Let me see:
Generador automático de descripciones de imágenes.

José Ignacio Bengoechea Isasa
Máster universitario en Ciencia de Datos (Data Science)
Area de minería de datos y machine learning

Anna Bosch Rué
Jordi Casas Roma
Octubre de 2018

FICHA DEL TRABAJO FINAL

Título del trabajo:	*Let me see: Generador automático de descripciones de imágenes*
Nombre del autor:	*José Ignacio Bengoechea Isasa*
Nombre del consultor/a:	*Anna Bosch Rué*
Nombre del PRA:	*Jordi Casas Roma*
Fecha de entrega (mm/aaaa):	01/2019
Titulación:	*Máster universitario en Ciencia de Datos (Data Science)*
Área del Trabajo Final:	*Minería de datos y machine learning*
Idioma del trabajo:	*Español*
Palabras clave	*Descripción imágenes; redes neuronales; visión artificial;*

Resumen del Trabajo (máximo 250 palabras): *Con la finalidad, contexto de aplicación, metodología, resultados i conclusiones del trabajo.*

El objetivo principal propuesto de este trabajo es diseñar e implementar un modelo, mediante redes neuronales que nos permita ser usado para generar descripciones automáticas de imágenes.

Para poder establecer el modelo se revisarán las líneas de investigación que se han producido durante los años en el area de descripciones de imágenes, lo que es conocido como *state of art.*

Para poder desarrollar el modelo se establecerá que tipo de dataset puede ser más conveniente para su uso, como pueda ser *Flickr* o *MSCOCO*.

Se usarán herramientas que estén presentes en el *state of art*, tales como *TensorFlow* o *Keras*. También se utilizarán modelos entrenados previamente para facilitar el desarrollo del trabajo.

La evaluación por métricas BLEU y ROUGE nos mostrará la cercanía de este trabajo con el estado del arte. Su integración con Telegram permite que cualquier usuario pueda probarlo.

Abstract (in English, 250 words or less):

The main objective proposed in this paper is to design and implement a model, using neural networks that allow us to be used to generate automatic descriptions of images.

In order to establish the model, the research lines that have been produced during the years in the area of image descriptions will be reviewed, which is known as state of art.

In order to develop the model, it will be established which type of dataset may be most convenient for its use, such as Flickr or MSCOCO.

Tools that are present in the state of art, such as TensorFlow or Keras, will be used. Also, previously trained models will be used to facilitate the development of the work.

The evaluation by BLEU and ROUGE metrics will show us the closeness of this work to the state of the art. Its integration with Telegram allows any user to test it.

ÍNDICE DE CONTENIDOS

ÍNDICE DE FIGURAS

A mi familia, sin su ánimo, apoyo y cariño no hubiera sido posible realizar y completar estos estudios de máster.

CAPÍTULO 1. INTRODUCCIÓN

CONTEXTO Y JUSTIFICACIÓN DEL PROYECTO

Este proyecto se realiza con el interés de poder desarrollar un algoritmo que permita generar descripciones en imágenes. Es decir, el algoritmo debe ser capaz no solo de identificar los objetos, o entes, que existan en la imagen, sino que también debe establecer qué tipo de acción están realizando esos objetos, o entes, y elaborar una frase coherente detallándola.

Este tipo de desarrollo es una extensión de los algoritmos de reconocimiento de imágenes. Sin embargo, antes de detallar las características de estos se hará un breve repaso sobre la evolución de estos.

En el año 2010 se convocó la primera edición del desafío de reconocimiento visual a gran escala de ImageNet[1]. El objetivo de este desafío era reconocer si existían objetos en la imagen de un dataset con 1,2 millones de fotos, clasificarlos, entre más de mil diferentes clases, y marcarlos visualmente.

En el año 2012 se presentó una nueva técnica, por parte de un equipo de la Universidad de Stanford, basada en reconocimiento de imágenes mediante el uso de redes convolucionales neuronales.

Esta técnica tuvo una amplia repercusión. Se paso de un error del 25,8% a un error del 16,4% en el algoritmo ganador. En años posteriores se ha popularizado el uso de soluciones basadas en redes convolucionales neuronales, y se ha conseguido minimizar el porcentaje de error, hasta el 2,3% de la edición de 2017.

Esta mejora en la capacidad de reconocimiento de imágenes está produciendo un cambio tecnológico a gran escala. Hay múltiples desarrollos donde se trata de automatizar la tarea de revisión de imágenes por múltiples motivos. Se pueden reseñar las siguientes noticias hechas públicas el último mes:

- Shell aplicara visión artificial e IA para proteger a los clientes a y sus servicios[2].
- El mercado global de tecnologías relativas a visión artificial llegara a 23 billones en 2023[3].

En el caso concreto de este trabajo el objetivo no se limita a reconocer imágenes. El sistema debe ser capaz de describir estas imágenes, para ello se puede aprovechar el potencial de las redes neuronales, y tratar de establecer una relación entre las imágenes y las descripciones, aprovechando que se puede tratar ambas mediante vectores.

El resultado que se quiere obtener es un modelo que nos permita describir imágenes, con un grado elevado de parecido a la realidad.

1.1. MOTIVACIÓN PERSONAL

El procesamiento de imágenes ha tenido una amplia repercusión en los últimos años. Mi motivación personal, a la hora de abordar este proyecto, es doble.

Por un lado, me interesa poder trabajar con bibliotecas de visión artificial, y adquirir practica y habilidad en la detección y resaltado de elementos de imágenes, principalmente por motivos laborales, ya que los procesos de reconocimiento automático se pueden aplicar en varios ámbitos, y este es un campo que no se ha tocado de manera directa en el máster de ciencia de datos hasta el momento.

Por otro lado, mi motivación es porque se puede trabajar con algoritmos que forman parte del area conocida como *Deep learning,* estos son, algoritmos que pueden utilizar múltiples capas de unidades de procesamiento no lineal, tales como redes neuronales.

Más concretamente, y de manera futura, se plantea este trabajo con el interés de poder utilizar, de manera conjunta, redes neuronales convolucionales, y redes neuronales recurrentes.

1.2. OBJETIVOS DEL PROYECTO

El objetivo principal propuesto de este trabajo es diseñar e implementar un modelo, mediante redes neuronales que nos permita ser usado para generar descripciones automáticas de imágenes.

Para poder llegar a este objetivo habrá que establecer una serie de objetivos relacionados:

- Analizar el estado del arte de este, es decir, la evolución que ha existido en la línea de investigación que se ha seguido y el estado actual en el que se encuentra.

- Desarrollo iterativo del modelo y de la evaluación de este.
- Documentación relativa al proceso empleado y a los resultados obtenidos.
- Documentación de presentación del trabajo.
- Publicación de la memoria de trabajo, y de la presentación en la plataforma *Open Access* de la UOC.

De manera opcional se establece un objetivo secundario de publicar un asistente, ya sea en una web, o en una red social como Twitter o Telegram, que realice descripciones de imágenes.

1.3. ENFOQUE Y MÉTODO SEGUIDO

En este trabajo se va a utilizar el modelo de procesos *CRISP-DM*, abreviatura de *Cross-industry standard process for data mining.* El motivo es que el uso de este estándar abierto permite establecer una metodología de trabajo iterativa, en la cual se va mejorando el modelo que se requiere para la funcionalidad del proyecto.

CRISP-DM consta de las siguientes fases:

- **Comprensión del negocio.** Donde se establecen los objetivos y requerimientos desde el punto de vista del negocio.
- **Comprensión de los datos**. Donde se capturan los datos y se establece familiaridad con los mismos para poder identificar los problemas.
- **Preparación de los datos.** Abarca todas las actividades necesarias para poder preparar el dataset final.
- **Modelado.** Es la fase en la que se aplican varias técnicas de modelado, y en la que se establecen los parámetros óptimos de las mismas.
- **Evaluación.** En esta fase se debe evaluar los modelos desde el punto de vista de los datos. Tras esta fase se debe llegar a una conclusión con los resultados.
- **Publicación.** Para poder obtener conocimiento de los datos es necesario que la información que se obtengan de ellos se organice y se presente de la manera adecuada. En este caso, se planteará hacer, como objetivo secundario, una web de consulta, o un asistente de Telegram o Twitter que etiquete imágenes.

CRISP-DM sigue un enfoque iterativo, existen varias fases en las que se aplican criterios iterativos para mejorar el proceso de obtención de información. Asimismo, tras la última fase se comprueba si los requerimientos se han validado, si no es así se implementa de nuevo el ciclo, tratando de mejorar la evaluación del modelo. Se puede apreciar en la siguiente figura el ciclo de vida de *CRISP-DM.*

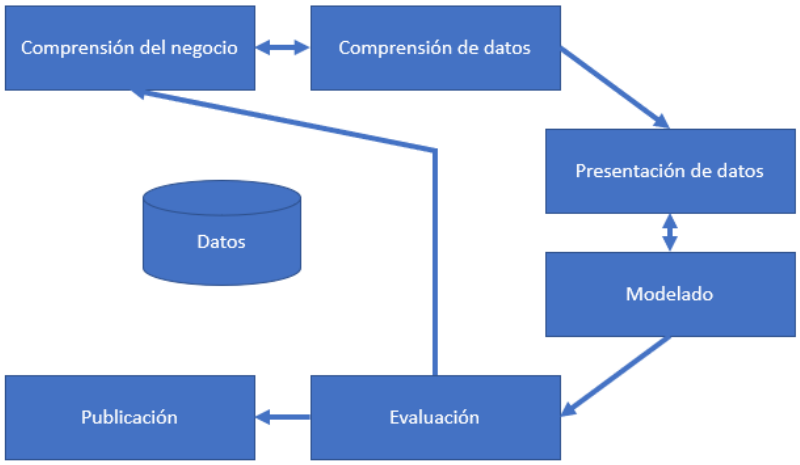


Figura 1. Ciclo de vida de CRISP-DM.

1.4. PLANIFICACIÓN DEL TRABAJO

El resumen de la planificación que se ha elaborado es la siguiente:

Fase	Fecha de fin	Descripción
Fase 1	30 de septiembre	Definición y planificación del trabajo final
Fase 2	21 de octubre	Estado del arte o análisis del mercado
Fase 3	16 de diciembre	Diseño e implementación del trabajo
Fase 4	6 de enero	Redacción de la memoria y de la presentación
Fase 5	13 de enero	Presentación y defensa del proyecto

La descomposición de las fases por detalles se realiza con la siguiente planificación:

FASE 1
Entrega: 30 de septiembre
Puntos que se deben realizar:

- Recogida de información relacionada con:
    - Metodologías de trabajo relacionadas con minería de datos.
    - Artículos relacionados de descripción automática de imágenes mediante redes neuronales.

Objetivos:

- Documentación relativa a la planificación y objetivos del trabajo final

FASE 2

Entrega: 21 de octubre.

Puntos que se deben realizar:

- Documentación de la evolución de los modelos relacionados con la descripción automática de imágenes, o el estado del arte.
- Obtención y preparación del dataset para las pruebas.
- Comienzo de las pruebas con redes neuronales y con reconocimiento de imágenes.
- Refinado de objetivos a cumplir, revisión de la planificación inicial.

Objetivos:

- Documentación relativa al análisis del estado del arte.
- Documentación modificada del plan de trabajo, con los objetivos parciales actualizados.

FASE 3

Entrega: 16 de diciembre.

Puntos que se deben realizar:

- Desarrollo del modelo del descriptor automático de imágenes.

Objetivos:

- Informe con el desarrollo de los modelos planteados.
- Código del modelo.

FASE 4

Entrega: 6 de enero

Puntos que se deben realizar:

- Desarrollo de la memoria, incluido el capítulo de evaluación de datos.

Objetivos:

- Memoria del trabajo.
- De manera optativa, integración del sistema en una web o en un asistente de una red social, como Twitter o Telegram.

FASE 5

Entrega: 13 de enero

Puntos que se deben realizar:

- Presentación del trabajo.
- Defensa del trabajo.
- Subida de los documentos relacionados con el mismo a la plataforma *Open Access* de la UOC.

Objetivos:

- Entrega final del trabajo en la plataforma *Open Access*.

CAPÍTULO 2. ANÁLISIS DEL ÁMBITO Y ESTADO DEL ARTE

2.1. TÉCNICAS PREVIAS AL DEEP LEARNING

Los algoritmos automáticos de descripción de imágenes, en ingles *image captioning,* han sufrido una evolución notable en la última década, a partir de la evolución del area conocida como *Deep learning*.

Para poder conseguir el objetivo de describir una imagen es necesario poder crear una sentencia que sea válida desde el plano semántico, y también precisa, desde el plano visual, esto implica que se deben correlacionar dos áreas aparentemente distintas, como son el entendimiento visual de la imagen, y el procesamiento lingüístico de los elementos que la componen. Es decir, se deben usar técnicas de visión artificial, y técnicas de procesamiento natural del lenguaje, combinadas.

Técnicas de descripción basadas en sustitución.

Estas técnicas partieron buscando las imágenes que son visualmente parecidas, junto con sus descripciones, del dataset de entrenamiento. Se forma un conjunto de imágenes candidatas de las que se escoge la imagen más parecida, así como su descripción

(Ordonez, et al, 2011)[4] utilizarón un dataset de un milllon de fotografias para poder establecer que fotografia se parecía a la que querian describir y asi poder usar su descripción. La comparación es semantica y se hace a traves de modelos de Markov.

Este metodo supone que siempre existe una imagen previa, que ya ha sido clasificada, que permite describir cualquier imagen que se desee, lo cual no es realista.

