

**Universidad Católica de Santa María**  
**Facultad de Ciencias e Ingenierías Físicas y**  
**Formales**  
**Escuela Profesional de Ingeniería Electrónica**



**“SUBTITULADO AUTOMÁTICO DE IMÁGENES UNA  
APLICACIÓN DE DEEP LEARNING”**

Tesis presentada por el Bachiller:

**Huanca Luque, Ademir Yuri**

para optar el Título Profesional de

**Ingeniero Electrónico con  
especialidad en Automatización  
y Control**

Asesora:

**Mgtr. Delgado Barra, Lucy**

**Arequipa – Perú**

**2021**

UCSM-ERP

**UNIVERSIDAD CATÓLICA DE SANTA MARÍA**  
**INGENIERIA ELECTRONICA**  
**TITULACIÓN CON TESIS**  
**DICTAMEN APROBACIÓN DE BORRADOR**

Arequipa, 06 de Noviembre del 2021

**Dictamen: 002512-C-EPIE-2021**

Visto el borrador del expediente 002512, presentado por:

**2012244091 - HUANCA LUQUE ADEMIR YURI**

Titulado:

**SUBTITULADO AUTOMATICO DE IMÁGENES UNA APLICACIÓN DE DEEP LEARNING**

Nuestro dictamen es:

**APROBADO**

**1475 - MALAGA CHAVEZ CESAR EDUARDO**  
**DICTAMINADOR**



**1531 - RIVERA CHAVEZ VICTOR HUGO**  
**DICTAMINADOR**



**1843 - URRUTIA ESPINOZA MARIO WILLIAM**  
**DICTAMINADOR**

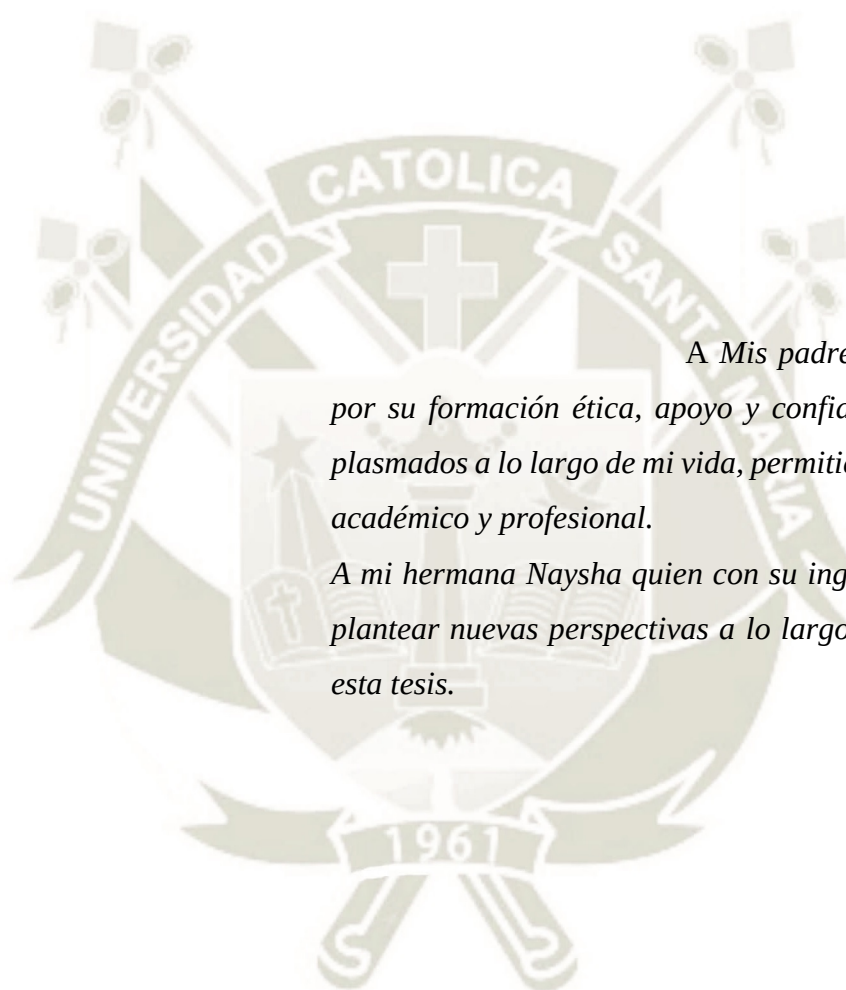


## Dedicatoria

*A mi tío Ing. Reynaldo León, mentor y guía quien con su saber y paciencia inspiró en mí el amor a la ciencia y la técnica desde muy temprana edad, sus conocimientos y consejos fueron esenciales en mi desarrollo académico y personal.*



## Agradecimiento



*A Mis padres Jesús y Soledad  
por su formación ética, apoyo y confianza incondicional  
plasmados a lo largo de mi vida, permitiendo mi desarrollo  
académico y profesional.*

*A mi hermana Naysha quien con su ingenio nato ayudó a  
plantear nuevas perspectivas a lo largo del desarrollo de  
esta tesis.*

## Resumen

El subtítulo autónomo de imágenes (Image captioning) requiere aplicar las áreas de visión artificial y procesamiento de lenguaje, el tema ha sido abordado por diferentes investigadores en el mundo, sin embargo la mayor parte de ellos son realizados en inglés, debido principalmente a que las bases de datos y las publicaciones se realizan en este idioma, no por ello debe dejarse al mundo hispanoparlante sin participación en este avance computacional. Se han realizado pocos estudios que aborden el subtítulo entre idiomas, siendo los mas recurrentes entre el chino e inglés, sin embargo los antecedentes para el español son prácticamente nulos.

En la presente tesis se aborda la investigación e implementación de un algoritmo basado en *Deep Learning*, utilizando CNNs y RNNs el cuál sea capaz de generar descripciones de las imágenes que le sean alimentadas.

Para conquistar el objetivo de esta tesis se propone la utilización de transferencia de aprendizaje en la extracción de características, aplicación de codificador y decodificador para el modelado de lenguaje y mecanismo de atención que permite localizar elementos resaltantes de las imágenes. Como base de datos se utiliza MSCOCO de Microsoft. Las oraciones resultantes se evalúan con BLEU y METEOR, el rendimiento del algoritmo se evalúa de acuerdo a su score en diferentes configuraciones variando EPOCHs, BATCHs y tamaño de dataset, a su vez se construyó una interfaz de usuario que permite realizar predicciones individuales de imágenes elegidas por el usuario.

### Palabras claves:

Subtítulo autónomo, Deep Learning, CNN, RNN, BLEU, MSCOCO

## Abstract

Image captioning applies the areas of artificial vision and language processing, the subject has been addressed by different researchers in the world, however most of them are conducted in English, mainly because the databases and publications are made in this language, not for this reason the Spanish-speaking world should be left without participation in this computational advance. Few studies have been carried out on subtitling between languages, the most recurrent being between Chinese and English; however, there are practically no precedents for Spanish.

This thesis deals with the research and implementation of an algorithm based on Deep Learning, using CNNs and RNNs, which is able to generate descriptions of the images that are fed to it.

To achieve the objective of this thesis we propose the use of transfer learning in the extraction of features, application of encoder and decoder for language modeling and attention mechanism that allows to locate salient elements of the images. Microsoft's MSCOCO is used as main database. The resulting sentences are evaluated with BLEU and METEOR, the performance of the algorithm is evaluated according to its score in different configurations varying EPOCHs, BATCHs and dataset size, also a user interface was built that allows individual predictions of images chosen by the user.

### Key words:

Image captioning, Deep Learning, CNN, RNN, BLEU, MSOCO

# Introducción

La inteligencia artificial ha visto un crecimiento exponencial en el último decenio estando presente en prácticamente todos los aspectos de la vida en forma de algoritmos de búsqueda de Google, filtros de correo, detección de voz y modelos de predicción en el sector financiero y logístico. En el área de visión artificial se aplica en el reconocimiento de personas, identificación de patrones laborales de riesgo e identificación de objetos en general, el estado del arte lo define la competencia anual de ImageNet donde hasta el 2020 las capacidades de reconocimiento de los algoritmos ya superaron a las humanas [32].

La identificación de imágenes y su clasificación es un problema aparentemente simple, siendo una de las primeras habilidades que se le enseña al ser humano y que aprende con facilidad a los 5 años, sin embargo, para una máquina la tarea es altamente compleja. El siguiente paso a la identificación de objetos o escenas es la generación de oraciones que describan lo observado, esta tarea se conoce como image captioning y sus aplicaciones van desde el subtitulado de imágenes de la web, asistencia a invidentes, telemedicina, etc. Google, Facebook y Yandex fueron de los pioneros en implementar sistemas de clasificación autónoma de imágenes originalmente utilizando sistemas poco eficientes como segmentación de imágenes, al día de hoy los algoritmos predominantes son basados en inteligencia artificial debido a su desempeño superior [38].

El subtitulado autónomo en el idioma español ha sido poco explorado por lo cual se propone un algoritmo de inteligencia artificial capaz de completar la tarea utilizando una base de datos en idioma inglés, a su vez se explora las distintas técnicas y métodos requeridos para su implementación, tales como redes neuronales recurrentes y convolucionales.

## Tabla de contenidos

|                                               |     |
|-----------------------------------------------|-----|
| Dedicatoria.....                              | iii |
| Agradecimiento .....                          | iv  |
| Resumen .....                                 | v   |
| Abstract.....                                 | vi  |
| Introducción.....                             | vii |
| CAPÍTULO I ASPECTOS GENERALES.....            | 1   |
| 1 Aspectos generales .....                    | 2   |
| 1.1 Planteamiento del problema .....          | 2   |
| 1.1.1 Trabajos previos .....                  | 2   |
| 1.2 Hipótesis.....                            | 3   |
| 1.3 Metodología.....                          | 3   |
| 1.4 Objetivos de la tesis.....                | 4   |
| 1.4.1 Objetivo general .....                  | 4   |
| 1.4.2 Objetivos específicos.....              | 4   |
| 1.4.3 Variables dependientes.....             | 4   |
| 1.4.4 Variables independientes.....           | 5   |
| 1.4.5 Justificación.....                      | 5   |
| CAPÍTULO II BASES TEÓRICAS.....               | 6   |
| 2 Bases teóricas .....                        | 7   |
| 2.1 Aprendizaje de máquina.....               | 7   |
| 2.1.1 Aprendizaje supervisado .....           | 7   |
| 2.1.2 Aprendizaje no supervisado .....        | 8   |
| 2.2 Redes neuronales artificiales .....       | 8   |
| 2.2.1 Neurona artificial.....                 | 9   |
| 2.2.2 Capas de una red neuronal.....          | 10  |
| 2.2.3 Funciones de activación .....           | 12  |
| 2.2.4 Entrenamiento de redes neuronales ..... | 15  |
| 2.3 Convolutional Neural Network (CNN).....   | 16  |
| 2.3.1 Capa de Convolucion .....               | 17  |
| 2.3.2 Capa de <i>Pooling</i> .....            | 19  |
| 2.3.3 Arquitecturas CNN.....                  | 20  |
| 2.4 RNNs y procesamiento de lenguaje .....    | 24  |

|                                                 |    |
|-------------------------------------------------|----|
| 2.4.1 RNNs para el modelado de secuencias.....  | 25 |
| 2.4.2 Estructura y flujo en RNNs .....          | 26 |
| 2.4.3 Unidad LSTM.....                          | 28 |
| 2.4.4 Unidad GRU.....                           | 29 |
| 2.5 Mecanismo de atención .....                 | 30 |
| 2.5.1 Mecanismo de atención de Bahdanau .....   | 31 |
| 2.6 Bases de datos .....                        | 31 |
| 2.6.1 Flickr8k/Flickr30k.....                   | 32 |
| 2.6.2 MSCOCO .....                              | 32 |
| 2.6.3 AIC .....                                 | 32 |
| CAPÍTULO III DISEÑO DEL SISTEMA .....           | 33 |
| 3 Diseño del sistema .....                      | 34 |
| 3.1 Modelo predictivo .....                     | 35 |
| 3.1.1 Traducción y preparación de dataset ..... | 36 |
| 3.1.2 Codificador.....                          | 40 |
| 3.1.3 Mecanismo de atención .....               | 43 |
| 3.1.4 Decodificador .....                       | 44 |
| 3.2 Mecanismos de evaluación.....               | 46 |
| 3.3 Ejecución de rutinas y entrenamiento.....   | 47 |
| 3.4 Interfaz gráfica .....                      | 50 |
| 3.4.1 Implementación .....                      | 51 |
| CAPÍTULO IV PRUEBAS Y RESULTADOS .....          | 57 |
| 4 Pruebas y resultados .....                    | 58 |
| CAPÍTULO V CONCLUSIONES Y RECOMENDACIONES ..... | 64 |
| 5 Conclusiones y recomendaciones .....          | 65 |
| CAPÍTULO VI ANÁLISIS ECONÓMICO.....             | 68 |
| REFERENCIAS .....                               | 70 |
| Glosario de términos y abreviaturas .....       | 75 |
| Anexo 1: Código.....                            | 77 |
| Anexo 2: Especificaciones de ordenador.....     | 91 |
| Anexo 3: Muestra de oraciones generadas .....   | 92 |



# **CAPÍTULO I**

## **ASPECTOS GENERALES**

## Aspectos generales

El área de visión artificial ha avanzado rápidamente con la aplicación de inteligencia artificial y *Deep Learning*; se tienen vehículos semi-autónomos, reconocimiento de rostros, gestos, estados de ánimo, etc. Estos algoritmos han desplazado total o parcialmente a personas que se encargaban de dichas tareas. Una de esas tareas es la descripción y subtítulo de imágenes.

### 1.1 Planteamiento del problema

En el último lustro se han implementado sistemas autónomos de subtítulo de imágenes con excelentes resultados. La mayor parte de estos en inglés y chino, el primero dado que las bases de datos se encuentran en este idioma y el segundo debido a alto interés en dominar la industria.

La aplicación del subtítulo autónomo no debe ser limitada por idioma. Por lo descrito se plantea el estudio e implementación de algoritmos *Deep Learning* para el reconocimiento, y rotulado de imágenes de propósito general, donde los textos generados se encuentren en español.

#### 1.1.1 Trabajos previos

La aplicación de redes neuronales al subtítulo de imágenes data de 2014 donde Kiros, R [12] planteó la utilización de redes convolucionales para esta tarea en reemplazo del uso de árboles semánticos. Los modelos planteados obtienen buenos resultados, generando oraciones inteligibles y acorde a la imágenes. El avance de la materia, sin embargo, no ha representado la implementación de estos para el mundo hispanohablante.

Uno de los pioneros en asumir el subtitulado automático entre distintos lenguajes (Cross lingual image captioning) fueron Miyazaki y Shimizu [18] quienes utilizaron el modelo propuesto por Vinalys 2015 y lo pre-entrenaron utilizando parte de la base de datos en inglés, para su base de datos en japonés utilizaron *crowdsourcing*, es decir múltiples colaboradores produjeron descripciones a imágenes facilitadas por el equipo.

Para generar oraciones en chino Li et al [16] construyeron un modelo donde utilizaron la traducción de texto directa del dataset Flickr8k, la traducción se realizó con las herramientas de Baidu y Google, dado que no se realizó revisión de la traducción del dataset las oraciones generadas por el modelo tienen errores gramaticales y estructuras poco convencionales. Por su lado investigadores de la Universidad de Renmin [14] utilizan un modelo CNN + LSTM y un clasificador neuronal para estimar automáticamente la fluidez de las oraciones de entrenamiento traducidas las que de poseer un coeficiente bajo de fluidez tiene un menor peso en el proceso de entrenamiento.

## 1.2 Hipótesis

Es posible implementar un algoritmo basado en *Deep Learning* capaz de identificar objetos, paisajes, entornos y en base a ellos generar descripciones inteligibles y semánticamente correctas de imágenes no especializadas dentro de la base de datos MSCOCO.

## 1.3 Metodología

De acuerdo a la hipótesis anteriormente indicada se procederá a la investigación de modelos de *Deep Learning* adecuados para la detección de imágenes y generación de oraciones coherentes que describan su contenido. Para esto se deberá entrenar al algoritmo con data relevante y adecuada para la tarea.

Se elige el dataset Common Objects in Context de Microsoft (MSCOCO) dada su extensión de más de 150 000 imágenes con cinco descripciones por cada una de ellas todas generadas por seres humanos, las imágenes de esta base de datos retratan escenas de la vida cotidiana sin abarcar nichos especializados como imágenes médicas o geológicas, por nombrar algunos ejemplos.

El algoritmo será codificado en lenguaje Python con el módulo Tensorflow, el cual domina este campo de investigación. Como hardware se utilizará una GPU Nvidia GTX 1660 Ti.

Se procederá a:

- Implementar algoritmo
- Revisar, evaluar y comparar resultados
- Contrastar hipótesis

## 1.4 Objetivos de la tesis

### 1.4.1 Objetivo general

Implementar un algoritmo de *Deep Learning* que permita subtítular imágenes entregadas, el sistema utilizará procesamiento de lenguaje para generar los subtítulos en español dado que todo el procesamiento se realizará utilizando recursos en idioma anglo.

### 1.4.2 Objetivos específicos

- Reconocer el modelo de *Deep Learning* adecuado para el subtítulado autónomo de imágenes.
- Comprender los distintos métodos de generación automática de subtítulos
- Implementar un codificador para la extracción de características de imágenes.
- Implementar un decodificador de lenguaje capaz de generar oraciones a partir de características extraídas por el codificador.
- Optimizar el uso de recursos computacionales en la implementación y entrenamiento del algoritmo.
- Reconocer la plataforma adecuada para el acercamiento de la tecnología al público.

### 1.4.3 Variables dependientes

- Tiempo de entrenamiento del modelo
- Velocidad de ejecución de algoritmos

- Calidad de subtítulos generados por el modelo.

#### 1.4.4 Variables independientes

- Capacidad computacional del ordenador utilizado para el entrenamiento e implementación, el ordenador debe contar con tarjeta gráfica dedicada con capacidad  $CUDA > 7.5$
- Tamaño de dataset utilizado en el desarrollo del modelo, de este depende la cantidad de información que el modelo puede extraer y luego extrapolar, medido en cantidad de imágenes y oraciones ingresados al algoritmo

#### 1.4.5 Justificación

La implementación de este modelo deja un precedente en la rama de investigación y permite optimizaciones para poder generar una base de datos comparable a MSCOCO completamente en español.



## **CAPÍTULO II BASES TEÓRICAS**

## Bases teóricas

### 2.1 Aprendizaje de máquina

Machine learning o aprendizaje de máquina es una rama de ciencias de la computación y es definido por Tom Mitchel como “El estudio de algoritmos de computación que mejoran automáticamente su rendimiento gracias a la experiencia. Se dice que un programa informático aprende sobre un conjunto de tareas, gracias a la experiencia y usando una medida de rendimiento, si su desempeño en estas tareas mejora con la experiencia.” [17].

Una de las primeras aplicaciones se dio en 1952 cuando Arthur Samuel ideó un juego de damas chinas que aprendía progresivamente de las jugadas de sus usuarios. En 2017 un algoritmo basado en *Deep Learning* era capaz de sobrepasar a modelos tradicionales. El aprendizaje de máquina puede dividirse en dos grandes enfoques, aprendizaje supervisado y no supervisado [36].

#### 2.1.1 Aprendizaje supervisado

El aprendizaje supervisado es el que predomina en las aplicaciones actuales, se considera como tal cuando se tiene una variable de entrada ( $x$ ) y una salida ( $y$ ) y se usa un algoritmo para obtener la función que permita obtener la salida deseada [33][26].

$$y = f(x) \quad (1)$$

El objetivo es aproximar la función de tal forma que se aproxime a la función original y en base a ello sea capaz de predecir la salida aún cuando se le provean nuevas variables de ingreso. En el proceso de entrenamiento se alimenta al algoritmo con datos y sus respectivos resultados esperados, el algoritmo itera hasta obtener las predicciones deseadas; se detiene el proceso de aprendizaje en cuanto el algoritmo alcanza niveles aceptables de rendimiento [33][26].

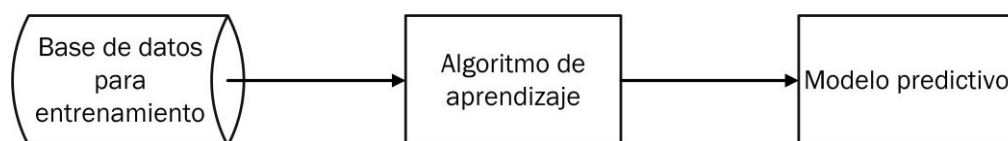


Figura 1: Diagrama de bloques de generación de modelo

*Fuente: Elaboración propia*

### 2.1.2 Aprendizaje no supervisado

Se considera aprendizaje no supervisado cuando sólo se alimenta al algoritmo con datos ( $x$ ) y no se le proporciona las salidas deseadas. El algoritmo deberá descubrir la estructura de los datos por su cuenta y presentar sus salidas. [20] Las principales aplicaciones del aprendizaje no supervisado son:

- **Agrupación:** Requiere descubrir conexiones inherentes dentro de la data, por ejemplo agrupar vehículos de acuerdo al tipo de combustible utilizado.
- **Asociación:** Requiere descubrir reglas que describen grandes porciones de datos, por ejemplo las personas que compran artículos X tienden a comprar artículos Y.

## 2.2 Redes neuronales artificiales

Una red neuronal artificial es una red de neuronas multicapa la cual cuenta con capa de ingreso, capa de salida y capas ocultas, está construida con una estructura similar al cerebro humano, cada neurona es responsable de procesar la información y transmitirla a otras capas. Una red neuronal debe pasar por un proceso de entrenamiento donde aprende a reconocer patrones, ya sean visuales, textuales o numéricos. Durante esta etapa el algoritmo compara sus resultados con los deseados y realiza los ajustes necesarios para aproximar su respuesta.

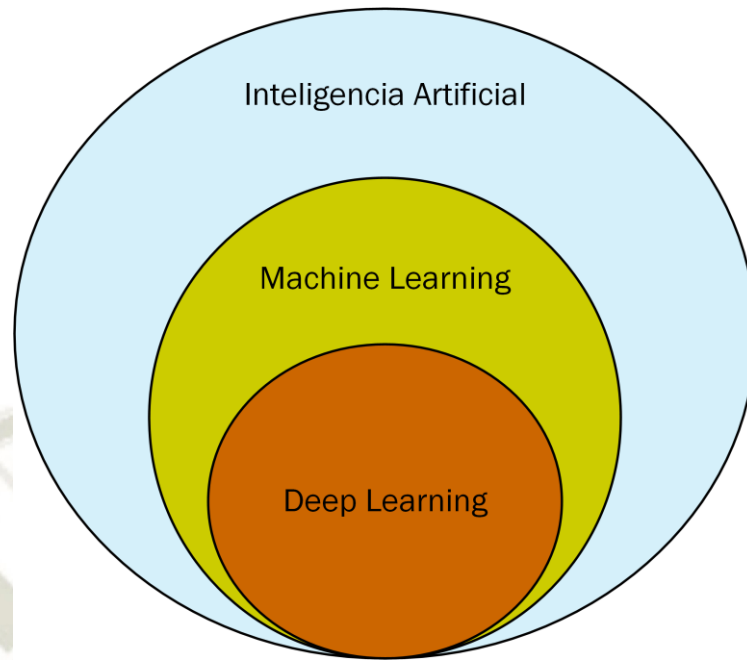


Figura 2: Jerarquía simplificada entre AI, ML y DL [21]

Formalmente una red neuronal es un modelo matemático para el procesamiento de información. Por el lado implementativo considerar que una red neuronal no es un programa específico sino un modelo capaz de procesar información o datos de entrada, sus características son:

- El procesamiento de información se da en células más simples llamadas neuronas.
- Las neuronas están conectadas entre sí, de tal manera que comparten información entre ellas.
- Las conexiones entre neuronas pueden ser fuertes o débiles, de ello dependerá la forma de cómo se procesa la información.
- Cada neurona tiene un estado interno el cual es determinado por las conexiones con otras neuronas.
- Cada neurona tiene una función de activación, que es calculada por sí misma y que determina la salida.

### 2.2.1 Neurona artificial

Una neurona es una función matemática que toma una o más entradas y produce una salida única.

