilizó la herramienta de inteligencia artificial conocida como Deepl.

Deepl, un sistema de traducción neuronal basado en técnicas de aprendizaje profundo se empleó para la traducción del texto del conjunto de datos. A diferencia de los enfoques tradicionales basados en reglas, Deepl adquiere su capacidad de traducción al analizar

grandes volúmenes de datos en múltiples idiomas y capturar los patrones lingüísticos inherentes. (¿Cómo funciona DeepL?, s/f)

En términos de funcionamiento, Deepl se basa en una arquitectura de red neuronal recurrente (RNN) que procesa el texto de entrada en fragmentos, generando así una representación semántica interna. Esta representación es posteriormente utilizada para generar el texto traducido en el idioma de destino. La red neuronal ha sido entrenada de forma "multilingüe", lo que le permite capturar las correspondencias entre palabras y frases de diferentes idiomas.

"Es bien sabido que la mayoría de los sistemas de traducción a disposición del público son modificaciones directas de la arquitectura Transformer. Las redes neuronales de DeepL también contienen partes de esta arquitectura, como los mecanismos de atención. No obstante, nuestras redes neuronales presentan diferencias significativas en topología que han implicado un avance notable en cuanto a calidad de traducción, en especial en comparación con la calidad de otros traductores automáticos. Estas diferencias en la calidad de la arquitectura se aprecian especialmente cuando entrenamos y comparamos internamente a nuestras redes neuronales con las redes más conocidas basadas en la arquitectura Transformer empleando los mismos datos." (¿Cómo funciona DeepL?, s/f)

Figura 4: Ejemplo imagen dataset MS-COCO

En la figura 4 podemos ver un ejemplo de las imágenes del dataset y sus respectivas descripciones son:

- Primer plano de recipientes de comida que contienen brócoli y pan.
- Comida presentada en bandejas de plástico de colores brillantes.
- Contenedores llenos de diferentes tipos de alimentos.
- Platos coloridos que contienen carne, verduras, frutas y pan.
- Un montón de bandejas con diferentes alimentos

En el conjunto de datos MS-COCO, se emplean cinco descripciones diferentes para cada

imagen con el propósito de enriquecer la diversidad lingüística y proporcionar una mayor consistencia en las anotaciones. Al incluir múltiples descripciones para una misma imagen, se busca mitigar posibles sesgos individuales y variaciones en la interpretación del contenido visual.

La consistencia en las anotaciones es fundamental para garantizar la calidad y confiabilidad de los resultados. Al contar con varias descripciones generadas por anotadores humanos, se evita depender únicamente de una perspectiva o interpretación específica. Esto ayuda a obtener anotaciones más precisas y representativas del contenido visual de las imágenes.

Además de la consistencia, las múltiples descripciones también fomentan la generalización en los modelos de descripción de imágenes. Al contar con diferentes perspectivas y expresiones lingüísticas, se promueve la capacidad del modelo para capturar la diversidad de los objetos, escenas y acciones representados en las imágenes. Esto ayuda a evitar el sobreajuste y a generar descripciones más variadas y contextualmente relevantes.

4.3. Preprocesamiento de los datos

4.3.1. Carga de datos

Las descripciones estaban en un formato inicial de Excel y las imágenes en una carpeta alojados en Google Drive. Se cargaron a la memoria de Google Colab, y se creó un diccionario que contiene la dirección de la imagen y sus 5 respectivas descripciones (Ver fig 5). Adicionalmente se le agrego a cada cadena de texto un “<start>” y “<end>” para poder compararlas con los resultados del modelo como se explicará más adelante.

```
def load_data(filename, start_row, end_row):
  traintext = pd.read_excel(filename)
  caption_mapping ={}
  text_data = []
  for ind, fila in traintext.iterrows():
    if ind < start_row:
            continue
    if ind > end_row:
            break
    id = fila['image_id']
    id = '/content/lina/MyDrive/archive/images/' + str(id).zfill(12)+ '.jpg'
    caption = fila['caption']
    caption = "<start> " + caption.strip() + " <end>"
    text_data.append(caption)
    if id not in caption_mapping:
     caption_mapping[id] = [caption]
    else:
       caption_mapping[id].append(caption)
  return caption_mapping, text_data
```

Figura 5: Bloque de código con el método load_data

4.3.2. Vectorización y limpieza de datos

```
def custom_standardization(input_string):
    lowercase = tf.strings.lower(input_string)
    return tf.strings.regex_replace(lowercase, "[%s]" % re.escape(strip_chars), "")


strip_chars = "!\"#$%&'()*+,-./:;<=>?@[\]^_`{|}~"
strip_chars = strip_chars.replace("<", "")
strip_chars = strip_chars.replace(">", "")

vectorization = TextVectorization(
    max_tokens=VOCAB_SIZE,
    output_mode="int",
    output_sequence_length=SEQ_LENGTH,
    standardize=custom_standardization,
)
vectorization.adapt(text_data)
```

Figura 6: Bloque de código con el método custom_standarization

La función `custom_standardization` (Ver fig. 6) toma una cadena de entrada y realiza los siguientes pasos de estandarización:

1. Conversión a minúsculas: La función `tf.strings.lower` se utiliza para convertir la cadena de entrada a minúsculas. Esto garantiza consistencia en el procesamiento y reduce la variabilidad causada por mayúsculas y minúsculas.