En una evolución de esta tecnica lo que se hace es utilizar las descripciones que ya existen para componer una nueva descripción. Más concretamente, (Gupta, et al, 2012)[5] usaron el toolkit NLP de Standford[6] para poder procesar las descripciones del dataset y crear una lista de frases para cada imagen.

En su metodo establecen que imágenes se parecen a la que estan procesando, y cuales son las descripciones relacionadas que son las mas relevantes. La descripción final se genera como una combinación de esas frases.

Estas técnicas generan frases gramaticalmente validas, pero en ocasiones no sirve ya que no tratan de determinar que objetos son los mas importantes de la imagen.

Técnicas de descripción basadas en plantillas.

Estas técnicas se basan en el uso de rellenar plantillas de frases, en esas plantillas hay huecos para describir los objetos, los atributos, o las acciones de la imagen. El proceso trata de determinar que objetos, atributos, y acciones existen en la imagen y los rellena.

(Yang, et al, 2011)[7] usan una tupla compuesta de cuatro términos (Nombre, Verbos, Escenas Preposiciones) para escribir la frase. Los autores establecen los objetos que existen en la imagen. A continuación, usan un modelo de lenguaje para establecer los verbos, las escenas, y las preposiciones. La tupla que tenga mayor probabilidad es seleccionada usando un modelo de Markov.

Estas técnicas pueden crear frases semánticamente validas, y donde existe coincidencias de objetos. Sin embargo, debido a las limitaciones de este método el desarrollo de descripciones sigue un esquema rígido, el cual a diferencia de las descripciones realizadas por humanos no se adapta a las características de la imagen.

2.2. TÉCNICAS BASADAS EN DEEP LEARNING

En el area conocida como *Deep Learning* ha habido un gran progreso desde el año 2012, el cual ha mejorado las técnicas de descripción automática de imágenes. Dentro de este area existen varias categorías de técnicas.

En este apartado se verá el uso por parte de varios autores de distintos tipos redes neuronales, la cual será ampliada en el capítulo 3.

- Redes neuronales convoluciones, o *CNN*. Son redes neuronales usadas en clasificación de imágenes.
- Redes neuronales recurrentes, o *RNN*. Son redes neuronales usadas para problemas relacionados con series temporales, o de traducción de idiomas.
- Redes *long-short term memory,* o *LSTM.* Son un tipo de red neuronal recurrente que usa unidades que evitan problemas de dependencias de larga duración.

Técnicas basadas en plantillas y sustitución ampliadas con *Deep Learning*.

Estas técnicas usan la metodología comentada en los capítulos anteriores, sustituyendo las herramientas de reconocimiento, tanto de imágenes, como de texto, por herramientas que pertenecen al area de *Deep Learning.*

(Karpathy, et al, 2014)[8] proponen un modelo para la sustitución bidimensional de imágenes y sentencias, usando un espacio combinado multimodal en el que se incluyen fragmentos de imágenes y de sentencias.

Para el reconocimiento de fragmentos de imágenes se utiliza una *RCNN* (*region convolutional neural network)*, y para el reconocimiento de fragmentos de frases se usa una red neuronal recurrente. Estos resultados se combinan en un espacio multimodal y posteriormente se evalua el par imagen-sentencia atribuyendo una puntuación en la evaluación.

Con el uso de las redes neuronales se observa que el rendimiento de las técnicas mejora significativamente. Sin embargo, utilizar estas redes dentro de las metodologías de sustitución o de plantillas no elimina las desventajas que se han comentado anteriormente.

Técnicas basadas en aprendizaje multimodal.

Hasta ahora se han visto métodos que tratan de detectar que objetos puede haber en la imagen y como relacionarlos con elementos de frases, pero basándose en técnicas de sustitución o de plantillas.

Un método multimodal puede dar más flexibilidad para crear las frases. En este modelo se extraen las características de la imagen, mediante una red neuronal convolucional. Posteriormente, se pasan las características de la imagen por un modelo de lenguaje, el cual realiza una correlación entre las características de la imagen y las palabras. Este modelo propondrá una predicción de palabras para crear la descripción.

Existe un artículo de Karpathy y Fei-Fei, (Karpathy, et al, 2015)[10], en el que se alinean caracteristicas de la imagen representadas a traves de una *RCNN*, con segmentos de frases representados por una red neuronal recurrente bidireccional. Esta información se envia a una arquitectura de red neuronal recurrente multimodal en la que se utilizan los alineamientos para poder obtener las descripciones candidatas. Este artículo tuvo cierta repercusión dentro de este area por sus buenos resultados.

(Mao, et al, 2015)[9] usaron una *CNN* para extraer objetos de la imagen. Lo usaron en combinación con una *RNN* para modelar diversas distribuciones de palabra, condicionadas a los objetos de la imagen, en un espacio multimodal donde se combinan ambas.

Técnicas basadas en encoder decoder

Este tipo de técnicas se basan en el éxito de las técnicas de traducción de idiomas, mediante redes neuronales encoder decoder. El problema de la descripción de imágenes sería equivalente al problema de traducción automática entre idiomas, donde la entrada es en este caso una imagen y la salida es la descripción de esta.

Se parte de dos redes, la codificadora que usa *CNN* y codifica la imagen. La red decodificadora que usa *RNN* y decodifica la imagen en una frase legible. En este tipo de planteamientos se suele utilizar una red *LSTM,* como decodificadora para generar las sentencias candidatas.

Existen varios trabajos prometedores relacionados con este area. Por ejemplo, existen trabajos de aprendizaje semisupervisado, es decir donde se tienen más imágenes sin descripciones que imágenes con descripciones.

(Pu, et al, 2016)[11] proponen un modelo de aprendizaje semisupervisado bajo la técnica encoder decoder. En este modelo usan una *CNN* para codificar imágenes, y un DGDN (*deep degenerative deconvolutional network)* para decodifcar las imágenes en descripciones.

(Jia, et al, 2015)[12] proponen extraer información semantica de las imágenes y añadir esta información a la red *LSTM* para poder generar las descripciones. Los resultados obtenidos en las metricas de evaluación por esta tecnica son muy prometedores.

Técnicas basadas en descripción guiada de imágenes

Una imagen está compuesta de un número elevado de elementos, muchos de ellos irrelevantes a la hora de describirla. Estas técnicas tratan de guiar la atención en los mecanismos encoder decoder para centrar la información en los aspectos más relevantes de la imagen.

(Xu, et al, 2017)[13] proponen un modelo encoder decoder guiado, el cual permite atender, de manera dinamica, regiones de imágenes. Para ello, se usara un vector de contexto, el cual nos permite penalizar con pesos diferentes regiones de la imagen. Esta técnica, junto con la de (Jia, et al, 2015)[12] obtiene los resultados mas interesantes, hasta el año 2017.

2.3. COMPARACIÓN DEL ESTADO DEL ARTE

Datasets existentes

- **MS COCO dataset.** Microsoft Common Objects in Context Dataset es un dataset masivo para reconocimiento de imagen, segmentación, y descripciones. Se compone de más de 300.000 imágenes, más de 2 millones de instancias, 80 categorías de objetos y 5 descripciones por imagen.

- **Flicker30k dataset.** Es un dataset para descripción automática de imágenes. Contiene más de 30 mil imágenes, con 158 mil descripciones suministradas por anotadores humanos.

- **Flicker8k dataset.** Es un dataset compuesto por 8000 imágenes, de las cuales el set de entrenamiento consta de 6000 imágenes, el set de validación consta de 1000 imágenes, y el set de evaluación de 1000 imágenes. Cada imagen contiene cinco descripciones anotadas por humanos.

- **Visual Genoma dataset.** Visual Genoma dataset consta de diferentes descripciones para múltiples regiones de la imagen. El dataset se descompone en siete regiones: descripciones de la región, objetos, atributos, relaciones, grafos de región, escenas de región, y pares pregunta respuesta. Este dataset consta de 108 mil imágenes. Cada imagen consta de 35 objetos y de 26 atributos.

Métricas de evaluación

Todas las métricas de evaluación que se verán se basan en la búsqueda de n-gramas. Un n-grama es una subsecuencia de n elementos que se da dentro de una secuencia.

- **BLEU.** Acrónimo de *BiLingual Evaluation Understudy*, es una métrica que permite evaluar la calidad del texto generado de manera automática. Se basa en comparar secuencias de n-gramas entre la frase candidata y las frases de referencia.

  Es una métrica popular porque fue pionera en la evaluación de texto generado de manera automática, y tiene una correlación considerable con el texto generado por humanos.

  Esta métrica genera un corpus separado por niveles de n-gramas, el cual representa la precisión entre las sentencias con la siguiente formula:

$$CP_n = \frac{\sum_i \sum_k min(h_k(c_i), max(h_k(s_{ij}))_{j \in m}}{\sum_i \sum_k h_k(c_i)}$$

Donde k define el conjunto de posibles n-gramas de longitud n. Esta métrica limita el número de apariciones de n-gramas mediante la siguiente formula:

$$b(C,S) = \begin{cases} 1 \; si \; l_c > l_s \\ e^{1-\frac{l_s}{l_c}} \; si \; l_c \leq l_s \end{cases}$$

Donde l_c es la longitud de las secuencias candidatas y l_s es la longitud de las sentencias del corpus. La puntuación de BLEU se computa mediante la media geométrica de la precisión entre las sentencias.

$$BLEU_N(C,S) = b(C,S) \exp\left(\sum_{n=1}^{N} \omega_n log CP_n(C,S\right)$$

Donde *N= 1,2,3,4* y representa cada una de las cuatro categorías de BLEU, las cuales dependen de la longitud de los n-gramas.

BLEU da una métrica de la precisión, es decir, una medida de cuantas veces aparece los n-gramas de la frase candidata en las frases de referencia.

- **ROUGE.** Acrónimo de *Recall-Oriented Understudy for Gisting Evaluation*. Es un conjunto de métricas usadas para medir la calidad del texto. Compara secuencias de palabras y pares de palabras con un conjunto de referencias creadas por humanos.

  ROUGE da una medida del *recall,* es decir, una medida de cuantas veces aparece los n-gramas de las frases de referencia en la frase candidata.

  Estos son las métricas relacionadas con ROUGE.

  $ROUGE_N$ muestra la relación en un n-grama entre los textos de referencias y el texto candidato.

$$ROUGE_N(c_i, S_i) = \frac{\sum_j \sum_k \min(h_k(c_i), h_k(s_{ij}))}{\sum_j \sum_j h_k(s_{ij})}$$

$ROUGE_L$ es una medida basada en la subsecuencia común más larga, o *LCS*. Este es un conjunto de palabras que ocurre en dos sentencias distintas, en el mismo orden.

$$ROUGE_L(c_i, S_i) = \frac{(1+\beta^2)R_l P_l}{R_l + \beta^2 P_l}$$

Donde:

$$R_l = max_j \frac{l(c_i, s_{ij})}{|s_{ij}|}$$
$$P_l = max_j \frac{l(c_i, s_{ij})}{|c_i|}$$

- **METEOR.** Acrónimo de Metric for Evaluation of Translation with Explicit ORdering. Es una métrica usada para evaluar la calidad del idioma traducido de manera automática.

  Se realiza una alineación entre las palabras del candidato y las frases de referencia, buscando una correspondencia 1:1. Esta alineación se calcula mientras se minimiza el número de bloques de pares idénticos y ordenados.

  Para establecer la alineación se usa condiciones de coincidencia exacta, o se buscan sinónimos en WordNet[14], una base de datos léxica en ingles realizada por la Universidad de Princeton.

  Dado un conjunto de alineaciones *m*, la métrica METEOR es la media armónica de precisión P_m y *recall,* R_m, entre la mejor puntuación de referencia y el candidato.

$$F_{mean} = \frac{P_m R_m}{\propto P_m + (1-\propto) R_m}$$

$$P_m = \frac{|m|}{\sum_k h_k(c_i)}$$
$$R_m = \frac{|m|}{\sum_k h_k(s_{ij})}$$

- **CIDEr$_n$.** Acrónimo de *Consensus based Image Description Evaluation*. Es una métrica de consenso automático para evaluar descripciones de imágenes. Compara segmentos de palabras con textos de referencia. Consigue llegar al consenso mediante el cálculo de la frecuencia inversa de documento *TF-IDF*, que en este caso es $g_k(s_{ij})$, por cada n-grama ω_k.

$$g_k(s_{ij}) = \frac{h_k(s_{ij})}{\sum_{\omega \in \Omega} h_l(s_{ij})} log\left(\frac{|I|}{\sum_{I_p \epsilon I} \min(1, \sum_q h_k(s_{pq}))}\right)$$

Donde Ω es el vocabulario de n-gramas *e I* es el conjunto de imágenes del dataset. El primer término representa la frecuencia inversa, *TF*, y el segundo la rareza, o *IDF.* El primer término es más elevado en combinaciones frecuentes, y el segundo equilibra el peso del n-grama.

Para determinar la métrica de *CIDEr$_n$* hay que determinar la similitud de coseno promedio entre la frase candidata y las frases de referencia.

$$CIDEr_n(c_i, s_{ij}) = \frac{1}{m} \sum_j \frac{g^n(c_i) g^n(s_{ij})}{\|g^n(c_i)\| \|g^n(s_{ij})\|}$$

Donde $g^n(c_i)$ es un vector formado por $g_k(c_i)$, lo que corresponde a los n-gramas de longitud n y $\|g^n(c_i)\|$ es la magnitud del vector.