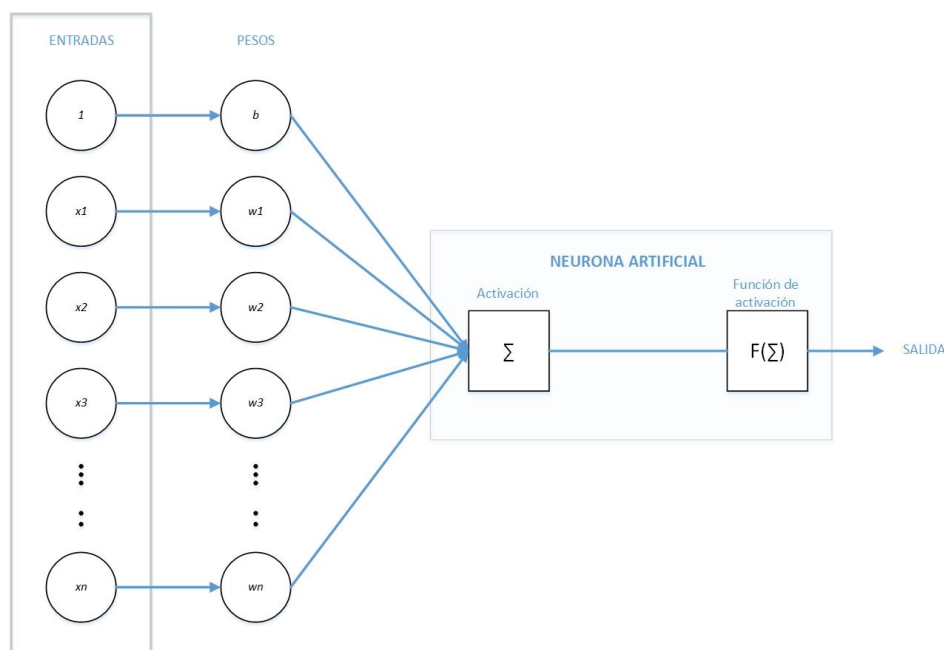


Figura 3: Diagrama de elementos de neurona artificial

*Fuente: Elaboración propia*

La neurona artificial puede definirse como:

$$y = f\left(\sum_i x_i w_i + b\right) \quad (2)$$

En esta ecuación  $x_i$  es el valor numérico que representa los datos de ingreso o las salidas de otras neuronas (parte de la red). Los pesos  $w_i$  son valores numéricos que representan la importancia de los datos de ingreso o importancia de la conexión entre neuronas. El valor  $b$  es llamado *bias* el cual es un elemento de ajuste en la función.

El resultado de la sumatoria de los valores anteriormente detallados se ingresan a la **función de activación**. Existen diferentes tipos de funciones de activación, todas deben cumplir con ser no lineales.

### 2.2.2 Capas de una red neuronal

Las neuronas artificiales se agrupan en capas para obtener mejores resultados, estas capas tienen diferentes funciones como filtrado, normalización o convolución. El conjunto organizado de capas conforma una red neuronal y puede contar con un número indefinido

de capas interconectadas. La capa de entrada representa la data de entrenamiento y las condiciones iniciales. En tareas de visión artificial si el ingreso es una imagen a escala de grises la salida de cada neurona en la primera capa corresponde a la intensidad de un píxel de la imagen. Normalmente la primera capa no es contada como parte de la red. Las neuronas de una cada capa pueden estar conectadas a neuronas de otras capas pero no a neuronas de la misma capa.

La necesidad de organizar las neuronas en capas se debe a que una neurona lleva información limitada (solo un valor). Al agrupar neuronas en capas sus salidas componen un vector, el cual de por si lleva más información no solo por la cantidad sino la relación entre los valores como un todo es también información relevante para una predicción acertada.

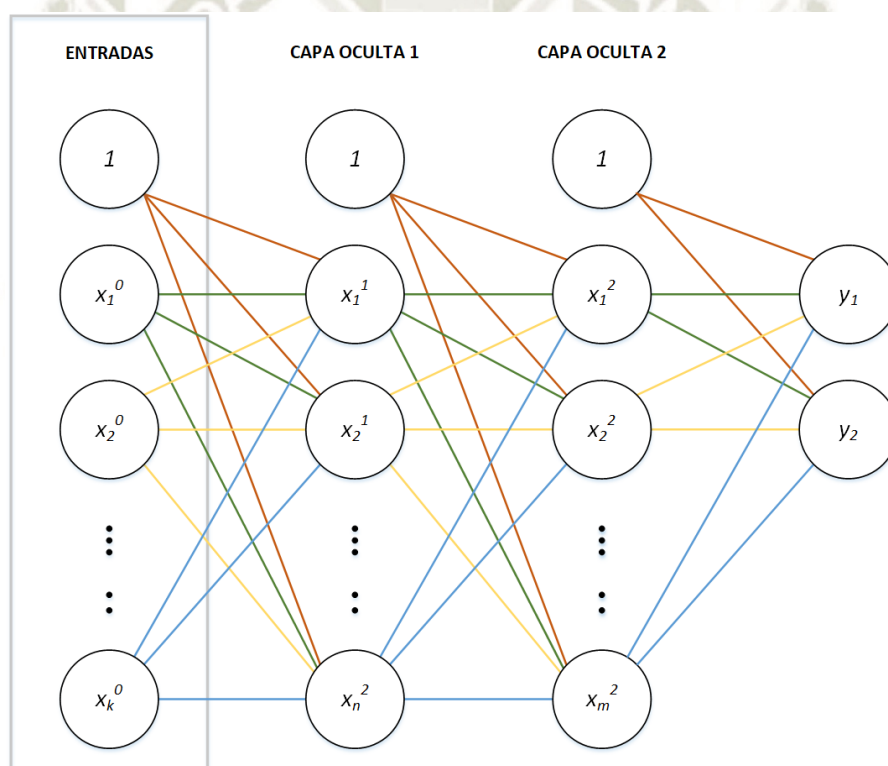


Figura 4: Red neuronal multicapa secuencial

*Fuente: Elaboración propia*

Las capas adicionales son llamadas *hidden layers* o capas ocultas. La imagen 4 muestra una red neuronal con 3-capas *fully connected* o totalmente conectadas donde dos de ellas se consideran ocultas. La capa de entrada tiene  $k$  neuronas de entrada, la primera capa

oculta tiene  $n$  neuronas ocultas y la segunda capa oculta cuenta con  $m$  neuronas. La salida, en el gráfico, son dos clases  $y_1$ ,  $y_2$ , en la parte superior se observa el *bias*. Cada neurona está conectada a todas las previas, por ello el nombre de totalmente conectadas. Cada conexión tiene su propio peso  $w$  no mostrado en el gráfico.

Una red neuronal puede contar con tres tipos de capas, capas de entrada que toman los datos brutos, capas ocultas que toman información de otras capas y de sí mismas, y capas de salida que realizan la predicción.

### 2.2.3 Funciones de activación

La función de activación es la que determina cómo se transformará la suma de *pesos* de las entradas para convertirse en salidas. La elección de la función de activación tiene gran impacto en las capacidades y rendimiento de la red neuronal, se puede utilizar diferentes funciones de activación en diferentes etapas del modelo.

Existen varios tipos de funciones de activación utilizables sin embargo solo un grupo pequeño es usado en aplicaciones prácticas.

#### 2.2.3.1 Sigmoidal

También llamada función logística, la cual convierte cualquier valor de entrada en salidas entre 0 y 1. Cuanto más positiva sea la entrada, la salida se acercará más a 1.0, cuanto más negativa la entrada la salida será más cercana a 0.0 [6]

La función Sigmoidal es:

$$f(x) = \frac{1}{1 + e^x} \quad (3)$$

Gráficamente:

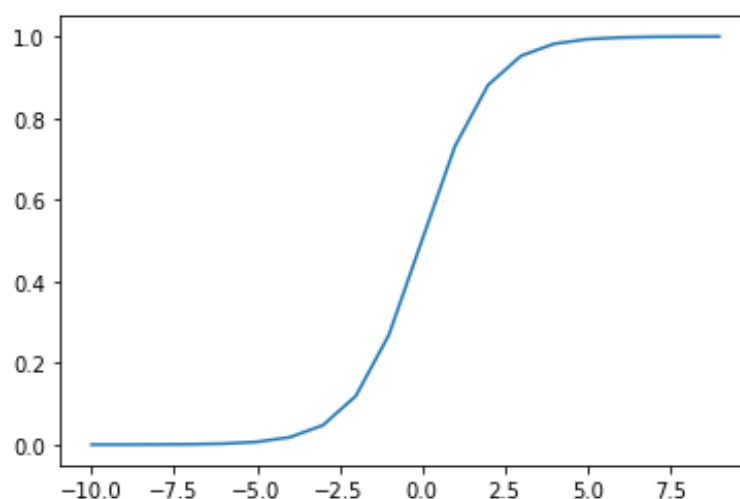


Figura 5: Gráfico de función Sigmoidal

*Fuente: Elaboración propia*

Esta función se usa normalmente en las capas de salida, se evita utilizar en las capas ocultas dado que en caso el valor de  $x$  sea mayor a 2 o menor a  $-2$  la salida será muy cercana a 1 y 0 respectivamente. Esto obstaculiza la habilidad de aprendizaje de la red y la ralentiza de manera considerable [6] [11].

#### 2.2.3.2 Tangente hiperbólica

Conocida comúnmente como  $Tanh(x)$ , su comportamiento es similar a la función sigmoidea, cuanto más positiva sea la entrada, la salida se acercará más a 1.0, cuanto más negativa la entrada la salida será más cercana a -1.0 [6].

La función tangente hiperbólica es:

$$f(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} \quad (4)$$

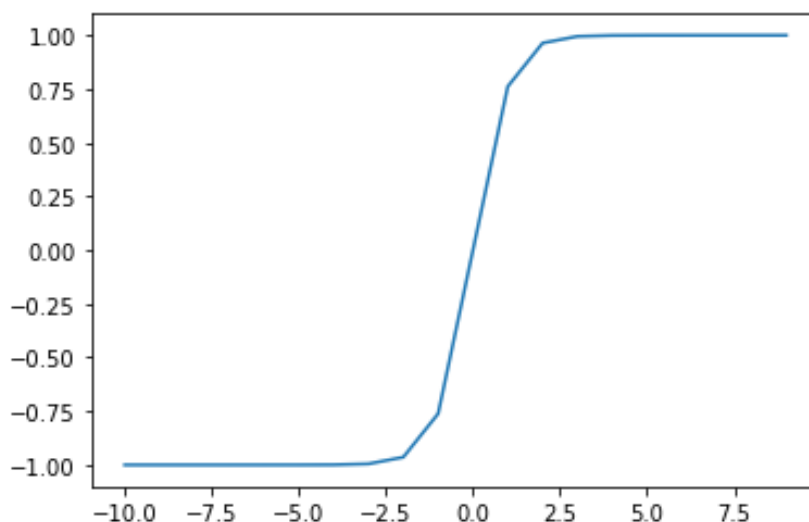


Figura 6: Gráfico de función Tangente hiperbólica

*Fuente: Elaboración propia*

### 2.2.3.3 Softmax

La función de activación Softmax normaliza un vector contenedor de K elementos e una distribución de probabilidades sobre K elementos. Por este motivo normalmente es utilizado en capas de salida para predecir la probabilidad de ser una clase deseada [6].

La función Softmax es:

$$f(x) = \frac{e^{x_i}}{\sum_{k=1}^K e^{x_k}} \quad (5)$$

### 2.2.3.4 RELU

Rectified linear unit, comúnmente utilizada en capas ocultas, la facilidad de implementación computacional la hacen preferible a funciones anteriormente revisadas [6].

La función ReLU es:

$$f(x) = \max(0, x) = \begin{cases} 0 & \text{si } x < 0 \\ x & \text{si } x \geq 0 \end{cases} \quad (6)$$

Gráficamente:

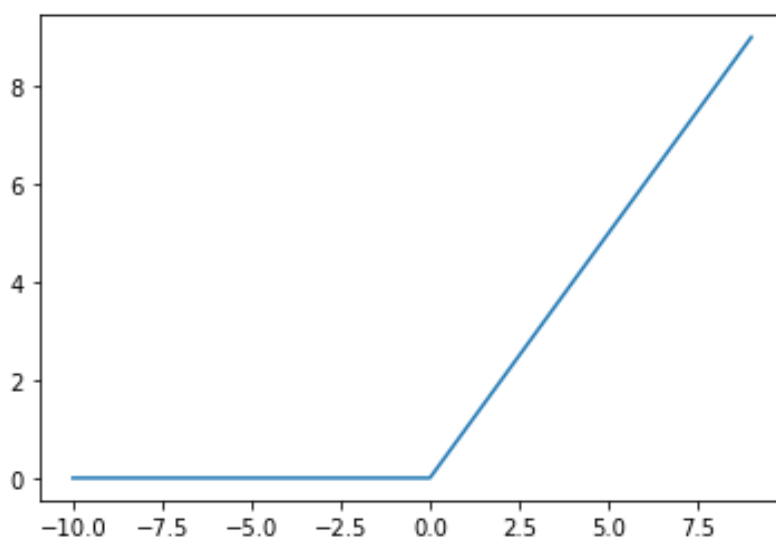


Figura 7: Gráfico de función ReLU

*Fuente: Elaboración propia*

Existen variaciones a la función ReLU las cuales se implementaron para incrementar la velocidad de entrenamiento de los modelos así como su rendimiento. Estos son Leaky ReLU, Parametric ReLU y Exponential Linear unit (ELU) [6].

#### 2.2.4 Entrenamiento de redes neuronales

Una vez definidos los parámetros anteriormente detallados; arquitectura, número de capas ocultas, número de neuronas por capa y las funciones de activación el modelo está listo para ser entrenado, este proceso consiste en la actualización iterativa de los valores internos. Tengamos en cuenta que toda red neuronal es una aproximación a función por lo que cada red neuronal no será exactamente la función deseada y tendrá un error en referencia a esta. En el proceso de entrenamiento el objetivo es reducir el valor de este error.

De manera simplista se puede reducir el proceso de aprendizaje al encontrar mínimos absolutos de las funciones requeridas y en base a estos calcular los coeficientes requeridos para recrear la función. A continuación se detallan los métodos para lo antes mencionado así como otros conceptos involucrados en este proceso [3].

- **Gradient descent:**

El descenso de gradiente es un método de optimización utilizado para obtener los

valores de los parámetros (coeficientes) de una función ( $f$ ) de tal manera que estos minimizan el costo de la función. En un modelo de aprendizaje de máquina el costo se calcula sobre toda la data de entrenamiento para cada iteración del algoritmo de descenso de gradiente. Una iteración del algoritmo es llamado un *Batch*. Entre los métodos más conocidos para realizar esta operación están la regresión lineal y la regresión logística [3].

El gradiente de una función en un punto  $x$  es el vector:

$$\frac{\partial C}{\partial x} = \left[ \frac{\partial C}{\partial x_1}, \frac{\partial C}{\partial x_2}, \dots, \frac{\partial C}{\partial x_m} \right] \quad (7)$$

- **Backpropagation:**

Uno de los principales elementos de una red neuronal, en resumen se encarga de retroesparcir a cada capa los valores aprendidos en cada iteración. Ello permite ajustar los valores (pesos) de las conexiones en la red con la finalidad de minimizar la diferencia entre la salida del vector de la red y la esperada [24]. En modelos de aprendizaje de máquina los valores a ajustar son los *pesos* y los *bias*.

## 2.3 Convolutional Neural Network (CNN)

Una Red Neuronal Convolutiva es la siguiente etapa en la evolución de las ANN. Cuentan con datos de entrada, pesos y *bias* tal como las redes neuronales tradicionales pero adicionan capas convolucionales en su estructura.

Las CNN se inspiran en los hallazgos realizados por Hubel y Wiesel en sus experimentos en la corteza visual de gatos. Ellos propusieron la existencia de células simples y complejas, las células simples se activan cuando el gato observa formas simples, las células complejas combinan la información de células simples y se activan cuando se detecta rotaciones o desplazamientos en el objeto original. En base a estas observaciones en 1998 Le Cun, Botton Bengio y Haffner propusieron las CNN [15].

Pese a que las CNN fueron propuestas en la década de los 90 no era posible utilizarlas dado que requiere gran cantidad de datos para su entrenamiento y por consecuencia alta capacidad computacional. El desarrollo de procesadores gráficos avanzados y la disponibilidad de gran cantidad de data permiten que las CNN sean implementadas en el mundo contemporáneo.

La convolución, que da nombre a este tipo de red neuronal, es la operación entre dos funciones que produce una tercera modificada. como operación está definida como:

$$s(t) = \sum_a I(a) \cdot K(t - a) \quad (8)$$

Contextualizando esto para visión artificial el primer elemento es la imagen de entrada y el segundo un filtro dedicado a la extracción de características.

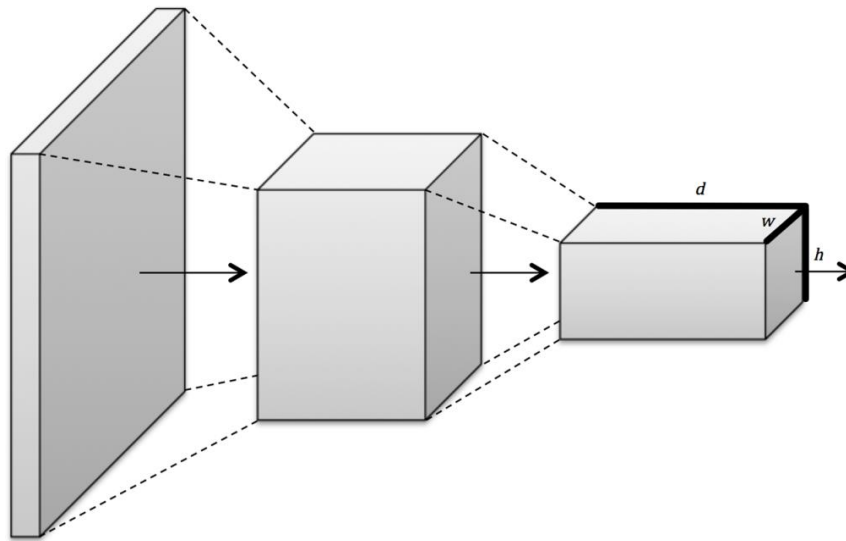


Figura 8: Variación de forma de data entre cada capa de CNN [3]

La arquitectura de una CNN comprende: Capas convolucionales o de filtrado(CONV), capas de pooling (POOL) y capas *fully connected* (FC) las cuales se detallan a continuación.

### 2.3.1 Capa de Convolucion

Esta capa encargada de filtrar las imágenes permite aislar características especiales dentro de estas. Para realizar esta detección se debe utilizar filtros diferentes para cada característica , en la 9 se ejemplifica la detección de horizontal o vertical, la información obtenida genera un mapa de características para cada filtro.

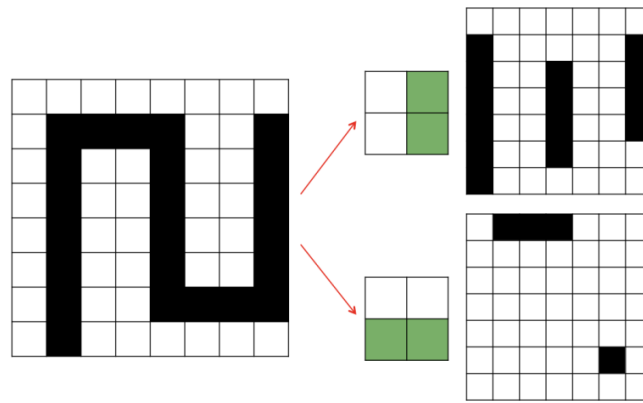


Figura 9: Uso de filtros para determinar orientación [3]

Denotamos el mapa de características  $h^t$  es una capa  $m$  como  $m^k$ . El filtro con los respectivos pesos es  $W$ , luego asumiendo que las neuronas del mapa de características tiene un *bias*  $b^k$ , el mapa puede expresarse como:

$$m_{ij}^k = f((W * x)_{ij} + b^k) \quad (9)$$

La ecuación describe de manera general a un filtro monodimensional, sin embargo, en aplicaciones de visión artificial los filtros deben operar, incluso, sobre los mapas de características generados en capas anteriores y de manera tridimensional por la naturaleza de las imágenes RGB. En el proceso de entrenamiento los valores asignados a estos filtros se actualizarán hasta un punto óptimo.

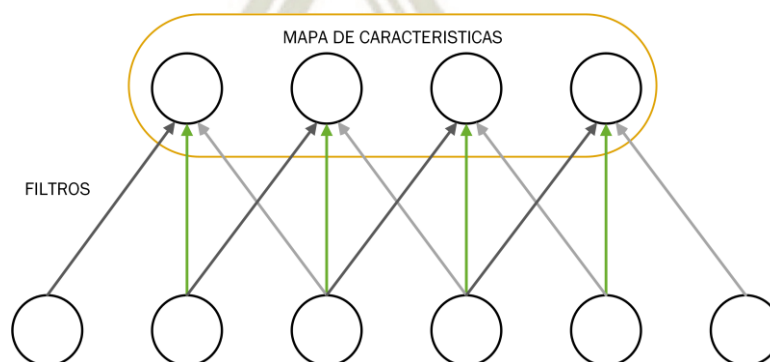


Figura 10: Mapa de características como red neuronal

*Fuente: Elaboración propia*

## 2.3.2 Capa de Pooling

Agregar más capas convolucionales incrementa la profundidad de la red neuronal lo que significa incremento en la cantidad de parámetros que la red debe optimizar. Cuanto mayor sea el número de parámetros a optimizar el proceso de aprendizaje de la red será más largo, las capas de pooling surgen como solución a ello dado que se encargan de reducir el número de parámetros a transferir a la siguiente capa. Este proceso se realiza aplicando una función estadística, como promedio o máximo para restringir los datos a transmitir a las subsiguientes capas. El objetivo es submuestrear las características producidas por las capas convoluciones encargadas de extraerlas de tal manera que se reduce la complejidad computacional, es práctica común incluir capas de pooling después de cada capa convolucional [8].

### 2.3.2.1 Max Pooling

La función de *max pooling* se desplaza a través del mapa creado en la capa convolucional y selecciona el valor de píxel máximo y reteniendo el valor para la siguiente etapa [8].

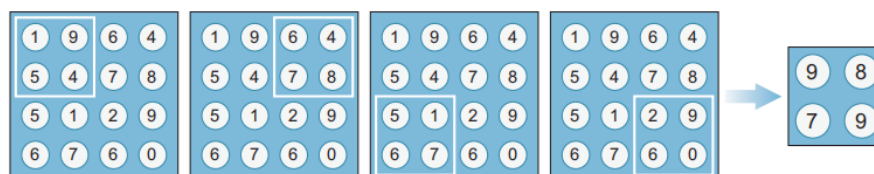


Figura 11: Filtro *Max pooling* 2 x 2, stride 2, reduce la matriz de 4x4 a 2x2 [7]

Una capa de pooling puede describirse a partir de dos parámetros:

- Su extensión espacial  $e$
- Su *stride* o pasos de desplazamiento  $s$

las dimensiones resultantes del mapa de características son:

$$\text{Ancho } w_{out} = \frac{w_{in}-e}{s} + 1$$

- Alto  $h_{out} = \frac{h_{in}-e}{s} + 1$

### 2.3.2.2 Average Pooling

Este tipo *subsampler* puede reducir el tamaño de la capa anterior en más de 2

dimensiones, para ello calcula el promedio de valores de todos los píxeles de la capa. Con ello puede tomar una matriz 3D y retorna un vector [6].

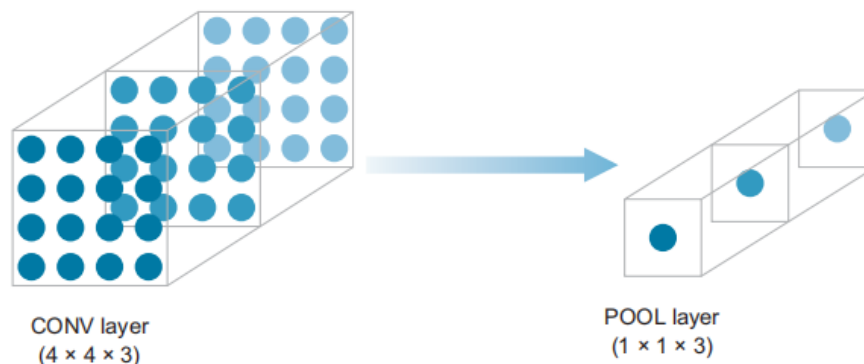


Figura 12: Filtro *Average pooling* retornando un vector a partir de matriz tridimensional [7]

### 2.3.3 Arquitecturas CNN

Las CNN son voluminosas en profundidad y requieren reiteradas calibraciones para elevar su nivel de respuesta, desde 2010 investigadores y empresas del rubro remiten sus mejores algoritmos de reconocimiento, selección y clasificación de imágenes al *ImageNet Large Scale Visual Recognition Challenge* (ILSVRC) donde el ganador actualiza el estado del arte en la materia. Hasta 2011 los algoritmos remitidos alcanzaban un margen de error top-5 de 25% utilizando técnicas convencionales de visión artificial o *shallow networks*, en 2012 se propuso la primera CNN para afrontar el desafío y obtuvo un error top-5 cercano al 15%, muy superior a sus contrapartes.