2. Eliminación de caracteres no deseados: La función `tf.strings.regex_replace` se utiliza para eliminar los caracteres no deseados de la cadena de entrada. En este caso, se utiliza una expresión regular junto con la variable `strip_chars` para especificar los caracteres que se deben eliminar. Estos caracteres son aquellos definidos en la cadena `strip_chars`, que incluyen símbolos de puntuación y otros caracteres especiales.

A continuación, se define un objeto `TextVectorization`. Este objeto se utiliza para vectorizar el texto, es decir, convertir las palabras en índices enteros para su procesamiento. Se configura con los siguientes parámetros:

- `max_tokens`: Especifica el número máximo de tokens únicos (palabras) que se utilizarán en el vocabulario. Este valor se establece en `VOCAB_SIZE`. Para el modelo VOCAB_SIZE usa un valor de 10.000. Este parámetro es importante porque se escogen la cantidad de opciones de palabras que tendrá el modelo para describir una imagen.

- `output_mode`: Define el modo de salida del vectorizador. En este caso, se establece en "int" para que las palabras se representen mediante índices enteros.

- `output_sequence_length`: Establece la longitud máxima de las secuencias de salida. Si las secuencias son más cortas, se rellenan con ceros. En este caso, se establece en `SEQ_LENGTH` (250).

- `standardize`: Se establece como la función `custom_standardization` definida anteriormente. Esta función se utilizará para estandarizar cada cadena de texto antes de la vectorización.

4.3.3. Ajuste de imágenes

Con la finalidad de aumentar la cantidad de datos y la diversidad de estos se realizaron transformaciones aleatorias en las imágenes utilizando la librería de Keras [18]. Se utilizaron 3 transformaciones:

RandomFlip("horizontal"): Esta transformación realiza un volteo horizontal aleatorio en las imágenes. Es decir, algunas imágenes se reflejarán horizontalmente, lo que crea versiones espejo de las imágenes originales.

RandomRotation(0.2): Esta transformación realiza una rotación aleatoria en las imágenes con un ángulo máximo de 0.2 radianes (aproximadamente 11.5 grados).

RandomContrast(0.3): Esta transformación ajusta aleatoriamente el contraste de las imágenes, aumentando o disminuyendo el contraste en un rango controlado. El valor de 0.3 indica que el contraste se ajustará en un rango de -0.3 a +0.3.

4.4. Construcción del modelo

4.4.1. Hiperparametros

Para el ajuste de los hiperparametros se realizó una investigación sobre el ajuste de estos en modelos de este tipo y teniendo en cuenta el dataset, la capacidad computacional, el tiempo, entre otros factores. Gracias a estas fuentes [21][22] se realizó una búsqueda manual sobre el modelo y a continuación se explicará la elección de cada hiperparametro.

IMAGE_SIZE (299, 299): Este define el tamaño de entrada al que se redimensionan las imágenes que entran al modelo. El tamaño de la imagen puede afectar la cantidad de detalles que el modelo puede capturar. Entre más grande se pueden obtener más características, pero también se puede volver más lento el tiempo de entrenamiento ya que requiere más recursos computacionales. Después de probarlo con tamaños de 224, 256 y 299 la diferencia no fue mucha en el tiempo y por eso se escogió 299 x 299.

VOCAB_SIZE (10000): Como se nombró arriba es el tamaño del vocabulario con el que cuenta el modelo. Es decir, cuántas palabras únicas se utilizarán en el vocabulario para generar las descripciones. Un vocabulario más grande puede permitir una mayor variabilidad en las descripciones generadas, pero también aumentará la complejidad computacional del modelo. Después de probarlo con 10.000, 15.000 y 20.000 de tamaño. La exactitud fue mayor al usar un vocabulario de 10.000 palabras. Esto puede ser debido a que el conjunto de datos MS-COCO no contiene una variedad suficiente de palabras únicas para justificar un vocabulario de 15,000 o 20,000 palabras. Si la mayoría de las descripciones en el conjunto de datos se componen de un conjunto limitado de palabras, aumentar el tamaño del vocabulario puede agregar palabras poco comunes o irrelevantes que no aportan información útil al modelo.

SEQ_LENGTH (25): También se nombró en el preprocesamiento de los datos como el tamaño máximo de la cadena de texto que devolverá el modelo. Esto puede ayudar a controlar la complejidad del modelo y facilitar el entrenamiento, pero también puede limitar la capacidad del modelo para describir imágenes más detalladamente. No obstante, este es el tamaño máximo de las descripciones de MS-COCO también, así que con la finalidad de que las secuencias sean coherentes y concretas se escogió este como el tamaño adecuado.

EMBED_DIM (1024): Es la dimensión de los embeddings que representan las palabras del modelo. Al tomar un valor grande se pueden capturar mejor las relaciones entre palabras. Después de probar con 256, 512, 768 y 1024 la exactitud del modelo mejoro con cada aumento en el valor y por esto se escogió 1024.

FF_DIM (1024): Representa la dimensión de las capas completamente conectadas en el modelo. Estas capas se utilizan para procesar y transformar las características extraídas de las imágenes y las incrustaciones de las palabras. Un valor mayor de FF_DIM puede permitir que el modelo aprenda representaciones más complejas. Después de probar con 256, 512, 768 y 1024 la exactitud del modelo mejoro con cada aumento en el valor y por esto se escogió 1024.