Comparativa de datasets y de métricas de evaluación

En este apartado se verán cuáles son las evaluaciones obtenidas en las métricas de los estudios comentados de los capítulos anteriores.

Las métricas que han sido utilizadas son:

- B1 para BLEU-1.
- B2 para BLEU-2.
- B3 para BLEU-3.
- MT para METEOR.

En negrita se marcan los mejores resultados de cada métrica. Cuanto más cercano este el resultado a uno, la métrica nos indica que el modelo se comporta mejor.

Resultados para Flickr8k

Categoría	Método	Flickr8k				
		B-1	B-2	B-3	B-4	MT
Aprendizaje multimodal	(Karpathy, et al, 2015)[10]	0.579	0.383	0.245	0.160	
	(Mao, et al, 2015)[9]	0.565	0.386	0.256	0.170	
Encoder decoder	(Jia et al, 2015)[12]	0.647	**0.459**	0.318	**0.216**	**0.202**
Atención guiada	(Xu, et al, 2017)[13]	**0.670**	0.457	**0.319**	0.213	

En este dataset se puede apreciar, en el caso de BLEU, que los trabajos de (Jia et al, 2015)[12] y de (Xu, et al, 2017)[13] monopolizan los mejores resultados con una ligera diferencia entre ambos.

Es lógico que el trabajo de Xu, del 2017 deba tener mejores resultados, ya que el modelo de atención guiada fue diseñado con la idea de mejorar los modelos previos y de focalizar la atención a las areas mas interesantes de la imagen.

Sin embargo, no hay una diferencia apreciable con el modelo de Jia, del 2015. El motivo es que en su base el modelo de Xu es un modelo basado en la arquitectura encoder-decoder, la cual ha encontrado su mejores resultados en estos valores.

Resultados para Flickr30k

Categoría	Método	Flickr30k				
		B-1	B-2	B-3	B-4	MT
Aprendizaje multimodal	(Karpathy, et al, 2015)[10]	0.573	0.369	0.240	0.157	
	(Mao, et al, 2015)[9]	0.600	0.410	0.280	0.190	
Encoder decoder	(Jia et al, 2015)[12]	0.646	0.446	0.305	0.206	**0.179**
Atención guiada	(Xu, et al, 2017)[13]	**0.670**	**0.457**	**0.314**	**0.213**	

En Flickr30k el dataset aumenta hasta las 30.000 imágenes. Es en este dataset donde se puede apreciar que el modelo de Xu, de 2017 mejora al modelo de Jia, del 2015. El motivo es que el modelo de Jia se comporta ligeramente peor que en el caso anterior.

Sin embargo, la diferencia no es excesiva. En el caso de BLEU-1, se pasa de 0.646 a 0.670, eso quiere decir que solo se aprecia la diferencia en 24 imágenes de cada mil. Si se observa BLEU-4, que es la métrica más interesante porque es la mayor, la diferencia pasa a ser de 7 imágenes de cada mil. En la práctica la calidad de las descripciones candidatas seguirá siendo muy parecida a la del modelo de 2015.

Resultados para MSCOCO

Categoría	Método	MSCOCO				
		B-1	B-2	B-3	B-4	MT
Aprendizaje multimodal	(Karpathy, et al, 2015)[10]	0.625	0.450	0.321	0.230	0.195
	(Mao, et al, 2015)[9]	0.670	0.490	0.350	0.250	
Encoder decoder	(Jia et al, 2015)[12]	0.670	0.491	**0.358**	**0.264**	0.227
Atención guiada	(Xu, et al, 2017)[13]	**0.718**	**0.504**	0.357	0.250	**0.230**

En el dataset MSCOCO existen 300.000 imágenes catalogadas. Aquí se puede apreciar una mejora de los modelos de Jia y de Xu en todas las métricas, con respecto a los valores del dataset de Flickr.

En el caso de Xu su precisión es mayor en BLEU-1 y en BLEU-2, en el caso de Jia el rendimiento es superior en BLEU-3 y BLEU-4. Se puede establecer el criterio de que BLEU-4 es la métrica más interesante, ya que contiene el mayor n-grama, por ello se podría considerar que las técnicas basadas en encoder-decoder aún no han sido superadas por las basadas en atención guiada, aunque es posible que sea una cuestión de tiempo.

Si se ampliasen los resultados incluyendo técnicas anteriores al año 2012, como las basadas en sustitución, se vería un salto considerable en los valores obtenidos. El uso de técnicas basadas en *Deep Learning* ha permitido mejorar los resultados en las evaluaciones.

2.4. INVESTIGACIONES FUTURAS POSIBLES

La descripción automática de imágenes es un campo en desarrollo constante donde se generan nuevas técnicas para mejorar su rendimiento. Dentro de este campo hay áreas donde es posible que aparezcan nuevas líneas de investigación:

- Las descripciones que se han realizado hasta ahora se focalizan en descripciones generales de la imagen, no se centran en los objetos de esta. Existe un artículo, de (Karpathy, et al, 2015)[15] acerca de *DenseNet* donde se analiza la importancia del análisis de imágenes con relación a los objetos y a las regiones.

  Es decir, para describir imágenes a un nivel humano y que estas descripciones sean aplicables a entornos y contextos de la vida real es necesario que estén fundamentadas en los objetos y regiones de la imagen.

- La investigación de las descripciones de imágenes mediante aprendizaje no supervisado es un area prometedora donde los resultados son muy esperados, ya que existe una limitación en los datasets etiquetados.

  Existen artículos basados en aprendizaje semisupervisado como el de (Pu, et al, 2016)[11] que tratan de establecer esta linea de trabajo.

- Estudio de imágenes de un campo concreto. Hasta ahora las técnicas que se han visto estudian imágenes generalistas. Es decir, no están especializadas en un campo, como puede ser la medicina.

  Existen estudios realizados para poder describir imágenes médicas, como el realizado por (Baoyo, et al, 2017)[16], que muestra la siguiente arquitectura basada en un modelo encoder-decoder.

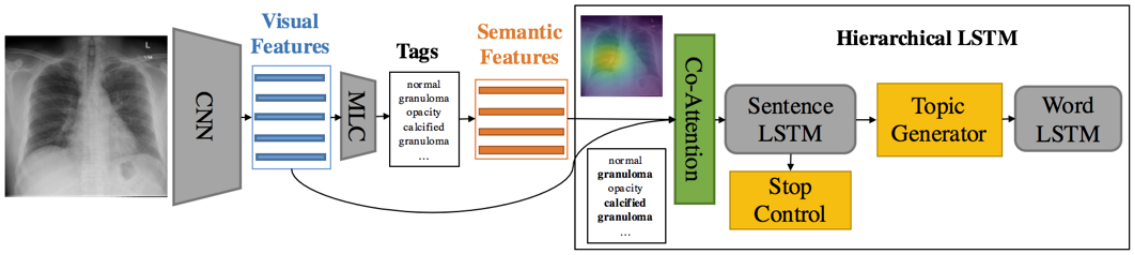


Figura 2. Arquitectura de un descriptor de imagenes medico.[16]

CAPÍTULO 3. ARQUITECTURA DEL MODELO

3.1. MODELO ENCODER-DECODER

El problema de la descripción de imágenes es equivalente al problema de traducción automática entre idiomas, donde la entrada es en este caso una imagen y la salida es la descripción de esta imagen.

Se utilizará un modelo basado en la técnica Encoder-Decoder como la planteada por (Vinyals, et al, 2016)[17] en su paper *Show and Tell: Lessons learned from the 2016 MSCOCO Image captioning Challenge.*

Este modelo coge como una entrada una imagen *I* y la entrena para maximizar la probabilidad $p(S|I)$ donde *S* es una secuencia de palabras generadas desde el modelo, y cada palabra ha sido creada en el entrenamiento.

La imagen *I* se pasa por una red convolucional neuronal, *CNN*, la cual ayuda a extraer las características más relevantes de la imagen. Estas características se pasan a una red neuronal recurrente, *RNN*, la cual ayuda a crear una sentencia de palabras asociadas a la imagen.

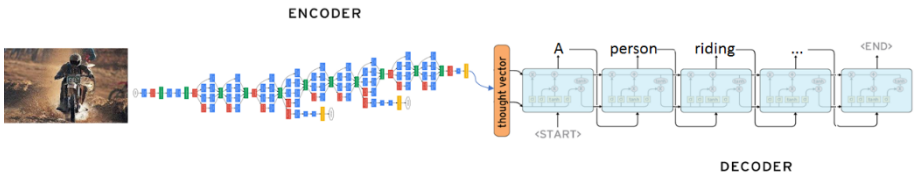


Figura 3. Esquema Encoder-Decoder.[19]

3.2. MODELO DE MEZCLA DE CARACTERÍSTICAS

La técnica Encoder-Decoder admite varios tipos de modelos que permiten la integración entre el Encoder y el Decoder.

El modelo de mezcla, o *Merge*, combina la forma codificada de la imagen con la forma codificada de las descripciones de texto. La combinación de ambas es usada por la capa Decoder para poder generar la descripción de la imagen.

Este modelo de arquitectura es el propuesto por (Tanti, et al, 2017)[18] en su paper *Where to put the Image in an Image Caption Generator*, en el cual comparan este modelo con otra alternativa, el modelo de inyección.

En su opinión "La arquitectura de mezcla tiene ventajas practicas, ya que el condicimiento por mezcla provoca que el vector de celda de estados de las *RNN* disminuye en cuatro veces su tamaño".

Esta es la figura del modelo planteado

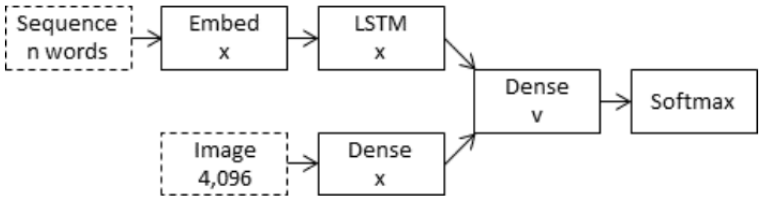


Figura 4. Arquitectura del modelo de mezcla usado. [18]

3.3. RED CONVOLUCIONAL NEURONAL

Las redes convolucionales neuronales, abreviadas como *CNN*, son un tipo de red neuronal que ha tenido un gran éxito en el campo de clasificación de imágenes, gracias al alto rendimiento de las GPU actuales.

En 1989 Yann LeCun realizo un sistema para reconocer números de códigos postales escritos a mano, mediante convoluciones, y mediante el algoritmo de *back-propagation.*

Existen cuatro elementos que caracterizan la arquitectura de una red neuronal convolucional.

1. El operador convolucional.
2. El operador agrupacional, también llamado *pooling.*
3. La capa de activación. Utiliza la función no lineal ReLU
4. La capa de clasificación, llamada *fully-connected layer.*

Estas operaciones son los bloques básicos que componen la red, por lo que entender su funcionamiento es importante para poder entender las *CNN*. Por ello se verán con más detalle.

Operador convolucional

La convolución es una operación matemática en la cual se extraen las características de la imagen. Para ello se utiliza un filtro, de dimensiones inferiores a las de la imagen, que se aplica por todos los puntos de esta para obtener un valor resultante con el que se crea una nueva matriz.

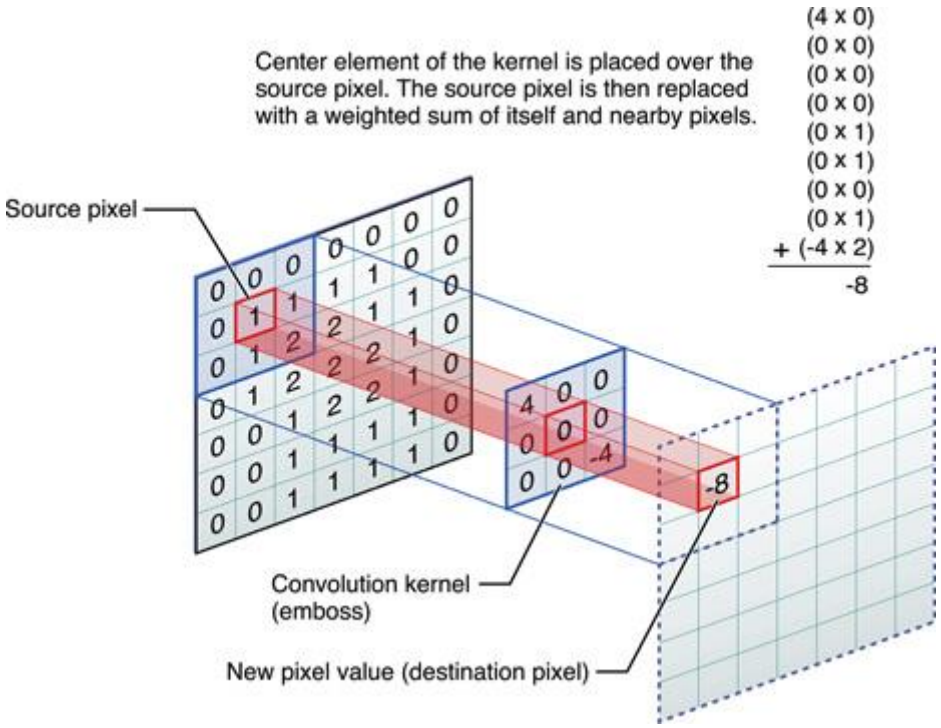


Figura 5. Ejemplo de operación convolución. [19]

Pooling

Esta operación reduce la dimensionalidad de la imagen preservando las características más importantes. Hay varios tipos de *pooling*. El más usado es el máximo por región, en el cual se reduce la dimensionalidad por regiones, quedándose con el valor máximo de cada región.