Los algoritmos remitidos desde tal fecha son referencia en precisión y calidad, siendo utilizados como punto de partida para otras investigaciones. A continuación se detallan las arquitecturas CNN destacadas.

#### 2.3.3.1 AlexNet

CNN diseñada por Alex Krizhevsky [13] ganadora de ImageNet Large Scale Visual Recognition Challenge donde el error del algoritmo alcanzó 15.3%, el segundo mejor alcanzó un error de 26.3% siendo un hito en el momento marcando una nueva tendencia que se mantiene hasta ahora.

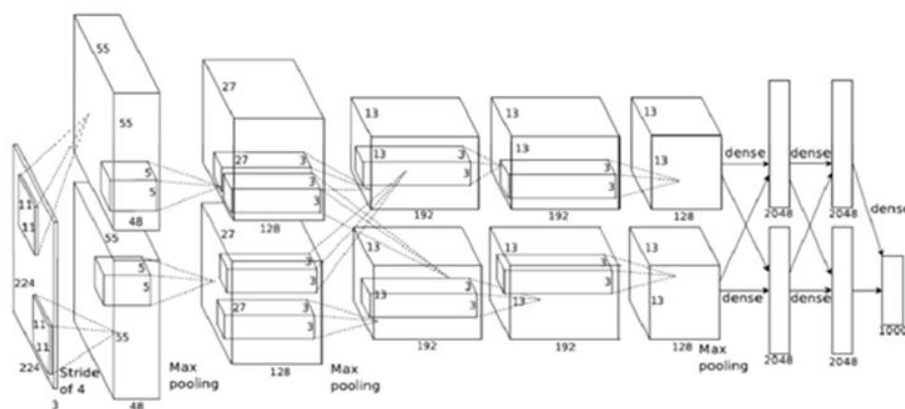


Figura 13: Estructura de AlexNet [25]

La arquitectura de AlexNet consta de cinco capas convolucionales, tres capas de max-pooling, dos capas de normalización, dos capas fully-connected y una capa de softmax. Como rectificador utilizó una función no lineal de activación ReLU. De acuerdo a los creadores el modelo requería de todas sus capas para su alto rendimiento siendo exigente en recursos de procesamiento, esto fue superado utilizando GPUs en el entrenamiento, siendo a su vez pioneros en ello. El modelo fue entrenado con 15 millones de imágenes en una tarjeta gráfica GTX 580 por cinco días.

En 2013 se proclamó ganadora la ZFnet [35] propuesta por Zeiler y Fergus, implementando mejoras y calibraciones en ciertos aspectos de AlexNet. El error conseguido por este modelo llegó a 11.2%. Pese a que su arquitectura varía poco con respecto a AlexNet el modelo solo fue entrenado con 1.3 millones de imágenes en tarjeta GTX 580 por veinte días. Utilizaron ReLU como función de activación.

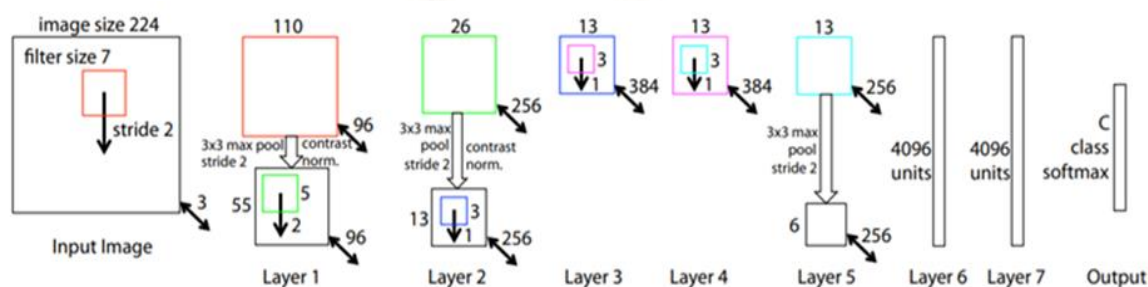


Figura 14: Estructura de ZFnet [35]

### 2.3.3.2 VGGnet

VGGNet [27] propuesta en 2014 por Simonyan *et al.* Su arquitectura se basa en Alexnet modificando los filtros de las primeras capas. Alcanzó un error de 7.3% en top-5 quedando en segundo lugar después de GoogleNet. Existen múltiples variantes como VGG16, VGG19, etc las cuales sólo difieren entre sí por el número de capas en la red neuronal.

### 2.3.3.3 GoogleNet /Inception

GoogLeNet o Inception fue el ganador del ILSVRC 2014 con error top-5 de 6.7% [29]. Este modelo fue pionero en separarse de la arquitectura de AlexNet planteando un paquete de 22 convoluciones en un nuevo módulo llamado *Inception Module*. Este modelo está compuesto de convoluciones paralelas y diferentes dimensiones de max-pooling.

- **Convolucion 1 x 1:** Este tipo de convolución es utilizado en GoogleNet para reducir el número de parámetros. La convolución se da entre la imagen de entrada y el filtro conv  $1 \times 1$  para crear una salida de dimensiones  $1 \times 1 \times n$  donde  $n$  es el número de filtros. Este filtro no extrae características en el plano pero si lo realiza en profundidad de la imagen (cruce de canales). Por ello el filtro  $1 \times 1$  no solo reduce dimensiones sino permite mayor aprendizaje a la red neuronal.

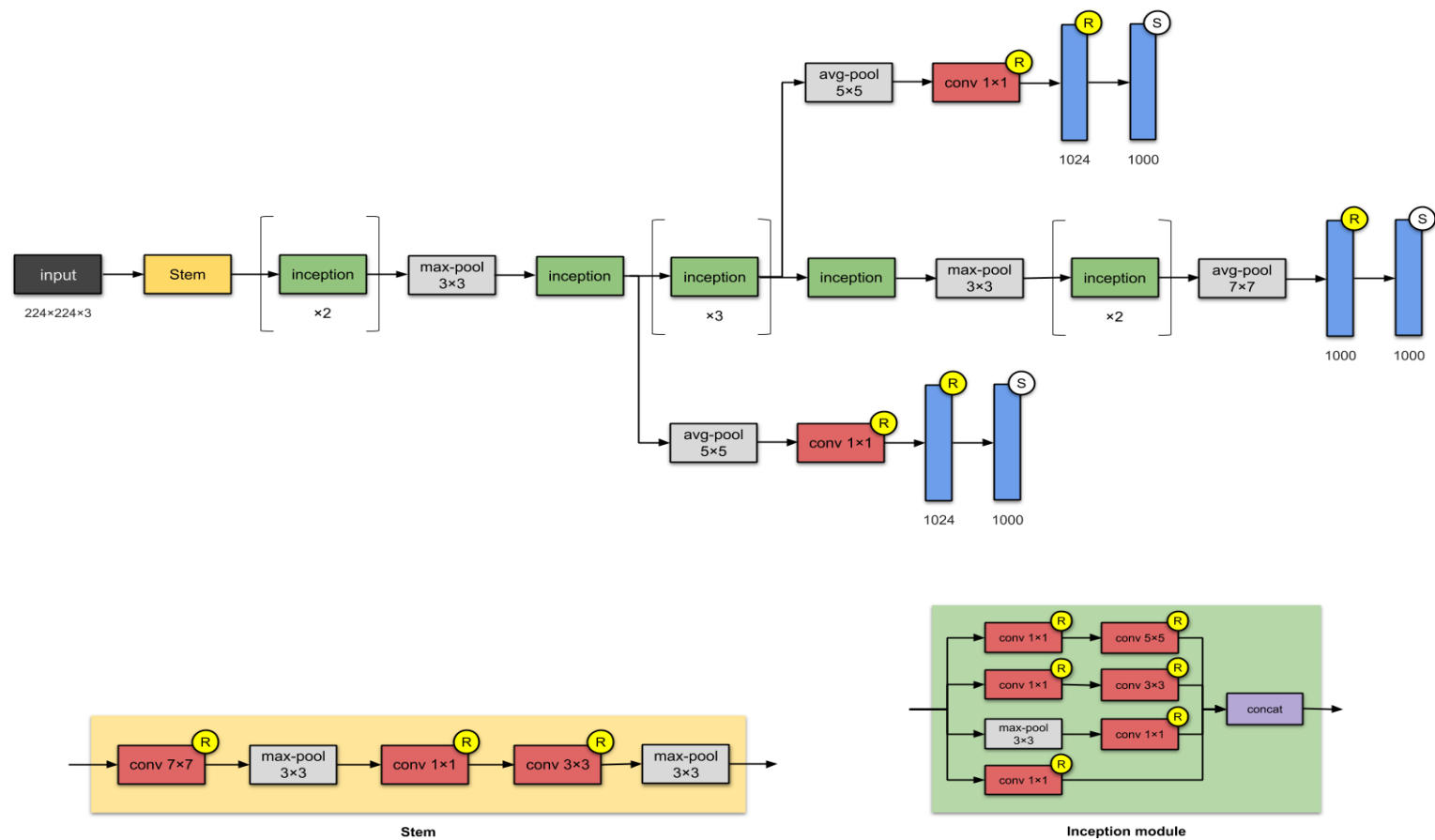


Figura 15: Estructura de Inception V1, extraído de [10] basado en [29]

- **Global Average Pooling:** En anteriores arquitecturas las capas *fully connected* se utilizan en la parte final del modelo; estas capas contienen gran parte de los parámetros extraídos en capas anteriores, que por su número implican alto coste computacional. En GoogLeNet se utiliza el *Global averga pooling* en la parte final de la red, esta capa toma un mapa de características de  $7 \times 7$  y lo reduce a  $1 \times 1$ , de tal forma se reduce la la cantidad de parámetros a tener en cuenta en el entrenamiento.
- **Módulo Inception:** Consta de convoluciones  $1 \times 1$ ,  $3 \times 3$ ,  $5 \times 5$  y max pooling de  $3 \times 3$  calculados en paralelo, los resultados son concatenados a la salida de cada módulo. El objetivo de este módulo es mejorar el rendimiento del algoritmo al reducir el número de parámetros a operar.

## 2.4 RNNs y procesamiento de lenguaje

Una de las ramas más exploradas de la inteligencia artificial es el procesamiento de imágenes, esta aborda la manipulación y filtrado necesario para extraer información de estas. Otra rama con popularidad es el Procesamiento Natural de Lenguaje, **NLP** por sus siglas en inglés, que aborda las técnicas necesarias para que las máquinas puedan comprender y generar textos con coherencia [28].

En esta sección se aborda la teoría en el diseño y manipulación de modelos requeridos para la implementación. En líneas generales se utiliza una estructura Codificador-Decodificador donde la tarea codificador la asume la CNN dedicada a la extracción de características, el decodificador es una RNN que realiza el modelado de la oración palabra por palabra, ambas deben estar interconectadas.

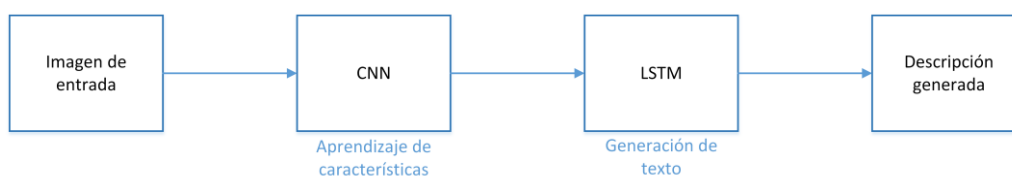


Figura 16: Estructura codificador - decodificador

*Fuente: Elaboración propia*

En el presente capítulo se abordan las principales técnicas y arquitecturas utilizadas

para la generación de descripciones de manera automática.

### 2.4.1 RNNs para el modelado de secuencias

En algoritmos típicos de aprendizaje de máquina se asume que la data de ingreso es independiente e idénticamente distribuida (**IDD**), es decir el orden en el cual se alimente la data al modelo no es relevante para los resultados. Las secuencias, por el contrario, dependen del orden de sus elementos. El modelado de secuencia tiene como principales aplicaciones la traducción, subtítulo automático y generación de textos.

Las secuencias se representan como:

$$x^{(1)}, x^{(2)}, \dots, x^{(T)} \quad (10)$$

Los exponentes indican el orden de las instancias y la longitud de la secuencia lo indica  $T$ . Es posible pensar en una secuencia como serie dependiente del tiempo donde cada punto de muestreo  $x^{(t)}$  corresponde a un momento en particular ( $t$ ) [22].

La imagen 17 detalla esta analogía donde  $x$  e  $y$  fluyen de acuerdo al eje de tiempo. Tanto  $x$  e  $y$  son secuencias.

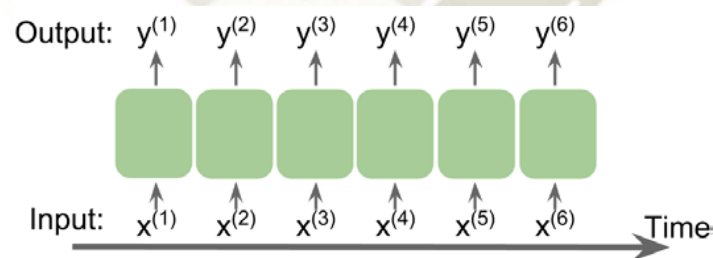


Figura 17: Secuencia dependiente del tiempo [22]

Los modelos redes neuronales presentados en el capítulo anterior, como CNNs, no son capaces de manejar esta clase de datos, para el tratamiento de estos se introducen las RNN (Recurrent Neural Networks).

Antes de discutir las características de las RNN definimos los tipos de relaciones, que de acuerdo a Ravichandiran [23] pueden dividirse en tres grupos:

- **Varias-a-una:** La data de entrada es una secuencia pero la salida es un vector de tamaño definido, no una secuencia. Por ejemplo, ingresa un texto y la salida es una etiqueta de clase.
- **Una-a-varias:** La data de entrada es estándar, no una secuencia, pero la salida es una secuencia. El subtítulo automático de imágenes es muestra de este tipo dado que se alimenta al algoritmo una imagen y la salida es una oración.
- **Varias-a-varias:** Tanto la entrada como la salida son secuencias. Esta categoría puede subdividirse de acuerdo a si la salida está o no sincronizada a las entradas. La clasificación en video es un ejemplo de la categoría sincronizada dado que a cada cuadro de video le corresponde una descripción. La traducción de textos es ejemplo de la categoría no síncrona, la data de entrada es una oración, cada palabra debe ser leída y procesada antes de generar la traducción.

## 2.4.2 Estructura y flujo en RNNs

La figura 18 muestra una comparativa entre una red neuronal estándar y una RNN, en ambos casos solo cuentan con una capa oculta, tanto los datos de entrada como de salida están compuestos por vectores. En una red neuronal estándar la información fluye en un solo sentido, de la capa de ingreso a la capa oculta y finalmente la capa de salida; por otro lado, en una RNN la capa oculta recibe información de la capa de entrada y de la capa oculta del proceso anterior.

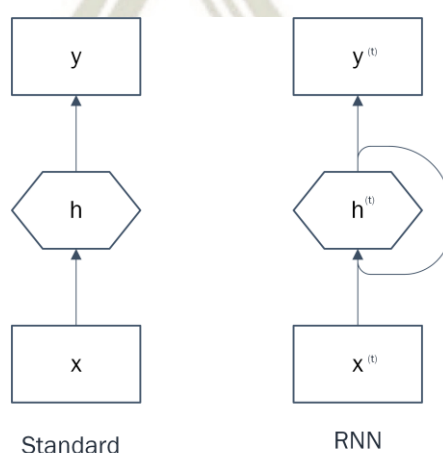


Figura 18: Comparación del flujo de datos entre una ANN y RNN

*Fuente: Elaboración propia*

El flujo de información entre procesos adyacentes en tiempo permite que la red neuronal tenga memoria de eventos pasados. Este flujo de información es representado mediante un *loop* o un bucle. Para examinar a detalle el flujo de información es posible desglosar la anterior representación como se muestra a continuación.

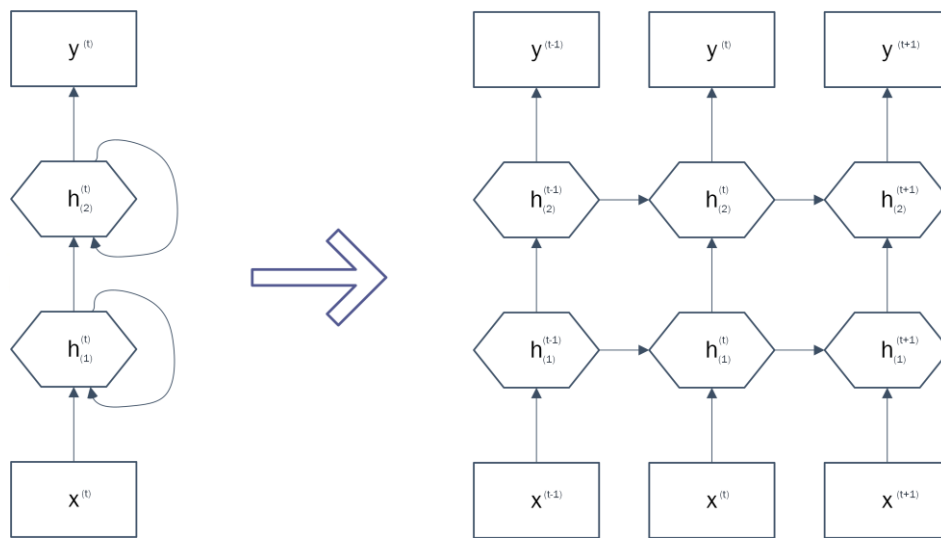


Figura 19: Desglose de una RNN

Fuente: Elaboración propia

En el primer paso de tiempo  $t = 0$ , los valores de la capa oculta son inicializados con valores al azar cercanos a cero. En los siguientes pasos, cuando  $t > 0$  las capas ocultas reciben información de la capa de entrada definida en ese tiempo  $x^{(t)}$  y de los valores previos de la capa oculta en  $t - 1$ , denotado como  $h^{(t-1)}$  [23].

Para denotar formalmente las ecuaciones de una RNN definimos:

- $W_{xh}$ : Matriz de pesos entre la capa de entrada  $x^{(t)}$  y la capa oculta  $h$
- $W_{hh}$ : Matriz de pesos asociados al bucle
- $W_{hy}$ : Matriz de pesos entre la capa oculta y la capa de salida

De acuerdo a ello la ecuación de valores de la capa oculta en un tiempo  $t$  es:

$$h^{(t)} = \phi_h(W_{xh}x^{(t)} + W_{hh}h^{(t-1)} + b_h) \quad (11)$$

$b$  representa el *bias* de la capa oculta y  $\phi_h(\cdot)$  es la función de activación.

Para el entrenamiento de este tipo de red neuronal se utiliza BTT (Backpropagation through time). Teóricamente realizar este procedimiento no altera los datos en el proceso de transferencia de información en el tiempo sin embargo, computacionalmente el proceso es exigente y exhibe pérdidas. Para solucionar este problema se implementaron las técnicas de TBPTT (Truncated backpropagation through time) y LSTM (Long short-term memory) .

### 2.4.3 Unidad LSTM

El término fue introducido por Hochreiter y Schmidhuber[9]. Un bloque de LSTM tiene una célula de memoria, que representa la capa oculta. El flujo de información en este bloque es controlado por las unidades  $\oplus$  *element-wise summation* y  $\otimes$  *element-wise product*. En una célula LSTM existen tres diferentes tipos de compuertas; compuerta de olvido, compuerta de entrada y compuerta de salida.

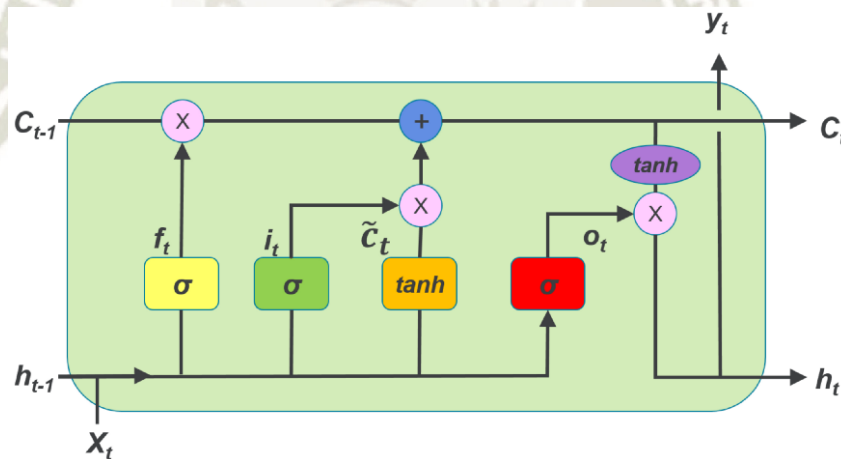


Figura 20: Diagrama neurona LSTM [31]

La compuerta de olvido ( $f$ ) permite a la célula de memoria reiniciar su estado evitando que crezca de manera indefinida. Esta compuerta decide qué información continuará en el proceso o si debe ser eliminada,  $f$  se denota como:

$$f_t = \phi(w_f[h_{t-1}, x_t] + b_f) \quad (12)$$

- La compuerta de entrada ( $i_t$ ) es la responsable de actualizar el estado de la célula.

$$i_t = \phi(w_i[h_{t-1}, x_t] + b_i) \quad (13)$$

El nodo de ingreso es:

$$\tilde{c}_t = \tanh(w_c[h_{t-1}, x_t] + b_c) \quad (14)$$

- La compuerta de salida ( $o_t$ ) decide cómo actualizar los estados de las unidades ocultas.

$$o_t = \phi(w_o[h_{t-1}, x_t] + b_o) \quad (15)$$

$b$  hace referencia al *bias* para cada una de las compuertas.

De acuerdo a las definiciones realizadas el estado de la célula en un momento  $t$  es

$$c_t = f_t \otimes c_{t-1} \oplus i_t \otimes \tilde{c}_t \quad (16)$$

Los valores de una capa oculta en un determinado momento son:

$$h_t = o_t \otimes \tanh(c_t) \quad (17)$$

#### 2.4.4 Unidad GRU

La GRU o *Gated Recurrent Unit* es una variante de la neurona LSTM introducida por Cho et al en 2014 [4]. En este modelo se combinan las compuertas de olvido y entrada en una sola llamada *update gate* o compuerta de actualización. El resultado es un modelo más simple, de mayor facilidad de implementación y menor exigencia computacional.

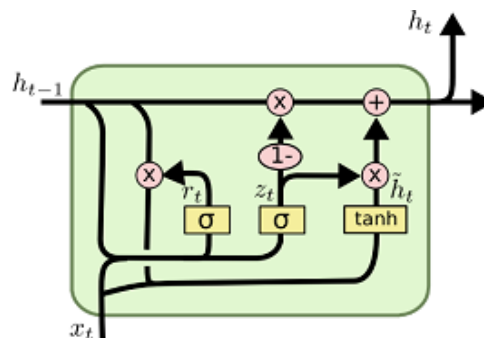


Figura 21: Diagrama neurona GRU [5]

Sus compuertas están representadas por:

$$z_t = \sigma(W_z \cdot [h_{t-1}, x_t y]) \quad (18)$$

$$r_t = \sigma(W_r \cdot [h_{t-1}, x_t]) \quad (19)$$

$$\tilde{h}_t = \tanh(W \cdot [r_t * h_{t-1}, x_t]) \quad (20)$$

$$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t \quad (21)$$

Donde  $r$  es *reset gate*,  $Z$  *update gate*, *reset gate* determina como serán combinados los nuevos valores con los mantenidos en memoria, la *update gate* determina cuánta información del estado anterior será retenida.

## 2.5 Mecanismo de atención

El mecanismo de atención nace del estudio de la visión humana la cual focaliza un objeto deliberadamente por su color o forma. Esta habilidad de auto-selección es llamada atención. La técnica fue originalmente propuesta para la traducción por inteligencia artificial.