BATCH_SIZE (128): Es un hiperparámetro que determina la cantidad de ejemplos de entrenamiento que se utilizan en cada paso de actualización de los pesos del modelo. Un tamaño de lote mayor implica que se procesan más ejemplos a la vez, lo que puede acelerar el entrenamiento, pero también requiere más memoria RAM o GPU. Un tamaño de lote más pequeño, por otro lado, puede reducir el consumo de memoria, pero también puede ralentizar el entrenamiento debido a la menor cantidad de ejemplos que se procesan en cada paso.

Es común utilizar tamaños de lote que sean potencias de 2 debido a que las operaciones vectorizadas, que son fundamentales para el rendimiento acelerado por GPU, a menudo funcionan de manera más eficiente con tamaños de lote que sean potencias de 2. Se probo

con tamaños de batches de potencia de 2 y se fue disminuyendo al encontrar un error de memoria, de esta forma se llegó al 128

Es importante destacar que cuando se utiliza un tamaño de lote más pequeño (por ejemplo, 128) en comparación con un tamaño de lote más grande (por ejemplo, 512), los resultados iniciales pueden parecer engañosamente prometedores. Durante las primeras épocas de entrenamiento, el modelo mostraba una mejora más rápida y aparentemente significativa en términos de exactitud. Esta aparente mejora inicial se debe a que, con un tamaño de lote más pequeño, el modelo está procesando menos ejemplos de entrenamiento en cada paso de actualización de los pesos, lo que puede conducir a una convergencia más rápida en la dirección correcta. Sin embargo, a medida que el entrenamiento continúa y el modelo explora una mayor variabilidad de datos y patrones complejos, el tamaño de lote más pequeño puede no proporcionar suficiente información para una actualización de pesos precisa. (Ver figura 7)

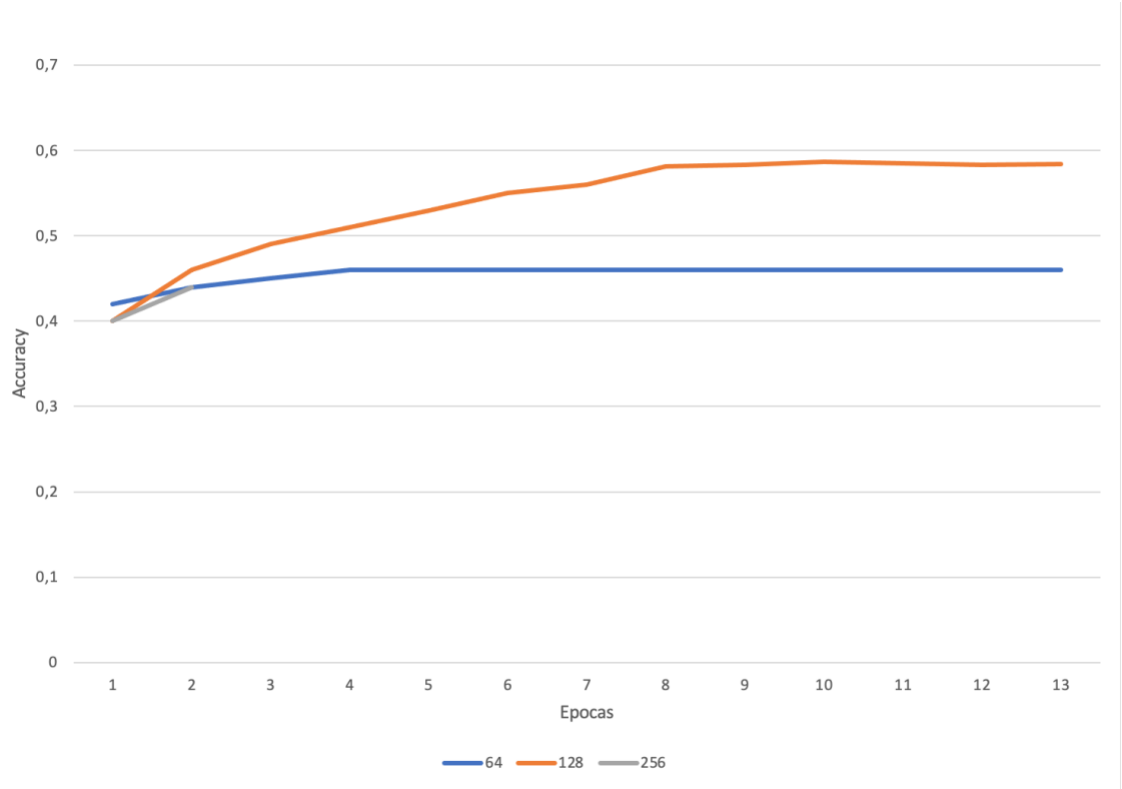

Figura 7: Grafica de comparación entre las épocas y la exactitud.

EPOCAS (15): Representa la cantidad de veces que el modelo pasará por todo el conjunto de datos de entrenamiento durante el proceso de entrenamiento. Cada época representa un ciclo completo de entrenamiento. El número de épocas debe ser lo suficientemente grande para que el modelo aprenda patrones complejos en los datos, pero también se debe tener cuidado para evitar el sobreajuste. El parámetro principal para decidir el número de épocas es el momento en el que el valor de exactitud al enfrentarse al modelo de validación se detiene o comienza a bajar, esto paso en el modelo entre la época 13 y 15 (Ver figura 7). También hicimos uso de la paciencia (3) para que el modelo se detenga si en 3 épocas seguidas los resultados no mejoran o empeoran.