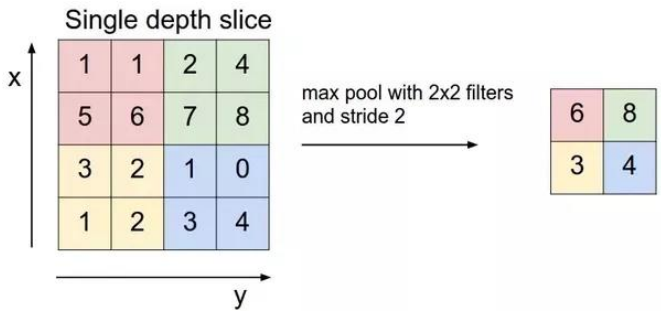


Figura 6. Ejemplo de pooling. [19]

Función de activación no lineal

Esta función de activación se aplica tras cada operación de convolución, y es la encargada de determinar cuándo se debe activar la salida de la neurona de una red.

Existen varias funciones de activación, como *sigmoid, tanh,* o *ReLU.* La función de activación más usada es *ReLU,* acrónimo de *Rectified Linear Unit*, esta es su función:

$$f(x) = \max(0, x)$$

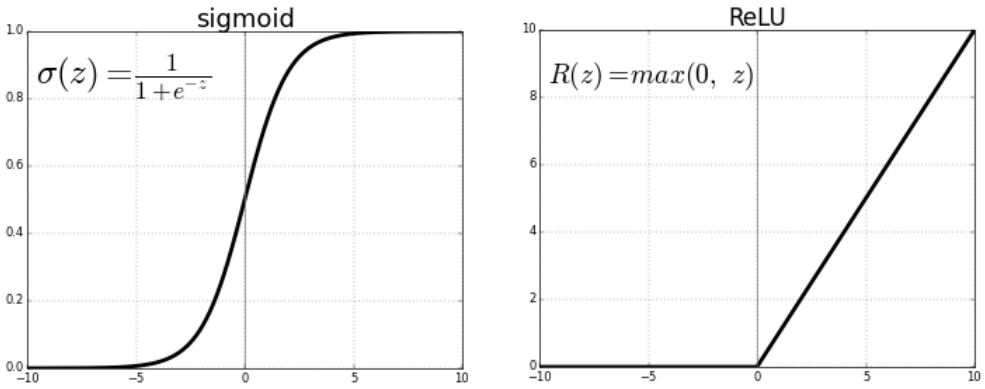


Figura 7. Comparación de operación sigmoide y ReLU. [19]

En el algoritmo de *back-propagation* es necesario calcular el gradiente de la función de activación para propagar el error. En el caso de *ReLu* el gradiente es 1 si el valor es positivo y 0 en otro caso, lo cual simplifica mucho el cálculo. Además, si la función no está activada el error no se propagará hacia atrás.

Fully-connect layer

Esta es la capa de clasificación de la salida, la cual utiliza una función de activación *SoftMax*. Se le llama como plenamente conectada porque todas las neuronas de la etapa previa están conectadas a todas las neuronas de esta etapa. En esta etapa se realiza la clasificación mediante el mapeado de las características que se han extraído en las capas anteriores.

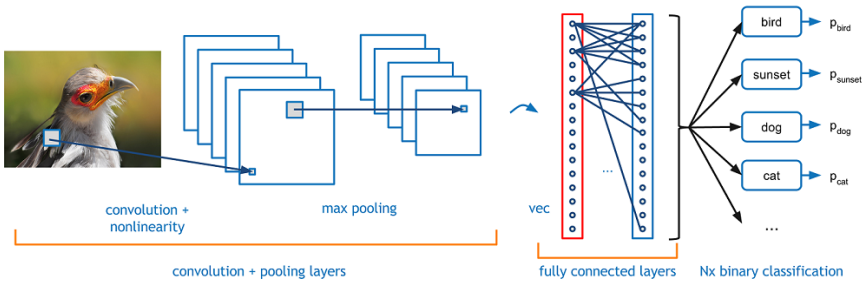


Figura 8. Ejemplo de fully connected layer. [19]

Modelos entrenados

Para facilitar la implementación se utilizarán los pesos de un modelo entrenado previamente, más concretamente el VGG16. Este es un modelo de 16 capas que fue utilizado por el equipo VGG en la competición ILSVRC-2014, y que quedo entre los cinco primeros.

3.4. RED RECURRENTE NEURONAL.

Las redes recurrentes neuronales, o *RNN*, son un tipo de redes neuronales en el que las conexiones entre nodos forman un grafo directo a lo largo de una secuencia. Esto les permite establecer un comportamiento dinámico a partir de una secuencia temporal. Fueron descubiertas por John Hopfield en 1982.

Son utilizadas tanto en reconocimiento de voz y escritura, como en traducción de idiomas. Como se puede apreciar en la siguiente figura, con cada nueva palabra de la secuencia, la red se va ampliando.

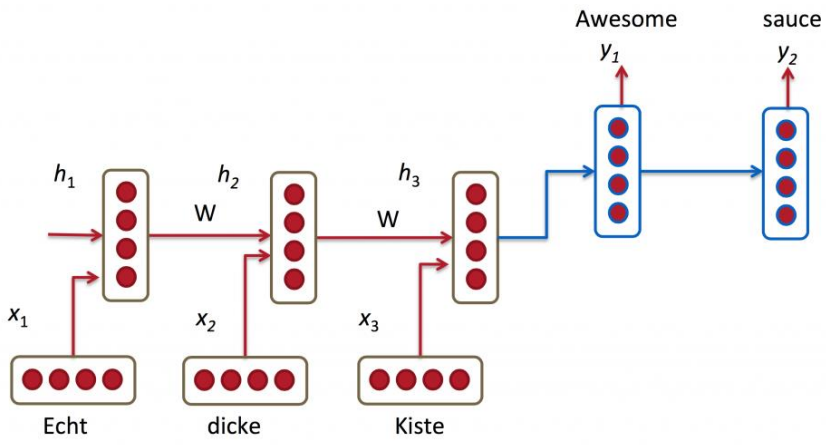


Figura 9. Ejemplo de red recurrente neuronal. [19]

Las *RNN* no tienen en cuenta las dependencias a largo plazo. Si la secuencia es lo suficientemente larga puede ocurrir que las palabras que más peso deban tener en la predicción hayan sido olvidadas por la *RNN.* Son muy sensibles a la longitud del intervalo.

Una solución a este problema viene gracias a las *long short-term memory,* o *LSTM.* Estas son redes neuronales recurrentes que recuerdan valores sobre intervalos de tiempo aleatorios, gracias a una celda de estado.

Estas redes son las que han propiciado el uso de *RNN* para reconocimiento de voz y escritura, así como para ser usadas en traducciones de idiomas, o incluso en problemas de series de tiempos.

Este es el esquema de una unidad de una red *LSTM*.

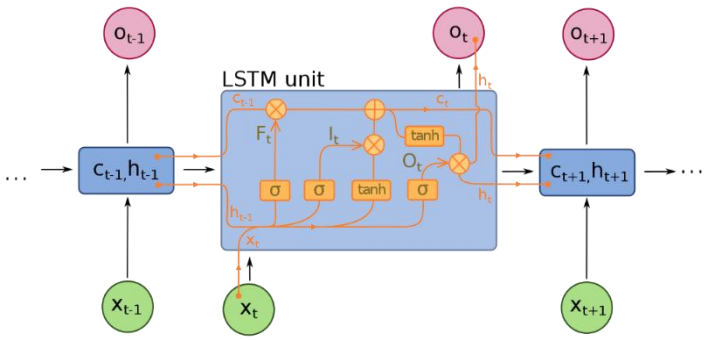


Figura 10. Unidad de red LSTM. [19]

Una unidad de red *LSTM* consta de dos bloques:

- Un módulo que contiene varios elementos:

    - Una puerta de entrada.
      Caracterizada por la función $i_t = \sigma(W_{ix} \circledast x_t + W_{im} \circledast m_{t-1})$,
      donde W representa el peso, m_{t-1} es la salida del bloque anterior en el instante $t-1$, y σ es el operador sigmoide.

    - Una puerta de olvido.
      Caracterizada por la función $f_t = \sigma(W_{fx} \circledast x_t + W_{fm} \circledast m_{t-1})$,
      donde f_t representa la puerta de olvido en el instante t.

    - Una puerta de salida.
      Caracterizada por la función $o_t = \sigma(W_{ox} \circledast x_t + W_{om} \circledast m_{t-1})$,
      donde o_t representa la puerta de salida en el instante t.

- Una celda de estado. La cual insensibiliza al *LSTM* del intervalo entre palabras.

  Esta caracterizada por la función $c_t = f_t \odot c_{t-1} + i_t \odot h(W_{cx}x_t + W_{cm}m_{t-1})$,

- Un bloque de activación. El bloque final es *SoftMax*. Este bloque normalizara la salida en el rango [0,1].

A la salida del bloque *SoftMax* se tendrá la predicción de la siguiente palabra. La red *LSTM* continuara hasta que encuentre un token finalizador, que en este caso se representa con la cadena “endcap“. Las palabras que se han producido representan la frase generada para una imagen.

CAPÍTULO 4. IMPLEMENTACIÓN DEL MODELO

4.1. DATASET

El dataset que se utilizará es *Flickr8k*. Este dataset se compone de 8000 imágenes, cada una con cinco descripciones. El dataset se encuentra organizado en 2 carpetas, una para imágenes y otra para textos. En esta última se pueden encontrar los siguientes ficheros:

- *Flickr8k.token.txt.* Contiene un identificador de imagen y cada una de las cinco descripciones de esa imagen.
- *Flickr_8k.trainImages.* Contiene los identificadores de 6000 imágenes que han sido divididos para ser usados como dataset de entrenamiento.
- *Flickr_8k.devImages.txt*. Contiene los identificadores de 1000 imágenes que han sido divididos para ser usados como dataset de desarrollo.
- *Flickr_8k.testImages.txt*. Contiene los identificadores de 1000 imágenes que han sido divididos para ser usados como dataset de evaluación.

Este dataset es una elección ideal por su no muy elevado tamaño, en comparación con otros como *Flicker30k* o *MSCOCO*, que tienen más de 30.000 imágenes, y supondrían un incremento exponencial en el tiempo de computación de las pruebas del modelo.

Si lo desea puede solicitar permisos para descargar el dataset desde esta dirección:

https://forms.illinois.edu/sec/1713398

4.2. CÓDIGO

La implementación del código se puede consultar y descargar online desde la siguiente dirección de GitHub. No se pueden incluir los modelos generados por limitaciones de tamaños de archivos subidos a GitHub.

https://github.com/Bengis/Let-me-see

En este apartado se verá la implementación realizada del modelo. El modelo de entrenamiento se divide en cuatro partes de código diferenciadas:

- Preprocesado. En este código se limpian las descripciones y se crean los conjuntos de entrenamiento, validación y evaluación.

- Encoder. El proceso de codificación de la red *CNN*, puede usar VGG o Inception V3 para extraer las características de las imágenes.
- Decoder. En este código se realiza la conversión de frases a secuencias de números, la mezcla con las características de la imagen extraídas anteriormente, y la creación del decoder que predecirá la frase candidata.
- Métricas. En este código se implementan las métricas BLEU y ROUGE que nos permiten evaluar la precisión y el *recall*, respectivamente, de las frases candidatas.
- Entrenamiento. En este código se lanza el entrenamiento y la evaluación para revisar en cada *epoch* cuál es el resultado obtenido y, si es necesario salirse mediante *EarlyStopping.*
- Integración con Telegram. En este código se realiza el objetivo opcional de integrar el modelo dentro de una red social, o web, que permita usarlo con nuevas imágenes.

4.3. PREPROCESADO

En este código, que se puede encontrar en el archivo “code/preprocessing.py”, se crea y limpian los dataset de entrenamiento y de evaluación. Se obtienen como resultados los siguientes ficheros

- TrainImages.txt. Descripciones del dataset de entrenamiento.
- TestImages.txt. Descripciones del dataset de evaluación.
- DevImages.txt. Descripciones del dataset de validación.
- TokenImages.txt. Descripciones del dataset completo.

Estas descripciones se han limpiado eliminando los puntos, los números, los espacios en blanco, y poniendo las descripciones en minúsculas. A continuación, se muestran los aspectos más relevantes del ´código.