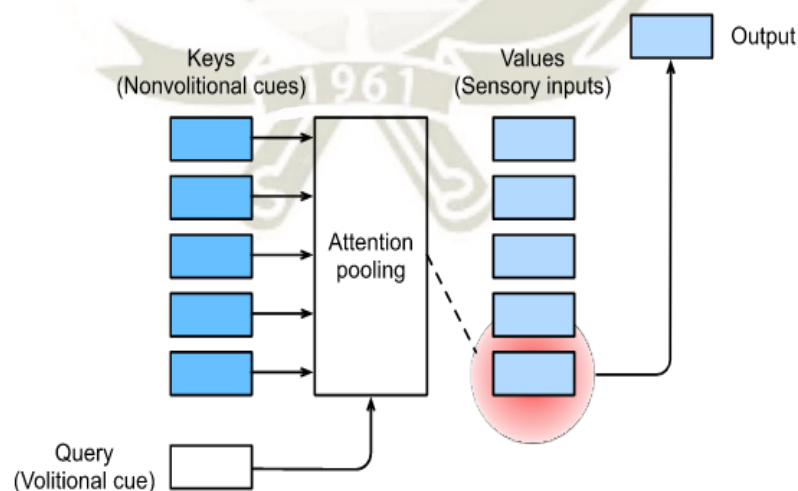


Figura 22: Diagrama de mecanismo de atención [36]

En un mecanismo de atención se distinguen tres elementos, **queries** que representan

el *bias* aplicado sobre la data ingresada, **values** hacen referencia a las entradas sensoriales y *keys* que hacen referencia a *bias* autodetectados.

### 2.5.1 Mecanismo de atención de Bahdanau

Este mecanismo de atención fue propuesto por Bahdanau [1] para modelos de procesamiento de lenguaje en 2014, en este mecanismo cuando se predice un token, si los tokens de ingreso no son considerados relevantes el modelo alinea (atiende) solo partes de la secuencia de ingreso que sean relevantes para la predicción en curso. Esto se logra tratando a la variable de contexto como salida de una capa de pooling.

La ecuaciones que representan a este mecanismo de atención son:

$$\alpha_{ts} = \frac{\exp(\text{score}(h_t, \bar{h}_s))}{\sum_{s'=1}^S \exp(\text{score}(h_t, \bar{h}_{s'}))} \quad (22)$$

$$c_t = \sum_s \alpha_{ts} \bar{h}_s \quad (23)$$

$$\alpha_t = f(c_t, h_t) = \tanh(W_c[c_t; h_t]) \quad (24)$$

$$\text{score}(h_t, \bar{h}_s) = v_a \tanh(W_1 h_t + W_2 \bar{h}_s) \quad (25)$$

## 2.6 Bases de datos

Las bases de datos o *dataset* son parte fundamental de la implementación de redes neuronales dado que estas deben entrenar con diversas casuísticas para ser capaces de "aprender" y esto solo se logra con grandes volúmenes de información.

Tanto como para la detección de objetos como para el rotulado autónomo existen bases de datos extensos y variados tales como MSCOCO, Flickr8k, PASCAL 1K, AI Challenger y STAIR Captions, además de bases de datos creadas por investigadores para fines específicos como análisis médico, y análisis de imágenes satelitales. En la base de datos cada imagen cuenta con, por lo menos, cinco oraciones que la describen. Para obtener descripciones independientes en cada imagen estas deben describirlas con sintaxis variada, esto se ejemplifica en la imagen. La tabla 1 detalla el tamaño de cada base de datos.

| Nombre    | Entrenamiento | Validación | Prueba |
|-----------|---------------|------------|--------|
| MSCOCO    | 82783         | 40504      | 40775  |
| Flickr8k  | 6000          | 1000       | 1000   |
| Flickr30k | 28000         | 1000       | 1000   |
| AIC       | 210000        | 30000      | 30000  |

Tabla 1: Cantidad de imágenes en distintas bases de datos

*Fuente: Elaboración propia*

### 2.6.1 Flickr8k/Flickr30k

Las fotos de esta base de datos provienen de Yahoo, este contiene 8000 fotos, 6000 para entrenamiento, 1000 para verificación y 1000 para pruebas, Flickr30K también tiene como origen fotos de Yahoo en su mayoría son imágenes de personas participando en actividades. Cada imagen cuenta con cinco descripciones manuales.

### 2.6.2 MSCOCO

Common Objects in context (Objetos comunes en contexto), base de datos desarrollada por Microsoft dirigida a la comprensión de escenas, contiene imágenes complejas del día a día y puede ser utilizado para modelos de reconocimiento, segmentación y descripción de imágenes. La base de datos utiliza la plataforma de crowdsourcing de Amazon la cual genera cinco descripciones para cada imagen [37].

### 2.6.3 AIC

Base de datos de origen Chino, es la primera en lenguaje diferente al inglés dedicada a la descripción automática. Contiene 210000 imágenes para entrenamiento y 30000 para verificación, al igual que MSCOCO cada imagen va acompañada de cinco descripciones que indican información de la imagen, escena, acciones, persona y otros. En comparación a bases de datos en inglés para este mismo propósito las oraciones chinas tienen mayor grado de flexibilidad en su sintaxis y lexicalización por lo cual la implementación de los algoritmos es más compleja.



## **CAPÍTULO III**

### **DISEÑO DEL SISTEMA**

## Diseño del sistema

Para alcanzar el objetivo de esta tesis, generar subtítulos automáticos de imágenes en lenguaje castellano, se propone un modelo basado en *Deep Learning* utilizando un extractor de características pre-entrenado, los pilares de este modelo se observan en la figura 23.

El tipo de aprendizaje adecuado para esta aplicación es el supervisado dado que por su naturaleza el modelo debe iterar y comparar sus resultados con los esperados, dentro del aprendizaje supervisado se prefiere el *Deep Learning* puesto que es el que mejor se desenvuelve en tareas de identificación de imágenes.

La arquitectura elegida para el modelo es codificador - decodificador donde el codificador es una CNN que extrae las características de las imágenes, el decodificador consta de una red neuronal GRU con memoria que genera las oraciones de acuerdo a las características extraídas y los coeficientes de atención. El coeficiente de atención se genera posterior a la extracción de características y es determinante para las oraciones.

El dataset utilizado es un fragmento de MSCOCO con las oraciones de descripción llevadas al español utilizando el API de traducción, basada en *Deep Learning*, de Yandex. El dataset es dividido en segmentos para entrenamiento y evaluación. La evaluación del modelo se realiza con las dos principales métricas para este tipo de modelos, BLEU a nivel corpus y METEOR a nivel de oraciones individuales.

Se implementó una interfaz de usuario gráfica (GUI) que permite interactuar con el modelo final, el usuario puede cargar una imagen desde un repositorio local o desde la web. Los resultados mostrados son la imagen con el gráfico de atención y el subtítulo generado.

La implementación se divide en tres etapas, el modelo de predicción, el cual requiere construir el modelo y entrenarlo, etapa de evaluación, donde se aplican métricas BLEU y METEOR, y etapa de GUI donde es posible invocar el modelo generado y genera

predicciones de imágenes a elección del usuario.

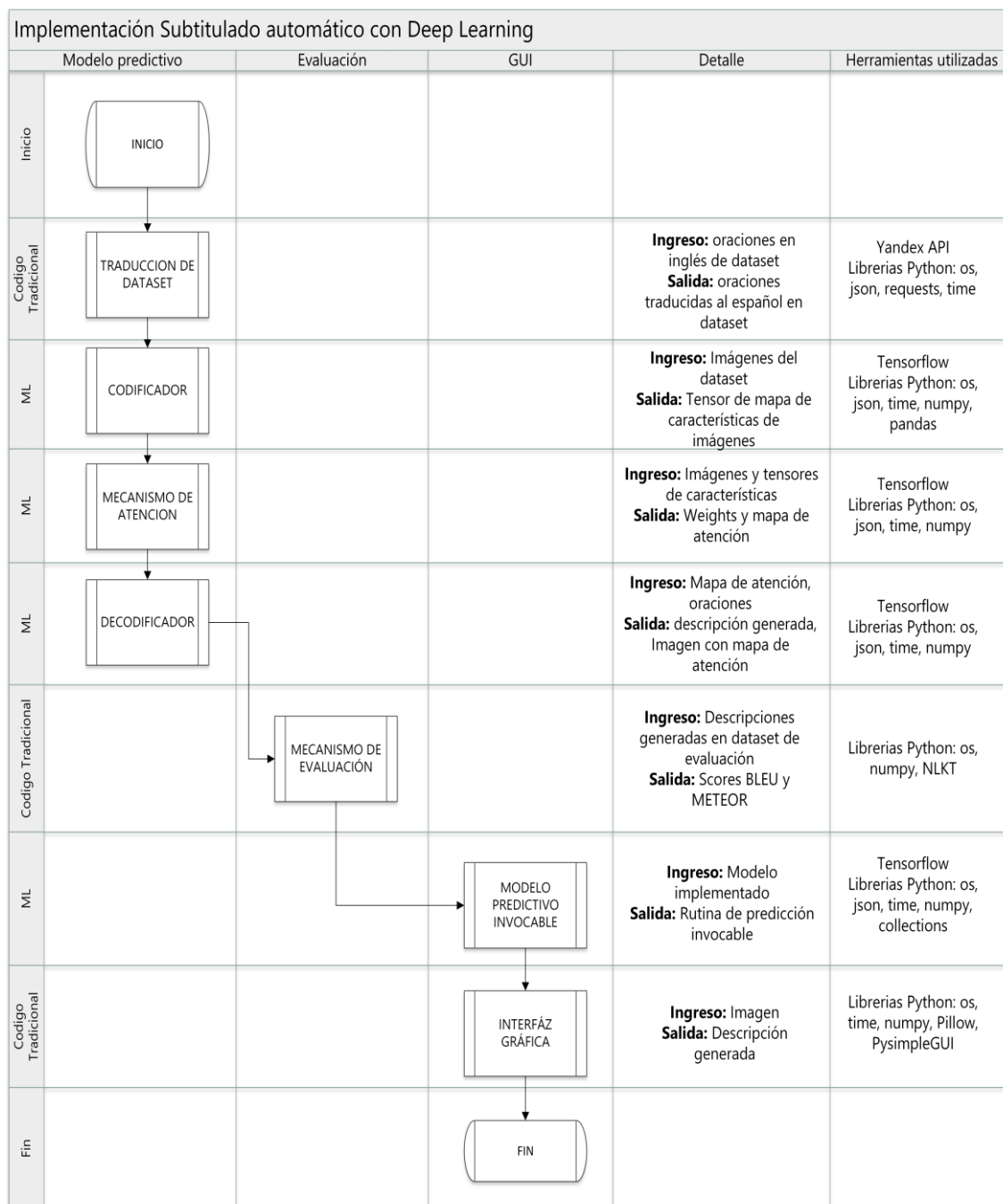


Figura 23: Rutinas implementadas

*Fuente: Elaboración propia*

### 3.1 Modelo predictivo

El modelo predictivo es el núcleo del sistema, este es el encargado de generar las descripciones a cada imagen que se le alimente, para alcanzar este objetivo el modelo debe

pasar por un proceso de entrenamiento donde el modelo actualizará parámetros internos de tal forma que aprenda a etiquetar imágenes.

Los principales bloques del modelo predictivo se abordan a continuación.

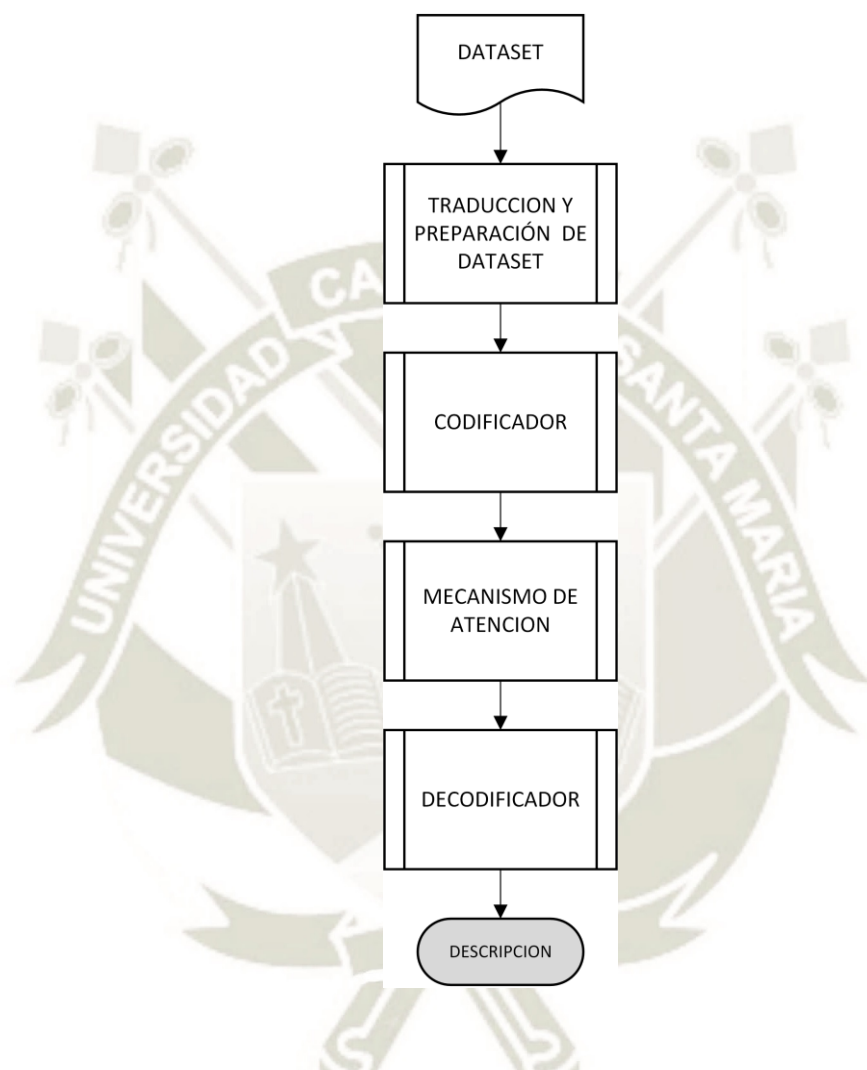


Figura 24: Modelo predictivo detallado

*Fuente: Elaboración propia*

### 3.1.1 Traducción y preparación de dataset

Para la tarea seleccionada existen datasets como Flickr, COCO, etc, existe también la posibilidad de crear un dataset desde cero lo cual significaría invertir en el subtítulo manual por humanos, lo que se descarta por el tiempo requerido y por el costo asociado. Para este proyecto no se pretende abarcar un tipo de imagen especializado (botánica,

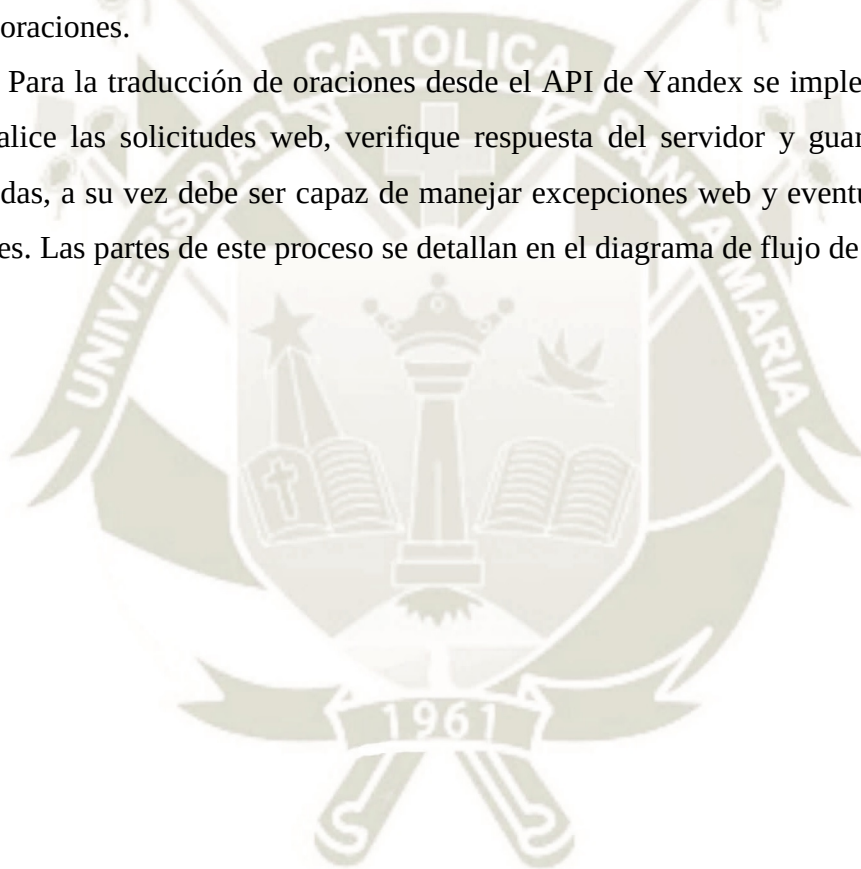
industrial, geológica, etc), por el contrario se dirige a imágenes comunes de la vida diaria; a estos parámetros se ajusta el dataset MSCOCO.

+

### 3.1.1.1 Traducción de dataset

El dataset con el que se trabajó consta de 20 000 imágenes con cinco subtítulos por cada una, el dataset es una porción de MSCOCO traducida utilizando el API de Yandex. Si bien las oraciones no son perfectas gramaticalmente, son inteligibles y permiten implementar el modelo. En el proceso de entrenamiento se probó con dataset entre 2000 y 20000 oraciones.

Para la traducción de oraciones desde el API de Yandex se implementa un código que realice las solicitudes web, verifique respuesta del servidor y guarde las oraciones traducidas, a su vez debe ser capaz de manejar excepciones web y eventuales pérdidas de paquetes. Las partes de este proceso se detallan en el diagrama de flujo de 24



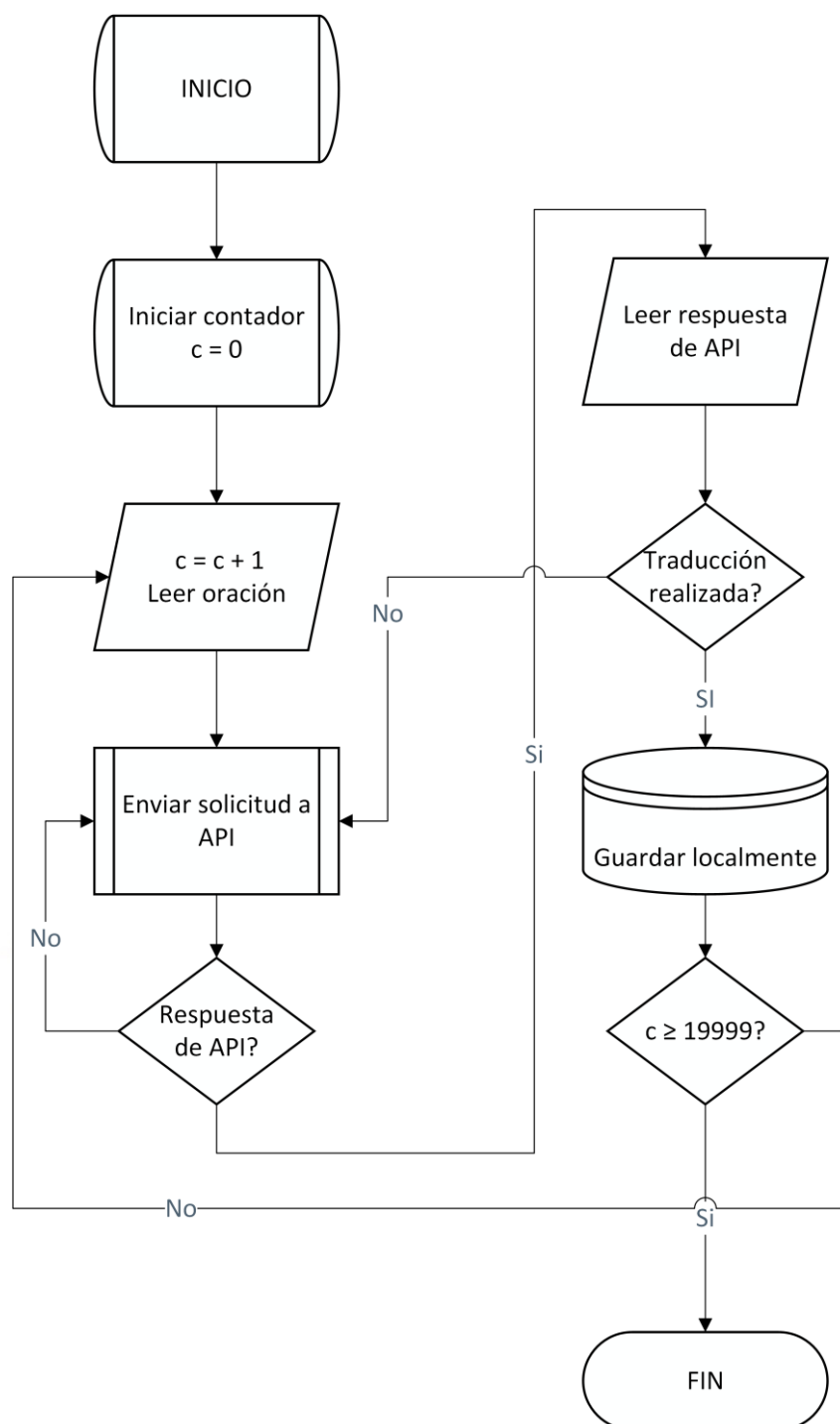


Figura 25: Diagrama de flujo de traducción de oraciones API Yandex

*Fuente: Elaboración propia*

### 3.1.1.2 Preparación de imágenes

Las imágenes del dataset MSCOCO no tiene una dimensión unificada y deben ser pre-procesadas a un tamaño de 299px × 299px y rango de píxeles entre 1 y -1 para poder

ser alimentadas a la red InceptionV3. Este procesamiento se realiza utilizando la librería propia de Keras.

### 3.1.1.3 Preparación de oraciones

Al igual que las imágenes las oraciones también deben ser pre-procesadas antes de ser ingresadas al modelo. Las oraciones deben ser limpiadas de símbolos de puntuación y exclamación y separadas en tokens <sup>1</sup>.

```
In [44]: real_caption
Out[44]: '<start> jugadores de tenis de pie en la cancha hablando entre sí <end>'

In [45]: real_caption=real_caption[8:-5]

In [46]: real_caption.split()
Out[46]:
['jugadores',
 'de',
 'tenis',
 'de',
 'pie',
 'en',
 'la',
 'cancha',
 'hablando',
 'entre',
 'sí']
```

Figura 26: Preparación de oraciones

*Fuente: Elaboración propia*

Posteriormente cada palabra *String* debe ser normalizada para que pueda ser interpretada por el ordenador. Para el normalizado se debe calcular la longitud de la oración más larga en el dataset, de acuerdo a ello se asignarán los coeficientes. Este proceso se realiza utilizando la librería *preprocessing.text* de Tensorflow.

| Oración     | <start> | un | camión | de | comida | rojo | estacionado | <end> |
|-------------|---------|----|--------|----|--------|------|-------------|-------|
| Normalizada | 3       | 2  | 124    | 5  | 58     | 93   | 183         | 4     |

Tabla 2: Tokenización y normalización de oración

*Fuente: Elaboración propia*

<sup>1</sup> Lista de strings que contienen solo una palabra

Se generó un gráfico de la frecuencia de palabras la cual se detalla en 26

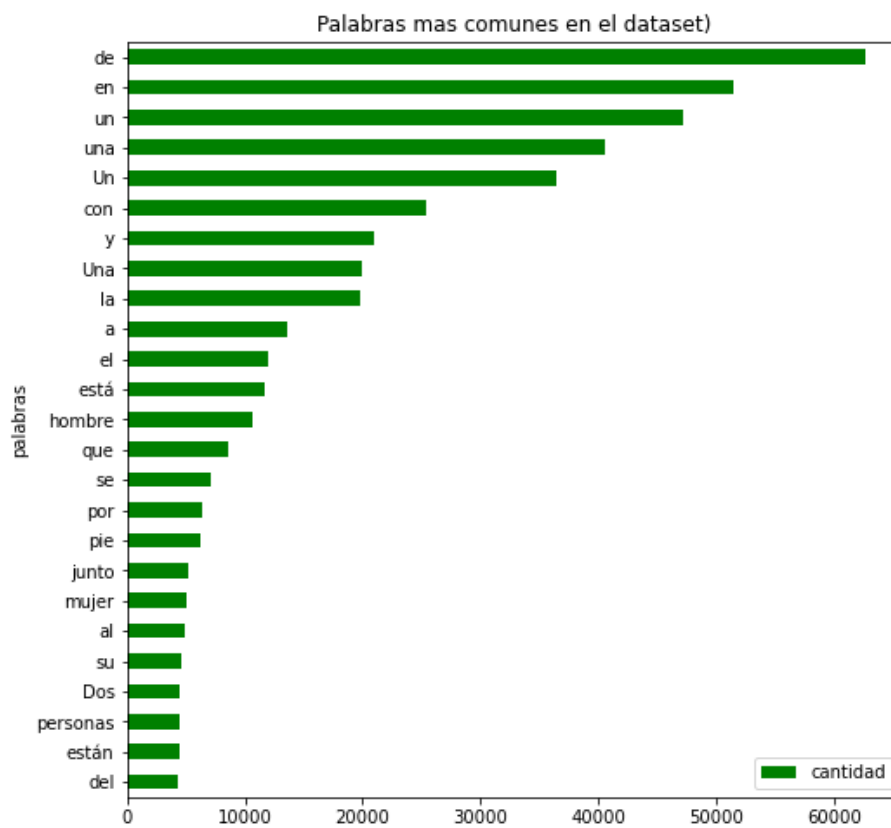


Figura 27: Palabras comunes en el dataset

*Fuente: Elaboración propia*

Una vez el dataset ha sido preprocesado debe ser dividido en segmentos de entrenamiento y validación, el grupo de entrenamiento netamente para ello y el de validación servirá para realizar métricas de evaluación objetiva y subjetiva del modelo. Se utiliza un ratio de 80-20, la división se realiza luego de mezclar el dataset de manera aleatoria para evitar que el modelo sufra de algún bias no deseado.