4.4.2. CNN

Como se tenía contemplado en el diseño se usó una red convolucional, para la elección de esta se consultó en diversas fuentes [23][24][25] y se probó con varias de ella cuando ya se tuvo todo el modelo creado. Las 3 con las que se obtuvieron mejores resultados fueron EfficientNetB0, Xception y VGG16. En la siguiente grafica (Ver figura 8) se pueden ver los resultados de la exactitud con el modelo de validación de cada red con el paso de las épocas.

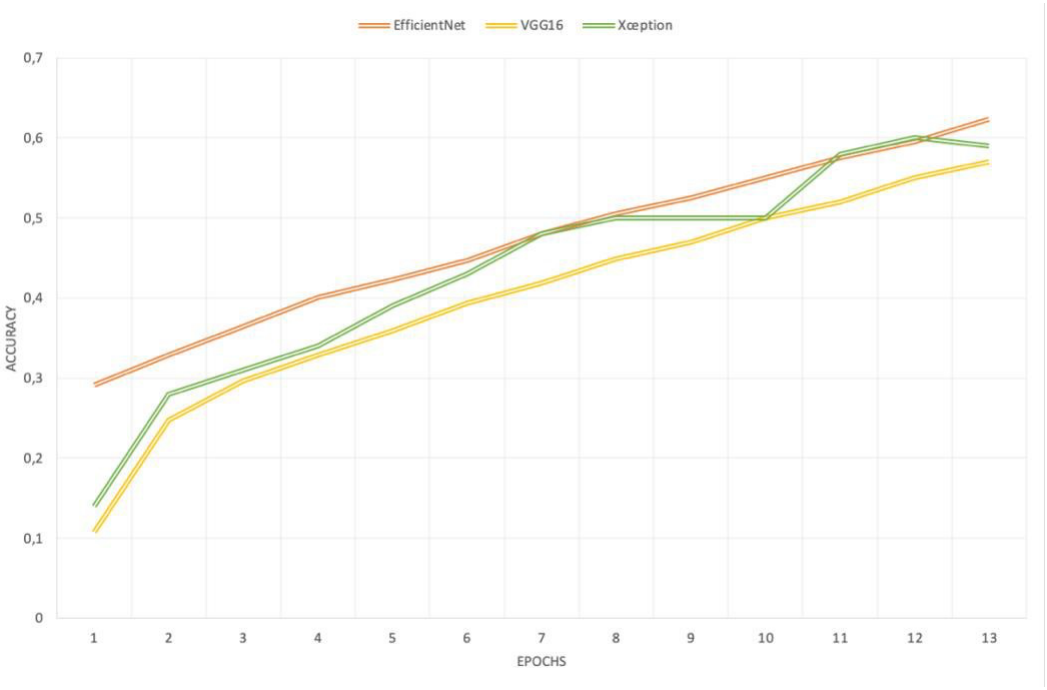

Figura 8: Grafica de comparación de las opciones de redes convolucionales.

Debido a esto se llegó a la conclusión de que la mejor opción para el modelo es EfficientNetB0. "EfficientNet es una arquitectura de red neuronal convolucional y un método de escalado que escala uniformemente todas las dimensiones de profundidad/ancho/resolución utilizando un coeficiente compuesto. A diferencia de la práctica convencional que escala arbitrariamente estos factores, el método de escala EfficientNet escala uniformemente el ancho, la profundidad y la resolución de la red con un conjunto de coeficientes de escala fijos." (EfficientNet, s/f)

```
def get_cnn_model():
    base_model = efficientnet.EfficientNetB0(
        input_shape=(*IMAGE_SIZE, 3), include_top=False, weights="imagenet",
    )
    base_model.trainable = False
    base_model_out = base_model.output
    base_model_out = layers.Reshape((-1, base_model_out.shape[-1]))(base_model_out)
    cnn_model = keras.models.Model(base_model.input, base_model_out)
    return cnn_model
```

Figura 9: Bloque de código con el método get_cnn_model.

Para usar esta red en nuestro Proyecto (Ver figura 9), primero instanciamos el modelo que tiene como argumentos el tamaño de la imagen y el color RGB, include_top=false, esto se refiere a que solo utilizamos la red para obtener las características de las imágenes pero no las predicciones (que es la última capa) y se incluyen los pesos preentrenados del

conjunto de datos de ImageNet. Se congelan los pesos para que no puedan ser cambiados y se ajusta la salida a la entrada del encoder (que es el siguiente paso).

4.4.3. Encoder-decoder

El encoder y decoder están respectivamente compuestos de 3 capas (Ver figura) las cuales son (Li, Yang, & Hu, 2022):

- MultiHeadAttention: Esta capa realiza la atención multi-cabezal, que permite que el modelo se centre en diferentes partes de las características de entrada para capturar relaciones significativas.
- LayerNormalization: Esta capa aplica la normalización de capa, lo que estandariza las activaciones de las capas para estabilizar el entrenamiento y mejorar la convergencia.
- Dense: Esta capa completamente conectada tiene una función de activación "relu" y se aplica después de la normalización de capa para introducir no linealidades y mayor capacidad de representación en el modelo.

En la arquitectura del Transformer se emplearon múltiples cabezas de atención. En particular, se utilizaron 8 cabezas de atención para el encoder y 12 cabezas de atención para el decoder.