En primer lugar, se añaden a todas las descripciones unas coletillas para indicar donde comienza la frase, con “startcap” y donde acaba con “endcap”.

```
for i in range(len(token)-1):
    id_capt = token[i].split("\t")
    id_capt[0] = id_capt[0][:len(id_capt[0])-2]
    if id_capt[0] not in captions:
        captions[id_capt[0]]=list()
    captions[id_capt[0]].append("startcap "+clean_caption(id_capt[1])+" endcap")
```

Se crea el dataset de entrenamiento, con aproximadamente el 75% de las imágenes, el dataset de validación, con el 12,5% de las imágenes, y el dataset de evaluación con el 12,5% restante de las imágenes.

```
file_train_id = open('../data/Flickr8k_text/Flickr_8k.trainImages.txt','r')
train_id=file_train_id.read().split('\n')[:-1]
train_captions=dict()
for key, value in captions.items():
    if key in train_id:
        if key not in train_captions:
            train_captions[key]=value

file_train_captions = open("../result/trainImages.txt",'w')
for tr_key, tr_value_list in train_captions.items():
    for tr_value in tr_value_list:
        file_train_captions.write(str(tr_key)+'\t'+str(tr_value)+'\n')
        file_train_captions.flush()
file_train_captions.close()
```

Asimismo, se limpian las descripciones con las siguientes condiciones:

- Se eliminan los dígitos y los signos de puntuación.
- Se ponen todos los caracteres en minúsculas.
- Se eliminan todos los caracteres que consten de una única letra.
- Se descartan los espacios en blanco que queden al comienzo y al final de las frases.

```
def clean_caption(description):
    description=description.lower()
    description=re.sub(r'\b\w\b', '', description)
    description=re.sub(r'\s+', ' ', description)
    description=re.sub('\d+', '',description)
    description=re.sub(r'[^a-zA-Z ]', '',description)
    description=" ".join(description.split())
    return description
```

4.4. ENCODER

En este código, que se puede encontrar en el archivo “code/encoder.py”, se generan y extraen las características de la imagen, usando para ella una red *CNN*. Se puede usar una opción de entre los siguientes modelos, lo cual ahorrará el tiempo de cálculo de la *CNN.*

- VGG-16. Modelo de 16 capas, creado por el equipo VGG, fue uno de los cinco primeros en la competición *ILSVRC-2014*. Los datos de entrada de la red son las imágenes de ambos datasets, que se redimensionan al tamaño 224*224. Los

datos de salida se extraen de la penúltima capa del modelo VGG16, lo que proporciona un array de 4096 dimensiones.

- Inception v3 es una evolución del modelo usado en la *ILSVRC-2012*, donde fue uno de los cinco primeros. En este caso el tamaño de las imágenes se redimensiona a 299*299. Los datos de salida se extraen de la penúltima capa lo que nos da un array de 2048 dimensiones.

Estos datos se guardan en un diccionario con el nombre de imagen como clave. Este diccionario se guarda en el fichero *"image-encodings.p"*. Estos son los aspectos más relevantes del código.

En primer lugar, se selecciona el tipo de modelo que se va a usar, ya sea VGG o Inception.

```
tModel=1 #0:VGG,1:InceptionV3
model= encode_image(tModel)
features=extract_features(tModel,model)
print('Extracted Features: %d' % len(features))
if tModel==0:
   dump(features, open('../result/image_encodings_vgg.p', 'wb'))
if tModel==1:
   dump(features, open('../result/image_encodings_inception.p', 'wb'))
```

A continuación, se crea una lista donde se almacenarán las características de las imágenes, esa lista se nutrirá con imágenes del dataset de entrenamiento, validación y evaluación.

```
   file_train = open("../data/Flickr8k_text/Flickr_8k.trainImages.txt")
   train_id=file_train.read().split('\n')[:-1]
   print("Train length:" + str(len(train_id)))

   print("Dev begin")
   file_dev = open("../data/Flickr8k_text/Flickr_8k.devImages.txt")
   dev_id=file_dev.read().split('\n')[:-1]
   file_dev.close()
   print("Dev length:" + str(len(dev_id)))

   print("Dev end")
   file_test = open("../data/Flickr8k_text/Flickr_8k.testImages.txt")
   test_id=file_test.read().split('\n')[:-1]
   file_test.close()
   print("Test length:" + str(len(test_id)))

   images = []
   images.extend(train_id)
   images.extend(dev_id)
```

```
    images.extend(test_id)
    print("Images length:" + str(len(images)))
```

Posteriormente, se extraen las características de cada imagen. En el caso de VGG se redimensionan a 224x224, en el caso de Inception se realizará la redimensión a 299x299. Estos tamaños se usan para adaptarlos a los que usan estos modelos.

```
def encode_process_VGG16(model, path):
    processed_img = image.load_img(path, target_size=(224,224))
    x = image.img_to_array(processed_img)
    x = np.expand_dims(x, axis=0)
    x = vgg16.preprocess_input(x)
    image_final = np.asarray(x)
    prediction = model.predict(image_final)
    return prediction

def encode_process_InceptionV3(model, path):
    processed_img = image.load_img(path, target_size=(299,299))
    x = image.img_to_array(processed_img)
    x = np.expand_dims(x, axis=0)
    x = inceptionv3.preprocess_input(x)
    image_final = np.asarray(x)
    prediction = model.predict(image_final)
    return prediction
```

4.5. DECODER

En este código, que se puede localizar en el fichero “code/decoder.py”, se han agrupado tres tareas del modelo:

- Convertir las secuencias de texto en números y codificar la red *LSTM*.
- Mezclar los datos de salida de la red *LSTM*, con los datos extraídos de la red *CNN*.
- Establecer el decoder que se encarga de predecir las palabras asociadas a la imagen introducida.

Para ello se usará un modelo de decoder que esta alimentado con un generador de datos, el cual da en cada paso una secuencia con la siguiente tupla:

- Una imagen.
- La descripción asociada a esa imagen, hasta la palabra anterior a la palabra que se quiere predecir.
- La palabra que se quiere predecir.

Para el caso de la descripción *"brown dog running on the beach"* queda la siguiente secuencia.

input_1	input_2	out
Image	-start-	brown
Image	-start, brown	dog
Image	-start-, brown, dog	running
Image	-start-, brown, dog, running	on
Image	-start-, brown, dog, running, on	the
Image	-start-, brown, dog, running, on	beach
Image	-start-, brown, dog, running, on, beach	-end-

Este son las partes más relevantes del código.

En primer lugar, se cargan y preparan los datos. En la preparación se utilizará un *tokenizer,* es decir, una función para demarcar y clasificar las frases. Eso generara dos variables importantes:

- Word_index. Es el índice de palabras que se van a utilizar.
- Vocab_size. Es el número de palabras distintas que existen.

```
def prepare_data(self):
    self.steps=len(self.train_captions)
    self.num_samples=0

    caption_length=[]
    for key, caption_list in self.train_captions.items():
        for caption in caption_list:
            self.num_samples+=len(caption.split())-1
            caption_length.append(len(caption.split()))

    self.max_length = max(caption_length)
    self.tokenizer=self.create_tokenizer()
    self.vocab_size = len(self.tokenizer.word_index)+1

    print("Length captions = " + str(len(self.train_captions)))
    print("Max length = " + str(self.max_length))
    print("Vocab size = " + str(self.vocab_size))
```

Las frases de referencia del dataset de entrenamiento se convierten en secuencias de números. Dependiendo de la cantidad de memoria que haya en el sistema se podrán convertir todas, o será necesario hacerlo de uno en uno utilizando un generador de datos.

```
def sequence_all(self, descriptions):
    in_1, in_2, out = list(), list(), list()
    for key, desc_list in descriptions.items():
        for desc in desc_list:
            seq = self.tokenizer.texts_to_sequences([desc])[0]
            for i in range(1, len(seq)):
                in_seq, out_seq = seq[:i], seq[i]
                in_seq = pad_sequences([in_seq], maxlen=self.max_length)[0]
                out_seq = to_categorical([out_seq], num_classes=self.vocab_size)[0]
                in_1.append(self.image_encodings[key][0])
                in_2.append(in_seq)
                out.append(out_seq)
    return np.array(in_1), np.array(in_2), np.array(out)

 def sequence_one(self, desc_list, img):
    in_1, in_2, out = list(), list(), list()
    for desc in desc_list:
        seq = self.tokenizer.texts_to_sequences([desc])[0]
        for i in range(1, len(seq)):
            in_seq, out_seq = seq[:i], seq[i]
            in_seq = pad_sequences([in_seq], maxlen=self.max_length)[0]
            out_seq = to_categorical([out_seq], num_classes=self.vocab_size)[0]
            in_1.append(img)
            in_2.append(in_seq)
            out.append(out_seq)
    return np.array(in_1), np.array(in_2), np.array(out)
```

Se define el modelo. Es necesario recordar que se está usando un modelo de mezcla donde se combinan las imágenes y las secuencias de texto obtenidas de la red *LSTM.*

Este modelo tiene una serie de parámetros que son configurables como se vio en el apartado de ajustes. El *learning rate,* el *dropout*, el número de salidas de la red *LSTM* y el número de *epochs.*

```
 def model(self, lr, dr, lstm):
    inputs1 = Input(shape=(4096,))
    fe1 = Dropout(dr)(inputs1)
    fe2 = Dense(lstm, activation='relu')(fe1)

    inputs2 = Input(shape=(self.max_length,))
    se1 = Embedding(self.vocab_size, lstm, mask_zero=True)(inputs2)
    se2 = Dropout(dr)(se1)
    se3 = LSTM(lstm)(se2)

    decoder1 = add([fe2, se3])
    decoder2 = Dense(lstm, activation='relu')(decoder1)
    outputs = Dense(self.vocab_size, activation='softmax')(decoder2)
```

```
    model = Model(inputs=[inputs1, inputs2], outputs=outputs)
    model.compile(loss='categorical_crossentropy', optimizer=Adam(lr=lr))
    return model
```

Las funciones de entrenamiento. Si no se disponen de 32 GB se recomienda usar la función *fit_generator* para evitar desbordamientos de memoria. Si se dispone de 32 GB se puede usar la función *fit_no_generator.* Es interesante indicar que inicialmente el modelo estaba configurado para usar *EarlyStopping* con *categorical_crossentropy,* que es la que suele utilizar en multiclases.

Obviamente esta no es la mejor métrica de validación para un descriptor de imágenes. Las mejores métricas son BLEU, o ROUGE. Por ello, se asocia a cada ciclo de entrenamiento, realizado con *fit_generator* o *fit_no_generator,* un ciclo de evaluación, que indique como va mejorando, o empeorando el valor de BLEU. Se usa un mecanismo de *EarlyStopping* basado en una de estas métricas, como veremos a continuación.

```
  def fit_generator(self, model,epochs):
    filepath = '../result/model-epoch{epoch:02d}-train_loss{loss:.2f}-val_loss{val_loss:.2f}.h5'
    #callbacks = [EarlyStopping(monitor='val_loss',patience=0),
    #ModelCheckpoint(filepath, monitor='val_loss', verbose=1, save_best_only=True, mode='min')]
    callbacks = [ModelCheckpoint(filepath, monitor='val_loss', verbose=1, save_best_only=True,
mode='min')]
    model.fit_generator(self.data_generator(self.train_captions), epochs=epochs,
steps_per_epoch=self.steps,callbacks=callbacks,
validation_data=self.data_generator(self.dev_captions),
validation_steps=len(self.dev_captions),shuffle=False)
    return model

  def fit_no_generator(self, model):
    filepath = '../result/model-epoch{epoch:02d}-train_loss{loss:.2f}-val_loss{val_loss:.2f}.h5'
    in1_train, in2_train, out_train = self.sequence_all(self.train_captions)
    in1_dev, in2_dev, out_dev = self.sequence_all(self.dev_captions)
    checkpoint = ModelCheckpoint(filepath, monitor='val_loss', verbose=1, save_best_only=True,
mode='min')
    model.fit([in1_train, in2_train], out_train, epochs=1, verbose=2, callbacks=[checkpoint],
validation_data=([in1_dev, in2_dev], out_dev),shuffle=False)
    return model
```

4.6. MÉTRICAS

En este código, que se puede localizar en el fichero “metrics.py”, se determina el resultado de la evaluación usando métricas BLEU y ROUGE, las cuales ya han sido

comentadas previamente. Se usarán las librerías *ntlk* y *py-rouge* para poder ejecutarlas. A continuación, se puede ver los aspectos más relevantes del código.

En primer lugar, se cargan los datos del conjunto de evaluación y se procesan realizando la predicción del modelo. Cada imagen tarda 1 segundo en ser procesada por el modelo.

```
np.random.seed(5050)
   actual_bleu, predicted_bleu = list(), list()
   actual_rouge, predicted_rouge = list(), list()
   decoder.load_test_data()
   #b = pb.ProgressBar(maxval=len(decoder.test_captions),widgets=[pb.Bar('=', '[', ']'), ' ',
pb.Percentage()])
   #b.start()
   #i=1
   print("Evaluating test images")
   for ts_key, ts_captions_list in decoder.test_captions.items():
      out = decoder.sequence_evaluation(model, decoder.image_encodings[ts_key])
      references = [d.split() for d in ts_captions_list]
      predicted_bleu.append(out.split())
      actual_bleu.append(references)
      predicted_rouge.append(out)
      actual_rouge.append(ts_captions_list)
```

Posteriormente, tras recibir el conjunto de frases candidatas se evalúan usando tanto BLEU como ROUGE.

```
print("\nEvaluation BLEU\n")
   bleu1=corpus_bleu(actual_bleu, predicted_bleu, weights=(1.0, 0, 0, 0))
   bleu2=corpus_bleu(actual_bleu, predicted_bleu, weights=(0.5, 0.5, 0, 0))
   bleu3=corpus_bleu(actual_bleu, predicted_bleu, weights=(0.3, 0.3, 0.3, 0))
   bleu4=corpus_bleu(actual_bleu, predicted_bleu, weights=(0.25, 0.25, 0.25, 0.25))
   print('BLEU-1: %f' % bleu1)
   print('BLEU-2: %f' % bleu2)
   print('BLEU-3: %f' % bleu3)
   print('BLEU-4: %f' % bleu4)
   # calculate ROUGE score
   evaluator = rouge.Rouge(metrics=['rouge-n', 'rouge-l', 'rouge-w'],
                     max_n=4,
                     limit_length=True,
                     length_limit=100,
                     length_limit_type='words',
                     apply_avg='Avg',
                     apply_best='Avg',
                     alpha=0.5,
                     weight_factor=1.2,
                     stemming=True)
   scores = evaluator.get_scores(predicted_rouge, actual_rouge)
```

```
print("\nEvaluation ROUGE with Avg\n")
for metric, results in sorted(scores.items(), key=lambda x: x[0]):
    print(prepare_results(metric, results['p'], results['r'], results['f']))
```

4.7. ENTRENAMIENTO

En este código, que se puede localizar en el fichero “train.py”, se realiza el proceso de entrenamiento, y también de evaluación. Cada vez que se procesa un *epoch* se deberá evaluar la métrica de este. De esta forma, se puede utilizar un mecanismo de *EarlyStopping*, como se indicó con anterioridad.