### 3.1.2 Codificador

El codificador de un sistema de etiquetado automático debe extraer características de las imágenes que se le alimenten y generar *insights* de ellas. Para la extracción de características utilizamos la técnica de transferencia de aprendizaje.

### 3.1.2.1 Transferencia de aprendizaje

La transferencia de aprendizaje es una técnica que permite utilizar un modelo ya generado y entrenado como punto de inicio de un segundo, es decir aplicar conocimiento previo para un problema similar. Dado que los modelos de CNN demandan un dataset amplio y variado para obtener buen rendimiento no es posible utilizarlas de manera competente con datasets pequeños o medianos. Para tareas de procesamiento de imágenes incluso el dataset de CIFAR con 50K imágenes es considerado pequeño. Obtener un dataset de tales dimensiones es difícil por lo que no siempre es posible obtener data específica para entrenar el modelo desde cero. En tales casos la transferencia de aprendizaje es una forma práctica de afrontar el problema.

En el subtitulado automático es común utilizarla para la extracción de características y luego alimentar al modelo procesador de lenguaje, el modelo por excelencia para esto es ImageNet entrenado en un dataset con más de un millón de imágenes. Los beneficios de este procedimiento incluyen la reducción del tiempo de entrenamiento del modelo completo y mejor rendimiento en general.

### 3.1.2.2 Extractor de características

Hasta hace diez años para la extracción de patrones se utilizaría detección de bordes, reconocimiento de colores y/patrones en histograma RGB u otras técnicas convencionales de visión artificial, el estado del arte es utilizar redes neuronales, entrenarlas y que estas aprendan a obtener las características de manera autónoma. Para esta tarea destacan las redes CNN remitidas al desafío de ImageNet que si bien fueron diseñadas para clasificar imágenes tienen dentro de su estructura capas destinadas a reconocer y extraer patrones de estas.

La extracción de características es la primera etapa del codificador y se implementa utilizando una CNN, se elige Inception V3 [30] o GoogleNet dado que es posible implementarla en sistemas de poca VRAM sin sacrificar la calidad de sus resultados, a diferencia de VGGNet que tiende a saturar el hardware por su profundidad.

En Python se declara una red neuronal sin parámetros a la cual se procede a cargar la estructura inception V3 e introducir pesos de ImageNet, este proceso implica descargar la estructura y pesos de su repositorio. Para que la información retribuida de esta CNN sea utilizable para nuestro modelo se requiere los resultados de la capa previa a la de salida, para

ello retira dicha capa de la red neuronal. Una vez inicializado el modelo se debe pasar todo el dataset a través de esta red para obtener las características contenidas en cada imagen. Tener en cuenta que este proceso no es entrenamiento sino la aplicación. La extracción de características puede dividirse en batches de acuerdo al tamaño del dataset y a las características del equipo donde se trabaja, para nuestro caso se utiliza un batch de 16.

Durante este proceso cada imagen ingresa a la red neuronal y es tratada como una matriz tridimensional, como tal las características extraídas son numéricas. Las características obtenidas son almacenadas en variables del tipo *tensors* de dimensiones [64,2048], los valores obtenidos son almacenados en el disco local para luego ser llamados desde el decodificador.

Como elemento final del codificador se incluye una capa *fully connected* donde se generarán *insights* de las características obtenidas. Esta capa es la única que se actualiza en el proceso de entrenamiento.

A continuación se presenta el pseudocódigo de lo detallado:

```

CODIFICADOR    {
    Extractor de características {

        Iniciar modelo vacío
        Cargar capas de GoogleNet al modelo
        Cargar pesos de ImageNet
        Definir capa de salida
    }
    Iniciar capa secundaria {

        Iniciar capa fully connected
    }
    Proceso invocable {

        Ingresar resultados de extracción de
        características a capa fully connected
        Función de activación ReLu
        Retornar resultado
    }
}

```

El codificador se implementa en dos partes, por lo cual al imprimir la estructura de este solo se muestra la última capa.

Model: "cnn\_encoder\_1"

| Layer (type)              | Output Shape | Param # |
|---------------------------|--------------|---------|
| dense_6 (Dense)           | multiple     | 524544  |
| Total params: 524,544     |              |         |
| Trainable params: 524,544 |              |         |
| Non-trainable params: 0   |              |         |

Figura 28: Estructura de codificador

*Fuente: Elaboración propia*

### 3.1.3 Mecanismo de atención

La primera implementación a subtitulado de imágenes la realizaron Xu et al [34] donde los autores exploran y comparan la atención *hard* y *soft*, su resultados lograron mejorar en promedio un punto en BLEU-4 y dos en METEOR.

En el presente proyecto se implementa el método de Bahadanau que combina los *insights* obtenidos en la extracción de características con las descripciones y genera un coeficiente de importancia de cada región de la imagen, este valor se alimenta a la siguiente capa oculta para predecir la siguiente palabra.

Esta etapa consta de tres capas densas, propias del método Bahadanau, y una capa oculta que combina los resultados de la primera y tercera capa.

El score se calcula de acuerdo a la fórmula [24], los pesos o *weights* de acuerdo a [22] y el vector de contexto de acuerdo a [21].

Pseudocódigo de mecanismo de atención:

```
MECANISMO DE ATENCION {  
  Iniciar capas {  
    Iniciar dos capas embebidas  
    Iniciar capa fully connected  
  }  
  Proceso invocable {  
    Calcular score de mecanismo de atención  
    Calcular pesos de mecanismo de atención  
    Calcular vector de contexto  
    Retornar vector de contexto y pesos  
  }  
}
```

### 3.1.4 Decodificador

En esta etapa el modelo procede a predecir las cadenas de palabras tomando como referencia los coeficientes de atención definidos en el proceso anterior. Utilizamos una arquitectura extraída de modelos de traducción de texto, donde el algoritmo debe almacenar el valor de la palabra anterior y en relación a esta predecir la siguiente para que la descripción sea coherente.

El decodificador consta de una capa embebida, una capa GRU encargada de predecir la cadena de palabras y dos *fully connected*. Se prefiere la GRU sobre la LSTM dado que esta última requiere mayor capacidad computacional. Se incluyen dos capas densas a la salida de la GRU.

En el proceso de entrenamiento se modifican los valores internos del decodificador de manera que sus predicciones se ajusten a los resultados esperados, de acuerdo al dataset. Los valores obtenidos se almacenan periódicamente para evitar pérdida de información por un eventual fallo y poder retomar el entrenamiento en dicho punto y para su posterior utilización en la interfaz gráfica.

```

DECODIFICADOR {
  Iniciar capas {
    Empatar con las dimensiones del codificador
    Iniciar capa embebida
    Iniciar capa GRU
    Iniciar capas fully connected
    Iniciar capa de mecanismo de atención
  }
  Proceso invocable {
    Obtener scores del mecanismo de atención
    Ingresar resultado a capa embebida
    Ingresar resultado a capa GRU
    Ingresar resultado a capas fully connected
    Retornar resultado y coeficientes de
    atención
  }
}

```

Al imprimir las capas del decodificador tensorflow muestra lo siguiente

Model: "rnn\_decoder\_1"

| Layer (type)                 | Output Shape | Param # |
|------------------------------|--------------|---------|
| embedding_1 (Embedding)      | multiple     | 1280256 |
| gru_1 (GRU)                  | multiple     | 1575936 |
| dense_7 (Dense)              | multiple     | 262656  |
| dense_8 (Dense)              | multiple     | 2565513 |
| bahdanau_attention_1 (Bahdan | multiple     | 394753  |
| Total params: 6,079,114      |              |         |
| Trainable params: 6,079,114  |              |         |
| Non-trainable params: 0      |              |         |

Figura 29: Capas del decodificador

*Fuente: Elaboración propia*

### 3.2 Mecanismos de evaluación

Los modelos de subtitulado automático utilizan métricas originalmente diseñadas para modelos de procesamiento de lenguaje, estos permiten evaluar con facilidad los resultados comparándolos con las descripciones de la base de datos utilizada para el entrenamiento. Estas métricas pueden utilizarse independientemente del lenguaje. Para la evaluación de las oraciones generadas se utilizan los métodos BLEU y METEOR.

El score BLEU fue propuesto por Papineni et al[19] en 2002 para la evaluación de traducciones. El algoritmo realiza la evaluación contando los n-gramas de la oración candidata con el texto de referencia, se realizan comparaciones de 1-gramas hasta 4-gramas, la comparación es independiente del orden de las palabras. El rango de evaluación de BLEU es de 0 a 1, muy pocas oraciones tendrán una evaluación de 1 dado que esto significaría que las oraciones son idénticas, incluso un ser humano no es capaz de obtener el *score* perfecto. METEOR [2] se propone para cubrir deficiencias que BLEU no aborda como la utilización de sinónimos o alteración del orden de palabras sin afectación en el significado [2].

El algoritmo implementado toma las imágenes del set de evaluación, no utilizadas en el entrenamiento, genera las descripciones y separa cada una de ellas en tokens. Se obtiene el índice correspondiente a las imágenes utilizadas y se retorna las descripciones del dataset, las cuales también deben ser *tokenizadas*, tanto las descripciones generadas y originales son almacenadas en una variable temporal. Finalmente se itera sobre todas las oraciones y se calcula su score, esta última función se realiza con la librería **nlkt**.

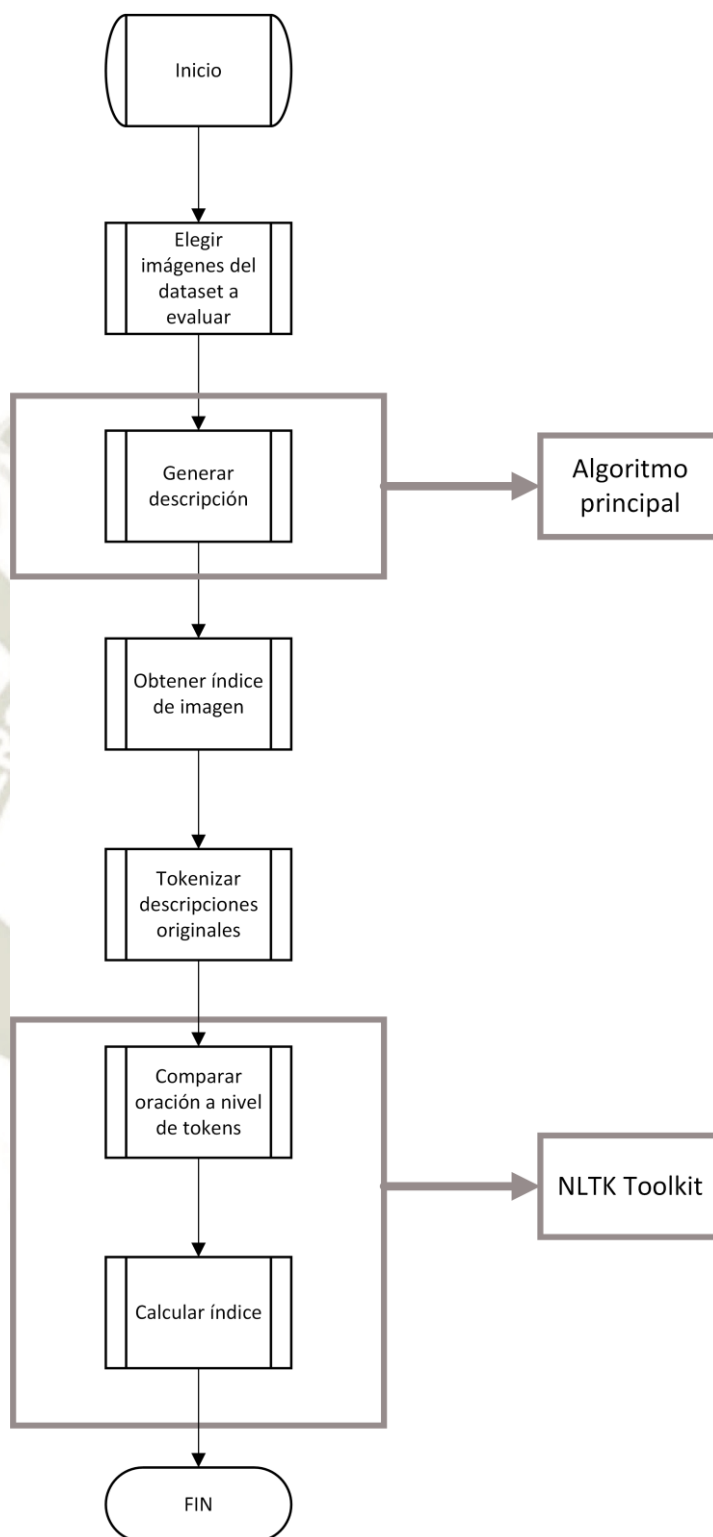


Figura 30: Diagrama de flujo para evaluación

*Fuente: Elaboración propia*

### 3.3 Ejecución de rutinas y entrenamiento

La ejecución de la rutina de traducción muestra las oraciones llevadas al castellano. Se debe contar con credencial de la API de lo contrario las solicitudes serán rechazadas. Las

oraciones traducidas se guardan en un archivo local con el comando “`json.dump()`”.

```

...: trad_yandex=traducir_yandex(train_captions[val],key)
...: temp=trad_yandex['text']
...: caption = f"<start> {temp[0]} <end>"
...: train_captions[val]=caption
...: print(val, train_captions[val])
...:
...: if val%5==0:
...:     with open(train_cap_1,'w') as f:
...:         json.dump(train_captions,f)
...:         print('*****CHECKPOINT GUARDADO*****', val)
...:
...: if val%50==0:
...:     print('Inicio de espera 30 seg')
...:     time.sleep(30)
...:     print('Fin de espera')
...:
-----
94801 <start> Una mano sosteniendo una copa de vino con un televisor de fondo. <end>

```

Figura 31: Proceso de traducción

*Fuente: Elaboración propia*

Al ejecutar la sección de extracción de características los tensores obtenidos se guardan con extensión “.numpy” para su posterior acceso.

| Name                                                                                                                    | Type     | Size   |
|-------------------------------------------------------------------------------------------------------------------------|----------|--------|
|  COCO_train2014_000000581903.jpg.npy | NPY File | 513 KB |
|  COCO_train2014_000000581882.jpg.npy | NPY File | 513 KB |
|  COCO_train2014_000000581835.jpg.npy | NPY File | 513 KB |
|  COCO_train2014_000000581738.jpg.npy | NPY File | 513 KB |
|  COCO_train2014_000000581719.jpg.npy | NPY File | 513 KB |
|  COCO_train2014_000000581708.jpg.npy | NPY File | 513 KB |
|  COCO_train2014_000000581697.jpg.npy | NPY File | 513 KB |
|  COCO_train2014_000000581605.jpg.npy | NPY File | 513 KB |
|  COCO_train2014_000000581499.jpg.npy | NPY File | 513 KB |
|  COCO_train2014_000000581495.jpg.npy | NPY File | 513 KB |
|  COCO_train2014_000000581392.jpg.npy | NPY File | 513 KB |
|  COCO_train2014_000000581302.jpg.npy | NPY File | 513 KB |

Figura 32: Archivos creados con características obtenidas

*Fuente: Elaboración propia*

Tras separar la data en entrenamiento y validación se visualiza el tamaño del dataset, 75 838 y 18 962 oraciones respectivamente.

```

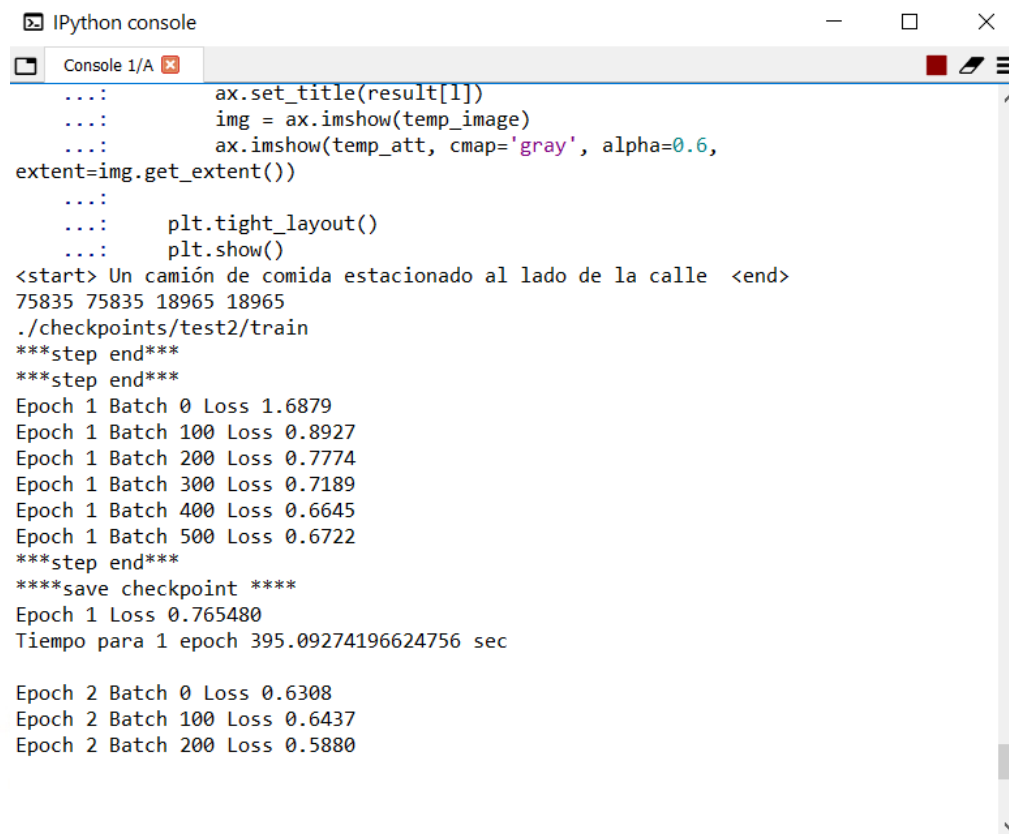
...:
...: print(len(ima_name_t), len(cap_t), len(ima_name_val), len(cap_val))
75838 75838 18962 18962
In [16]: |

```

Figura 33: Tamaño de sets de entrenamiento y validación.

*Fuente: Elaboración propia*

Realizado el pre-procesamiento y extracción de características se ejecuta el bucle de entrenamiento, a continuación se detalla partes de este proceso.



```

IPython console
Console 1/A
...: ax.set_title(result[1])
...: img = ax.imshow(temp_image)
...: ax.imshow(temp_att, cmap='gray', alpha=0.6,
extent=img.get_extent())
...:
...: plt.tight_layout()
...: plt.show()
<start> Un camión de comida estacionado al lado de la calle <end>
75835 75835 18965 18965
./checkpoints/test2/train
***step end***
***step end***
Epoch 1 Batch 0 Loss 1.6879
Epoch 1 Batch 100 Loss 0.8927
Epoch 1 Batch 200 Loss 0.7774
Epoch 1 Batch 300 Loss 0.7189
Epoch 1 Batch 400 Loss 0.6645
Epoch 1 Batch 500 Loss 0.6722
***step end***
***save checkpoint ***
Epoch 1 Loss 0.765480
Tiempo para 1 epoch 395.09274196624756 sec

Epoch 2 Batch 0 Loss 0.6308
Epoch 2 Batch 100 Loss 0.6437
Epoch 2 Batch 200 Loss 0.5880

```

Figura 34: Inicio del proceso de entrenamiento

*Fuente: Elaboración propia*

```

Epoch 8 Batch 300 Loss 0.4412
Epoch 8 Batch 400 Loss 0.4282
Epoch 8 Batch 500 Loss 0.4380
Epoch 8 Loss 0.445544
Tiempo para 1 epoch 231.98910355567932 sec

Epoch 9 Batch 0 Loss 0.4326
Epoch 9 Batch 100 Loss 0.4270
Epoch 9 Batch 200 Loss 0.4403
Epoch 9 Batch 300 Loss 0.4278
Epoch 9 Batch 400 Loss 0.4144
Epoch 9 Batch 500 Loss 0.4584
Epoch 9 Loss 0.432130
Tiempo para 1 epoch 226.01037788391113 sec

Epoch 10 Batch 0 Loss 0.4272
Epoch 10 Batch 100 Loss 0.4016
Epoch 10 Batch 200 Loss 0.4198
Epoch 10 Batch 300 Loss 0.4040
Epoch 10 Batch 400 Loss 0.4106
Epoch 10 Batch 500 Loss 0.4217
Epoch 10 Loss 0.420516
Tiempo para 1 epoch 226.91148471832275 sec

```

```

****save checkpoint ****
Epoch 396 Loss 0.123801
Time taken for 1 epoch 221.12090373039246 sec

Epoch 397 Batch 0 Loss 0.1214
Epoch 397 Batch 100 Loss 0.1135
Epoch 397 Batch 200 Loss 0.1194
Epoch 397 Batch 300 Loss 0.1247
Epoch 397 Batch 400 Loss 0.1191
Epoch 397 Batch 500 Loss 0.1271
Epoch 397 Loss 0.118701
Time taken for 1 epoch 218.27991247177124 sec

Epoch 398 Batch 0 Loss 0.1266
Epoch 398 Batch 100 Loss 0.1185
Epoch 398 Batch 200 Loss 0.1262
Epoch 398 Batch 300 Loss 0.1132
Epoch 398 Batch 400 Loss 0.1130
Epoch 398 Batch 500 Loss 0.1238
Epoch 398 Loss 0.117248
Time taken for 1 epoch 217.97986960411072 sec

Epoch 399 Batch 0 Loss 0.1144
Epoch 399 Batch 100 Loss 0.1158
Epoch 399 Batch 200 Loss 0.1227
Epoch 399 Batch 300 Loss 0.1140
Epoch 399 Batch 400 Loss 0.1052
Epoch 399 Batch 500 Loss 0.1179
Epoch 399 Loss 0.117102
Time taken for 1 epoch 222.41144037246704 sec

Epoch 400 Batch 0 Loss 0.1172
Epoch 400 Batch 100 Loss 0.1263
Epoch 400 Batch 200 Loss 0.1130
Epoch 400 Batch 300 Loss 0.1076
Epoch 400 Batch 400 Loss 0.1217
Epoch 400 Batch 500 Loss 0.1202
Epoch 400 Loss 0.119785
Time taken for 1 epoch 233.75817370414734 sec

```

Figura 35: Proceso de entrenamiento y guardado en 400 Epoch

*Fuente: Elaboración propia*

En caso se tenga una interrupción en el proceso de entrenamiento es posible retomarlo desde el último checkpoint guardado. El entrenamiento de 400 Epoch tomó 28 horas de trabajo ininterrumpido. Culminado el entrenamiento el modelo está listo para las pruebas respectivas, esto se aborda en el capítulo de pruebas y resultados.