El decoder consta de dos capas principales de atención:

1. Capa de Self-Attention (Auto atención): Esta capa de atención, similar a la del encoder, recibe como entrada las incrustaciones de palabras generadas en pasos anteriores dentro del decoder. Esta auto atención permite que el modelo se centre en diferentes partes de la secuencia de palabras generadas para capturar relaciones contextuales dentro de la descripción en proceso de construcción.

2. Capa de Atención hacia el Encoder: Esta capa de atención recibe como entrada el resultado del encoder, que son las características visuales extraídas por la red convolucional a partir de las imágenes. La capa de atención hacia el encoder permite que el modelo se enfoque en diferentes partes de las características visuales para generar descripciones que se ajusten mejor a la información visual de las imágenes.

Al utilizar más cabezas de atención en el decoder en comparación con el encoder, se le da al modelo una mayor capacidad para capturar patrones complejos en las secuencias de palabras generadas y comprender las relaciones entre las características visuales y el texto descriptivo. La combinación de autoatención y atención hacia el encoder en el decoder permite que el modelo tome decisiones informadas sobre qué información visual es relevante para cada palabra en la descripción generada.

4.4.4. Modelo final y beam search

Para la unificación de las capas y la compilación del modelo se utilizó la clase ImageCaptioningModel de Keras.

Como paso final se creó un Beam Search (Shambharkar et al., 2021), esta toma como entrada las imágenes (inputs) y las procesa a través del modelo CNN para obtener las incrustaciones de la imagen. Luego, estas incrustaciones se pasan por el encoder para obtener la representación codificada de la imagen. Luego, el algoritmo procede a generar la descripción palabra por palabra en un bucle que se repite hasta alcanzar la longitud máxima de la secuencia. En cada paso del bucle, el algoritmo explora diferentes secuencias candidatas y evalúa su probabilidad en función de las predicciones de probabilidad del modelo de decoder. Para cada candidato se agrega una nueva palabra posible de acuerdo con las predicciones del modelo y se calcula un nuevo puntaje de probabilidad acumulativa. Una vez que se ha alcanzado la longitud máxima de la secuencia o todas las secuencias tienen la etiqueta especial <end>, el algoritmo selecciona la secuencia más probable y la devuelve como la descripción final generada. Debido a que el modelo solía repetir palabras se implementó una estrategia de penalización (Ver figura 10) que se aplica durante el proceso de beam search para evitar la generación repetida de palabras en la descripción. El penalizador se utiliza para penalizar las secuencias que contienen palabras duplicadas, lo que ayuda a mejorar la coherencia y diversidad de las descripciones generadas. Cuando se está construyendo una nueva secuencia candidata, se verifica si la palabra que se agregaría a la secuencia candidata ya está presente en la secuencia hasta ese momento. Si la palabra ya está presente en la secuencia, se aplica una penalización de valor -10 al puntaje de probabilidad acumulativa de esa secuencia candidata.

```
predictions = self.decoder(tokenized_caption, encoded_img, training=False, mask=mask)
sorted_indices = np.argsort(predictions[0, i, :].numpy())[::-1][:beam_size]

for idx in sorted_indices:
    candidate_caption = np.append(caption, self.vocab_tensor[idx])
    candidate_score = score + tf.math.log(predictions[0, i, idx])

    penalty = 0
    if self.vocab_tensor[idx] in candidate_caption[:-1]:
        penalty = -10

    all_candidates.append((candidate_caption, candidate_score + penalty))
```

Figura 10: Bloque de código utilizado para implementar beam search.

Para la tasa de aprendizaje se usa la clase LRSchedule de Keras, esta se estableció en 1e-4 para el entrenamiento del modelo, se experimentó con 1e-3, 1e-4 y 1e-5 y combinándolos con diferentes tamaños de lote, pero 1e-4 fue el elegido gracias a que con este se encontró un calentamiento gradual y no afecto las métricas del modelo.

Para el entrenamiento del modelo (Ver figura 11) se utilizaron los datasets de entrenamiento y validación previamente preprocesados, se utilizaron 15 épocas como se ajustó en los hiperparametros (Como se puede ver en la figura el entrenamiento se detuvo en la época 14 debido a que no hubo mejoría respecto a los resultados frente al set de

validación en las últimas 3 épocas) y se incluyó la paciencia. Debido a que se obtuvieron los resultados deseados se usó este modelo para el prototipo y para las pruebas que se nombraran más adelante.

```
caption_model.fit(
    train_dataset,
    epochs=EPOCHS,
    validation_data=valid_dataset,
    callbacks=[early_stopping],
)
```

```
Epoch 1/15
12/12 [==============================] - 263s 19s/step - loss: 26.8892 - acc: 0.2151 - val_loss: 18.9829 - val_acc: 0.4036
Epoch 2/15
12/12 [==============================] - 133s 11s/step - loss: 18.0116 - acc: 0.3434 - val_loss: 15.3495 - val_acc: 0.4651
Epoch 3/15
12/12 [==============================] - 137s 12s/step - loss: 15.3832 - acc: 0.4020 - val_loss: 13.8304 - val_acc: 0.4918
Epoch 4/15
12/12 [==============================] - 137s 11s/step - loss: 13.7391 - acc: 0.4323 - val_loss: 12.8458 - val_acc: 0.5141
Epoch 5/15
12/12 [==============================] - 137s 11s/step - loss: 12.4119 - acc: 0.4670 - val_loss: 11.8987 - val_acc: 0.5393
Epoch 6/15
12/12 [==============================] - 137s 11s/step - loss: 11.1143 - acc: 0.4999 - val_loss: 11.2732 - val_acc: 0.5557
Epoch 7/15
12/12 [==============================] - 137s 11s/step - loss: 9.9562 - acc: 0.5376 - val_loss: 10.7223 - val_acc: 0.5682
Epoch 8/15
12/12 [==============================] - 137s 12s/step - loss: 8.7989 - acc: 0.5692 - val_loss: 10.4159 - val_acc: 0.5801
Epoch 9/15
12/12 [==============================] - 137s 12s/step - loss: 7.6845 - acc: 0.6055 - val_loss: 10.2314 - val_acc: 0.5819
Epoch 10/15
12/12 [==============================] - 137s 12s/step - loss: 6.5723 - acc: 0.6389 - val_loss: 10.2233 - val_acc: 0.5834
Epoch 11/15
12/12 [==============================] - 137s 11s/step - loss: 5.5055 - acc: 0.6727 - val_loss: 10.2122 - val_acc: 0.5878
Epoch 12/15
12/12 [==============================] - 137s 11s/step - loss: 4.5045 - acc: 0.7097 - val_loss: 10.3506 - val_acc: 0.5850
Epoch 13/15
12/12 [==============================] - 136s 11s/step - loss: 3.5814 - acc: 0.7364 - val_loss: 10.4581 - val_acc: 0.5838
Epoch 14/15
12/12 [==============================] - 137s 12s/step - loss: 2.7023 - acc: 0.7674 - val_loss: 10.6262 - val_acc: 0.5847
<keras.callbacks.History at 0x7f50173ac400>
```

Figura 11: Resultados exactitud y perdida para el entrenamiento del modelo.

4.5. Desplegar el modelo

Una vez que el entrenamiento del modelo se completó exitosamente en Google Colab, se procedió a guardar el modelo entrenado en formato SavedModel de TensorFlow. Este formato permite almacenar el modelo junto con sus pesos y configuración para su posterior uso y despliegue en otras aplicaciones.

Luego, se desarrolló una aplicación web utilizando Flask, que ofrece una interfaz fácil de usar para los usuarios. La aplicación permite a los usuarios cargar imágenes en formato JPEG a través de la interfaz (Ver figura 12). Al presionar un botón, la imagen se pasa al modelo descargado y entrenado previamente. El modelo genera una descripción para la imagen y la devuelve a la interfaz, donde se muestra al usuario (Ver figura 13).

Figura 12: Interfaz de la aplicación creada.

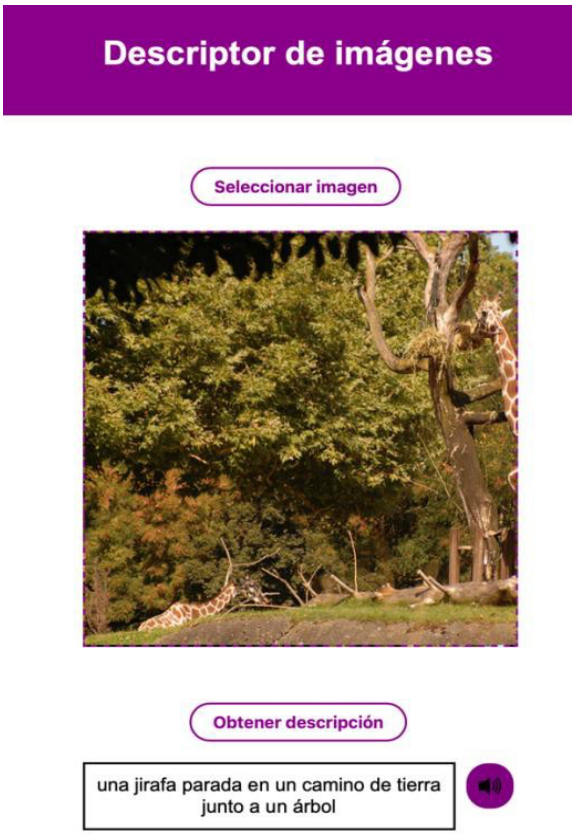

Figura 13: Funcionamiento a través de la interfaz de la aplicación creada.

5. Validación

5.1. Métodos

Para evaluar el modelo creado se usaron dos métricas comúnmente usadas en los modelos de Image captioning (Cui, Y., Yang, G., Veit, A., Huang, X., & Belongie, S., s/f). Estas son:

BLEU (Bilingual Evaluation Understudy): La métrica compara una traducción automática con una o más traducciones de referencia y calcula la similitud entre ellas. Cuanto mayor sea el valor de BLEU, mejor será la calidad de la traducción automática en comparación con las traducciones de referencia.

El cálculo de BLEU se basa en la coincidencia de n-gramas (secuencias de n palabras contiguas) entre la traducción automática y las traducciones de referencia. Para cada n-grama, se cuenta cuántas veces aparece en la traducción automática y en las referencias, y luego se calcula la precisión dividiendo el número de ocurrencias coincidentes por el número total de ocurrencias en la traducción automática.

Luego, BLEU calcula un puntaje de brecha para tener en cuenta la longitud de las traducciones. Si la traducción automática es más corta que las referencias, se penaliza para evitar que el modelo simplemente genere traducciones cortas para maximizar la precisión de coincidencia.

Finalmente, se combina la precisión de coincidencia de n-gramas y el puntaje de brecha para obtener el puntaje BLEU final. Cuanto más se asemeje la traducción automática a las referencias en términos de n-gramas y longitud, mayor será el puntaje BLEU. (Papineni et al., 2001)

METEOR (Metric for Evaluation of Translation with Explicit ORdering): A diferencia de BLEU, que se basa en el cálculo de coincidencias de n-gramas, METEOR toma en cuenta tanto la coincidencia de palabras individuales como la similitud global entre la traducción automática y las referencias humanas. METEOR es considerado una métrica más completa y sofisticada que BLEU, ya que toma en cuenta la similitud de cadenas, la coherencia de la traducción y la calidad gramatical. (Banerjee, S., & Lavie, A., 2005)

Las dos métricas dan valores de 0 a 1, siendo 1 una descripción perfecta y 0 que no hay coincidencia en absoluto. Como se puede ver, aunque estas métricas buscan entender la similitud entre las predicciones y las referencias tampoco son un 100% confiables y dependen de muchos factores, por eso se recomienda usar más de una, que fue lo que se hizo en este caso.

5.2. Validación resultados

Para evaluar el comportamiento del modelo se utilizaron las dos métricas nombradas en el punto anterior. Se uso el dataset de test que contiene 10.000 descripciones y para las dos métricas el resultado es el promedio de resultados para cada imagen y sus respectivas descripciones y predicciones. Para BLEU el resultado fue 0.57 (Ver figura 14) y para METEOR el resultado fue 0.62 (Ver figura 15). Es un resultado coherente con lo buscado y que demuestra que aún quedan cosas por mejorar, pero aun así las descripciones son coherentes y logran brindar en la mayoría de los casos una descripción acertada de lo que sucede.

BLEU

```
[ ] from nltk.translate.bleu_score import SmoothingFunction, corpus_bleu, sentence_bleu

    predicted_tokens = [caption.split() for caption in captionsPredicted.values()]
    reference_tokens = [[caption.split() for caption in references] for references in captionsLoaded.values()]
    smooth_func = SmoothingFunction()

    # Calcular el puntaje BLEU promedio
    bleu_score = corpus_bleu(reference_tokens, predicted_tokens, smoothing_function=smooth_func.method1)

    print("BLEU Score:", bleu_score)

    BLEU Score: 0.5723008271808738
```

Figura 14: Puntuación de BLEU

▾ METEOR

```
import nltk
from nltk.translate.meteor_score import meteor_score

nltk.download('wordnet')
captionsLoaded_strings = list(captionsLoaded.values())
captionsPredicted_strings = list(captionsPredicted.values())

scores = []
for i in range(len(reference_tokens)):
    score = meteor_score(reference_tokens[i], predicted_tokens[i])
    scores.append(score)

# Calcular el puntaje METEOR promedio
meteor_score_promedio = sum(scores) / len(scores)

print("METEOR Score Promedio:", meteor_score_promedio)
```

```
METEOR Score Promedio: 0.6283387851057024
```

Figura 15: Puntuación de METEOR

6. Conclusiones

6.1. Discusión

Se construyo una aplicación basada en una arquitectura encoder-decoder capaz de generar textos alternativos con un 62% de exactitud en promedio para imágenes de cualquier tipo, proporcionando a los usuarios descripciones contextuales del contenido visual. Una de las principales contribuciones de este trabajo es el desarrollo de un modelo de image captioning específico para el idioma español. La mayoría de los modelos existentes se han centrado en inglés, y la disponibilidad de un modelo preciso en español puede tener un gran impacto en aplicaciones y tareas que requieren procesamiento de lenguaje natural en este idioma.

La elección de la arquitectura ha demostrado ser efectiva para capturar relaciones a largo plazo entre las imágenes y sus descripciones en español. Esta combinación de técnicas ha permitido obtener resultados prometedores y puede ser una base sólida para futuras investigaciones en el campo de la descripción de imágenes. A su vez, la integración de una red convolucional preentrenada junto con la atención ha mejorado la capacidad del modelo para comprender y describir el contenido visual de las imágenes.

La inclusión de la funcionalidad de lectura en voz alta hace que la aplicación sea accesible para personas con discapacidades visuales. Esto permite que los usuarios con dificultades para leer las descripciones visualmente puedan escuchar las descripciones generadas por el modelo, mejorando significativamente la experiencia para ellos y logrando el objetivo inicial de contribuir en la generación de textos alternativos para las personas con discapacidad.

6.2. Trabajo futuro

Una de las limitaciones más grandes del proyecto fueron los conjuntos de datos, se necesitan conjuntos de datos más grandes y diversos con descripciones humanas español.

Un conjunto de datos más grande y variado permitirá entrenar el modelo en una mayor variedad de contextos y temas, lo que potencialmente mejoraría la capacidad del modelo para describir imágenes en diferentes dominios.

Otra característica de este tipo de modelos puede ser que se adapten al contexto o dominio específico de la imagen. Por ejemplo, si la imagen proviene de una tienda de deportes, el modelo podría utilizar esa información como guía para generar descripciones más específicas y relevantes relacionadas con el contexto.

Se puede considerar la posibilidad de aplicar el modelo a problemas específicos dentro de la atención médica, el turismo, la educación o cualquier otro campo que requiera descripciones de imágenes en español.