A continuación, se ve el código. Los valores que se utilizan para este cálculo son valores ajustado por búsqueda de parámetros, que se verá en el capítulo siguiente. Posteriormente, se lanza un ciclo completo de entrenamiento y evaluación. A la finalización de este se guardan los datos en un log.

```
def train():
    lr=0.00051
    dr=0.35
    lstm=400
    epochs=20
    for i in range(epochs):
        dc = decoder.decoder()
        model = dc.model(lr, dr, lstm)
        model=dc.fit(model,epochs)
        metrics.evaluate_model(model,dc)
```

4.8. INTEGRACIÓN CON TELEGRAM

De manera opcional se ha fijado el objetivo de integrar el sistema con una web o red social. Para ello se ha elegido Telegram, que es una red social desarrollada en 2013, que permite el desarrollo e integración de asistentes externos.

Desde mi punto de vista es una red ideal para cualquier prototipo, ya que permite mandar nuevas imágenes sin necesidad de tener que montar la infraestructura necesaria de subida de imágenes que requiere una web, ni de estar expuesto a la visibilidad que se sufre en una red a la que se pueda acceder sin inicio de sesión, como Twitter.

En el siguiente capítulo se indicará como se puede utilizar este asistente y mandarle imágenes. En este capitulo el objetivo es estudiar como se ha implementado, ya que a diferencia de los apartados anteriores será necesario extraer las características de la imagen mediante la red *CNN.*

El código esta presente en el fichero “code/letmeseebot.py”. Estos son los aspectos mas relevantes del mismo.

En primer lugar, es necesario reducir en todo lo posible el tiempo de espera. Por ello se deberá cargar en memoria el modelo del encoder, que será el modelo *VGG,* el modelo del decoder, y también el *tokenizer* del decoder. En el siguiente capitulo veremos las razones por las que se ha adoptado VGG, más concretamente en el apartado donde se compara con Inception.

```
print("Loading data...")
keras.backend.clear_session()
start = time.time()
dc = decoder.decoder()
filename = '../result/model-epoch01-train_loss4.63-val_loss4.09-best.h5'
model = load_model(filename)
model._make_predict_function()
vgg = vgg16.VGG16()
vgg.layers.pop()
vgg = Model(inputs=vgg.inputs, outputs=vgg.layers[-1].output)
vgg._make_predict_function()
end = time.time()
print("Data loaded!.\nTime: %0.2f seconds." % (end - start))
```

Posteriormente, se establece la conexión con la red de Telegram. Esta red permite revisar los mensajes que mandan los usuarios al bot de dos formas, mediante una técnica de *pooling*, interrogando a la red social, o bien mediante eventos, en cuyo caso será el sistema el que nos avise.

La mejor opción es mediante eventos, pero requiere un certificado de seguridad, y un dominio propio, que no podemos cumplir. Por ello, se usará la técnica de *pooling.* Cada mensaje recibido se procesa por la siguiente función, que establece si el mensaje enviado contiene una foto, en cuyo caso se extraen las características y se predice la frase candidata.

```
handle(msg):
    folder="../data/telegram/"
    content_type, chat_type, chat_id = telepot.glance(msg)
    line= str(msg['from']['username']) + ";" + str(msg['from']['first_name'])  + ";" +
str(msg['from']['language_code']) + ";" + str(msg['date'])
    if content_type != 'photo':
        bot.sendMessage(chat_id, "Please send me a photo. I make captions!")
    if content_type == 'photo':
        start = time.time()
        bot.sendMessage(chat_id, "Pretty photo. Let me see. Give me a second, please...")
```

```
    photo_id=msg['photo'][2]['file_id']
    filename=folder + photo_id + ".jpg"
    bot.download_file(photo_id, filename)
    img=extract_features(filename)
    out=predict_model(model,dc,img)
    end = time.time()
    line=line+";" + str(out) + ";" + str(end-start)
    bot.sendMessage(chat_id, "I see this: **" + str(out) + "** " + str(round(end-start,2)) + " seconds")
f=open("../data/telegram/log.txt", "a+")
f.write(line+"\n")
f.close
```

Para extraer las características será necesario convertir las imágenes a la resolución 224 x 224. Por ello, no se realizarán predicciones de imágenes inferiores a esa resolución.

```
def extract_features(filename):
    processed_img = image.load_img(filename, target_size=(224, 224))
    x = image.img_to_array(processed_img)
    x = np.expand_dims(x, axis=0)
    x = vgg16.preprocess_input(x)
    image_final = np.asarray(x)
    feature = vgg.predict(image_final)
    return feature
```

CAPÍTULO 5. EVALUACIÓN DE RESULTADOS

5.1. AJUSTE DE PARÁMETROS

Para mejorar la evaluación del modelo se realizará una búsqueda de los parámetros óptimos más relevantes del mismo. Para ello se disponen de varias opciones de búsqueda:

- Búsqueda manual. En este caso se seleccionan por cada parámetro relevante un conjunto de valores. Se realiza el entrenamiento del modelo por cada valor y se estudia la evaluación. En función del resultado que se obtenga en la métrica de la evaluación se decide el parámetro que deba ser usado.

- Búsqueda aleatoria. En este caso la selección de valores de cada parámetro es aleatoria dentro de un rango. Se puede usar para valores lineales una distribución normal adaptada al rango de valores que se quieran obtener.

- Mediante herramientas de búsqueda. Como la función *GridSearchCV* de la *scikit-learn.* Esta función realiza una búsqueda exhaustiva en función de un estimador, a través de un conjunto de valores.

De estas tres opciones se va a utilizar una combinación de la primera y la segunda. Se usarán valores seleccionados de un rango, que estén ordenados de menor a mayor, para poder visualizar el efecto que producen en la métrica de la evaluación.

Se usará un conjunto reducido del dataset para poder realizar de manera ágil las iteraciones de comprobación. Mas concretamente será el conjunto de validación del dataset, que este compuesto por 1000 imágenes.

Con respecto a la métrica se usará BLEU, ya que tras las pruebas se ha visto que es más sensible que ROUGE a la modificación de parámetros. Más concretamente será BLEU-1 ya que al usar un dataset más reducido habrá situaciones en las que BLEU-2, BLEU-3 y BLEU-4 tengan valores muy cercanos a cero.

En relación con los parámetros de búsqueda se han seleccionado los siguientes, ya que su modificación, aunque no sea muy grande produce un efecto significativo en la métrica.

- **Velocidad de aprendizaje, o *Learning rate*.** Esta es una variable muy importante en cualquier red neuronal, ya que afecta directamente al descenso del gradiente. Un valor alto puede provocar que no se llegue al mínimo de la función, y que haya un fallo de convergencia. Un valor bajo puede provocar un aumento muy pequeño del descenso del gradiente, y que haya que realizar más pasos, lo que implica un mayor coste computacional.

- **Índice de abandono, o *Dropout*.** Esta es una técnica que permite reducir el *overfitting*, o sobre entrenamiento, en redes neuronales. Fue propuesta por (Srivastava, et al, 2014)[4]. Para ello, lo que hace es eliminar neuronas en las capas ocultas y visibles de la red neuronal. Esta tecnica permite aumentar el rendimiento de las redes neuronales, pero hay que tener en cuenta que provoca un aumento del ruido en el descenso del gradiente.

- **Tamaño de salidas de la capa LSTM.** Este tamaño define la cantidad de unidades de las que se compondrá la red recurrente neuronal. Un mayor tamaño puede mejorar el rendimiento, pero supondrá un incremento en el coste computacional.

- **Número de epochs.** Esta variable indica el número de veces que se debe aplicar el algoritmo de *backpropagation* en la red neuronal. Permite establecer la actualización de los pesos de esta red neuronal.

Se hicieron pruebas modificando el tamaño del *batch*, y el *momentum* de la red neuronal, pero se pudo comprobar en esas pruebas que los valores por defecto, que son 32 y 1 respectivamente, son los mejores.

Learning rate

Para poder determinar el valor optimo del *learning rate* hay que tener en cuenta que esta es una variable logarítmica, es decir, que existe un amplio conjunto de valores realmente pequeños.

Existe una técnica establecida por Leslie N. Smith en su artículo: “Cyclical Learning Rates for Training Neural Networks” que consiste en entrenar el modelo desde un valor bajo del *learning rate* e irlo elevando exponencialmente en cada secuencia. Luego se realiza una gráfica donde se compara la perdida con el valor del *learning rate.*

En este caso se hará una variación. Para ello, se seleccionan 10 valores de *learning rate* en un intervalo, y se buscan los máximos que haya en BLEU-1. Una vez hallado un máximo se hará un zoom logarítmico cerca de ese máximo y se estudiarán los valores de BLEU-1 en ese nuevo intervalo. Así sucesivamente hasta que no haya una mejora en

la escala del *learning rate.* Los valores máximos de BLEU-1 que se obtienen son bajos, en torno a 0.40, ya que solo se está usando un *epoch* de entrenamiento, para reducir el tiempo de búsqueda.

En primer lugar, se hicieron pruebas en el intervalo [0-1], donde se apreció un máximo por debajo del valor 0.01. Continuamos en el intervalo [0-0.01], donde se puede apreciar que hay un máximo cerca del punto 0.002.

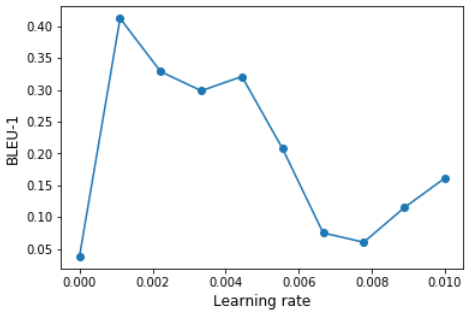


Figura 11. Evolución del learning rate entre 0 y 0.01.

Se amplía la gráfica en torno a punto 0.002, donde se puede ver que el valor de BLEU-1 asciende cerca de 0.00025 y encuentra su máximo un poco antes de 0.00075.

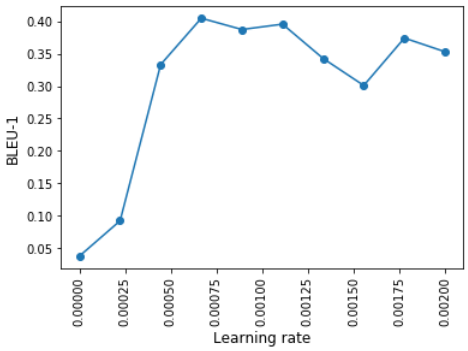


Figura 12. Evolución del learning rate entre 0 y 0.002.

Se amplia de nuevo la gráfica entre esos dos valores y se observa un máximo en torno a 0.00050.

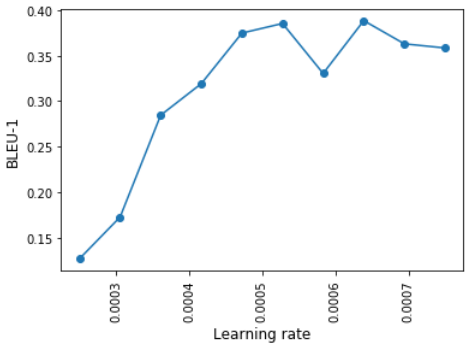


Figura 13. Evolución del learning rate entre 0.00025 y 0.00075.

Se vuelves a ampliar, por última vez y ya se localiza el valor del máximo que se halla en 0.00051.

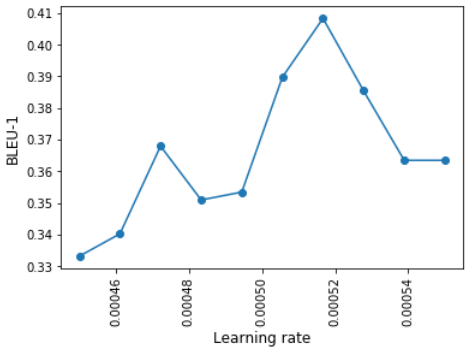


Figura 14. Evolución del learning rate entre 0.00045 y 0.00055.

Dropout

El *dropout* representa el porcentaje de unidades que serán descartadas en la red neuronal, junto con sus conexiones. En este caso se aplica *dropout* a la salida de la red convolucional, y también a la salida de la red *LSTM.*

Se estudia la variación de la métrica BLEU-1 cuando el porcentaje decae desde 0.25 hasta 0.5. Se puede apreciar un máximo de la métrica para un *dropout* de 0.35.

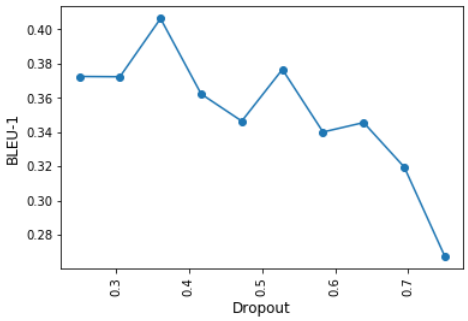


Figura 15. Evolución del dropout.

Tamaño de salidas de la capa LSTM

El tamaño de salidas de la capa LSTM indica el número de unidades de las que se compone esta red. Se puede apreciar que al aumentar este número aumenta el valor de la métrica. Sin embargo, un número elevado de unidades incrementa el tiempo de manera notable. En este caso se usará un valor de 400 unidades.

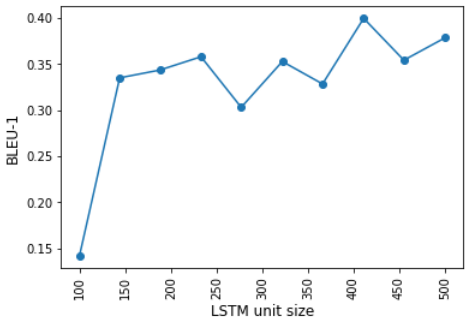


Figura 16. Evolución de la tamaño de salidas de la red LSTM.

Número de epochs

En *Deep learning* un *epoch* es un ciclo completo de propagación hacia adelante y hacia atrás. Al finalizar el *epoch* se actualizan los pesos de las matrices de la red neuronal.

En este código se utiliza un mecanismo de parada llamado *EarlyStopping* implementado por *keras.* Este mecanismo verifica la pérdida del error en el conjunto de validación, y en caso de que esa pérdida no se reduzca se para el modelo.

Sin embargo, se puede observar la variación de la métrica BLEU-1 con respecto al número de *epochs* en la siguiente gráfica, donde se llegan a realizar iteraciones completas de hasta 60 *epochs.*

Se puede observar que a partir de las iteraciones donde hay 20 *epochs* no mejora la métrica, ya que el sistema esta sobreentrenado. Por ello, se usarán 20 *epochs* como valor máximo de este parámetro, pero condicionado al mecanismo *EarlyStopping.*

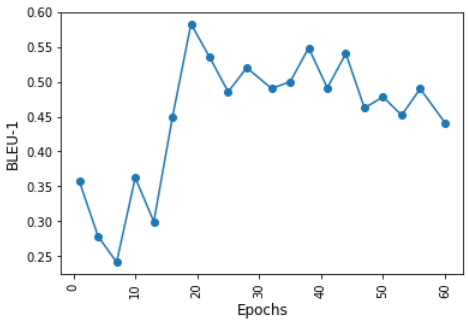
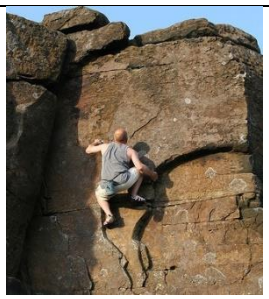


Figura 17. Evolución del número de epochs.

5.2. RENDIMIENTO CUALITATIVO

El rendimiento cualitativo permite evaluar los resultados del modelo de una manera cualitativa, es decir, estableciendo lo bien o mal que se adapta la imagen desde un acercamiento realizado a través de un juicio de calidad humano.

Como se puede ver en la siguiente tabla, nuestro modelo genera descripciones que están comprendidas en un amplio rango, desde descripciones completas sin errores, a descripciones que no tienen nada en común con la imagen.

Descripciones sin errores			
	Black dog is jumping over hurdle	Boy is jumping into the air	Man is climbing rock
Descripciones con errores menores			
	Two boys are playing soccer	Two dogs are playing in the grass	Baby is eating with baby
Hay algo relacionado con la imagen			
	Man is sitting on the edge of water	Two girls are playing soccer	Two men are playing basketball
No hay nada relacionado con la imagen			
	Two dogs are playing in the grass	Group of people are sitting on the beach	Man in black shirt is sitting on the water

Se puede observar que el modelo identifica un objeto principal de la escena, ya sea una persona, o un animal, y también identifica el ambiente de la escena, que puede ser sobre hierba, en la playa, en el agua. En otro tipo de imágenes se identifica la acción, que puede ser jugar al baloncesto o al futbol.

El modelo puede acertar algunos de estos elementos, o incluso todos, sobre todo cuando las imágenes son similares, o de la misma temática, a las que se han usado en el entrenamiento, o puede que no acierte con ninguno, sobre todo cuando los objetos de las imágenes se pueden confundir con otros, o cuando son objetos que no han llegado a ser usados en el entrenamiento apenas.

Para poder hacer un análisis de las imágenes de evaluación lo que se hace es extraer todas las imágenes a una carpeta y cambiarles el nombre uniendo la frase candidata y el nombre de la imagen mediante el código que está en el fichero “predict.py”, del cual se pueden ver los fragmentos más interesantes del código a continuación.

Carga de datos del modelo y puesta en marcha del decoder.

```
print("Loading data...")
start = time.time()
filename = '../result/model-epoch01-train_loss3.27-val_loss3.79-best.h5'
model = load_model(filename)
dc = decoder.decoder()
end = time.time()
print("Data loaded!.\nTime: %0.2f seconds." % (end - start))
```

Selección de imágenes que pertenecen al conjunto de evaluación.

```
images=list()
file='../result/testImages.txt'
test_files= pd.read_csv(file, delimiter='\t')
for i in range(int(len(test_files))/10):
   if test_files.iloc[i][0] not in images:
      images.append(test_files.iloc[i][0])
```

Llamada a la predicción de las imágenes y guardado de las mismas en una carpeta, junto con la descripción candidata. Cada imagen se procesa en un periodo cercano a un segundo.

```
for image in images:
   start = time.time()
   out=predict_model(model,dc,image)
   out=out.replace(' ', '-')
   img=mpimg.imread("../data/Flickr8k_Dataset/" + image)
```

```
    mpimg.imsave("../result/images/"+ out + "-" + image, img, )
    imgplot = plt.imshow(img)
    plt.show()
    end = time.time()
    print("Time: %0.2f seconds.\nCaption: %s." % (end - start,out))
```

Por último, la función de predicción donde se crea la secuencia de evaluación en el decoder. Tras obtener la frase candidata se quitan las coletillas “startcap” y “endcap” de la frase.

```
def predict_model(model, decoder, img):
    out = decoder.sequence_evaluation(model, decoder.image_encodings[img])
    out = re.sub('(\s*)startcap(\s*)','',out)
    out = re.sub('(\s*)endcap(\s*)','',out)
    return out
```

5.3. RENDIMIENTO CUANTITATIVO

Antes de ver las métricas es necesario indicar que el equipo principal en el que se han hecho las pruebas tiene una CPU i5-4200, con 8 GB de memoria. Cada *epoch* del entrenamiento ha supuesto unos 2200 segundos, mientras que cada fase de evaluación supone unos 700 segundos.

El tiempo promedio de predicción de una imagen es de un segundo. Obviamente para poder entrenar en este equipo ha sido necesario utilizar el generador de datos que se comentó en el capítulo anterior, con la función *fit_generator.*

Asimismo, también se han hecho pruebas en un equipo remoto con 32 GB y una CPU similar a un i3-3200. En este caso el tiempo de entrenamiento era mas largo de unos 3000 segundos, y el tiempo de evaluación era de unos 800 segundos. En este equipo se podía usar la función *fit_no_generator,* que no depende de un generador de datos.

En ninguno de estos equipos se ha podido usar la GPU, lo cual hubiese supuesto una reducción considerable del tiempo de entrenamiento.

En esta sección se analiza el rendimiento cuantitativo viendo los valores obtenidos en la evaluación mediante el uso de métricas como BLEU o ROUGE.

BLEU

BLEU es una medida cuantitativa ampliamente utilizada en problemas de descripción de imágenes. Representa la precisión de la evaluación, es decir, una medida de cuantas veces aparece los n-gramas de la frase candidata en las frases de referencia.

Se puede ver una comparativa entre los valores iniciales que se han obtenido, sin realizar ningún ajuste para un *epoch*, y los valores ajustados que se han obtenido usando 14 *epochs* con *EarlyStopping.*

BLEU	Valores iniciales	Valores ajustados
BLEU-1	0.569416	0.601986
BLEU-2	0.307419	0.346363
BLEU-3	0.203145	0.240815
BLEU-4	0.089569	0.114119

Estos son los parámetros ajustados.

Learning rate	0.00051
Dropout	0.35
Tamaño unidad LSTM	400
Epochs	14

Como se puede apreciar ha habido una mejora aproximada del 8% en la métrica del BLEU. En mi opinión, es una buena mejora si solo se tiene en cuenta la métrica. Cada iteración de entrenamiento supone cerca de 2200 segundos, con lo que la creación del modelo ajustado se alargó unas 9 horas. Existe un compromiso entre el tiempo de cálculo y los mejores valores posibles.

A continuación, se puede ver la comparación de estos valores con el estado del arte.

BLEU	Valores ajustados	(Jia et al, 2015)[12]	(Karpathy, et al, 2015)[10]
BLEU-1	0.601986	0.647	0.579
BLEU-2	0.346363	0.459	0.383
BLEU-3	0.240815	0.318	0.245
BLEU-4	0.114119	0.216	0.160

Como se puede apreciar este modelo aún podría mejorarse hasta llegar al estado del arte de la técnica encoder-decoder, representada por el trabajo de Jia. Hay varias partes donde se puede mejorar el modelo, las cuales se comentarán con más detalle en el apartado de futuras mejoras.

Si se compara con el trabajo de (Karpathy, et al, 2015)[10] se puede apreciar que el resultado no difiere mucho, aunque en mi opinión hay que mejorar el porcentaje de precisión de BLEU-4.

ROUGE

ROUGE es una medida cuantitativa que representa el *recall* de la evaluación, es decir, una medida de cuantas veces aparece los n-gramas de las frases de referencia en la frase candidata. Sin embargo, es posible también obtener medidas de precisión con ROUGE, como vimos en capítulos anteriores.

A continuación, se puede ver una comparativa entre los valores iniciales que se obtenían, sin realizar ningún ajuste para un *epoch*, y los valores ajustados que se han obtenido usando 14 *epochs* con *EarlyStopping.*

RECALL	Valores iniciales	Valores ajustados
ROUGE-L	0.40	0.41
ROUGE-1:R	0.35	0.36
ROUGE-2:R	0.05	0.07
ROUGE-3:R	0.01	0.02
ROUGE-4:R	0.00	0.01

PRECISION	Valores iniciales	Valores ajustados
ROUGE-L	0.41	0.45
ROUGE-1:R	0.36	0.41
ROUGE-2:R	0.06	0.08
ROUGE-3:R	0.02	0.03
ROUGE-4:R	0.00	0.01

Se pueden ver mejoras suaves en *recall*, y más apreciables en precisión entre los valores iniciales y los ajustados. Hay que tener en cuenta que a la hora de ajustar los parámetros se han buscado aquellos que mejoraban más a BLEU, no a ROUGE.

Asimismo, se puede ver la comparación de estos valores con el estado del arte. Lamentablemente solo ha sido posible encontrar un artículo donde se haya usado esta métrica, y es con su variante de precisión.

PRECISION	Valores ajustados	(Jia et al, 2015)[12]
ROUGE-L	0.45	0.50

El resultado da un valor elevado de ROUGE, pero existe una diferencia en torno al 10% del mismo, bastante superior a la diferencia que existía en BLEU-1 con (Jia et al, 2015)[12], que era en torno al 8%. Esto es posible reducirlo si se aplican las futuras mejoras que se comentan en el próximo capitulo.

5.4. COMPARACIÓN VGG INCEPTION

Como se ha indicado en puntos anteriores este modelo permite extraer características tanto del modelo entrenado de VGG como del modelo entrenado de Inception.

La implementación de Inception se ha comentado en el capítulo 4, donde se indica que para poder extraer las características de Inception se aplican una serie de cambios en el modelo.

- Las imágenes se adaptan al tamaño prefijado por Inception. El tamaño es de 299 x 299 pixeles.
- El modelo se modifica para que se adapte a la penúltima capa del modelo de Inception, la cual tiene 2048 dimensiones, en lugar de las 4096 de VGG.

A continuación, se pueden ver cuáles son los mejores valores que se han obtenido con cada modelo para las métricas de BLEU. En el caso de VGG fueron necesarios 14 *epochs* y en el caso de Inception fueron necesarios 16 *epochs.*

BLEU	VGG	Inception V3
BLEU-1	0.601986	0.598153
BLEU-2	0.346363	0.336717
BLEU-3	0.240815	0.234640
BLEU-4	0.114119	0.104637

Como se puede apreciar los valores obtenidos por Inception no mejoran a los de VGG. Existen varias razones que pueden explicarlo.

- En primer lugar, no se están usando los pesos con los que se ganó en la competición ImageNet. Si se utilizan esos pesos da un fallo en el modelo por las dimensiones de salida que se obtienen.

  Una solución sería realizar una redimensión con la salida obtenida con esos pesos y realizar pruebas. Esta opción implica utilizar una nueva capa, *fully connected layer,* que conecte la salida del *CNN* con la capa de mezcla.

- En segundo lugar, el ajuste de parámetros que se ha realizado fue hecho para VGG. Estos parámetros no están optimizados para Inception. Esto implica que se debe hacer un nuevo estudio de parámetros personalizado para Inception.

Lamentablemente por limitaciones de tiempo no se han podido probar estas opciones, pero se dejan indicadas para futuras mejoras del modelo.

5.5. PREDICCIÓN CON NUEVAS IMÁGENES EN TELEGRAM

Hasta ahora se han realizado pruebas de evaluación con las imágenes del conjunto de evaluación. Sin embargo, el sistema debe permitir que se evalúen imágenes nuevas. Para ello se implementará un código que permita al sistema ser usado desde un asistente, o *bot*, de Telegram cuyo alias es @Let_Me_See_Bot.

El funcionamiento es muy sencillo, solo hay que buscar el *bot* en Telegram, darle al botón iniciar y mandarle una imagen con un tamaño superior a 224 x 224 pixeles. El *bot* nos devolverá la descripción que predice y el tiempo que ha tardado en realizarla. En la siguiente figura se puede ver un ejemplo, donde se han requerido menos de 2 segundos.

Figura 18. Ejemplo de uso del bot de Telegram.

Tras realizar pruebas se puede sacar varias conclusiones:

- El tiempo de respuesta del asistente no suele exceder los 3 segundos. Es un tiempo razonable, sobre todo si se tiene en cuenta que se debe bajar la imagen, procesarla en el encoder, y luego procesar las características en el decoder para obtener la respuesta.
- El modelo responde bien si las imágenes utilizadas son parecidas a las del dataset de entrenamiento. En la figura anterior se puede ver que reconoce un objeto y reconoce el fondo, pero no identifica correctamente al objeto, que es un zorro, en su lugar lo llama perro. El motivo es que no ha sido entrenado con imágenes de zorros.

- El asistente se ejecuta en el equipo remoto de 32 GB, donde consume unos 4 GB de memoria. Para facilitar el tiempo de respuesta se tiene precargado en memoria el modelo del encoder, el modelo del decoder, y el vocabulario disponible.

5.6. VALIDACIÓN CRUZADA

Para poder estudiar la estabilidad del modelo se han realizado pruebas mediante validación cruzada, o *cross-validation.* Esta técnica lo que hace es particionar el conjunto de entrenamiento en una serie de bloques, por ejemplo, cuatro bloques, de los cuales tres se usan para entrenamiento y uno para validación. Posteriormente, se hace la media de la perdida obtenida y se obtiene la desviación.

En nuestro caso, se han usado cinco bloques. Tres para el conjunto de entrenamiento, uno para el conjunto de validación, y uno para el conjunto de evaluación. Como los dataset de entrenamiento, validación y evaluación ya habían sido creados en el código de preprocesamiento lo que hacemos es usar el conjunto de datos completo, el cual esta en el fichero *tokenImages.txt*, el cual fue también limpiado en ese apartado.

Usaremos la función *kfold* de *scikit-learn*, la cual nos divide el conjunto de datos en 5 bloques. Se realizan cinco iteraciones del modelo en las que se anotan los valores de BLEU y ROUGE. Tras ello se determina el valor medio y la desviación. Estos son los valores obtenidos para el modelo que habíamos ajustado.

BLEU	Media	Desviación	ROUGE	Media	Desviación
BLEU-1	0.600382	2,36%	**ROUGE-1:R**	0.35	0,66%
BLEU-2	0.343839	2,52%	**ROUGE-2:R**	0.05	0,55%
BLEU-3	0.232930	2,88%	**ROUGE-3:R**	0.01	0,98%
BLEU-4	0.113429	3,02%	**ROUGE-4:R**	0.01	1,00%

CAPÍTULO 6. FUTURAS MEJORAS Y CONCLUSIONES

6.1. FUTURAS MEJORAS

Existen varias mejoras que se pueden aplicar a estos resultados que se han obtenido.

Modelos entrenados. En primer lugar, hay que analizar porque los resultados con Inception V3 no han mejorado los de VGG-16. Inception V3 es un modelo más grande y más elaborado que VGG y debería establecer una mejora en la evaluación.

Hay una serie de puntos que se han comentado en el apartado anterior que deben ser revisados para mejorar su rendimiento.

Asimismo, no se tiene por qué limitar los modelos pre entrenados de imágenes a estos dos. Se pueden realizar pruebas con otros modelos como ResNet-152.

Mejora del modelo. El modelo que se usa es un modelo muy sencillo, donde solo usamos una capa en la red LSTM. El uso de un mayor número de capas podría aumentar el rendimiento de este.

Refinar el vocabulario. Se ha utilizado un vocabulario extraído del conjunto de entrenamiento, con cerca de 8.000 palabras, muchas de ellas mal escritas. Se puede eliminar las palabras mal escritas para mejorar el rendimiento.

Utilizar mecanismos de atención. Los mecanismos de atención como (Xu, et al, 2017)[13] ayudan a mejorar los resultados de evaluación del trabajo. Sin embargo, el uso de estos mecanismos va a requerir tener que usar entrenamiento mediante GPU para poder obtener los resultados en un tiempo razonable.

Otros datasets. La adaptación a Flickr30k es trivial. Sin embargo, va a requerir de más potencia computacional. Cada *epoch* en Flickr8k suponen cerca de 40 minutos ya que no se cuenta con soporte de GPU.

Un nuevo dataset con más datos mejorará los resultados cualitativos, pero esta opción no debería ser prioritaria, ya que lo importante es mejorar el modelo antes de cambiar el dataset.

6.2. CONCLUSIONES

Personalmente estoy satisfecho con los resultados obtenidos en este trabajo. Debemos partir de la premisa de mi desconocimiento de redes neuronales al comenzar el mismo. En menos de dos meses he pasado de no conocer apenas las redes neuronales a poder realizar ajustes en las mismas. Tras muchas horas de pruebas y de evaluaciones el código se ha ido acercando, poco a poco, al estado del arte.

Mi impresión es que si hubiese dispuesto de más tiempo podría superar mediante un ajuste más fino los resultados de (Karpathy, et al, 2015)[10], sobre todo en BLEU-4 y encarar las mejoras comentadas en el apartado anterior para poder acercarme en todo lo posible a los resultados obtenidos por (Jia et al, 2015)[12].

Sin embargo, hay que ser realista y ver no solo las limitaciones en tiempo, sino tambien en recursos. En clasificación de imágenes se gana mucho tiempo si se dispone de equipos con GPU, lo cual no ha sido el caso, y ha habido que realizar entrenamientos muy largos, de mas de 10 horas, que podrian haber sido reducidos sustancialmente en el caso de disponer de estos recursos.

Mi opción inicial cuando iba a realizar este TFM era realizar un sistema de clasificación basado en imágenes, es un modelo muy documentado y que consigue resultados muy favorables en las métricas, pero creo que un descriptor de imágenes es un reto mas abierto, y que aún no ha conseguido ser completado por la comunidad, por lo que es susceptible de ser modificado y cambiado para tratar de mejorarlo.

El poder usar distintos tipos de redes neuronales y de establecer arquitecturas de mezcla, o de inyección es una demostración de las tremendas capacidades que tenemos al combinar distintos tipos de información en las redes neuronales. La experimentación con estos modelos es sorprendente y por ello me satisface haber podido seleccionar este area.

Por último, el poder integrar este modelo en un asistente o bot de una red social es muy divertido, y permite evaluar el sistema ante un juez mas exigente, que es el ser humano. Los resultados de las métricas pueden parecer buenos, ya que se acercan al estado del arte, pero es el uso en un sistema el que determina la viabilidad del modelo mediante evaluación cualitativa. En este caso se aprecia que el sistema es bueno cuando se le pasan imágenes similares del dataset del que se nutre.

GLOSARIO

- CNN. *Convolutional neural network.* Red neuronal convolucional. Son redes neuronales muy efectivas para visión artificial.
- CPU. *Central processing unit.* Unidad de procesamiento central. Es el procesador principal de un sistema informático, que está especializado en resolver un número reducido de operaciones altamente complejas.
- GPU. *Graphics processing unit.* Unidad de procesamiento gráfico. Es un procesador auxiliar de un sistema informático, especializado en resolver un numero altamente elevado y paralelo de operaciones matriciales sencillas. Nos permite procesar la red neuronal en el menor tiempo posible.
- Keras. Es una librería de código abierto que nos permite ejecutar nuestro modelo en sistemas como TensorFlow, Microsoft Cognitive Toolkit o Theano.
- Inception. Es un modelo de red neuronal convolucional, desarrollado por Google, que quedo segundo en la competición de 2015 de ImageNet.
- LSTM. *Long-short term memory.* Es una red neuronal recurrente que es menos sensible al intervalo de la secuencia.
- RNN. *Recurrent neural network.* Red neuronal recurrente. Es un tipo de red neuronal que procesa secuencias de entradas. Es usada en traducción de idiomas y series de tiempo.
- TensorFlow. Es una biblioteca, desarrolla por Google, de código abierto para aprendizaje automático. Puede ser ejecutada en CPUs y GPUs, en estas últimas mediante extensiones CUDA.
- VGG. Es un modelo de red neuronal convolucional que obtuvo el primer y segundo en las competiciones de ImageNet de localización y de clasificación, respectivamente. Hay 2 modelos, uno de 16 capas y otro de 19 capas que usan pequeños filtros convolucionales de tamaño 3x3 y 1x1.

REFERENCIAS

[1] Image-net.org

[2] Global-Market-for-Machine-Vision-Technologies-to-Reach-24-8-Billion-by-2023

[3] Shell-mining-oil-gas-azure-databricks

[4] Ordonez, V., Kulkarni, G., & Berg, T. (2011). IM2text: Describing images using 1 million captioned photographs. Advanced in Neural Information Processing Systems, pp. 1143-1151. [Fecha de consulta 15 de diciembre de 2018].

[5] Gupta, A., Verma, T., & Jawahar, V. (2012). Choosing linguistics over vision to describe images. Proceedings of the 26th AAAI Conference on Artificial Intelligence Vol. 5 pp. 606-612. [Fecha de consulta 15 de diciembre de 2018].

[6] Stanford CoreNLP

[7] Yang, Y., Teo, C., Daume, H., & Aolimono, Y. (2011). Corpus-guided sentence generation of natural images. Proceedings of the 2011 Conference in Empirical Methods in Natural Language Processing, pp. 444-454. [Fecha de consulta 15 de diciembre de 2018].

[8] Karpathy, A., Joulin, A., & LI, F. (2014). Deep fragment embeddings for bidirectional image sentence mapping. Advances in Neural Information Processing Systems 27, Vol 3, pp. 1889-1897. [Fecha de consulta 16 de diciembre de 2018].

[9] Mao, J., Xu, W., Yang, Y., Wang, J., Huang, Z., & Yuille, A. (2015). Deep captioning with multimodal neural networks. Proceedings on the 31st International Conference on Machine Learning, pp. 595-603. [Fecha de consulta 16 de diciembre de 2018].

[10] Karpathy, A., & Fei-Fei, L. (2015). Deep Visual-Semantic Alignments for Generating Image Descriptions. IEEE Transactions on Pattern Analysis and Machine Intelligence, Volume 39 Issue 4, pp. 664-676 . [Fecha de consulta 16 de diciembre de 2018].

[11] Pu, Y., Gan, Z., Henao, R., Yuan, X., LI, C., Stevens, A., & Carin, L. (2016). Variational autoencoder for deep learning of images, labels and captions. Advances in Neural Information Processing Systems 29, pp. 2352-2360. [Fecha de consulta 16 de diciembre de 2018].

[12] Jia, X., Gavves, E., Fernando, B., & Tuytelaars, T. (2015). Guiding the long-short term memory model for image caption generation. IEEE International Conference on Computer Vision, pp. 2407-2415. [Fecha de consulta 17 de diciembre de 2018].

[13] Xu, K., Ba, J., Kiros, R., Cho, K., Courville, A., Salakhutdinov, R., Bengio, Y. (2017). Show, attend and tell: Neural image caption generation with visual attention. ArXiv preprint 1502.03044. Digital edition. [Fecha de consulta 17 de diciembre de 2018].

[14] WordNet Princeton

[15] Karpathy, A., Fei-Fei, L. & Johnson, J. (2016)., Densecap: Fully convolutional localization networks for dense captioning, in: IEEE Conference on Computer Vision and Pattern Recognition, 2016, pp. 4565–4574. [Fecha de consulta 17 de diciembre de 2018].

[16] Baoyo, J., Pengtao, X., & Xing, E. (2017). On the Automatic Generation of Medical Imaging Reports. arXiv:1711.08195. Digital edition. [Fecha de consulta 17 de diciembre de 2018].

[17] Vinyals, O. T. (2015). Show and Tell: A neural image Caption generator. arXiv, 1411.4555. Digital edition. [Fecha de consulta 18 de diciembre de 2018].
[18] Tanti, M. G. (2018). Where to put the Image in an Image Caption Generator. arXiv:, 1703.09137. Digital edition. [Fecha de consulta 6 de diciembre de 2018].
[19] Stanford.edu

BIBLIOGRAFÍA ADICIONAL

- Bai, S., & An, S. (2018). A survey on automatic image caption generation. *Neurocomputing 311,* pp. 291-304.
  [Fecha de consulta 25 de diciembre de 2018].
- Hossain , Z., Sohel, F., Shiratuddin, M., & Laga, H. (2018). A comprehensive study of deep learning for image captioning. *arXiv:1810-04020v1*. Digital edition.
  [Fecha de consulta 15 de diciembre de 2018]
- Shahebaz, M. (2018). Introduction to Image Caption Generation using the Avenger's Infinity War Characters. [en línea] Medium.com
  https://medium.com/analytics-vidhya/introduction-to-image-caption-generation-using-the-avengers-infinity-war-characters-6f14df09dbe5
  [Fecha de consulta 5 de diciembre de 2018]