### 3.4 Interfaz gráfica

Para la interacción con usuarios finales se ha implementado una GUI utilizando la librería PysimpleGUI de Python. Los valores (pesos) obtenidos en el proceso de entrenamiento fueron almacenados localmente y son llamados mediante funciones. La GUI tiene como argumentos de ingreso una URL de imagen o un archivo local, las salidas son la oración generada, gráfico de atención e imagen original. No es posible aplicar métricas como BLEU o METEOR dado que las imágenes a ingresar no cuentan con oraciones de referencia.

### 3.4.1 Implementación

La interfaz gráfica requiere explotar las características de la programación orientada a objetos para lo cual procedemos a aislar en clases. Las partes a aislar son:

- Modelo de subtitulado
- Solicitudes web
- Conversión a PNG y reducción de tamaño

Al modelo de *Deep Learning* creado se le retira sus capacidades de aprendizaje y solo se mantiene lo necesario para que genere los subtítulos es decir se debe mantener codificador, decodificador, modelo de atención y valores obtenidos del entrenamiento ya cargados. Se crea el codificador con las mismas características que el modelo de entrenamiento; extractor de características Inception V3 con pesos de ImageNet y capa *fully connected*. El decodificador mantiene su estructura al igual que el modelo de atención, declarado todo el modelo se cargan los valores finales obtenidos en el entrenamiento. Se mantiene la función de evaluación Este objeto se guarda con el nombre "*cargar<sub>m</sub>odelo.py*" y puede ser llamado desde otro archivo mediante importación.

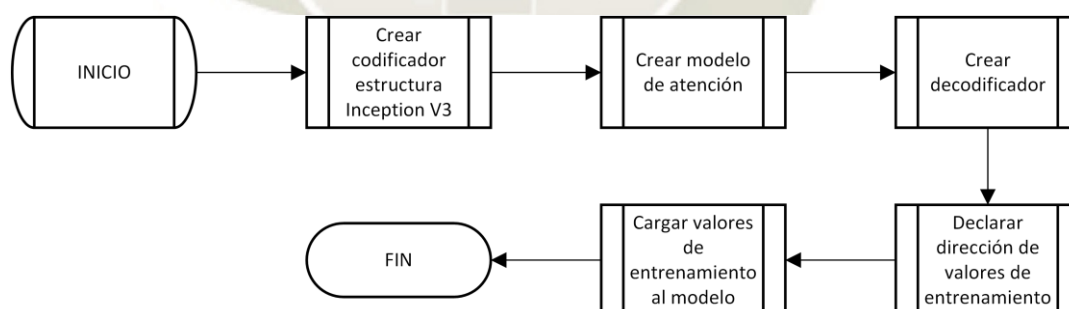


Figura 36: Diagrama de flujo creación de interfaz gráfica

Fuente: Elaboración propia

Las solicitudes web también requieren una instancia separada, originalmente se usó la función de *keras getfile* que permite descargar la imagen, guardarla y pasarla directamente a las capas de procesamiento, sin embargo esta función manifestó errores al realizar solicitudes a ciertas webs. Para solucionar esto se utiliza la librería *requests*, la imagen obtenida es almacenada en un repositorio local y se retorna la dirección.

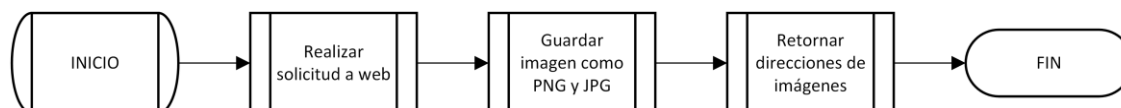


Figura 37: Diagrama de flujo de solicitud web

*Fuente: Elaboración propia*

Las imágenes a mostrar deben ser constantes en tamaño para no alterar las dimensiones de la ventana y facilitar la calidad de visualización. En la reducción de tamaño se debe mantener la proporción de la imagen, todas la imágenes a mostrar son de 500 px de base y altura variable, se utilizan las funciones *resize* de la librería *pillow*, se retorna al programa principal la dirección donde se guardó la imagen.

La construcción de la GUI inicia con la definición del *layout* a utilizar, trabajamos con dos columnas 32. La primera columna debe contener tres bloques diferenciados, área de ingreso de datos (URL o dirección), área para mostrar imagen original y área para mostrar el subtítulo generado, la segunda columna solo mostrará el gráfico de atención generado para la imagen evaluada.



Figura 38: Layout de GUI

*Fuente: Elaboración propia*

El área de ingreso de datos consta de una línea editable donde se puede copiar una URL o buscar directamente en los archivos locales, dado que cada opción requiere tratamiento diferente se les asigna diferenciadores para el llamado de la función correcta, las opciones se muestran mediante *combo-box*. El botón 1 cumplirá la función de abrir el explorador de windows y mostrar las carpetas donde el usuario puede elegir la imagen, el botón 2 tendrá la función de generar el subtítulo y el gráfico de atención asociado, es decir activar este botón deberá descargar la imagen, reducir su tamaño y mostrar los resultados, esto se logra llamando a los objetos indicados líneas arriba.

Solo existen tres acciones que el usuario puede realizar, seleccionar opción (URL o archivo), buscar archivo y generar descripción, el bucle principal itera entorno a estas tres acciones y espera cambio en el estado de alguno de ellos para ejecutar el código respectivo, mientras no se seleccione alguna opción la interfaz se mantiene a la espera, cerrar la ventana cancela la ejecución del programa.

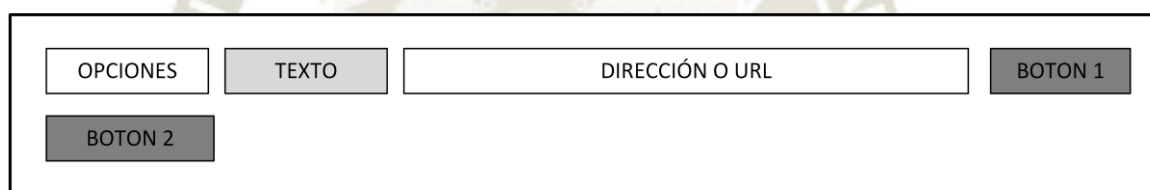


Figura 39: Layout de opciones

*Fuente: Elaboración propia*

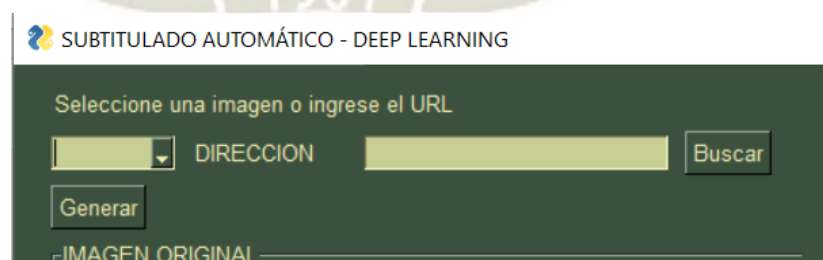


Figura 40: Opciones implementadas en GUI

*Fuente: Elaboración propia*

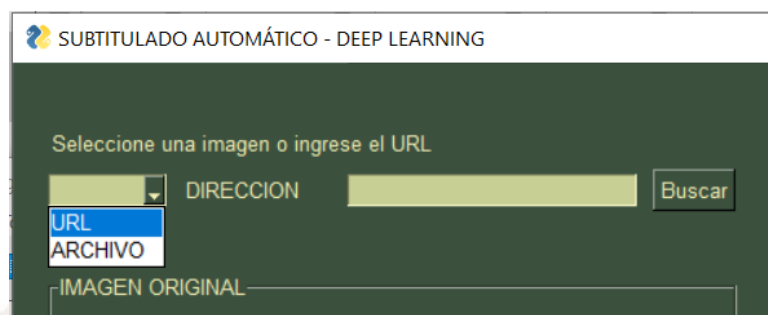


Figura 41: Elección de URL o Archivo en GUI

*Fuente: Elaboración propia*

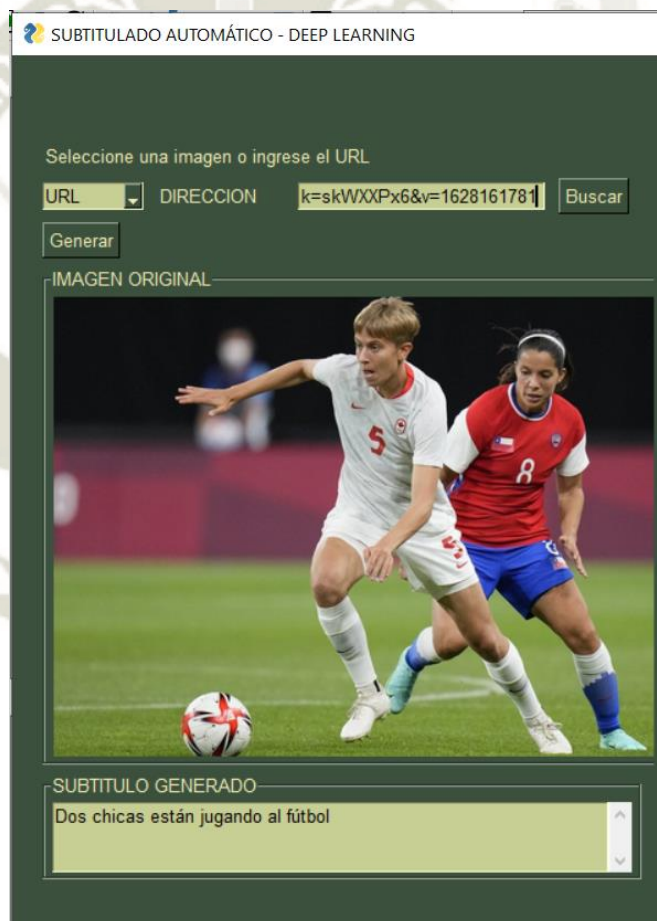


Figura 42: Generación de subtítulo y ploteo de imagen

*Fuente: Elaboración propia*

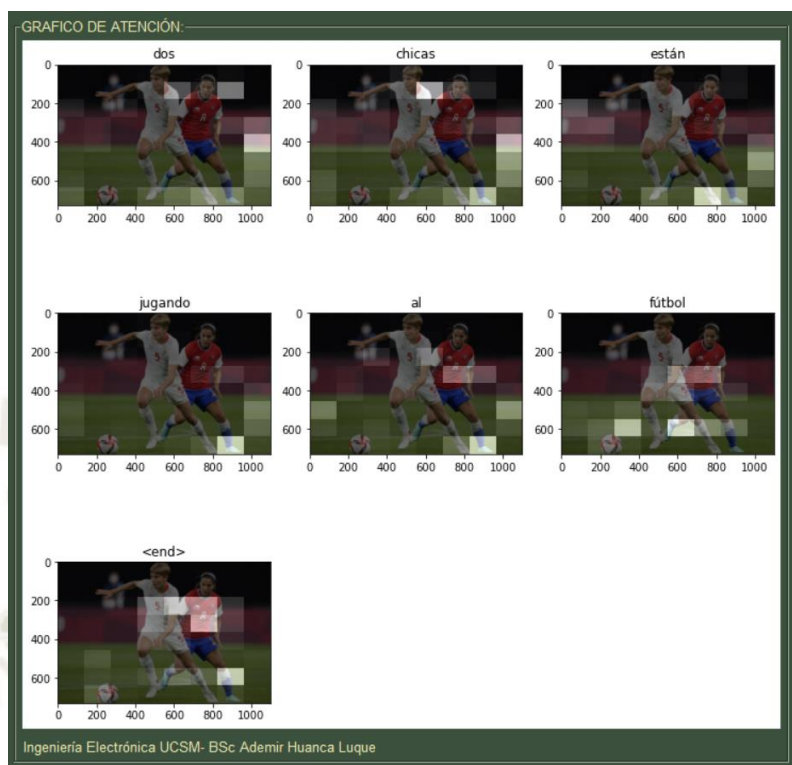


Figura 43: Gráfico de atención mostrado en GUI

*Fuente: Elaboración propia*

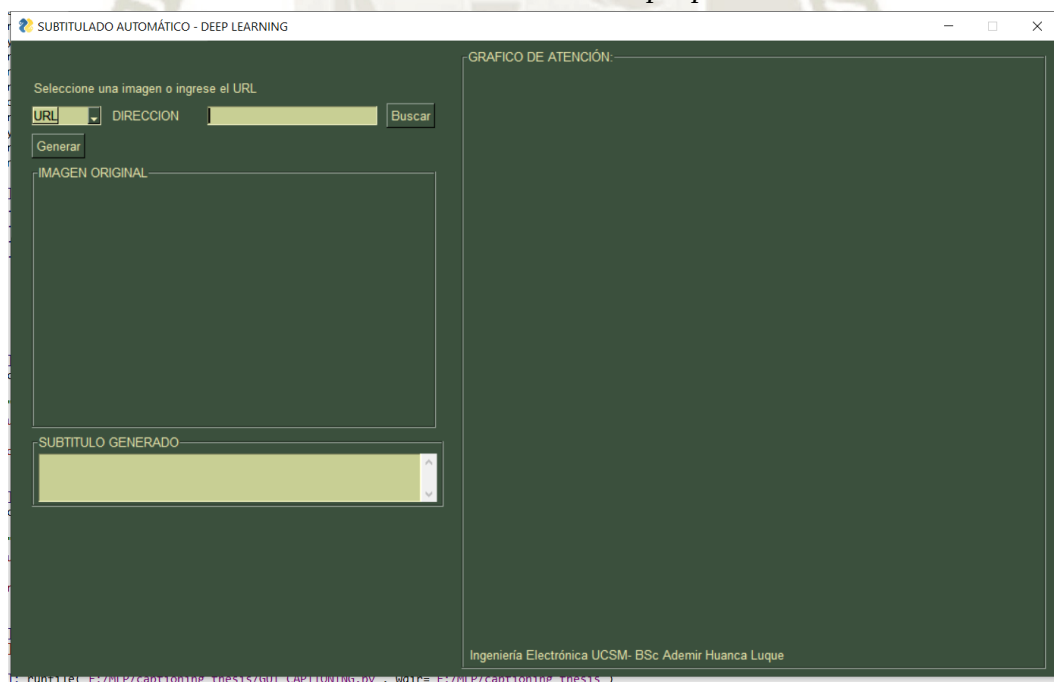


Figura 44: GUI implementado

*Fuente: Elaboración propia*

Los bloques de ploteo de imagen y subtítulo generado son de salida, la data a mostrar se actualiza de acuerdo a los resultados obtenidos. Al declarar un modelo de *Deep Learning* como una clase, python ejecuta todo el código contenido en dicho archivo, el

modelo creado es profundo y su carga no es ágil.

Es posible cambiar el tipo de letra y colores de fondo de la interfaz y otros elementos cosméticos, para el presente se eligió la combinación de verdes 'Dark Green 2'.

Las predicciones realizadas en imágenes fuera del dataset pueden resultar no muy cercanas a la realidad, este fenómeno puede corregirse ampliando la base de datos y mejorando las oraciones de descripción mediante *crowdsourcing* o métricas de evaluación de estas que permitan descartarlas en el proceso de entrenamiento.

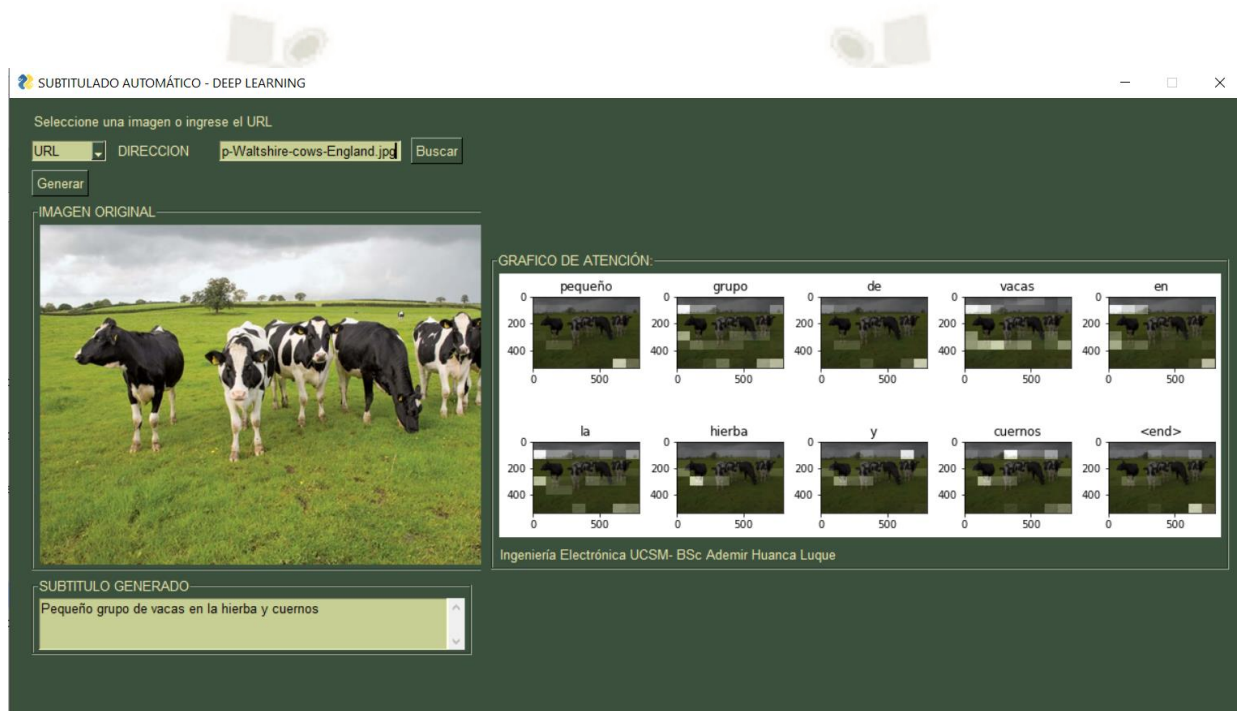


Figura 45: GUI en funcionamiento

*Fuente: Elaboración propia*

En caso se elija la opción de imágenes desde URL éstas son almacenadas localmente en un carpeta predeterminada.



## **CAPÍTULO IV**

### **PRUEBAS Y RESULTADOS**

## Pruebas y resultados

Se realizaron pruebas con distintos tamaños de dataset, variaciones en tamaño de *batch* y EPOCHS. El uso de un dataset de entrenamiento pequeño implica que tal modelo no sea capaz de extrapolar lo aprendido a imágenes externas al dataset de validación, un dataset mayor mejora esta característica pero no está libre de fallas.

La ejecución de la rutina principal muestra el progreso de entrenamiento, pérdida promedio por epoch y tiempo tomado para ello.

Se realizó un entrenamiento piloto con 10 000 imágenes y 20 EPOCHS, es posible observar la evolución de este y el error final alcanzado, este modelo respondía a las imágenes dentro de su dataset sin embargo sus respuestas a imágenes externas no era destacable, para mejorar ello se amplió el tamaño de la base de datos y entrenamiento.

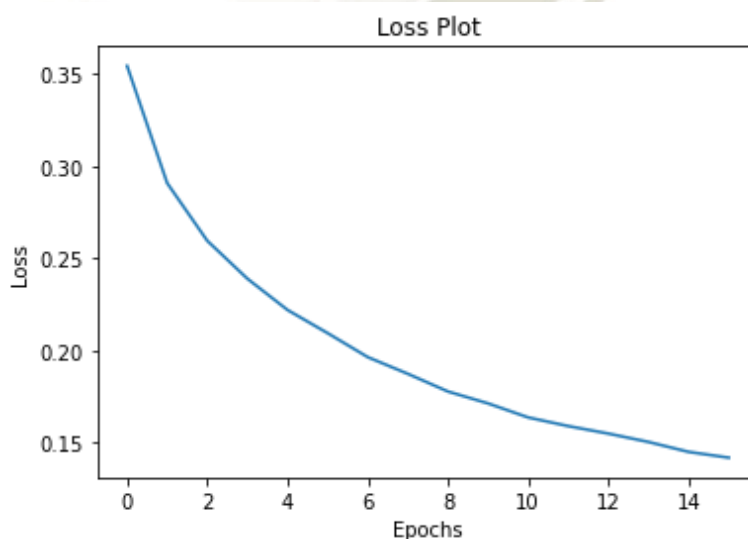


Figura 46: Pérdida en el entrenamiento epochs = 20, loss = 0.141977

*Fuente: Elaboración propia*

Se realizaron entrenamiento con mayor número de imágenes en el dataset y más iteraciones, los gráficos de comparación de entrenamiento revelan que a partir de 250 EPOCHs el modelo no optimiza sus resultados de manera significativa por lo que el entrenamiento pasado este umbral sólo representa pérdida de capacidad computacional.

La evaluación objetiva del modelo se realiza utilizando el algoritmo para BLEU y METEOR detallado en el capítulo anterior. La ejecución de este se muestra en la imagen siguiente.

```

....
.... meteor= meteor_score(real_capt2,capt )
....
.... print('BLEU-2: %f' % bleu[1])
.... print('BLEU-1: %f' % bleu[0])
.... print('BLEU-3: %f' % bleu[2])
.... print('BLEU-4: %f' % bleu[3])
.... print('METEOR: %f' % meteor)
Descripcion real: <start> tres autobuses de dos pisos se sientan a un lado de la carretera <end>
Descripcion generada: tres autobuses de dos pisos se sientan a un lado de una intersección <end>
BLEU-2: 1.000000
BLEU-1: 1.000000
BLEU-3: 0.971812
BLEU-4: 0.923473
METEOR: 0.845836
Descripcion real: <start> un perro saltando en el aire para atrapar un frisbee <end>
Descripcion generada: un perro sostiene un frisbee <end>
BLEU-2: 0.000000
BLEU-1: 0.269597
BLEU-3: 0.000000
BLEU-4: 0.000000
METEOR: 0.307377
Descripcion real: <start> surtido de vinos dulces y flores en una mesa con platos <end>
Descripcion generada: una mesa blanca cubierta con botellas de vino <end>
BLEU-2: 0.790569
BLEU-1: 0.875000
BLEU-3: 0.705432
BLEU-4: 0.594604
METEOR: 0.777345

```

Figura 47: Cálculo de scores METEOR y BLEU

*Fuente: Elaboración propia*

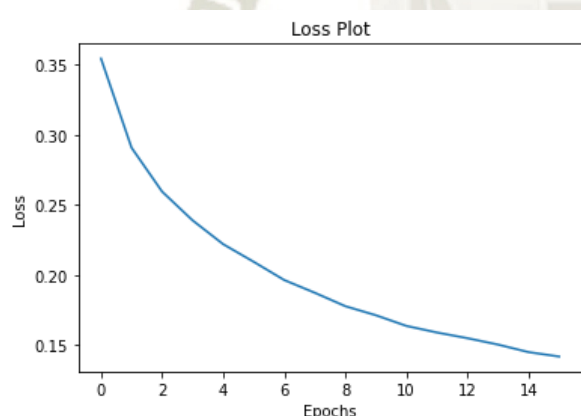
Una vez iterado sobre todo el dataset de validación se obtiene el coeficiente para el modelo, en la tabla 3 se detalla la evolución de scores obtenidos de acuerdo al tamaño de dataset y epochs de entrenamiento, así mismo se muestra la pérdida final del modelo con respecto al dataset de entrenamiento.

| Tamaño de Dataset | 10 000 | 10 000 | 13 000 | 20 000 | 20 000 |
|-------------------|--------|--------|--------|--------|--------|
| EPOCHS            | 50     | 400    | 200    | 200    | 400    |
| LOSS              | 0.2077 | 0.0752 | 0.1091 | 0.1931 | 0.1172 |
| BLEU-1            | 0.3628 | 0.3739 | 0.3443 | 0.3793 | 0.5538 |
| BLEU-2            | 0.2084 | 0.2228 | 0.1968 | 0.2294 | 0.4379 |
| BLEU-3            | 0.1457 | 0.1594 | 0.1388 | 0.1678 | 0.3951 |
| BLEU-4            | 0.0666 | 0.0764 | 0.0642 | 0.0825 | 0.2950 |
| METEOR            | -      | -      | 0.3948 | 0.3650 | 0.3541 |

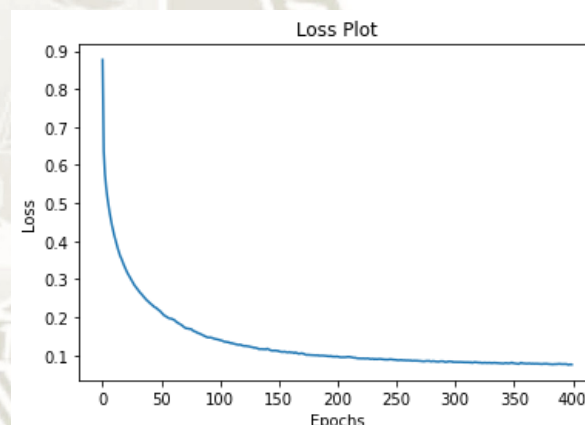
Tabla 3: Evolución de modelo

*Fuente: Elaboración propia*

Los gráficos de pérdida obtenidos a través de las distintas configuraciones muestran la evolución del mismo.