7. Referencias

1. **Ghandi, T., Pourreza, H., & Mahyar, H. (2023).** *Deep Learning Approaches on Image Captioning: A Review.* McMaster University, Hamilton, Ontario, Canada.
2. **Sutskever, I., Vinyals, O., & Le, Q. V. (2014).** *Sequence to Sequence Learning with Neural Networks.* Google.
3. **Nash, R., & O'Shea, K. (2015).** *An Introduction to Convolutional Neural Networks.* Department of Computer Science, Aberystwyth University, Ceredigion; School of Computing and Communications, Lancaster University, Lancashire.
4. **Ambalina, L. (s/f).** *5 Papers on CNNs Every Data Scientist Should Read.* KDnuggets. Recuperado el 1 de agosto de 2023, de https://www.kdnuggets.com/2020/04/5-papers-cnns-data-scientist.html
5. **Muñoz, E. (2020, octubre 11).** *A Guide to the Encoder-Decoder Model and the Attention Mechanism.* Better Programming. https://betterprogramming.pub/a-guide-on-the-encoder-decoder-model-and-the-attention-mechanism-401c836e2cdb
6. **Brownlee, J. (2017, octubre 13).** *How Does Attention Work in Encoder-Decoder Recurrent Neural Networks.* Machinelearningmastery.com. https://machinelearningmastery.com/how-does-attention-work-in-encoder-decoder-recurrent-neural-networks/
7. **Goodfellow, I., Pouget-Abadi, J., & Mirza, M. (2022).** *Generative Adversarial Nets.* Université de Montreal.
8. **Paischer, F., Adler, T., Hofmarcher, M., & Hochreiter, S. (2022).** *Generative Adversarial Nets.* SITTA: A Semantic Image-Text Alignment for Image Captioning.
9. **Vijayaraju, N. (2019).** *Image Retrieval Using Image Captioning.* San Jose State University Library.
10. **"Write Good Alt Text to Describe Images."** (s/f). Harvard.edu. Recuperado el 21 de agosto de 2023, de https://accessibility.huit.harvard.edu/describe-content-images
11. **García, C. (2020).** *MS-COCO-ES: Spanish COCO Captions [Data set].*
12. **Keras: Deep Learning for Humans. (s/f).** Keras.Io. Recuperado el 15 de agosto de 2023, de http://keras.io
13. **"¿Cómo Funciona DeepL?"** (s/f). Deepl.com. Recuperado el 21 de agosto de 2023, de https://www.deepl.com/es/blog/how-does-deepl-work
14. **"EfficientNet."** (s/f). Paperswithcode.com. Recuperado el 28 de agosto de 2023, de https://paperswithcode.com/method/efficientnet
15. **Li, K., Yang, R., & Hu, X. (2022).** *An Efficient Encoder-Decoder Architecture with Top-Down Attention for Speech Separation.* BNRist, Tsinghua University, Beijing 100084, China.
16. **Shambharkar, P. G., Kumari, P., Yadav, P., & Kumar, R. (2021).** *Generating Caption for Image Using Beam Search and Analyzation with Unsupervised Image Captioning Algorithm.* 2021 5th International Conference on Intelligent Computing and Control Systems (ICICCS), 857–864.

17. **Cui, Y., Yang, G., Veit, A., Huang, X., & Belongie, S. (s/f).** *Learning to Evaluate Image Captioning.* Cornell University.
18. **Papineni, K., Roukos, S., Ward, T., & Zhu, W.-J. (2001).** *BLEU: A Method for Automatic Evaluation of Machine Translation.* Proceedings of the 40th Annual Meeting on Association for Computational Linguistics - ACL '02, 311–318.
19. **Banerjee, S., & Lavie, A. (2005).** *METEOR: An Automatic Metric for MT Evaluation with Improved Correlation with Human Judgments.* Proceedings of the ACL Workshop on Intrinsic and Extrinsic Evaluation Measures for Machine Translation and/or Summarization, 65–72.
20. **Dabian, D. A., & Peña Moyano, F. Y. (2020).** *Prevalencia y Causas de Ceguera y Discapacidad Visual en Colombia.* Ciencia y Tecnología para la Salud Visual y Ocular.
21. **"Hyperparameter Tuning over an Attention Model for Image Captioning."** (2021, noviembre). Researchgate.net. https://www.researchgate.net/publication/356462509_Hyperparameter_Tuning_over_an_Attention_Model_for_Image_Captioning
22. **Srinivasan, L., Sreekanthan, D., & Amutha, A. L. (s/f).** *Image Captioning - A Deep Learning Approach.* Ripublication.com. Recuperado el 28 de agosto de 2023, de https://www.ripublication.com/ijaer18/ijaerv13n9_102.pdf
23. **Zhao, P., Li, C., Rahaman, M. M., Xu, H., Yang, H., Sun, H., Jiang, T., & Grzegorzek, M. (s/f).** *A Comparative Study of Deep Learning Classification Methods on a Small Environmental Microorganism Image Dataset (EMDS-6): From Convolutional Neural Networks to Visual Transformers.* Microscopic Image and Medical Image Analysis Group, MBIE College, Northeastern University.
24. **Chollet, F. (2016).** *Xception: Deep Learning with Depthwise Separable Convolutions.* Google, Inc.
25. **Simonyan, K., & Zisserman, A. (2014).** *Very Deep Convolutional Networks for Large-Scale Image Recognition.* Visual Geometry Group, Department of Engineering Science, University of Oxford.