[ 20 EPOCH]



[ 400 EPOCH]

Figura 48: Comparación de gráficos de entrenamiento con 10 000 imágenes

*Fuente: Elaboración propia*

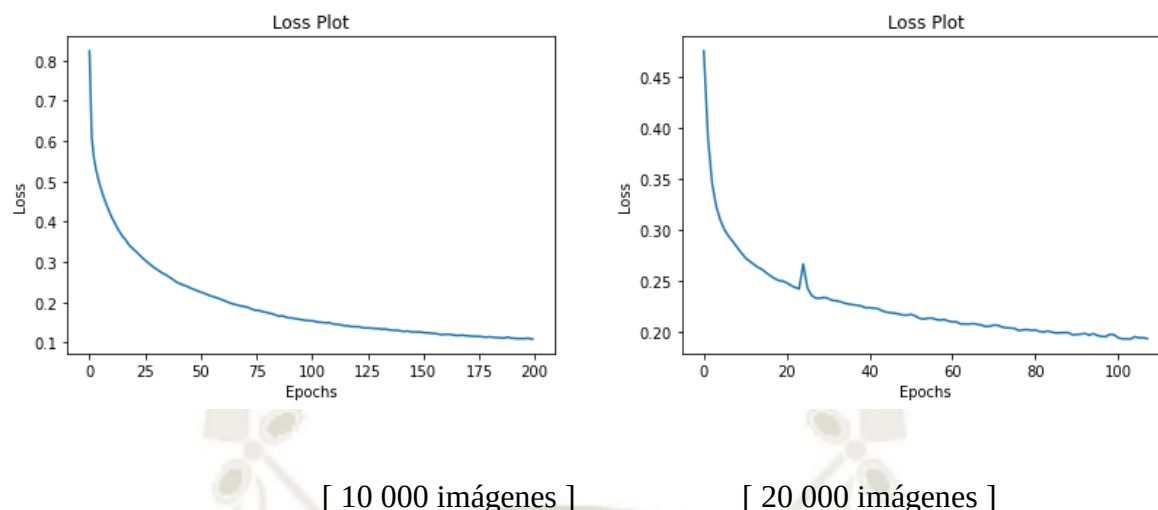


Figura 49: Comparación de gráficos de entrenamiento a 200 EPOCH.

*Fuente: Elaboración propia*

Ciertas descripciones logran identificar un primer plano de las imágenes mostradas sin embargo no detecta o lo hace erróneamente con los complementos.

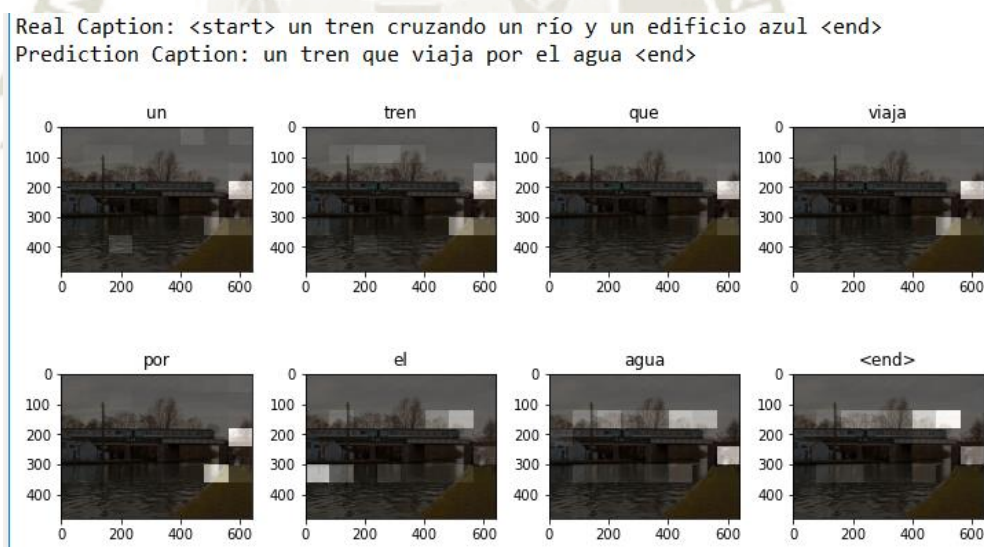


Figura 50: Predicción realizada por el modelo

*Fuente: Elaboración propia*

Las descripciones generadas no siempre son inteligibles ni coherentes a pesar que el modelo exhiba un coeficiente de evaluación aceptable.

*Fuente: Elaboración propia*

El modelo es capaz de detectar la presencia de una persona y del fondo sin embargo, no siempre, determina correctamente el género de la persona en la imagen, a su vez puede obviar información contenida en la imagen.



*Fuente: Elaboración propia*

SUBTITULADO AUTOMÁTICO - DEEP LEARNING

Seleccione una imagen o ingrese el URL

URL  DIRECCION

IMAGEN ORIGINAL



SUBTITULO GENERADO

Un rebaño de ovejas se encuentra junto a un pequeño

Figura 53: Predicción desde GUI

*Fuente: Elaboración propia*

Como evaluación subjetiva se tomaron quince oraciones generadas al azar y se pudo ver aciertos en nueve de ellas, representando un **60%** (Anexo 4) .La evaluación subjetiva no es determinante ni recomendable, se exhorta referirse a los scores que son estado del arte, BLEU y METEOR.



## **CAPÍTULO V**

# **CONCLUSIONES Y RECOMENDACIONES**

## Conclusiones y recomendaciones

Se ha explorado e implementado un algoritmo de subtítulo automático basado en *Deep Learning* utilizando librerías de Python y base de datos en idioma inglés, los resultados obtenidos muestran la viabilidad de lo propuesto, la investigación puede ser sujeta a mejoras y actualizaciones. De lo implementado se concluye que:

1. La evaluación subjetiva y los scores BLEU y METEOR indican que si es posible implementar un algoritmo basado en *Deep Learning* capaz de generar subtítulos de imágenes en castellano, aún cuando el dataset original se encuentre en otro idioma.
2. Se verificó la viabilidad de la arquitectura codificador-decodificador para tareas de subtítulo automático, es posible explorar otras arquitecturas y determinar su rendimiento en referencia al implementado.
3. El uso de transferencia de aprendizaje en la detección de características facilita la implementación del modelo, obviando la necesidad de entrenar y optimizar una red neuronal adicional de detección y clasificación.
4. El decodificador con mecanismo de atención permite que el algoritmo sea capaz de priorizar elementos de la imagen y generar descripciones en torno a estos.
5. Los tiempos de entrenamiento crecen con el incremento de la base de datos y los batches sin embargo, más tiempo en entrenamiento no significa directamente un modelo mejor desarrollado.

6. La arquitectura codificador - decodificador es simple pero eficiente, es posible elegir otras que pueden generar mejores resultados sin embargo el proceso de implementación incrementa en complejidad.
7. El modelo puede ser optimizado mediante el proceso de *fine-tuning* , agregando o quitando capas, funciones de activación o incrementando la profundidad o ancho del modelo.
8. Para un mayor desempeño se puede obtener un dataset con descripciones realizadas por hablantes nativos del castellano lo cual mejoraría la sintaxis y coherencia de las oraciones obtenidas por el algoritmo.
9. El desempeño también puede mejorarse implementado un evaluador de fluidez de las oraciones traducidas y de acuerdo al score obtenido restringir o reforzar su injerencia en el entrenamiento del algoritmo.
10. En cada entrenamiento el modelo desarrolla diferentes características, aprende, cosas distintas por lo que ningún modelo final es igual a otro pese a que se utilice la misma configuración y la misma base de datos.
11. El modelo ha sido desarrollado en entorno de PC de escritorio utilizando efectivamente 8 GB de un total de 24 GB de RAM y 6 GB de VRAM, características con las que cuentan celulares de gama alta por lo que es posible de escalar a dispositivos móviles utilizando Tensorflow Lite u otras herramientas.
12. La construcción de la GUI se realizó en entorno nativo de Python, como trabajo futuro puede convertirse todo el paquete en un ejecutable (.exe) que permitiría su portabilidad y facilidad de utilización.
13. El repositorio colaborativo de Google, Colab, permite desarrollar algoritmos de *Deep Learning* escritos en python con hardware remoto competente pudiendo reducir costos asociados.





## **CAPÍTULO VI**

### **ANÁLISIS ECONÓMICO**

El proyecto realizado ha sido, en su totalidad, implementado con software (Anaconda, Docker), lenguaje de programación (Python) y librerías de libre utilización (Tensorflow, nlkt, PysimpleGUI, pandas, etc), los costos asociados al proyecto se reducen a horas hombre, adquisición de hardware con capacidad CUDA > 7.5 y suscripción a API Yandex Translate.

| Ítem         | Descripción                     | Unidad de medida | Cantidad | Costo unitario | Total             |
|--------------|---------------------------------|------------------|----------|----------------|-------------------|
| 1            | Código ML                       | hora-hombre      | 30       | S/55.00        | S/1,650.00        |
| 2            | Código GUI y rutinas auxiliares | hora-hombre      | 15       | S/35.00        | S/525.00          |
| 3            | Ordenador                       | un               | 1        | S/6,300.00     | S/6,300.00        |
| 4            | Acceso a API Yandex             | sub              | 1        | S/320.00       | S/320.00          |
| <b>Total</b> |                                 |                  |          |                | <b>S/8,475.00</b> |

Tabla 4: Costos

*Fuente: Elaboración propia*

## REFERENCIAS

- [1] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*, 2014.
- [2] Satanjeev Banerjee and Alon Lavie. Meteor: An automatic metric for mt evaluation with improved correlation with human judgments. In *Proceedings of the acl workshop on intrinsic and extrinsic evaluation measures for machine translation and/or summarization*, pages 65–72, 2005.
- [3] Nikhil Buduma and Nicholas Locascio. *Fundamentals of Deep learning: Designing next-generation machine intelligence algorithms*. " O'Reilly Media, Inc.", 2017.
- [4] Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, and Yoshua Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555*, 2014.
- [5] Alexander Jakob Dautel, Wolfgang Karl Härdle, Stefan Lessmann, and Hsin-Vonn Seow. Forex exchange rate forecasting using deep recurrent neural networks, Mar 2020.
- [6] JAY DAWANI. *Hands-on mathematics for deep learning*. PACKT Publishing Limited, 2020.
- [7] Mohamed Elgendy. *Deep Learning for Vision Systems*. Simon and Schuster, 2020.
- [8] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning*. MIT press, 2016.

- [9] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.
- [10] Raimi Karim. Illustrated: 10 cnn architectures, Nov 2020.
- [11] Nikhil Ketkar. *Deep learning with python: A hands-on introduction*. Apress, 2017.
- [12] Ryan Kiros, Ruslan Salakhutdinov, and Rich Zemel. Multimodal neural language models. In *International conference on machine learning*, pages 595–603. PMLR, 2014.
- [13] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, 25:1097–1105, 2012.
- [14] Weiyu Lan, Xirong Li, and Jianfeng Dong. Fluency-guided cross-lingual image captioning. In *Proceedings of the 25th ACM international conference on Multimedia*, pages 1549–1557, 2017.
- [15] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.
- [16] Xirong Li, Weiyu Lan, Jianfeng Dong, and Hailong Liu. Adding chinese captions to images. In *Proceedings of the 2016 ACM on International Conference on Multimedia Retrieval*, pages 271–275, 2016.
- [17] Tom M Mitchell et al. Machine learning. 1997.
- [18] Takashi Miyazaki and Nobuyuki Shimizu. Cross-lingual image caption generation. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1780–1790, 2016.

- [19] Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. Bleu: a method for automatic evaluation of machine translation. In *Proceedings of the 40th annual meeting of the Association for Computational Linguistics*, pages 311–318, 2002.
- [20] Jason Brownlee PhD, Sep 2019.
- [21] EDWARD RAFF. *Inside deep learning: Math, algorithms, models*. O'REILLY MEDIA, 2021.
- [22] Sebastian Raschka and Vahid Mirjalili. Python machine learning: Machine learning and deep learning with python. *Scikit-Learn, and TensorFlow. Second edition ed*, 2017.
- [23] Sudharsan Ravichandiran. *Hands-on deep learning algorithms with python: master deep learning algorithms with extensive math by implementing them using tensorflow*. Packt Publishing Ltd, 2019.
- [24] David Rumelhart, Geoffrey Hinton, and Ronald Williams. Learning representations by back-propagating errors. In *Letter to Nature*, pages 533–536, 1986.
- [25] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, 115(3):211–252, 2015.
- [26] Stuart Russell and Peter Norvig. Artificial intelligence: a modern approach. 2002.
- [27] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.
- [28] Anubhav Singh and Rimjhim Bhadani. *Mobile Deep Learning with TensorFlow Lite, ML Kit and Flutter: Build scalable real-world projects to implement end-to-end neural networks on Android and iOS*. Packt Publishing Ltd, 2020.

- [29] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1–9, 2015.
- [30] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2818–2826, 2016.
- [31] Divyanshu Thakur. Lstm and its equations, Jul 2018.
- [32] Michael Thomsen. Microsoft’s deep learning project outperforms humans in image recognition, Feb 2015.
- [33] Ivan Vasilev, Daniel Slater, Gianmario Spacagna, Peter Roelants, and Valentino Zocca. *Python Deep Learning: Exploring deep learning techniques and neural network architectures with Pytorch, Keras, and TensorFlow*. Packt Publishing Ltd, 2019.
- [34] Kelvin Xu, Jimmy Ba, Ryan Kiros, Kyunghyun Cho, Aaron Courville, Ruslan Salakhudinov, Rich Zemel, and Yoshua Bengio. Show, attend and tell: Neural image caption generation with visual attention. In *International conference on machine learning*, pages 2048–2057. PMLR, 2015.
- [35] Matthew D Zeiler and Rob Fergus. Visualizing and understanding convolutional networks. In *European conference on computer vision*, pages 818–833. Springer, 2014.
- [36] Aston Zhang, Zachary C Lipton, Mu Li, and Alexander J Smola. Dive into deep learning. *arXiv preprint arXiv:2106.11342*, 2021.
- [37] Lin, T.Y., Maire, M., Belongie, S., Hays, J., Perona, P., Ramanan, D., Dollár, P., & Zitnick, C. Microsoft coco: Common objects in context. In *European conference on*

*computer vision* (pp. 740–755), 2014.

- [38] Wang, H., Zhang, Y., & Yu, X. An overview of image caption generation methods. *Computational intelligence and neuroscience*, 2020.
- [39] Hossain, M. Z., Sohel, F., Shiratuddin, M. F., & Laga, H. A comprehensive survey of deep learning for image captioning. In *ACM Computing Surveys* (pp. 1 - 36), 2019.



## Glosario de términos y abreviaturas

|           |                                                             |
|-----------|-------------------------------------------------------------|
| AIC       | Attributes in Crowd                                         |
| ANN       | Artificial Neural Network                                   |
| Baidu     | Empresa tecnológica y buscador basado en China              |
| Benchmark | Prueba de rendimiento                                       |
| Bias      | Tendencia en favor de algún resultado en específico         |
| BLEU      | Bilingual Evaluation Understudy                             |
| BTT       | Backpropagation Through Time                                |
| CIFAR     | Canadian Institute for Advanced Research                    |
| CNN       | Convolutional Neural Network                                |
| CUDA      | Compute Unified Device Architecture                         |
| DL        | Deep Learning                                               |
| Flickr    | Repositorio virtual de imágenes                             |
| GPU       | Graphics processing unit                                    |
| ILSVRC    | ImageNet Large Scale Visual Recognition Challenge           |
| ImageNet  | Base de datos de imágenes                                   |
| METEOR    | Metric for Evaluation of Translation with Explicit Ordering |
| ML        | Machine Learning                                            |
| MS COCO   | Microsoft Common Objects in Context                         |
| NLKT      | <i>Librería</i> , Natural Language Toolkit                  |
| NLP       | Natural Language Processing                                 |
| PASCAL    | Base de datos de imágenes para detección de objetos         |
| Python    | Lenguaje de programación interpretado                       |
| RGB       | Red Green Blue                                              |

|            |                                                                        |
|------------|------------------------------------------------------------------------|
| RNN        | Recurrent Neural Networks                                              |
| STAIR      | Software Technology and Artificial Intelligence<br>Research Laboratory |
| Tensorflow | Plataforma de desarrollo de aprendizaje de máquina                     |
| VRAM       | Video Random Access Memory                                             |
| Yandex     | Empresa tecnológica y buscador basada en Rusia                         |



## Anexo 1: Código

### RUTINA TRADUCCIÓN YANDEX

```
import collections
import random
import json
from PIL import Image
import requests
import time

#TRADUCIR ORACIONES YANDEX IDE
key = '*****'
train_cap_1='E:/MLP/COCO_TRA_1/train_captions_20k_trad.json'

with open(train_cap_1, 'r') as f:
    train_captions = json.load(f)

def traducir_yandex(text,key):
    parameters = {
        'key' : key,
        'text' : text[8:-6],
        'lang' : 'en-es',
        'format' : 'plain',
        'options' : '1',
    }
    print("-----")
    response =
requests.get("https://translate.yandex.net/api/v1.5/tr.json/translate
", params=parameters)
    print("---")
    response_dict = response.json()
    return response_dict

for val in range(0,96000):

    trad_yandex=traducir_yandex(train_captions[val],key)
    temp=trad_yandex['text']
    caption = f"<start> {temp[0]} <end>"
    train_captions[val]=caption
    print(val, train_captions[val])

    if val%5==0:
        with open(train_cap_1,'w') as f:
            json.dump(train_captions,f)
            print('*****CHECKPOINT GUARDADO*****', val)

    if val%50==0:
        print('Inicio de espera 30 seg')
        time.sleep(30)
        print('Fin de espera')
```

## RUITNA PRINCIPAL GUI

```
import PySimpleGUI as sg
import os.path
from threading import Thread
from time import sleep
import jpg_to_png
import capt_esp
import date_name_u

sg.theme('Dark Green 2')

im_path_att=""
im_path = ""

file_list_column = [
    [ sg.Text("Seleccione una imagen o ingrese el URL")
    ],
    [ sg.Combo(['URL', 'ARCHIVO'],key="OPTION"),
      sg.Text("DIRECCION",size=(11,1)),
      sg.In(size=(25, 1), enable_events=True, key="-DIR-"),
      sg.FileBrowse("Buscar",key="-IN-"),

    ],
    [
    [
      sg.Button("Generar",key="-GEN-"),

    ],
    [
      sg.Frame("IMAGEN ORIGINAL",layout =[[sg.Image(key="-ORIGINAL-",size=(410,250))]])
    ],
    [
      sg.Frame("SUBTITULO GENERADO",layout= [[
        sg.Multiline(key="-CAPTION-",size = (57,1))]])
    ]
    ]
]

image_viewer_column = [
    [sg.Image(key="-ATTENTION-",size=(600,600))],
    [sg.Text("Ingeniería Electrónica UCSM- BSc Ademir Huanca Luque",key="-DAT-")]
]

# ----- Full layout -----

layout = [
    [
        sg.Column(file_list_column,size=(450,600)),
        #sg.VSeparator(),
        sg.Frame("GRAFICO DE ATENCIÓN:",image_viewer_column,
size=(600,600)),
    ]
]

window = sg.Window("SUBTITULADO AUTOMÁTICO - DEEP LEARNING", layout)
```

```
while True:
    event, values = window.read()
    if event == "Exit" or event == sg.WIN_CLOSED:
        break
    elif event == "-GEN-":
        option=values["OPTION"]
        print(option)
        if option == "URL":
            suf = 1
        if option == "ARCHIVO":
            suf = 0

        im_path=values["-DIR-"]
        print(im_path)
        print(suf)

        print("-GEN-")
        des=date_name_u.date_name()
        cer = capt_esp.test_im(im_path,des,suf) #TEST WITH URL

        path_att=cer[1]+".png"
        path=cer[0]
        caption=cer[2]
        caption=caption[:-5]
        caption=caption.capitalize()
        print(path)
        print(path_att)
        n_path = jpg_to_png.jpgtopng(path)
        print(n_path)
        window["-CAPTION-"].update(caption)
        window["-ORIGINAL-"].update(n_path)
        window["-ATTENTION-"].update(path_att)

window.close()
```

## RUTINAS AUXILIARES GUI

```
import datetime
from PIL import Image
import os

def date_name():
    current_date_and_time = datetime.datetime.now()
    current_date_and_time_string = str(current_date_and_time)
    cc=current_date_and_time_string[5:-7]
    bb=cc.replace(" ", "_")
    bb=bb.replace(":", "_")
    bb=bb.replace("-", "_")
    return bb

def jpgtopng(im_path_jpg):
    new_path= "E:/MLP/captioning thesis/IMAGES/"
    basewidth=500
    #basehsize=350
    im=Image.open(im_path_jpg)
    wpercent = (basewidth / float(im.size[0]))
    #basepercent = (basehsize/float(im.size[0]))
    hsize = int((float(im.size[1]) * float(wpercent)))
    #basew = int((float(im.size[1])*float(basepercent)))
    im = im.resize((basewidth, hsize), Image.ANTIALIAS)
    #im = im.resize((basew, basehsize), Image.ANTIALIAS)

    base=os.path.basename(im_path_jpg)
    os.path.splitext(base)
    name = os.path.splitext(base)[0]

    im.save(new_path+name+".png")
    png_path=new_path + name + ".png"
    return png_path

import os
os.environ['TF_CPP_MIN_LOG_LEVEL'] = '3'
import tensorflow as tf
import matplotlib.pyplot as plt
# Scikit-learn includes many helpful utilities
from sklearn.model_selection import train_test_split
from sklearn.utils import shuffle
from tqdm import tqdm
import collections
import random
import re
import numpy as np
import time
import json
from glob import glob
from PIL import Image
from keras.models import load_model
import pickle
import keras
import cargar_modelo
import random
from in_req import req_im

def test_im(image_url,a,suf):
```

```

if suf ==1:
    #PROBAR CON IMAGEN DESDE URL
    image_path = req_im(image_url,str(suf))

    #image_extension = image_url[-4:]
    #image_path =
tf.keras.utils.get_file('image'+a+image_extension, origin=image_url)

#CAMBIAR A ARCHIVO
else:
    image_path = image_url

result, attention_plot = cargar_modelo.evaluate(image_path)
print ('Prediction Caption:', ' '.join(result))
caption=(' ' .join(result))
type(caption)
print("GUARDADO: "+caption)
final_image = cargar_modelo.plot_attention(image_path, result,
attention_plot,a)
# opening the image
Image.open(image_path)
att_path=final_image[1]

return image_path, att_path,caption
#CONTADOR DE PALABRAS PARA HISTOGRAMA
l = list()
for a in range(len(img_name_vector)):
    c =train_captions[a]
    x = c.split()
    l.append(x)

contador = Counter(x for xs in l for x in set(xs))
import pandas
df = pandas.DataFrame.from_dict(contador, orient='index')
df.plot(kind='bar')
"""

```

## RUTINA DE APRENDIZAJE

```
import os
os.environ['TF_CPP_MIN_LOG_LEVEL'] = '3'
import tensorflow as tf
import random
import matplotlib.pyplot
import numpy as np
import time
import keras
import collections
import json
from nltk.translate.bleu_score import sentence_bleu
from nltk.translate.bleu_score import corpus_bleu
from PIL import Image

#Limitar el uso de RAM de GPU actualmente en 4000 MB
gpus = tf.config.experimental.list_physical_devices('GPU')
if gpus:
    try:
        tf.config.experimental.set_virtual_device_configuration(gpus[0],
[tf.config.experimental.VirtualDeviceConfiguration(memory_limit=4300)
])
    except RuntimeError as e:
        print(e)

#Cargado de biblioteca
ima_namea='E:/MLP/COCO_TRA_1/imagenes.json'
train_capta='E:/MLP/COCO_TRA_1/subtitulos.json'

with open(ima_namea, 'r') as f:
    ima_name_vector = json.load(f)

with open(train_capta, 'r') as f:
    train_capt = json.load(f)

# Obtener imagenes
e_train = sorted(set(ima_name_vector))

ima_bdat = tf.data.Dataset.from_tensor_slices(e_train)
ima_bdat =
ima_bdat.map(cargar_imagen,num_parallel_calls=tf.data.experimental.AU
TOTUNE).batch(16)

for ima, path in ima_bdat:
    path_carc = ima_feat_exc_model(ima)
    path_carc = tf.reshape(path_carc, (path_carc.shape[0], -1,
path_carc.shape[3]))

    for bf, p in zip(path_carc, path):
        path_carc = p.numpy().decode("utf-8")
        np.save(path_carc, bf.numpy())
###-----FIN EXTRACCION DE CARACTERISTICAS-----
-

# Verificar el mayor número de caracteres
def calc_mx_length(tensor):
```

```

        return max(len(t) for t in tensor)
# Elegir las 4000 de mayor longitud
top_p = 4000
#eliminar elementos de puntuación y no discernir entre mayusculas y
minusculas
tokenzr =
tf.keras.preprocessing.text.Tokenizer(num_words=top_p,oov_token="<unk
>",filters='!"#$%&()*+.,-/:;=?@[\\]^_`{|}~',lower=True)
tokenzr.fit_on_texts(train_capt)
train_seqs = tokenzr.texts_to_sequences(train_capt)
cap_vect = tf.keras.preprocessing.sequence.pad_sequences(train_seqs,
padding='post')
mx_length = calc_mx_length(train_seqs)

i2c_vect = collections.defaultdict(list)
for ima, cap in zip(ima_name_vector, cap_vect):
    i2c_vect[ima].append(cap)

#Config
t_batch = 128
t_bufer = 300
emb_dim = 256
units = 512
t_vocab = top_p + 1
num_steps = len(ima_name_t) // t_batch

ft_shape = 2048
att_ft_shape = 64

# Cargar charactersiticas
def load_func1(ima_name, cap):
    ima_tensor = np.load(ima_name.decode('utf-8')+'.npy')
    return ima_tensor, cap
bdata = tf.data.Dataset.from_tensor_slices((ima_name_t, cap_t))

bdata = bdata.map(lambda itema, itemb: tf.numpy_function(load_func1,
[itema, itemb], [tf.float32, tf.int32]),
                num_parallel_calls=tf.data.experimental.AUTOTUNE)

bdata = bdata.shuffle(t_bufer).batch(t_batch)
bdata = bdata.prefetch(buffer_size=tf.data.experimental.AUTOTUNE)

class BahdanauAtt(tf.keras.Model):
    def __init__(self, units):
        #Inicializar tres capas necesarias para Bahadanau
        super(BahdanauAtt, self).__init__()
        self.C1 = tf.keras.layers.Dense(units)
        self.C2 = tf.keras.layers.Dense(units)
        self.F = tf.keras.layers.Dense(1)

    def call(self, caractrs, hidden):
        #caractrs = salida del encoder#
        #hidden = capa oculta de salida del decodificador
        #return = context_vector y attention_weights procesados por el
mecanismo
        hidden_with_time_axis = tf.expand_dims(hidden, 1)

        #El score se calcula de acuerdo a la formula detallada

```

```

        BahdanauAtt_hidden_layer = self.C1(caractrs) +
self.C2(hidden_with_time_axis )
        BahdanauAtt_hidden_layer = (tf.nn.tanh(BahdanauAtt_hidden_layer))
        score = self.F(BahdanauAtt_hidden_layer)
        #Los attention_weights se calculan de acuerdo a la formula
        att_weights = tf.nn.softmax(score, axis =1)
        ctext_vector = att_weights * caractrs
        ctext_vector = tf.reduce_sum(ctext_vector, axis =1)

        return att_weights, ctext_vector

class CNN_enco(tf.keras.Model):
    def __init__(self, emb_dim):
        super(CNN_enco, self).__init__()
        self.fc = tf.keras.layers.Dense(emb_dim)

    def call(self, k):
        k = self.fc(k)
        k = tf.nn.relu(k)
        return k

class RNN_deco(tf.keras.Model):
    def __init__(self, emb_dim, units, t_vocab):
        super(RNN_deco, self).__init__()
        #Conectar las cpas de características con la RNN en las mismas
dimensiones
        self.units = units
        #Iniciar una capa embebida
        self.embedding = tf.keras.layers.Embedding(t_vocab, emb_dim)
        #Iniciar capa GRU
        #return_sequences= True representa cada salida de la GRU
        #return_state = True, representa que se retorna la salida de la
GRU y la última capa oculta
        #recurrent_initializer = 'glorot_uniform' , indica que el método
de inicialización se distribuye uniformememnte
        self.gru = tf.keras.layers.GRU(self.units,return_sequences=True,
return_state=True, recurrent_initializer='glorot_uniform')
        #Iniciar dos capas Fully connected
        self.fc1 = tf.keras.layers.Dense(self.units)
        self.fc2 = tf.keras.layers.Dense(t_vocab)
        #Iniciar capa de mecanismo de atención
        self.BahdanauAtt = BahdanauAtt(self.units)

    def call(self, k, caractrs, hidden):
        #Utilizar función de atención para obtener scores y vector de
contexto
        ctext_vector, att_weights = self.BahdanauAtt(caractrs, hidden)
        #Ingresar la forma de la salida obtenida a trafes de la capa
embebida (batch_size, 1, embedding_dim)
        k = self.embedding(k)
        #conectar k y el resultado del mecanismo de atención a una nueva
salida k, forma de la salida: (batch_size, 1, embedding_dim +
hidden_size)
        k = tf.concat([tf.expand_dims(ctext_vector, 1), k], axis =-1)
        #Introducir k a la capa de GRU
        output, state = self.gru(k)
        #Introducir k a capa fully connected forma de la salida
(batch_size, max_length, hidden_size)

```

```

        k = self.fc1(output)
        #Cambiar la forma de k para ingresarla a la segunda capa fully
        connected, forma de la salida (batch_size * max_length, hidden_size)
        k = tf.reshape(k, (-1, k.shape[2]))
        #Ingresar k a la segunda capa fully connected, forma de la salida
        (batch_size * max_length, vocab)
        k = self.fc2(k)

        #Retornar el resultado del codificado, weights del mecanismo de
        atención y estado de capa oculta de GRU
        return k, state, att_weights

    def reset_est(self, t_batch):
        #Inicializar el tensor de pesos (weights) de la capa oculta de
        GRU a valores en cero
        return tf.zeros((t_batch, self.units))

encoder = CNN_enco(emb_dim)
decoder = RNN_deco(emb_dim, units, t_vocab)

#Metodo de optimización y loss function
#Se elige optimizador Adam
optimizador = tf.keras.optimizers.Adam()
#Se utiliza sparse categorical cross entropy como funcion de perdida
(loss function)
loss_vector =
tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True,
reduction='none')

#El valor de perdida debe medirse cada vez que se obtiene un
resultado parcial
def loss_fgra(real, pred):
    #Se comapra el resultado predecido con el resultado real
    # tf.math.equal compara el real y 0
    mk1 = tf.math.logical_not(tf.math.equal(real, 0))
    #utilizar funcion para calcular loss
    loss_ = loss_vector(real, pred)
    mk1 = tf.cast(mk1, dtype=loss_.dtype)
    loss_ *= mk1

    return tf.reduce_mean(loss_)

path_save = "./checkpoints/test1/train"
print(path_save)
checkpoint_s = tf.train.Checkpoint(encoder=encoder, decoder=decoder,
optimizer = optimizador)
checkpoint_s_manager = tf.train.CheckpointManager(checkpoint_s,
path_save, max_to_keep=5)

recover_epoch = 0
if checkpoint_s_manager.latest_checkpoint:
    recover_epoch = int(checkpoint_s_manager.latest_checkpoint.split('-'
)[-1])
    # restaurar entrenamiento en checkpoin guardado
    checkpoint_s.restore(checkpoint_s_manager.latest_checkpoint)

loss_graph = []

```

```
#----- PROCESO -----
@tf.function
def train_etapa(ima_tensor, target):
    #perdida inicial = 0
    loss = 0
    #Iniciar los estados ocultos del decoder
    hidden = decoder.reset_est(t_batch=target.shape[0])
    #Definir la primera descripción (texto) que ingresa al decoder,
    #definido por <start>
    dec_input = tf.expand_dims([tokenzr.word_index['<start>']] *
    target.shape[0], 1)

    #Abrir un administrador de contexto para guardar el gradiente
    with tf.GradientTape() as tape:
        #Usar el encoder para procesar la imagen ingresada
        caractrs = encoder(ima_tensor)

        #iniciar el uso del decoder para decodificar ciclicamente, el tamaño
        #de este es la longitud maxima del tensor de descripciones
        for x in range(1, target.shape[1]):
            #Usar el decoder para obtener la primera predicción y el
            #tensor implícito
            prediccion, hidden, _ = decoder(dec_input, caractrs,
            hidden)

            #calcular la perdida del proceso de decodificación
            loss += loss_fgra(target[:, x], prediccion)
            #Se usa teacher_forcing para definir la entrada de la
            #siguiente decodificación
            dec_input = tf.expand_dims(target[:, x], 1)

        #Después de completar un bucle se calcula la pérdida
        t_loss = (loss / int(target.shape[1]))
        #Obtener variables del modelo de entrenamiento
        trainable_var = encoder.trainable_variables +
        decoder.trainable_variables
        #usar el administrador de gradientes para resolver la gradiente de la
        #variable (parametro)
        grads = tape.gradient(loss, trainable_var)
        #Actualizar parametros basados en la gradiente
        optimizador.apply_grads(zip(grads, trainable_var))

        print('***step end***')
    #Retornar la perdida promedio en la oración trabajada
    return loss, t_loss

#----- ENTRENAMIENTO-----
#Numero de repeticiones del proceso
epochs = 10

#Bucle de entrenamiento
for epoch in range(recover_epoch, epochs):
    #Guardar el tiempo de inicio de cada ronda de entrenamiento
    t_inicio = time.time()
    #Perdida total al inicio de ronda es cero
    t_loss = 0

    #Bucle a través de cada batch de dataset
    for (batch, (ima_tensor, target)) in enumerate(bdata):
```

```

        #llamar a train_etapa para obtener la perdida total y
        promedio del batch
        batch_loss, t_loss = train_etapa(ima_tensor, target)
        #Sumar la perdida promedio del batch al total de batches
        t_loss += t_loss

        if batch % 100 == 0:
            print ('Epoch ', epoch + 1, 'Batch ', batch, 'Loss ',
batch_loss.numpy() / int(target.shape[1]))

#Plotear perdida promedio de la ronda
    loss_graph.append(t_loss / num_steps)
#cada 10 epochs guardar el progreso de entrenamiento
    if epoch % 10 == 0:
        print('****save checkpoint **** ')
        checkpoint_s_manager.save()
#Mostrar Epoch y perdida promedio
    print ('Epoch', epoch + 1, 'Loss', t_loss/num_steps)
#mostrar tiempo tomado para la ronda
    print ('Tiempo para 1 epoch {} sec\n'.format(time.time() -
t_inicio))

#BLEU EVALUATION
def BLEU_ev(originales, predicted):
    #CALCULA BLEU SCORE A NIVEL ORACION, originales es una lista de
    listas de las oraciones originales, predicted es un string
    for val in range(len(originales)):
        sen = originales[val][8:-5]
        sen = sen.replace(".", "")
        sen = sen.split()
        originales[val]=sen
    predicted = predicted.split()
    bleu1= sentence_bleu(originales, predicted, weights=(1.0, 0,
0,0))
    bleu2= sentence_bleu(originales, predicted, weights=(0.5, 0.5,
0,0))
    bleu3= sentence_bleu(originales, predicted, weights=(0.3, 0.3,
0.3 ,0))
    bleu4= sentence_bleu(originales, predicted, weights=(0.25, 0.25,
0.25, 0.25))
    return bleu1,bleu2,bleu3,bleu4

# descripciones del set de validacion
total_captions = list()
predicted = list()
for a in range(len(ima_name_val)):
    print(a)
    image = ima_name_val[a]
    real_caption = ' '.join([tokenizr.index_word[i] for i in
cap_val[a] if i not in [0]])
    #oraciones para bleu
    ima_index= ima_name_vector.index(image)
    real_captions =list()

    for i in range(4):
        if ima_index >=49999:
            i= i-1

```

```
temp= train_capt[ima_index+i].split()
real_captions.append(temp)

total_captions.append(real_captions)

out_result, att_graph = evaluate(image)
predicted.append(out_result[:-1])
print("Oración generada: ", ' '.join(out_result))
print(a, "finished")

print('BLEU-1: %f' % corpus_bleu(total_captions, predicted,
weights=(1.0, 0, 0, 0)))
print('BLEU-2: %f' % corpus_bleu(total_captions, predicted,
weights=(0.5, 0.5, 0, 0)))
print('BLEU-3: %f' % corpus_bleu(total_captions, predicted,
weights=(0.3, 0.3, 0.3, 0)))
print('BLEU-4: %f' % corpus_bleu(total_captions, predicted,
weights=(0.25, 0.25, 0.25, 0.25)))
```



## RUTINA EVALUACIÓN BLEU Y METEOR

```
#BLEU EVALUATION
def BLEU_ev(originales, predicted):
    #CALCULA BLEU SCORE A NIVEL ORACION, originales es una lista de
    listas de las oraciones originales, predicted es un string
    for val in range(len(originales)):
        sen = originales[val][8:-5]
        sen = sen.replace(".", "")
        sen = sen.split()
        originales[val]=sen
    predicted = predicted.split()
    bleu1= sentence_bleu(originales, predicted, weights=(1.0, 0,
0,0))
    bleu2= sentence_bleu(originales, predicted, weights=(0.5, 0.5,
0,0))
    bleu3= sentence_bleu(originales, predicted, weights=(0.3, 0.3,
0.3 ,0))
    bleu4= sentence_bleu(originales, predicted, weights=(0.25, 0.25,
0.25, 0.25))
    return bleu1,bleu2,bleu3,bleu4

# captions on the validation set
total_captions = list()
predicted = list()
for a in range(len(img_name_val)):
    print(a)
    #rid = np.random.randint(0, len(img_name_val))
    image = img_name_val[a]
    real_caption = ' '.join([tokenizer.index_word[i] for i in
cap_val[a] if i not in [0]])
    #oraciones para bleu
    ima_index= img_name_vector.index(image)

    real_captions =list()

    for i in range(4):
        if ima_index >=49999:
            i= i-1

        temp= train_captions[ima_index+i].split()
        real_captions.append(temp)

    total_captions.append(real_captions)

    result, attention_plot = evaluate(image)
    predicted.append(result[:-1])
    print("Oración generada: ", ' '.join(result))
    print(a, "finished")

print('BLEU-1: %f' % corpus_bleu(total_captions, predicted,
weights=(1.0, 0, 0, 0)))
print('BLEU-2: %f' % corpus_bleu(total_captions, predicted,
weights=(0.5, 0.5, 0, 0)))
print('BLEU-3: %f' % corpus_bleu(total_captions, predicted,
weights=(0.3, 0.3, 0.3, 0)))
print('BLEU-4: %f' % corpus_bleu(total_captions, predicted,
weights=(0.25, 0.25, 0.25, 0.25)))
```

## RUTINAS AUXILIARES

```
import datetime
from PIL import Image
import os

def date_name():
    current_date_and_time = datetime.datetime.now()
    current_date_and_time_string = str(current_date_and_time)
    cc=current_date_and_time_string[5:-7]
    bb=cc.replace(" ", "_")
    bb=bb.replace(":", "_")
    bb=bb.replace("-", "_")
    return bb

def jpgtopng(im_path_jpg):
    new_path= "E:/MLP/captioning thesis/IMAGES/"
    basewidth=500
    #basehsize=350
    im=Image.open(im_path_jpg)
    wpercent = (basewidth / float(im.size[0]))
    #basepercent = (basehsize/float(im.size[0]))
    hsize = int((float(im.size[1]) * float(wpercent)))
    #basew = int((float(im.size[1])*float(basepercent)))
    im = im.resize((basewidth, hsize), Image.ANTIALIAS)
    #im = im.resize((basew, basehsize), Image.ANTIALIAS)

    base=os.path.basename(im_path_jpg)
    os.path.splitext(base)
    name = os.path.splitext(base)[0]

    im.save(new_path+name+".png")
    png_path=new_path + name + ".png"
    return png_path

#CONTADOR DE PALABRAS PARA HISTOGRAMA
l = list()
for a in range(len(img_name_vector)):
    c =train_captions[a]
    x = c.split()
    l.append(x)

contador = Counter(x for xs in l for x in set(xs))
import pandas
df = pandas.DataFrame.from_dict(contador, orient='index')
df.plot(kind='bar')
"""
```

## Anexo 2: Especificaciones de ordenador

Computer: ASUS TUF Gaming FX505DU\_FX505DU

CPU: AMD Ryzen 7 3750H (Picasso, PCO-B1)  
2300 MHz (23.00x100.0) @ 1309 MHz (13.11x99.9)

Motherboard: ASUS FX505DU

Chipset: AMD Promontory/Bixby FCH

Memory: 24576 MBytes @ 1198 MHz, 17-17-17-39  
- 8192 MB PC25600 DDR4 SDRAM - Micron 8ATF1G64HZ-3G2J1  
- 16384 MB PC19200 DDR4 SDRAM - Samsung M471A2K43CB1-CRC

Graphics: NVIDIA GeForce GTX 1660 Ti [ASUS]  
NVIDIA GeForce GTX 1660 Ti, 6144 MB GDDR6 SDRAM

Graphics: AMD Radeon RX Vega 10 (Picasso) [ASUS]  
AMD Radeon Vega, 512 MB DDR4 SDRAM

Drive: HP SSD S700 500GB, 488.4 GB, Serial ATA 6Gb/s @ 6Gb/s

Drive: Micron\_2200V\_MTFDHBA512TCK, 500.1 GB, NVMe

Sound: NVIDIA TU116 - High Definition Audio Controller

Sound: AMD Raven/Renoir/Van Gogh/Cezanne/Rembrandt - Audio Processor - HD Audio Controller

Network: RealTek Semiconductor RTL8168/8111 PCI-E Gigabit Ethernet NIC

Network: RealTek Semiconductor, Device ID: C822

Network: 802.11n USB Wireless LAN Card

OS: Microsoft Windows 10 Professional (x64) Build 19042.1165 (20H2)

## Anexo 3: Muestra de oraciones generadas

```
.... for i in range(15):
.... rid = np.random.randint(0, len(img_name_val))
.... image = img_name_val[rid]
.... ima_index= img_name_vector.index(image)
.... result, attention_plot = evaluate(image)
.... real_caption = ' '.join([tokenizer.index_word[i] for i in cap_val[rid]
if i not in [0]])
.... print ('Descripcion real:', real_caption)
.... print ('Descripcion generada:', ' '.join(result))
.... #plot_attention(image, result, attention_plot)
.... real_captions =list()
.... real_capt = list()
.... capt=' '.join(result)[:5]
.... for i in range(5):
.... temp= train_captions[ima_index+i]
.... real_capt.append(temp)
.... #temp2 =temp.split()
.... #real_captions.append(temp2)
....
.... bleu = BLEU_ev(real_capt,capt)
.... real_capt2 = list()
.... for i in range(5):
.... temp= train_captions[ima_index+i][8:-5]
.... real_capt2.append(temp)
.... meteor= meteor_score(real_capt2,capt )
....
.... print('BLEU-2: %f' % bleu[1])
.... print('BLEU-1: %f' % bleu[0])
.... print('BLEU-3: %f' % bleu[2])
.... print('BLEU-4: %f' % bleu[3])
.... print('METEOR: %f' % meteor)
```

Descripcion real: <start> tres autobuses de dos pisos se sientan a un lado de la carretera <end>

Descripcion generada: tres autobuses de dos pisos se sientan a un lado de una intersección <end>

BLEU-2: 1.000000

BLEU-1: 1.000000

BLEU-3: 0.971812

BLEU-4: 0.923473

METEOR: 0.845836

Descripcion real: <start> un perro saltando en el aire para atrapar un frisbee <end>

Descripcion generada: un perro sostiene un frisbee <end>

BLEU-2: 0.000000

BLEU-1: 0.269597

BLEU-3: 0.000000

BLEU-4: 0.000000

METEOR: 0.307377

Descripcion real: <start> surtido de vinos dulces y flores en una mesa con platos <end>

Descripcion generada: una mesa blanca cubierta con botellas de vino <end>

BLEU-2: 0.790569

BLEU-1: 0.875000

BLEU-3: 0.705432

BLEU-4: 0.594604

METEOR: 0.777345  
 Descripción real: <start> matrícula de minnesota en la parte trasera de una motocicleta <end>  
 Descripción generada: hay una motocicleta con una matrícula de minnesota <end>  
 BLEU-2: 0.731925  
 BLEU-1: 0.750000  
 BLEU-3: 0.734265  
 BLEU-4: 0.680375  
 METEOR: 0.815529  
 Descripción real: <start> un niño pequeño se sienta solo en un banco del parque <end>  
 Descripción generada: una persona que está en un banco <end>  
 BLEU-2: 0.491957  
 BLEU-1: 0.644123  
 BLEU-3: 0.359610  
 BLEU-4: 0.000000  
 METEOR: 0.426136  
 Descripción real: <start> una almohada negra está acostada en una cama completamente blanca <end>  
 Descripción generada: una cama con dos almohadas y una almohada decorativa con sábanas a cámara <end>  
 BLEU-2: 0.720577  
 BLEU-1: 0.692308  
 BLEU-3: 0.717331  
 BLEU-4: 0.667274  
 METEOR: 0.768846  
 Descripción real: <start> un hombre que está parado en la tierra <end>  
 Descripción generada: un hombre caminando por un camino <end>  
 BLEU-2: 0.453174  
 BLEU-1: 0.477688  
 BLEU-3: 0.359116  
 BLEU-4: 0.000000  
 METEOR: 0.480769  
 Descripción real: <start> una habitación en una antigua casa <unk> con muebles de época y alfombras <end>  
 Descripción generada: una habitación en su interior <end>  
 BLEU-2: 0.164971  
 BLEU-1: 0.180717  
 BLEU-3: 0.150955  
 BLEU-4: 0.000000  
 METEOR: 0.209472  
 Descripción real: <start> un avión de pasajeros con dos motores volando en el cielo <end>  
 Descripción generada: un gran jet volando por el cielo <end>  
 BLEU-2: 0.567505  
 BLEU-1: 0.743038  
 BLEU-3: 0.414834  
 BLEU-4: 0.000000  
 METEOR: 0.420755  
 Descripción real: <start> se ha utilizado una lente ojo de pez para <unk> un aparador con varios objetos decorativos <end>  
 Descripción generada: un escritorio de madera con el <unk> <end>  
 BLEU-2: 0.654654  
 BLEU-1: 0.857143  
 BLEU-3: 0.589149  
 BLEU-4: 0.454994  
 METEOR: 0.711429  
 Descripción real: <start> árboles altos y <unk> en un área tipo <unk> con hierba alta muestra un par de elefantes caminando <end>

Descripción generada: árboles altos y <unk> <unk> con árboles en <unk> <end>

BLEU-2: 0.333487

BLEU-1: 0.497133

BLEU-3: 0.000000

BLEU-4: 0.000000

METEOR: 0.217544

Descripción real: <start> un bateador con el número 54 está intentando un swing <end>

Descripción generada: detrás de él <end>

BLEU-2: 0.000000

BLEU-1: 0.064648

BLEU-3: 0.000000

BLEU-4: 0.000000

METEOR: 0.057471

Descripción real: <start> una mujer está de pie en una silla roja <end>

Descripción generada: una dama de rojo de <unk> de pie en un lado de una pared <end>

BLEU-2: 0.468807

BLEU-1: 0.571429

BLEU-3: 0.301191

BLEU-4: 0.000000

METEOR: 0.391579

Descripción real: <start> un oso negro está saliendo del bosque <end>

Descripción generada: un oso caminando por el bosque mirando a la caramelo mirando a través de los árboles en el día en el día en el suelo <end>

BLEU-2: 0.447214

BLEU-1: 0.480000

BLEU-3: 0.412322

BLEU-4: 0.312085

METEOR: 0.652174

Descripción real: <start> este jet lleva un transbordador espacial <end>

Descripción generada: un avión a reacción volando por el cielo <end>

BLEU-2: 0.372923

BLEU-1: 0.551561

BLEU-3: 0.307476

BLEU-4: 0.000000

METEOR: 0.332836

In [47]: