UNIVERSIDAD DE CHILE
FACULTAD DE CIENCIAS FÍSICAS Y MATEMÁTICAS
DEPARTAMENTO DE INGENIERÍA ELÉCTRICA

BÚSQUEDA POR TEXTO DE IMÁGENES SIN ETIQUETAR

MEMORIA PARA OPTAR AL TÍTULO DE
INGENIERO CIVIL ELÉCTRICO

BASTIÁN LUCIANO DÍAZ RENJIFO

PROFESOR GUÍA:
JUAN BARRIOS NUÑEZ

MIEMBROS DE LA COMISIÓN:
CRISTÓBAL SILVA PÉREZ
ANDRÉS CABA RUTTE

SANTIAGO DE CHILE
2019

RESUMEN DE LA MEMORIA PARA OPTAR
AL TÍTULO DE INGENIERO CIVIL ELÉCTRICO
POR: BASTIÁN LUCIANO DÍAZ RENJIFO
FECHA: 2019
PROF. GUÍA: JUAN BARRIOS NUÑEZ

BÚSQUEDA POR TEXTO DE IMÁGENES SIN ETIQUETAR

En la era de internet es el contenido multimedia el que concentra la mayoría del tráfico y se espera que para el 2020 este tipo de datos tenga un tráfico de más de 100,000 petabytes al mes.

Esto junto con el creciente uso de redes sociales ha provocado que buscar imágenes sea una práctica habitual. Esta situación está resuelta para imágenes que tienen texto asociado como las que se encuentran en Google. ¿Pero qué pasa si se necesita buscar en una base de datos de imágenes que no tienen texto asociado? Por ejemplo, las imágenes subidas a una red social, o las carpetas del computador que tienen las imágenes de las vacaciones.

Esta es la principal pregunta que motiva este trabajo, el cual está basado principalmente en las investigaciones de Snoek, Dong y Li quienes aplican la técnica word2visualvec, donde la idea principal es entrenar una red neuronal para que aprenda a transformar el espacio de los descriptores de texto al espacio de los descriptores de imágenes, conservando la relación semántica entre el texto y las imágenes.

Para el entrenamiento de la red neuronal se usó el dataset MSCOCO, el cual consiste en más de 80,000 imágenes para entrenar y validar el modelo y más de 40,000 para ser usadas como conjunto de prueba. Además de las fotos, este dataset contiene 5 descripciones de texto para cada una de ellas. Así en total se tienen más de 400.000 ejemplos de texto-imagen para entrenar la red neuronal.

Los experimentos se dividieron en 3 partes que se diferencian en el tipo de descriptor de texto y descriptor de imagen usados. Para el texto se usaron los descriptores TF-IDF, R-TF-IDF y word2vec y para las imágenes se usaron los descriptores VLAD y Deep Features (basado en la VGG16). En total se entrenaron 21 modelos, 18 de ellos fueron entrenados en inglés y 3 en español. La evaluación de los modelos consideró el tiempo de entrenamiento, el costo en el conjunto de validación y el Average Precision de los resultados retornados.

Los resultados obtenidos son prometedores y permiten sacar conclusiones muy importantes. Dentro de ellas es que el impacto de los descriptores de texto se refleja mayormente en el tiempo de entrenamiento, donde el descriptor word2vec es el más rápido de entrenar. Mientras que los descriptores de imágenes impactan significativamente en la relevancia de los resultados, siendo Deep Features basados en la red pre-entrenada VGG16 el que tuvo los mejores resultados, alcanzando una efectividad de 20 % en coincidencias exactas y 68 % en coincidencias parciales.

Future is now

Agradecimientos

A mis compañeros de trabajo que me vieron llegar varias veces trasnochado después de una noche trabajando en la memoria.

A mis amigos que entendieron cada vez que dije que no iba a salir para poder trabajar en la tesis.

A mi profesor guía Juan Manuel Barrios, porque fue en su curso que surgió esta idea y después de un par de iteraciones resulto en un tema de memoria muy entretenido y desafiante.

Y por último, a mi familia que son la mayor motivación que tengo para superar los desafíos que la vida me presenta.

Tabla de Contenido

1. Introducción **1**
- 1.1. Motivación . . . 1
- 1.2. Objetivo General . . . 2
- 1.3. Objetivos Específicos . . . 2

2. Marco Teórico **3**
- 2.1. Perceptrón . . . 3
- 2.2. Redes Neuronales . . . 4
- 2.3. Redes Neuronales Convolucionales . . . 6
- 2.4. VGG16 . . . 8
- 2.5. Deep Features . . . 8
- 2.6. SIFT . . . 9
- 2.7. VLAD . . . 11
- 2.8. Análisis de Componentes Principales . . . 13
- 2.9. Stopwords y Stemming . . . 14
- 2.10. TF-IDF . . . 14
- 2.11. N-Gramas . . . 15
- 2.12. Word2vec . . . 16
- 2.13. Búsqueda de vecinos más cercanos . . . 17
- 2.14. Average Precision at k . . . 18

3. Revisión Bibliográfica **19**
- 3.1. Content Based Image Retrieval . . . 19
  - 3.1.1. Query by Example . . . 20
  - 3.1.2. Query by Sketch . . . 20
  - 3.1.3. Query by Description . . . 21
- 3.2. Estado del Arte . . . 22

4. Descripción e Implementación del Sistema **24**
- 4.1. Descripción general del Sistema . . . 24
- 4.2. Implementación . . . 26
  - 4.2.1. Dataset . . . 26
  - 4.2.2. Descriptores de Texto . . . 26
  - 4.2.3. Descriptores de Imágenes . . . 27
  - 4.2.4. Embedding . . . 27
  - 4.2.5. Software . . . 28

4.3. Diseño de Experimentos 28
4.3.1. Evaluación 29

5. Experimentos y Análisis de Resultados **31**
5.1. Texto Inglés - VLAD 31
5.1.1. Reducción de dimensionalidad 31
5.1.2. Caracterización de Modelos 32
5.1.3. Entrenamiento 32
5.1.4. Evaluación AP@k 34
5.2. Texto Inglés - VGG16 36
5.2.1. Reducción de dimensionalidad 36
5.2.2. Caracterización de Modelos 36
5.2.3. Entrenamiento 37
5.2.4. Evaluación AP@k 39
5.3. Texto Español - VGG16 41
5.3.1. Reducción de dimensionalidad 41
5.3.2. Caracterización de Modelos 41
5.3.3. Entrenamiento 41
5.3.4. Evaluación AP@k 43
5.3.5. DEMO: Imágenes Propias 45
5.4. Problemas Encontrados y Posibles Soluciones 45

6. Conclusiones y Trabajo Futuro **47**
6.1. Conclusiones 47
6.2. Trabajo Futuro 48

Bibliografía **48**

A. Evolución de Entrenamientos **53**

B. Resultado de Entrenamientos **55**

Índice de Tablas

2.1. Arquitectura red VGG16 8
2.2. Ejemplo de cálculo de Average Precision 18

4.1. Primer grupo de experimentos 28
4.2. Segundo grupo de experimentos 28
4.3. Tercer grupo de experimentos 29

5.1. Dimensión de los descriptores de texto y la cantidad de parámetros de los modelos generados para el primer grupo de experimentos. 32
5.2. Resultados del entrenamiento del sistema Texto Inglés - VLAD 34
5.3. Dimensión de los descriptores de texto y la cantidad de parámetros de los modelos generados para el segundo grupo de experimentos. 37
5.4. Resultados del entrenamiento del sistema Texto Inglés - VGG16 39
5.5. Dimensión de los descriptores de texto y la cantidad de parámetros de los modelos generados para el tercer grupo de experimentos. 41
5.6. Resultados del entrenamiento del sistema Texto Español - VGG16 42

B.1. Resultados del entrenamiento del sistema Texto Inglés - VLAD 55
B.2. Resultados del entrenamiento del sistema Texto Inglés - VGG16 55

Índice de Ilustraciones

1.1. Resultado de busqueda de "horses on the beach" 1

2.1. Representación gráfica de un perceptrón 3
2.2. Representación gráfica de una red neuronal 4
2.3. Overfitting 5
2.4. Early stopping 6
2.5. Representación gráfica de max pooling 7
2.6. Arquitectura de CNN (LeCun et al. [27] 7
2.7. Extracción de Deep Features (Zeiler and Fergus [43]) 9
2.8. Filtrado de Pirámide Espacial (Almeida et al. [2]) 10
2.9. Orientación del punto de interés 10
2.10. Descriptor sift 11
2.11. Resultado de SIFT (Almeida et al. [2]) 11
2.12. VLAD 12
2.13. Ejemplo de PCA 13
2.14. Ejemplo de BoW 15
2.15. Ejemplo de N-gramas 15
2.16. Skipgram model 16
2.17. Resultados de skipgram model 17
2.18. K-Means Tree 17
2.19. Ejemplo de documentos retornados por un sistema CBIR 18

3.1. Sistema CBIR con Query by Example 20
3.2. Sistema CBIR con Query by Sketch 20
3.3. Ejemplo de búsqueda por Sketch en Pepe Ganga 21
3.4. Sistema CBIR con Query by Description 21
3.5. Ejemplo de QBD en *Google Photos* 22

4.1. Representación gráfica del espacio compartido entre texto e imágenes (Dong et al. [13]) 24
4.2. Representación gráfica del modelo genérico 25
4.3. Búsqueda de imágenes 25
4.4. Ejemplo de dataset MSCOCO 26
4.5. Ejemplo de evaluación exacta y parcial 30

5.1. Varianza explicada por las componentes principales del descriptor VLAD . . 31

5.2. Entrenamiento de red neuronal usando TF-IDF, VLAD y distintos valores de Dropout . . . 33
5.3. Entrenamiento de red neuronal usando R-TF-IDF y VLAD . . . 33
5.4. Entrenamiento de red neuronal word2vec y VGG16 . . . 34
5.5. Resultados de Average Precision bajo una coincidencia exacta . . . 35
5.6. Resultados de Average Precision bajo una coincidencia parcial . . . 35
5.7. Ejemplos de las imágenes retornadas por la red entrenada con R-TF-IDF (inglés) y VLAD . . . 35
5.8. Varianza explicada por las componentes principales del descriptor VGG16 . . . 36
5.9. Entrenamiento de red neuronal usando TF-IDF y VGG16 . . . 38
5.10. Entrenamiento de red neuronal usando R-TF-IDF y VGG16 . . . 38
5.11. Entrenamiento de red neuronal usando word2vec y VGG16 . . . 38
5.12. Resultados de Average Precision bajo una coincidencia exacta . . . 39
5.13. Resultados del retrieval bajo una coincidencia parcial . . . 40
5.14. Ejemplos de las imágenes retornadas por la red entrenada con R-TF-IDF (inglés) y VGG16 . . . 40
5.15. Entrenamiento de red neuronal usando TF-IDF, descriptor VGG16 y Dropout 0.2 . . . 42
5.16. Entrenamiento de red neuronal usando R-TF-IDF, descriptor VGG16 y Dropout 0.2 . . . 42
5.17. Entrenamiento de red neuronal usando word2vec, descriptor VGG16 y Dropout 0.2 . . . 43
5.18. Resultados del retrieval bajo una coincidencia exacta . . . 43
5.19. Resultados del retrieval bajo una coincidencia parcial . . . 44
5.20. Ejemplos de las imágenes retornadas por la red entrenada con R-TF-IDF (español) y VGG16 . . . 44
5.21. Ejemplo del búsqueda en imágenes propias . . . 45

A.1. Entrenamiento de red neuronal usando R-TF-IDF - VLAD y distintos valores de Dropout . . . 53
A.2. Entrenamiento de red neuronal usando word2vec - VLAD y distintos valores de Dropout . . . 53
A.3. Entrenamiento de red neuronal usando TF-IDF - VGG16y distintos valores de Dropout . . . 53
A.4. Entrenamiento de red neuronal usando R-TF-IDF - VGG16 y distintos valores de Dropout . . . 54
A.5. Entrenamiento de red neuronal usando word2vec - VGG16 y distintos valores de Dropout . . . 54

Capítulo 1

Introducción

1.1. Motivación

En la era de internet es el contenido multimedia el que concentra la mayoría del tráfico, y se espera que para el 2020 este tipo de datos tenga un tráfico de más de 100,000 petabytes al mes (Cisco [10]).

Esto junto con el creciente uso de redes sociales ha provocado que compartir y buscar imágenes sea una práctica habitual. En la figura 1.1 se tienen los resultados de la búsqueda de "horses on the beach". Si se toma un poco de atención, se puede ver que el título de la primera imagen también es "horses on the beach". Este es el principio que usa el buscador para entregar los resultados, esto es, hacer un match entre el texto ingresado en el buscador y el texto asociado a las imágenes.

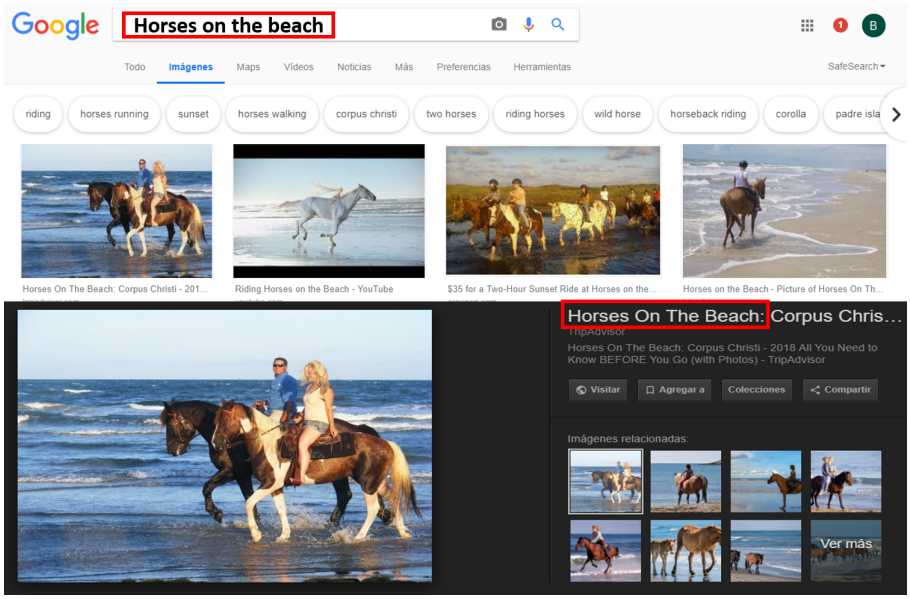


Figura 1.1: Resultado de busqueda de "horses on the beach"

¿Pero qué pasa si se necesita buscar en una base de datos de imágenes que no tienen texto asociado? Por ejemplo, las imágenes subidas a una red social, o las carpetas del computador que tienen las imágenes de las vacaciones. Ésta es la principal pregunta que motiva este trabajo, donde el foco está en entrenar un modelo que permita realizar búsquedas por texto de imágenes sin etiquetar.

Para enfrentar este problema se usa un enfoque de recuperación de información, donde el contenido visual y textual de los documentos se representan por vectores, conocidos como descriptores, y la búsqueda consiste en localizar descriptores similares a un descriptor de consulta de acuerdo a alguna función de distancia.

De esta manera, para buscar imágenes por medio de texto se requiere poder comparar descriptores que representan el texto de la consulta con los descriptores que representan el contenido visual de las imágenes. Esto hace necesario implementar un método que transforme el espacio vectorial de los descriptores de texto en el espacio vectorial de los descriptores de imágenes.

1.2. Objetivo General

El objetivo de este trabajo es implementar un método que permita transformar descriptores de texto en descriptores visuales, para buscar imágenes relevantes a partir de una consulta de texto tanto en inglés como español.

1.3. Objetivos Específicos

1. Implementar los descriptores de texto TF-IDF, R-TF-IDF (TF-IDF con vocabulario reducido) y word2vec en inglés y español.
2. Implementar los descriptores de imágenes VLAD y Deep Features.
3. Diseñar y entrenar una red neuronal que convierta descriptores de texto (TF-IDF, R-TF-IDF y word2vec) a descriptores de imágenes (VLAD y Deep Features).
4. Usar la métrica Average Precision at k (k = 1 y 10) para evaluar la relevancia de las imágenes retornadas por los modelos entrenados.

Capítulo 2

Marco Teórico

2.1. Perceptrón

El perceptrón es la unidad básica de las redes neuronales y está inspirado en el funcionamiento de las neuronas biológicas (Caparrini [6]). Éste consta de una capa de entrada, donde cada señal x_{i} es ponderada por un peso independiente $w_{\mathrm{i}j}$, y luego son sumadas. Finalmente, el resultado de la suma entra a una función de activación para generar la salida del perceptrón. De esta manera un perceptrón es representado por la ecuación 2.1 y se puede ver en la figura 2.1. Las funciones de activación más populares hoy en día son la ReLU y Softmax definidas por las ecuaciones 2.2 y 2.3 respectivamente.

$$y = f_{activacion}(\sum_{\mathrm{i}=1}^{N} x_{\mathrm{i}} \cdot w_{\mathrm{i}}) \tag{2.1}$$

$$\mathrm{ReLU}(x) = max(0, x) \tag{2.2}$$

$$Softmax_{\mathrm{i}}(x) = \frac{\exp x_{\mathrm{i}}}{\sum x_{\mathrm{i}}} \tag{2.3}$$

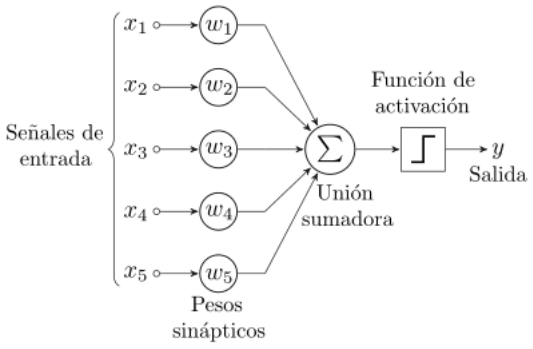


Figura 2.1: Representación gráfica de un perceptrón

2.2. Redes Neuronales

Las redes neuronales son un conjunto de perceptrones agrupados en una arquitectura de capas como se puede ver en la figura 2.2. Este tipo de redes se llaman Multi Layer Perceptron Fully Connected (MLP) porque todos los perceptrones de la capa anterior están conectados a todos los perceptrones de la capa siguiente.

Las MLP están compuestas por una capa de entrada, una o más capas ocultas y una capa de salida tal como se ve en la figura 2.2. La capa de entrada corresponde a los datos que serán usados para entrenar la red. En cambio, la o las capas ocultas son perceptrones que ajustan sus pesos a medida que aprenden de los datos de entrenamiento. Por último, la capa de salida también ajusta sus pesos en el entrenamiento, pero además tiene la función de entregar la salida final del modelo, esto puede ser la clasificación de una imagen o la predicción de una serie de tiempo.

El número de capas ocultas y la cantidad de unidades en estas son hiper-parámetros que deben ser definidos manualmente, aunque existen ciertas heurísticas para comenzar. Vale la pena mencionar que el número de capas y la cantidad de unidades impacta directamente en el número de parámetros. Por ejemplo, si una imagen en blanco y negro de tamaño 100×100 es entregada a una MLP con 10 unidades en la capa oculta, esta tendrá $100 \times 100 \times 10 = 100,000$ parámetros solo en la capa oculta.

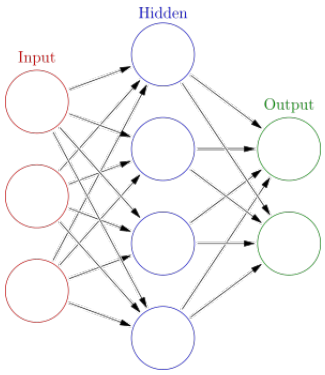


Figura 2.2: Representación gráfica de una red neuronal

Las redes neuronales pueden ser usadas para resolver distintos problemas de aprendizaje supervisado como por ejemplo clasificación de imágenes o predicción de series de tiempo. Para esto deben ser entrenadas usando datos que representen el fenómeno y también se necesita la variable objetivo que es lo que se desea predecir.

El algoritmo que se utiliza para entrenar las redes neuronales se llama backpropagation, el cual ajusta los pesos de la red neuronal para minimizar una función de costo (que depende del tipo de problema). El funcionamiento de este algoritmo se puede resumir en los siguientes pasos:

1. Se entrega un dato de entrenamiento a la red y se hace una predicción
2. Se compara el valor que la red predijo con el valor real que debió entregar usando una función de costo adecuada para el problema (por ejemplo, Cross Entropy para

clasificación o RMSE para regresión)

3. Se calcula el gradiente de la función de costo y se propaga a las unidades de la red
4. Se actualizan los pesos de manera proporcional al gradiente propagado a las unidades

Es fácil darse cuenta de que calcular el gradiente para cada dato de entrenamiento es una operación costosa, sobre todo, cuando se tiene grandes volúmenes de datos. Para solucionar esto, se utiliza el entrenamiento por batches, el cual consiste en acumular el error de predicción para un determinado número de ejemplos, y luego, se actualizan los pesos.

Otro problema que surge a partir de esta arquitectura es que debido a la gran cantidad de parámetros que tienen las redes neuronales, la posibilidad de overfitting es muy grande. El overfitting es cuando un modelo es sobreentrenado y comienza a memorizar los datos, perdiendo su capacidad de generalización. Esto se puede ver gráficamente en la figura 2.3. Aquí, la línea verde representa un modelo que memorizo los datos, y en efecto, se tiene overfitting, mientras que la línea negra representa un modelo entrenado correctamente.

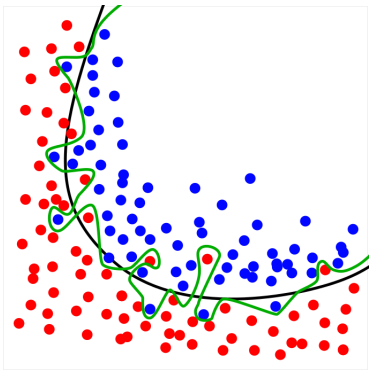


Figura 2.3: Overfitting

Para disminuir el riesgo de overfitting, se usan dos técnicas: Early Stopping y Dropout. La técnica *Early Stopping* (Caruana et al. [7]) consiste en entrenar la red y al mismo tiempo se evalúa su desempeño, de esta manera, si la función de costo en el conjunto de entrenamiento está bajando, pero en el conjunto de validación comienza a subir, significa que la red ha comenzado a memorizar los datos y se debe detener su entrenamiento. Esto se muestra en la figura 2.4.

El Dropout (Srivastava et al. [40]) es otra técnica que ayuda a reducir el riesgo de overfitting. La idea principal es asignar una probabilidad de Dropout que significa que cada unidad tendrá dicha probabilidad de desactivarse y no actualizará sus pesos al recibir un dato de entrenamiento. Con esto se busca que neuronas entre capas no queden asociadas fuertemente, lo que evita tener salidas fijas para cada dato de entrada y por tanto se previene el sobreajuste. Esta técnica se usa solo durante el entrenamiento de la red, mientras que para hacer predicciones todas las neuronas están activas.

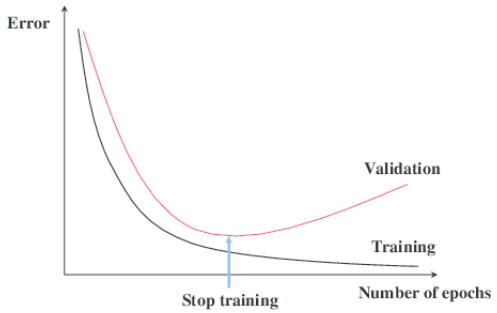


Figura 2.4: Early stopping

2.3. Redes Neuronales Convolucionales

Las redes neuronales convolucionales (CNN por su sigla en inglés) son similares a las MLP, pero están optimizadas para que los datos de entrada sean imágenes de manera que el número de parámetros no aumente de manera descontrolada al tener un input de gran dimensión (Karpathy [23]).

Para lograr esto las CNN hacen uso de capas de convolución y capas de pooling. Las capas de convolución están formadas por filtros de tamaño $alto \times ancho \times canales$ que hacen convolución con la imagen de entrada. De esta manera, si una imagen de tamaño $32 \times 32 \times 3$ pasa por una capa convolucional de 12 filtros de $3 \times 3 \times 3$, la salida será de tamaño $32 \times 32 \times 12$. Los pesos que conforman los kernels son aprendidos durante el entrenamiento e intuitivamente, la red aprenderá filtros que destaquen patrones dentro de la imagen, como orientaciones dominantes, colores o formas (LeCun et al. [27]).

Las capas convolucionales tienen 3 hiper-parámetros: profundidad, stride y padding. La profundidad de una capa convolucional corresponde a la cantidad de filtros que se aplicarán sobre la imagen de entrada, y como resultado es la profundidad del volumen de salida de esta capa, como se vio en el ejemplo anterior. El valor del stride define cada cuantos pixeles se va a aplicar el filtro. Los valores más comunes son 1 y 2. Finalmente, el padding consiste en rellenar el borde de la imagen con valores (comúnmente 0) para que el alto y ancho de la imagen de entrada sea igual al ancho y alto de la imagen de salida. Una animación de cómo opera la capa de convolución se puede ver en *http://cs231n.github.io/convolutional-networks/*.

La capa de pooling tiene el objetivo de reducir la dimensión de la representación de la imagen de manera progresiva. Esto lo hace agrupando los valores de la imagen ya sea tomando el promedio de un cuadrante o el máximo (esta operación es la más popular conocida como max-pooling). El significado de esta operación resulta en la búsqueda de patrones mas grandes y complejos dentro de una imagen. La capa de pooling tiene dos hiper-parámetros: stride (cada cuantos pixeles se aplica el agrupamiento) y el tamaño del cuadrante. La representación gráfica de esta operación se puede ver en 2.5

A modo de resumen, las redes neuronales convolucionales son redes compuestas por capas de convolución, capas de pooling y capas fully connected. La primera tiene el objetivo de aprender a capturar los patrones más importantes dentro de las imágenes, la segunda se aplica para encontrar patrones cada vez más grandes y complejos en las imágenes, y la tercera es

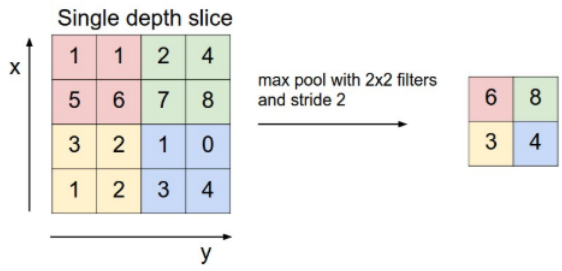


Figura 2.5: Representación gráfica de max pooling

la que resuelve el problema final, ya sea la clasificación de la imagen o una regresión. Todas estas etapas se pueden ver en la figura 2.6.

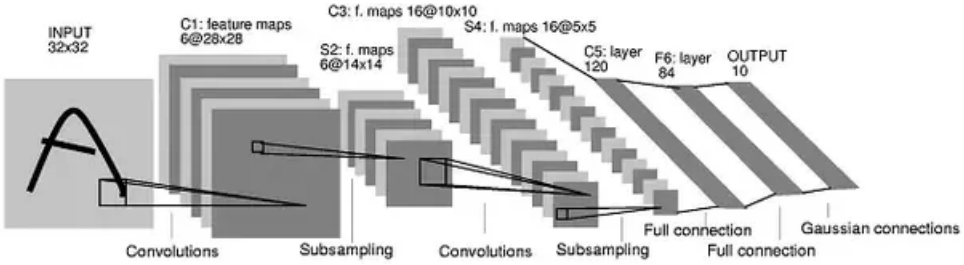


Figura 2.6: Arquitectura de CNN (LeCun et al. [27]

2.4. VGG16

La red VGG16 pertenece a la familia de las CNN, pero tiene la particularidad de que es muy profunda, específicamente contiene 16 capas que deben ser entrenadas, lo que se traduce en más de 138 millones de parámetros (Simonyan and Zisserman [39]). En la tabla 2.1 se puede ver la arquitectura de esta red.

De la arquitectura de la red se puede observar que todos los filtros de las capas convolucionales son de tamaño 3×3, los cuales aumentan en número a medida que se avanza en las capas. Estas capas convolucionales usan un padding = 1, lo que provoca que el tamaño de la imagen de entrada sea igual al tamaño de la imagen de salida. Por otro lado, las capas de max-pooling usan un stride = 2, reduciendo el tamaño de la imagen de entrada a esta capa a la mitad.

Entrenar estas redes desde cero tiene un gran costo computacional, por lo tanto, para hacer uso de esta, se usan los pesos pre-entrenados en el dataset ImageNet (Deng et al. [11]), el cual consiste en más de 1 millón de imágenes que pertenecen a 1,000 clases distintas.

Numero Capa	VGG16
1	input 224x224 imagen RGB
2	conv3-64
3	conv3-64
4	max-pooling
5	conv3-128
6	conv3-128
7	conv3-128
8	max-pooling
9	conv3-256
10	conv3-256
11	conv3-256
12	max-pooling
13	conv3-512
14	conv3-512
15	conv3-512
16	max-pooling
17	conv3-512
18	conv3-512
19	conv3-512
20	max-pooling
21	FC - 4,096
22	FC - 4,096
23	FC - 1,000
24	Softmax

Tabla 2.1: Arquitectura red VGG16

2.5. Deep Features

Un método efectivo para aplicar redes neuronales convolucionales en datasets pequeños es usar redes pre-entrenadas (Chollet [8]). Una red pre-entrenada es una red que fue previamente entrenada con un gran volumen de datos. Si este dataset fue lo suficientemente grande y representativo, entonces, los atributos espaciales que aprendió la red (filtros de las capas convolucionales) pueden ser usados de manera genérica para resolver otros problemas de visión computacional.

Por ejemplo, si se entrena una CNN como la VGG16 en ImageNet (Deng et al. [11]), dataset que contiene fotos de animales tales como perros, gatos, cebras y objetos como autos

y casas, los pesos que aprendió pueden ser usados como punto de partida para entrenar la misma red, pero para resolver otro tipo de problemas, como puede ser la clasificación de distintas razas de perro.

De la misma manera, los pesos aprendidos por la red pueden ser vistos como extractores de características, es decir, los valores de los pixeles de la imagen original pasan a través de la red mientras es transformada en información que le permite a la capa fully connected clasificar correctamente.

Por ejemplo, si se corta la red VGG16 pre-entrenada en Imagenet en la capa 21 (ver tabla 2.1), se tiene que la salida de la red será una representación de los patrones de la imagen de entrada que le permite a las dos últimas capas realizar la clasificación. Es justamente este proceso lo que se llama extracción de deep features, donde una red pre-entrenada es cortada en una capa intermedia para obtener un vector de baja dimensionalidad que contiene información relevante de la imagen de entrada. Esto se puede ver en la figura 2.7, donde la imagen de los perros es reducida a una nueva imagen donde el fondo es totalmente eliminado y resaltan los patrones comunes como las orejas, la lengua, nariz entre otros.

Figura 2.7: Extracción de Deep Features (Zeiler and Fergus [43])

2.6. SIFT

SIFT (de la sigla en inglés Scale Invariant Feature Transform) es un algoritmo que tiene por objetivo detectar puntos de interés en una imagen y crear un descriptor de cada uno de ellos, de tal manera que estos sean invariantes a la escala y a las rotaciones, es decir, un punto de interés sigue siendo un punto de interés en la imagen rotada o reducida. Este algoritmo esta compuesto por 5 pasos que se revisan a continuación (Lowe [29]):

1. **Filtrado de pirámide espacial**: Es fácil notar que para detectar pequeños detalles se debe tomar una escala espacial pequeña mientras que para detectar grandes patrones se necesita una escala espacial más grande. Este principio es el que opera en la primera etapa del algoritmo SIFT. Esta etapa construye progresivamente un set de imágenes difuminadas por filtros gaussianos de σ variable para resaltar detalles a distintas escalas. Luego se toma la diferencia entre estas imágenes para obtener una "diferencia de

gaussianas". Este proceso se repite para subsamplings de la imagen original, de esta forma, el algoritmo se vuelve invariante a la escala. Este proceso se puede ver en la figura 2.8.

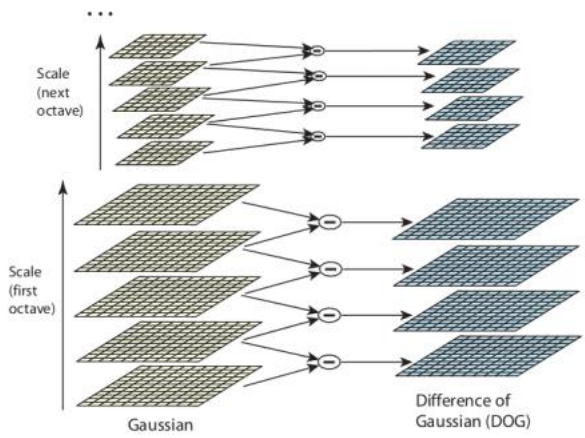


Figura 2.8: Filtrado de Pirámide Espacial (Almeida et al. [2])

2. **Detección de local extrema**: Los puntos llamados local extrema son aquellos puntos negros rodeados de puntos blancos (máximo) y puntos blancos rodeados de puntos negros (mínimo). El radio que define la vecindad de un punto está definido por el valor de σ del filtro usado. De esta manera, la vecindad de un punto es pequeña cuando σ es pequeño y viceversa. Todos estos puntos son candidatos a ser puntos de interés.
3. **Localización de puntos de interés**: Esta etapa filtra los candidatos para quedarse solo con los puntos de interés reales. Para esto, calcula una expansión de Taylor de segundo orden de la vecindad del punto, y si el valor de la expansión de Taylor del candidato es mayor a 0.03 entonces se considera un punto de interés. Si no supera ese umbral el candidato es descartado. El valor 0.03 es el que se usa en la presentación de este algoritmo.
4. **Calculo de orientación**: Para que el algoritmo sea invariante a la rotación, se debe calcular la orientación del punto de interés. Esto se hace calculando los gradientes de cada punto en la vecindad del punto de interés, luego se hace un histograma de las direcciones de los gradientes y finalmente, la dirección que tiene la mayoría de los puntos es la orientación del punto de interés. Esto se puede ver en la figura 2.9.

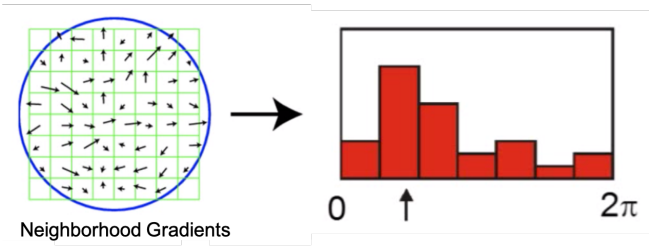


Figura 2.9: Orientación del punto de interés

5. **Descriptor local**: Finalmente, se debe crear un descriptor para cada punto de interés detectado. Al igual que la etapa anterior, esto se hace creando histogramas de las direcciones de los gradientes, pero esta vez se divide la vecindad en 4 cuadrantes,

generando 4 histogramas con 8 direcciones cada uno. Con esto el descriptor de un punto de interés tiene $4 \times 4 \times 8 = 128$ dimensiones.

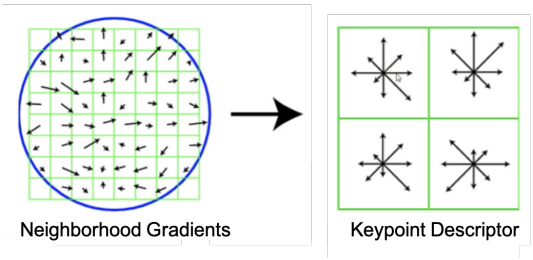


Figura 2.10: Descriptor sift

Finalmente, en la figura 2.11 se puede ver cada uno de los puntos de interés detectados por SIFT, junto a la escala a la que fueron detectados que es proporcional al radio de los círculos dibujados y su orientación está indicada por la línea dentro de los círculos. Se puede ver como pequeños detalles como esquinas y bordes fueron detectados al igual que patrones más planos como la superficie de la infraestructura y nubes.

Figura 2.11: Resultado de SIFT (Almeida et al. [2])

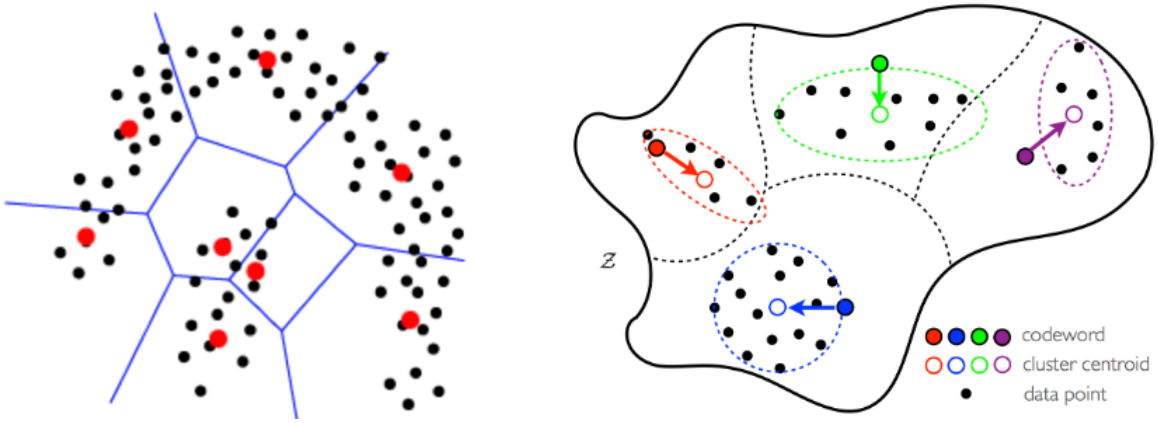
2.7. VLAD

VLAD (Vector of Locally Aggregated Descriptors) es un algoritmo desarrollado por Jegou et al. [21] y tiene por objetivo agregar descriptores locales como SIFT para crear un descriptor global. A continuación, se detallan los pasos que componen este algoritmo:

1. **Calculo de descriptores locales**: Primero se deben calcular los descriptores locales (por ejemplo SIFT) de todas las imágenes del conjunto de entrenamiento.
2. **Codebook**: Una vez se han calculado todos los descriptores locales, se hace un clustering sobre ellos y los centroides de los cluster son llamadados visual words. El número de clusters es un parámetro de este método y comúnmente usa 64.

3. **Vector de diferencias**: Una vez se obtienen los centros de los clusters, se pueden agregar los vectores locales. La forma de hacer esto, es que cada descriptor local es asociado al cluster más cercano. Luego se calcula la diferencia entre el valor del descriptor local y el centro del cluster. Esta diferencia es asociada al cluster en cuestión. Este proceso es repetido para todos los descriptores locales de una imagen. Finalmente, todas las diferencias asociadas a un cluster son sumadas, y luego, las diferencias totales de cada cluster son concatenadas para formar el descriptor global de la imagen. por ejemplo, si se usa SIFT de 128 dimensiones y un codebook de 64 palabras, el vector global de la imagen tendrá $128 \times 64 = 8,192$ dimensiones.

En la figura 2.12 se puede ver de manera gráfica como se calcula un descriptor VLAD en un espacio de 2 dimensiones. En la figura 2.12a, los puntos negros son los descriptores locales de todas las imágenes del conjunto de entrenamiento y los puntos rojos son los centroides de los cluster encontrados en este espacio. En la figura 2.12b, los puntos negros son los descriptores locales de una imagen en particular y los puntos con bordes de color son los centroides calculados en el paso anterior. Cada uno de los puntos es asociado a su centroide más cercano, que se puede ver con los puntos de color sólido y las flechas que indican su cluster más cercano. Esto se hace para todos los puntos y se calcula la distancia entre cada uno de los puntos y sus centroides. Por último, todas las distancias de un centroide son sumadas y son concatenadas en un solo vector que es de dimensión $k \times dim$ donde k es el número de centroides y dim la dimensión del espacio de los descriptores locales.



(a) Cálculo de centroides

(b) Diferencia de vectores

Figura 2.12: VLAD

La intuición detrás de esta representación es que el clustering es capaz de capturar los patrones de las imágenes del conjunto de entrenamiento de manera general, pero luego al tomar la diferencias entre los descriptores locales de una imagen en particular y los centroides ya calculados se tiene qué tan alejada y en qué dirección está la imagen de la imagen promedio del conjunto de entrenamiento.

2.8. Análisis de Componentes Principales

El análisis de componentes principales o PCA por su sigla en inglés, es una técnica no supervisada de reducción de dimensionalidad. Esta técnica tiene por objetivo encontrar las direcciones de mayor variabilidad en los datos. Por ejemplo en la figura 2.13 los datos están graficados en sus coordenadas originales (x,y), sin embargo, al usar PCA se encuentran un nuevo eje de coordenadas (PCA1,PCA2) de tal forma que al proyectar los datos sobre el eje PCA1 se pierde mucha menos información que al proyectarlos sobre el eje x o y.

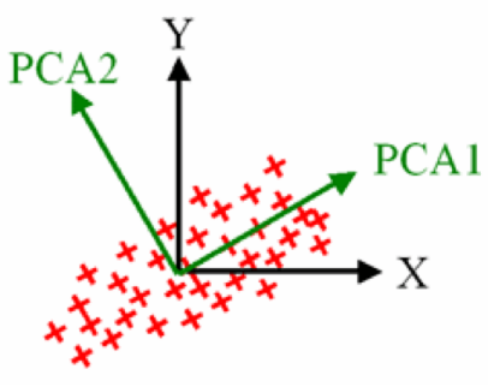


Figura 2.13: Ejemplo de PCA

Para calcular las componentes principales se considera un conjunto de datos X en formato de tabla dado por:

$$X = \begin{bmatrix} x_{11} & \dots & x_{1m} \\ \vdots & \ddots & \\ x_{n1} & & x_{nm} \end{bmatrix} \tag{2.4}$$

Donde cada columna representa un atributo y cada fila representa una observación. Es fundamental que las columnas tengan media 0 (Orchard [32]), en caso contrario, se les debe restar el promedio. Luego se puede calcular la matriz de varianza-covarianza empírica como:

$$S = \frac{1}{n-1} X^T X = V D V^T \tag{2.5}$$

El último término de la ecuación 2.5 corresponde a la diagonalización de la matriz de varianza-covarianza, donde V es la matriz de vectores propios ortogonales ordenados de tal forma que:

$$|\lambda_1| > |\lambda_2| > |\lambda_3| > ... > |\lambda_m| \tag{2.6}$$
$$V = [v_1, v_2, ..., v_m]$$

Donde λ_i es el valor propio asociado al vector propio v_i. Además, los vectores propios están alineados con las direcciones de máxima varianza de los datos. De hecho, la variabilidad de los datos proyectados en un vector propio es proporcional al valor del valor propio respectivo.

Como los vectores propios están ordenados en función de los valores propios de manera creciente, se está ordenando directamente la contribución de cada proyección a la variabilidad de los datos. De esta forma, solo se deben seleccionar los k vectores propios que expliquen la varianza deseada.

Cuando ya se han encontrado las componentes principales P, se deben transformar los datos haciendo una proyección sobre esta nueva base ortogonal según la siguiente ecuación.

$$X_{pca} = XP \tag{2.7}$$

2.9. Stopwords y Stemming

Para poder trabajar con texto, es necesario hacer un pre-procesamiento como por ejemplo pasar todo a minúscula, y que palabras como *perro* y *perros* sean entendidos como un único término. A continuación, se detallan dos etapas del procesamiento de texto que consisten en remover stopwords y hacer un stemming.

1. **Stopwords**: Las stopwords son aquellas palabras que no tienen ningún significado semántico como artículos (el, la, etc.) y pronombres (ellos, ellas, etc.) que deben ser eliminadas del texto. Si bien no existe una lista definitiva de estas palabras hay librerías como NLTK (Bird and Loper [4]) que entregan un listado de stopwords para distintos idiomas incluyendo español e inglés.
2. **Stemming**: El stemming es un método para transformar las palabras a su raíz. Este método consiste en un conjunto de reglas secuenciales, como por ejemplo, transformar todas las palabras en pasado a presente, o en el caso del inglés eliminar el termino *ing* de las palabras terminadas en *ing*. El método más popular de stemming es el **Algoritmo de Porter** (Porter [35]), pero se han hecho algunas mejoras que son capturadas por el **Snowball Stemmer** (Porter [36]).

2.10. TF-IDF

El descriptor TF-IDF significa *Term Frecuency - Inverse Document Frecuency* y es un método que transforma texto en un vector (Salton and Buckley [38]). Esta técnica vive en el paradigma de los modelos de *Bag of Words* (BoW) donde el texto es representado en base a la frecuencia de sus palabras como se puede ver en la figura 2.14.

Sin embargo, el descriptor TF-IDF agrega mucha más información que solo la frecuencia de las palabras. Son dos ideas las que sustentan el vector TF-IDF:

1. Las palabras que se repiten mucho en un documento son importantes (término TF)
2. Las palabras que se repiten en muchos documentos no aportan información para diferenciarlos entre ellos (término IDF)

Figura 2.14: Ejemplo de BoW

Para calcular el valor de cada término se debe usar la ecuación 2.8, donde w_{ij} es el valor de la palabra i en el documento j, $frec_{ij}$ es la frecuencia el término i en el documento j , n_i es la cantidad de veces que el termino i se repite a lo largo de los documentos y N es la cantidad total de documentos. Así la componente $1 + \log(frec_{ij})$ representa la importancia de la palabra en el documento (TF) y la componente $\log(N/n_i)$ es la importancia de la palabra en el conjunto de entrenamiento. De esta forma si una palabra se repite mucho en un documento su valor será alto, pero si una palabra aparece en todos los documentos no aporta información y su valor será 0.

$$w_{ij} = \begin{cases} (1 + \log(frec_{ij})) * \log(N/n_i), & \text{if } frec_{ij} \geq 0 \\ 0, & \text{si no} \end{cases} \tag{2.8}$$

2.11. N-Gramas

Los N-Gramas son combinaciones entre palabras adyacentes en un documento, como se aprecia en la figura 2.15. Este método es muy usado en text mining para ampliar el vocabulario y capturar la relación entre palabras adyacentes y así las representaciones generadas por ejemplo con TF-IDF tienen más información semántica.

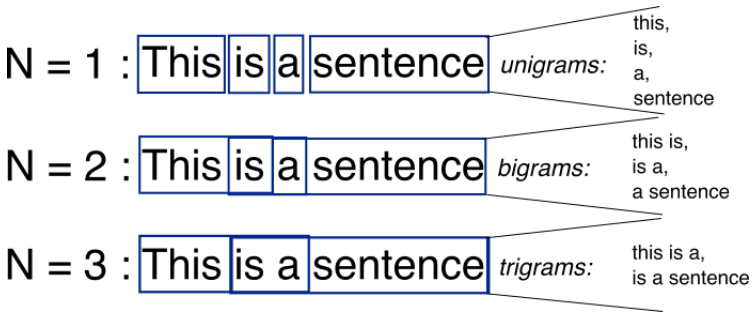


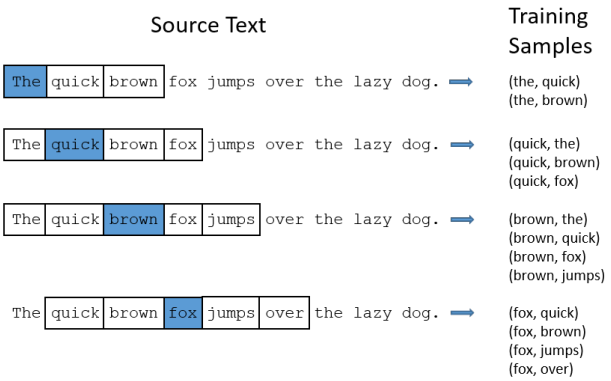
Figura 2.15: Ejemplo de N-gramas

2.12. Word2vec

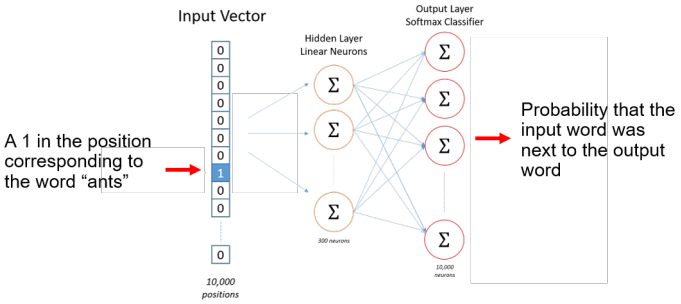
Una de las principales desventajas de los métodos clásicos de representación de texto como TF-IDF es que no existe similitud de palabras. Por ejemplo, con el descriptor TF-IDF las palabras perro y mascota son igual de distintas que las palabras perro y camión. Es claro notar que las palabras perro y camión son muy diferentes, mientras que perro y mascota tienen un significado más cercano. Esta es la principal motivación de los modelos de word2vec, donde palabras con un significado semántico similar son cercanas entre sí y palabras con un significado semántico diferentes son distantes (Mikolov et al. [30]).

El término word2vec engloba un conjunto de modelos que intentan satisfacer las condiciones presentadas anteriormente. A continuación, se describe el modelo skipgram que es el usado en este trabajo. El modelo skipgram está basado en redes neuronales con una capa oculta que se llama capa de embedding ya que es la capa que crea los vectores densos para cada palabra.

Como toda red neuronal se necesitan datos de entrenamiento. El conjunto de entrenamiento consiste en palabras adyacentes dentro de una ventana de contexto (el tamaño de esta ventana de contexto es un hiper-parámetro del modelo). En la figura 2.16a se puede ver cómo queda el conjunto de entrenamiento al aplicar una ventana de contexto de tamaño 2, donde la palabra central es el input y las palabras adyacentes son el target.



(a) Ejemplos de entrenamiento



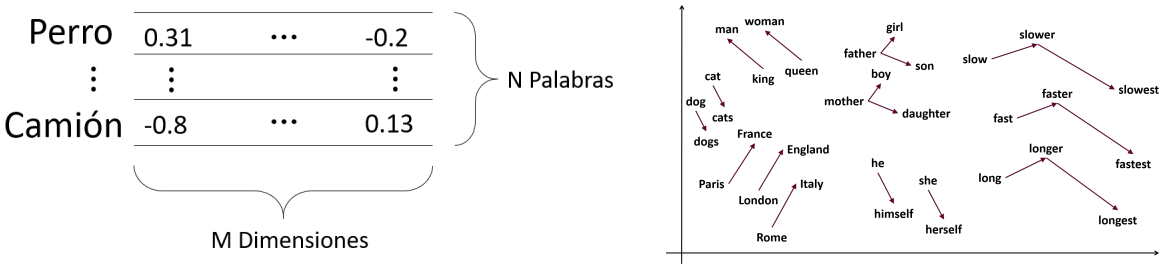
(b) Entrenamiento del modelo skipgram

Figura 2.16: Skipgram model

En la figura 2.16b está la arquitectura de la red. El número de unidades ocultas es la cantidad de dimensiones que tendrá cada palabra. En la práctica se recomienda usar entre 50 y 300 neuronas. La entrada y la salida de la red es una representación binaria donde va un 1 en la posición que corresponde a la palabra de entrada y salida. De esta forma, debido a como se forma el dataset de entrenamiento las palabras que aparecen en un mismo contexto tendrán representaciones similares.

Como resultado del entrenamiento se tiene una representación densa para cada palabra capturada por la capa oculta de la red neuronal, lo que se traduce en una *look up table*

como en la figura 2.17a para un corpus de N palabras y una red con M unidades en la capa oculta. Por último, en la figura 2.17b hay un ejemplo de cómo al graficar las representaciones obtenidas en un espacio de 2 dimensiones las palabras que tienen un significado semántico similar son cercanas. Uno de los resultados más notables de este método es que se pueden hacer operaciones aritméticas entre palabras obteniendo resultados que siguen teniendo un significado como por ejemplo $king - man + woman = queen$



(a) Look up table de palabras y sus vectores

(b) Espacio semántico de word2vec

Figura 2.17: Resultados de skipgram model

2.13. Búsqueda de vecinos más cercanos

La búsqueda de los vecinos más cercanos (KNN por su sigla en inglés) permite encontrar en una base de datos los elementos mas cercanos a un elemento de consulta (Imandoust and Bolandraftar [20]). Para realizar esta comparación de elementos se definen 3 conceptos clave: descriptores, distancia e índice.

Los descriptores son vectores que describen a los elementos de una base de datos y están directamente relacionados a la extracción de características. Por ejemplo, para el texto están los descriptores TF-IDF, word2vec, entre otros. Y para las imágenes existen descriptores como VLAD y Deep Features.

La distancia es una medida objetiva de que tan diferentes son dos vectores. Así, dos vectores son parecidos cuando la distancia entre ellos es pequeña. Las medidas de distancia más populares son la Euclidiana y Distancia Coseno.

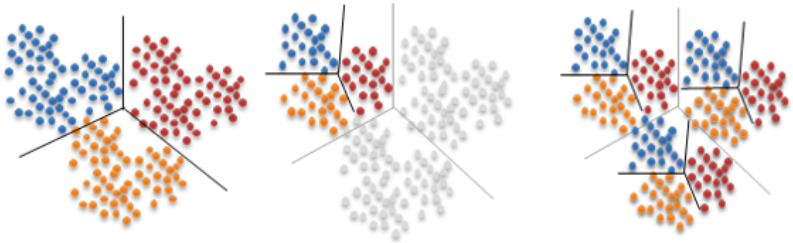


Figura 2.18: K-Means Tree

Finalmente, los índices son una estructura de datos que permiten comparar los elementos de una base de datos de manera eficiente. Un ejemplo de esto son los K-Means Trees. El

índice K-Means Tree es un tipo de índice multi-dimensional que aplica K-Means sobre los datos de manera recursiva para particionar el espacio los datos. En el ejemplo de la figura 2.18 se puede ver este proceso donde primero se aplica K-Means (con k = 3) sobre todos los datos, luego se aplica en cada cluster de manera independiente y así sucesivamente. Esto permite que el elemento de consulta solo sea comparado con los elementos de la región con el centroide más parecido a él y no contra toda la base de datos.

De esta manera el funcionamiento de un buscador queda definido por los siguientes pasos:

1. Cálculo de los descriptores de la base de datos
2. Definición de función de distancia
3. Construcción de índice
4. Consultar la base de datos

2.14. Average Precision at k

Average Precision at k (o $AP@k$) es una medida de evaluación de algoritmos de ranking y está compuesta por dos partes: $P@k$ y Average. $P@k$ se define como la proporción del número documentos relevantes dentro de los primeros k documentos retornados por el sistema. Lo anterior se debe hacer para un conjunto de consultas que se le hace al sistema, las cuales son promediadas para obtener el valor final de $AP@k$.

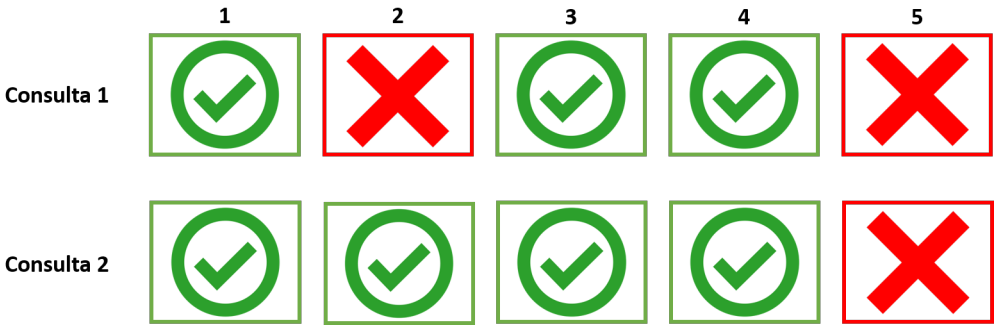


Figura 2.19: Ejemplo de documentos retornados por un sistema CBIR

En el ejemplo de la figura 2.19 se tiene que los cuadrados con un check son resultados correctos y los que tienen una equis son incorrectos. Así los resultados de AP@k son los siguientes:

	P@1	P@2	P@3	P@4	P@5
Consulta 1	1	0.5	0.6	0.75	0.6
Consulta 2	1	1	1	1	0.8
AP@k (Promedio)	1	0.75	0.8	0.875	0.7

Tabla 2.2: Ejemplo de cálculo de Average Precision

Capítulo 3

Revisión Bibliográfica

3.1. Content Based Image Retrieval

Content Based Image Retrieval (CBIR por su sigla en inglés) consiste en la búsqueda o recuperación de imágenes a partir de su contenido. Entre las distintas componentes de un CBIR se destacan las siguientes:

- Almacenamiento
- Extracción de características
- Indexamiento
- Interfaz de usuario

El **almacenamiento** consiste en crear la base de datos que va a ser consultada. En este caso, la base de datos contiene todas las imágenes que serán consultadas por el o los usuarios. La **extracción de características** es el núcleo de estos sistemas y consiste en capturar la información contenida en las imágenes (esto se lleva a cabo usando distintos métodos, dos de ellos descritos en el siguiente capítulo). El **indexamiento** permite que las consultas a la base de datos se resuelvan de manera eficiente y en el menor tiempo posible. Finalmente, la **interfaz de usuario** es la parte visual de este sistema que permite a los usuarios hacer consultas a la base de datos y ver los resultados.

A continuación, se describen 3 enfoques distintos de sistemas de CBIR. El primero, es uno de los enfoques más comunes que consiste en búsqueda de imágenes a partir de una imagen de ejemplo. El segundo, consiste en entregar un dibujo de lo que se desea buscar y el sistema debe encontrar las imágenes que coinciden con el dibujo. Finalmente, el último enfoque es la principal motivación de este trabajo y consiste en encontrar imágenes a partir de una descripción de texto.

3.1.1. Query by Example

En un sistema CBIR con Query by Example (QBE por su sigla en inglés) la consulta es una imagen de muestra y el sistema debe retornar imágenes similares a la consulta. Un ejemplo de esto está en la figura 3.1, donde la imagen de consulta es un león acostado, y el sistema encontró otros leones acostados en la base de datos. Para lograr esto, el sistema compara los atributos de las imágenes de la base de datos con los atributos de la imagen consultada y retorna las imágenes cuyos atributos sean más parecidos a los de la imagen consultada.

Un caso de uso de este sistema es el llamado *Visual Search* usado en diferentes retails donde los clientes pueden subir una foto de un producto y la página retorna todos los productos que visualmente son parecidos.

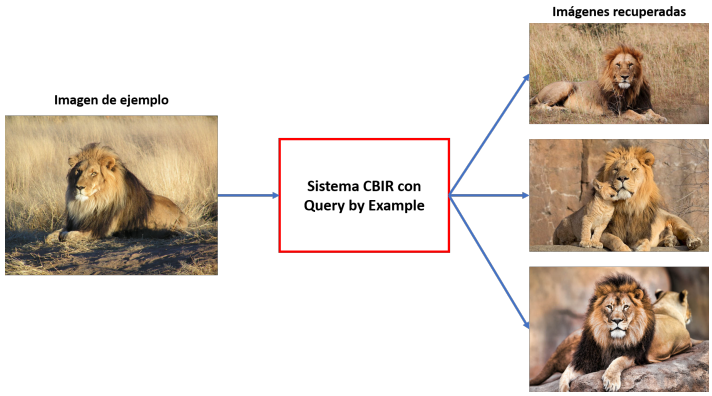


Figura 3.1: Sistema CBIR con Query by Example

3.1.2. Query by Sketch

En un sistema CBIR con Query by Sketch (QBS por su sigla en inglés) la consulta es un dibujo (sketch) y el sistema debe retornar las imágenes que coincidan con el dibujo. En la figura 3.2 se tiene un ejemplo de esta situación, donde la consulta es un dibujo de un avión y el sistema retorna imágenes de aviones reales.

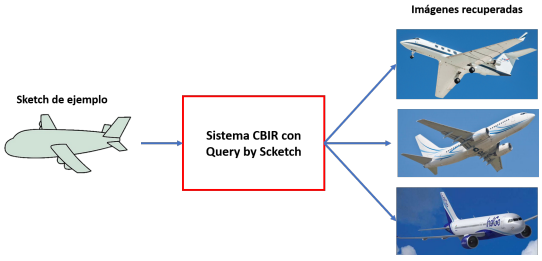


Figura 3.2: Sistema CBIR con Query by Sketch

A diferencia del caso anterior, este problema tiene un grado de dificultad más alto ya que la extracción de características debe ser tal que la información que se extrae de un dibujo

(que puede ser de buena o mala calidad) sea comparable con la información que se extrae de imágenes reales.

Un caso de uso de este sistema se puede ver en la tienda *Pepe Ganga* (*www.pepeganga.com*) donde se puede hacer un dibujo y la página retorna los productos similares al sketch como se muestra en la figura 3.3.

Figura 3.3: Ejemplo de búsqueda por Sketch en Pepe Ganga

3.1.3. Query by Description

El tipo de sistema abordado en este trabajo son los sistemas CBIR con Query by Description (QBD por su sigla en inglés), cuya particularidad es que la consulta consiste en una descripción (texto) y el sistema debe encontrar las imágenes que coinciden con aquella descripción. En la figura 3.4 hay un ejemplo donde la consulta es *Horses on the beach* y el sistema retorna las imágenes coinciden con la descripción.

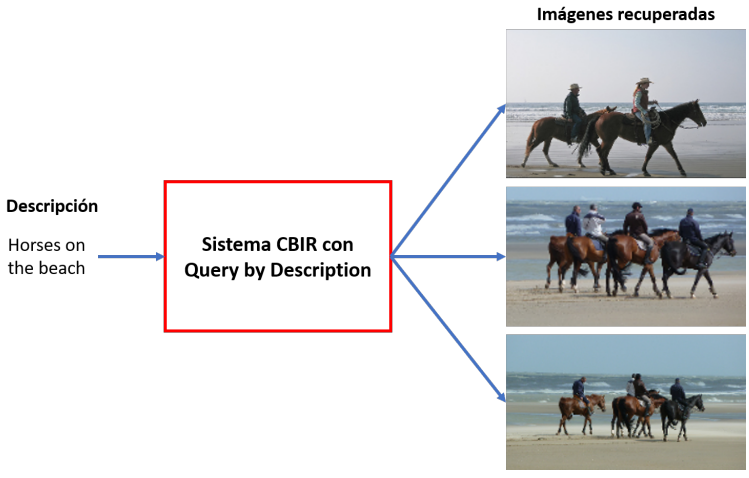


Figura 3.4: Sistema CBIR con Query by Description

A diferencia de los dos casos anteriores, el dominio de la consulta (texto) es diferente al dominio de la base de datos (imágenes). Esta es la principal dificultad de este tipo de sistemas

ya que las características que se extraen del texto no son directamente comparables con las características que se extraen de las imágenes.

Un caso de uso de este sistema es la búsqueda en imágenes propias, como las fotos de las vacaciones, viajes, etc. Esto está siendo fuertemente explotado por *Google Photos* que permite buscar en fotos propias diferentes objetos, acciones, entre otros, como se puede ver en el ejemplo de la figura 3.5.

Figura 3.5: Ejemplo de QBD en *Google Photos*

3.2. Estado del Arte

A continuación, se presentan distintos trabajos que abordan el problema de Image Retrieval donde las principales diferencias radican en cómo se calculan los descriptores de las imágenes, del texto y en cómo se modela el espacio que asocia estas variables.

Uno de los descriptores de imágenes más populares es el descriptor local SIFT debido a sus propiedades de invarianza al escalamiento, rotación, entre otras. Esta es la principal motivación de Bakar et al. [3] y Almeida et al. [2] quienes utilizan este descriptor para construir un sistema CBIR de *query by example*. Estos trabajos son tomados como referencia para implementar un CBIR con *query by description* usando el descriptor local SIFT.

Múltiples autores como Gong et al. [14], Klein et al. [26], Gong et al. [15], Hardoon et al. [16] y Hwang and Grauman [19] se basan en el método Canonical Correlation Analysis (CCA). Este método consiste en contruir un espacio conjunto que maximiza la correlación entre la representación vectorial de las imágenes y del texto asociado. Las principales diferencias en estos trabajos radican en pequeñas variaciones a este método, así como su eficiencia computacional, además de leves cambios en la forma de representar la información, como por ejemplo usar Fisher Vectors sobre los descriptores.

Por otra parte, Johnson et al. [22] propone un método basado en grafos para capturar la información de los objetos, sus atributos y relaciones espaciales dentro de una imagen para crear una representación semántica de esta. La principal desventaja de este método radica

en la constitución del conjunto de entrenamiento, el cual debe contener las imágenes y sus respectivos grafos, algo que según este trabajo no existe, y por lo tanto crean un conjunto de datos manualmente de solo $5,000$ imágenes. Y por otra parte, en una aplicación real las consultas son en lenguaje natural, por lo que se hace indispensable un modelo que transforme la frase en lenguaje natural a una estructura de grafos.

Un método novedoso es presentado por Quinn et al. [37] y Karpathy et al. [24]. Al igual que en el caso anterior, este trabajo busca capturar los objetos, atributos y relaciones existentes en las imágenes y sus descripciones, pero en vez de usar grafos usan un detector de objetos (RCNN) para fragmentar las imágenes y una técnica presentada en Huang et al. [18] para capturar la información del texto. Estos trabajos indican que reconocer objetos dentro de una imagen mejora los resultados, sin embargo, usar las posiciones relativas de los objetos dentro de una imagen fueron probadas en un dataset limitado, por lo que no es posible generalizar.

Un nuevo enfoque totalmente distinto es presentado por Dong et al. [12] (2016) y Dong et al. [13] (2018), donde no confían en que descriptores locales sean capaces de capturar la información semántica de las imágenes. En cambio, debido al éxito de Deep Learning en tareas de clasificación de imágenes es que deciden usar redes pre entrenadas para hacer la extracción de características de las imágenes. Luego, usan una red neuronal cuyo objetivo es aproximar dicho vector de características a partir de la representación vectorial del texto asociado. De esta forma, este trabajo dice ser el primero en resolver el problema de Image Retrieval directamente en el espacio de características visuales y además son capaces de generalizar de imágenes a vídeos. Este trabajo está fuertemente inspirado en esta investigación.

Capítulo 4

Descripción e Implementación del Sistema

4.1. Descripción general del Sistema

Este trabajo está basado principalmente en la investigación de Dong et al. [13] quien propone usar una red neuronal para transformar el espacio de los descriptores de texto al espacio de los descriptores de imágenes. Esta transformación está ilustrada en la figura 4.1, donde el espacio de las imágenes está representado por la superficie azul, el espacio del texto está representado por la superficie blanca y la transformación del espacio del texto al espacio de las imágenes está representado por las líneas que unen ambas superficies. En el caso ideal, las descripciones que hacen match con una imagen son mapeadas a una zona cercana a la imagen.

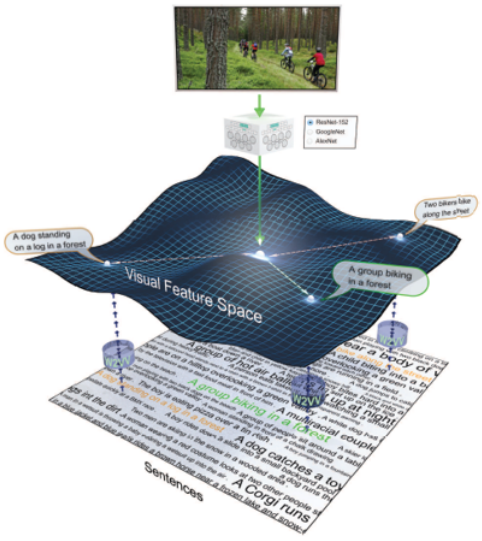


Figura 4.1: Representación gráfica del espacio compartido entre texto e imágenes (Dong et al. [13])

Para lograr esto, se debe crear el espacio de los descriptores de texto e imágenes por separado, y luego, entrenar una red neuronal que transforme el dominio del texto al dominio de las imágenes. Un ejemplo de ésto se puede ver en la figura 4.2, donde la entrada de la red neuronal es la representación vectorial de una descripción, en este caso *Horses on the beach* y la salida de la red neuronal debe ser la representación vectorial de la imagen que coincide con dicha descripción.

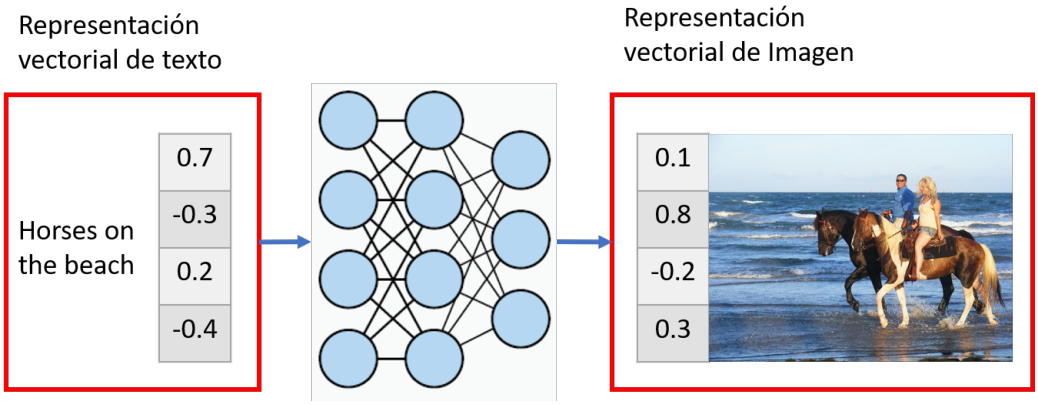


Figura 4.2: Representación gráfica del modelo genérico

Para encontrar las imágenes que coinciden con una descripción de texto, se debe entregar la representación vectorial de la descripción (T) a la red neuronal ya entrenada, la cual entregara su equivalente en el dominio de las imágenes (T'). Luego, se hace una búsqueda de los vecinos más cercanos a T' que son las imágenes cuyo significado semántico es cercano al significado semántico de la descripción. Todo este proceso está ilustrado en la figura 4.3, donde primero se calcula el descriptor de la palabra *Surf*, luego la red neuronal ya entrenada calcula su homólogo en el espacio de imágenes, y finalmente, se buscan los vecinos más cercanos que en este caso resultan ser imágenes de personas practicando surf.

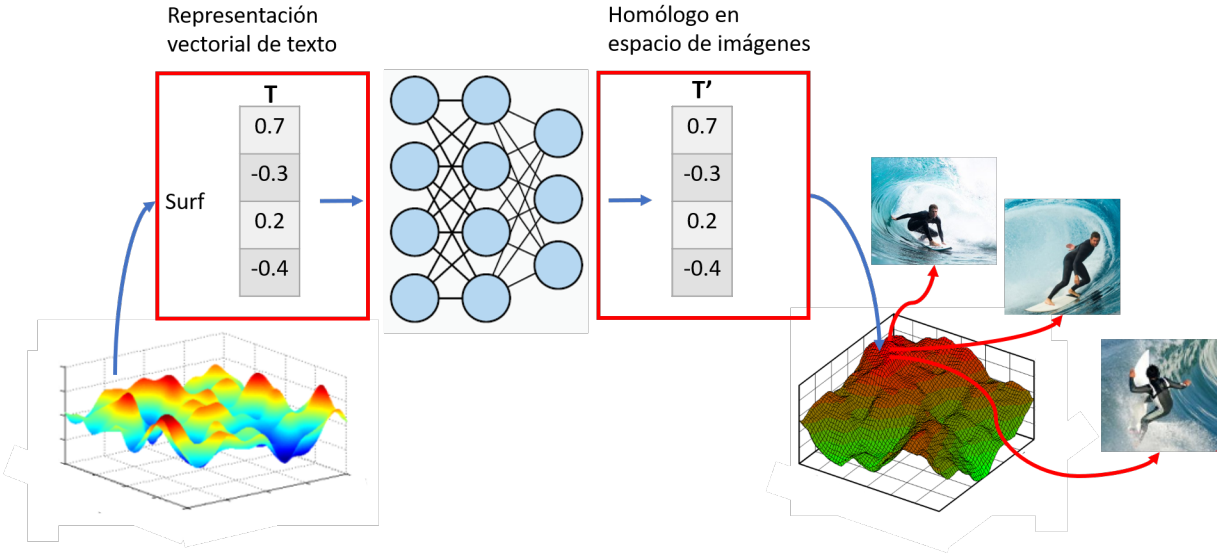


Figura 4.3: Búsqueda de imágenes

4.2. Implementación

4.2.1. Dataset

El conjunto de datos usado para entrenar la red neuronal es el dataset MSCOCO (Lin et al. [28]). Este dataset tiene 82,783 imágenes para entrenamiento y validación y 40,504 para el conjunto de prueba. Las imágenes contienen escenas cotidianas, animales, objetos, comida, deportes y lugares.

Además, cada una de ellas contiene metadata asociada, como bounding boxes, clases de los objetos, pose de las personas y 5 descripciones (en inglés) de texto creadas por personas distintas. De esta forma, al ser 5 descripciones por imagen se tiene un total de $411,365$ ejemplos para entrenamiento y validación. Durante el entrenamiento se usó un $10\,\%$ de los datos como conjunto de validación. En la figura 4.4 se puede ver un ejemplo de una imagen que pertenece al dataset MSCOCO y las descripciones de texto que están definidas para ella.

A man is skate boarding down a path and a dog is running by his side.
A man on a skateboard with a dog outside.
A person riding a skate board with a dog following beside.
This man is riding a skateboard behind a dog.
A man walking his dog on a quiet country road.

Figura 4.4: Ejemplo de dataset MSCOCO

4.2.2. Descriptores de Texto

Antes de poder calcular los descriptores de texto, el corpus fue pre-procesado. El pre-procesamiento consistió en transformar todas las letras a minúscula. Luego se removieron las stopwords (Ver sección 2.9). Posteriormente, se hizo un stemming con Snowball Stemmer (Ver sección 2.9). Luego se usaron los siguientes descriptores para transformar el texto procesado a vectores:

1. **TF-IDF** (Ver sección 2.10): Entrenado con todo el conjunto de entrenamiento usando unigramas y bigramas.
2. **R-TF-IDF** (Ver sección 2.10): Entrenado solo con aquellos unigramas y bigramas que aparecen al menos 3 veces en el conjunto de entrenamiento.

3. **Word2vec** (Ver sección 2.12): Se uso un modelo pre-entrenado con el corpus de Wikipedia en inglés.

Para entrenar un modelo en español fue necesario traducir el conjunto de entrenamiento. Para esto, se dividió el corpus en 10 particiones. Luego se usó la función *Translate* de *Google Spread Sheets* para traducir cada una de las particiones en paralelo.

Luego, al igual que en inglés se removieron las stopwords y se hizo stemming con Snowball Stemmer en su versión en español. Finalmente, los descriptores TF-IDF y R-TF-IDF fueron entrenados de la misma forma, pero el modelo de word2vec usado es un modelo pre-entrenado con el corpus de Wikipedia en español.

4.2.3. Descriptores de Imágenes

En el caso de las imágenes, estas fueron reducidas a un tamaño de 224×224 a las cuales se les aplicó los siguientes descriptores:

1. **VLAD** (Ver sección 2.7): Se calcularon los descriptores SIFT para cada imagen generando en total $28,223,652$ vectores. De estos vectores se seleccionaron $2,490,000$ de forma aleatoria a los cuales se les aplicó K-means para establecer la posición de 60 centroides. Con la posición de los centroides establecida, se calculó el descriptor VLAD de 7,680 dimensiones para cada imagen. Luego cada descriptor VLAD se normalizó con norma L2 como lo recomienda Jegou et al. [21]. Finalmente, para reducir la dimensionalidad del descriptor a 128 dimensiones se usó PCA (2.8). De aquí en adelante cada vez que se mencione al descriptor VLAD se hace referencia a este descriptor de 128 dimensiones.
2. **Deep Features** (Ver sección 2.5): Para calcular Deep Features se resto a cada una de las imágenes una constante que representa el valor promedio de las imágenes de ImageNet. Luego, se uso la capa 21 (de acuerdo a la tabla 2.1) de la red VGG16 pre-entrenada en ImageNet para extraer Deep Features. De esta forma, cada imagen es representada por un vector de 4,096 dimensiones. Para reducir la dimensionalidad de este descriptor a 128 dimensiones se usó PCA. De aquí en adelante cada vez que se mencione al descriptor VGG16 se hace referencia a este descriptor de 128 dimensiones.

4.2.4. Embedding

La arquitectura de la red neuronal que se usó es una red MLP con dos capas ocultas. La primera capa tiene 150 unidades, la segunda tiene 130 unidades y finalmente la capa de salida tiene 128 unidades. La función de activación de las capas ocultas es ReLU, mientras que la capa de salida no tiene función de activación ya que se trata de un problema de regresión. La función de costo usada fue RMSE y los pesos de la red fueron entrenados con el optimizador ADAM (Kingma and Ba [25]). Finalmente, el criterio para detener el entrenamiento fue un aumento consecutivo en el costo de validación durante 3 épocas.

Por último, para realizar las búsquedas de manera eficiente se aplicó el índice K-Means Tree con 40 centroides al descriptor VLAD y VGG16. Este índice se creó con la distancia coseno que es una de las más usadas para hacer búsqueda de imágenes.

4.2.5. Software

El cálculo de los descriptores de texto TF-IDF y R-TF-IDF se calcularon usando la librería **NLTK** (Bird and Loper [4]). Los modelos pre-entrenados en inglés y español de word2vec se usaron con la librería **FastText** (Mikolov et al. [31]). El cálculo de los descriptores VLAD se hizo con **OpenCV** (Bradski and Kaehler [5]). La arquitectura de la red, el entrenamiento y los descriptores VGG16 se calcularon usando **Keras** (Chollet et al. [9]) con **TensorFlow** (Abadi et al. [1]). El indexamiento y las búsquedas se hacen con el módulo **BallTree** de **SKlearn** [Pedregosa et al. [34]]. Por último, el entrenamiento de los modelos se llevó a cabo en la plataforma **CoLab** de Google la cual provee una GPU Tesla K80.

4.3. Diseño de Experimentos

Los experimentos se dividieron en 3 grupos de acuerdo a los descriptores de texto y descriptores de imágenes utilizados. A continuación se detalla en que consiste cada uno de ellos.

1. **Texto Inglés - VLAD**: El primer grupo de experimentos consiste en usar los descriptores TF-IDF, R-TF-IDF y word2vec para el corpus en inglés, y el descriptor VLAD para las imágenes. Además, se aplicó 3 niveles de Dropout: 0.2; 0.5 y 0.8, de esta forma, se entrenaron 9 modelos diferentes. Esto está resumido en la tabla 4.1.

Grupo	Descriptor de Texto	Descriptor de Imagen	Dropout
Texto Inglés - VLAD	TF-IDF (inglés)	VLAD	0.2 - 0.5 - 0.8
	R-TF-IDF (inglés)	VLAD	0.2 - 0.5 - 0.8
	word2vec (inglés)	VLAD	0.2 - 0.5 - 0.8

Tabla 4.1: Primer grupo de experimentos

2. **Texto Inglés - VGG16**: El segundo grupo de experimentos consiste en usar los descriptores TF-IDF, R-TF-IDF y word2vec para el corpus en inglés, y el descriptor VGG16 para las imágenes. Además, se aplicó 3 niveles de Dropout: 0.2; 0.5 y 0.8, de esta manera, se entrenaron 9 modelos diferentes. Esto está resumido en la tabla 4.2.

Grupo	Descriptor de Texto	Descriptor de Imagen	Dropout
Texto Inglés - VGG16	TF-IDF (inglés)	VGG16	0.2 - 0.5 - 0.8
	R-TF-IDF (inglés)	VGG16	0.2 - 0.5 - 0.8
	word2vec (inglés)	VGG16	0.2 - 0.5 - 0.8

Tabla 4.2: Segundo grupo de experimentos

3. **Texto Español - VGG16**: El tercer grupo de experimentos consiste en usar los descriptores TF-IDF, R-TF-IDF y word2vec para el corpus en español, y el descriptor VGG16 para las imágenes. En está ocasión solo se aplicó un nivel de Dropout 0.2, generado 3 modelos diferentes. Esto está resumido en la tabla 4.3.

Grupo	Descriptor de Texto	Descriptor de Imagen	Dropout
Texto Español - VGG16	TF-IDF (español)	VGG16	0.2
	R-TF-IDF (español)	VGG16	0.2
	word2vec (español)	VGG16	0.2

Tabla 4.3: Tercer grupo de experimentos

4.3.1. Evaluación

La evaluación de los modelos se hizo bajo tres métricas:

1. Tiempo de entrenamiento
2. Valor mínimo obtenido por la función de costo en el conjunto de validación durante la etapa de entrenamiento.
3. Average Precision

El tiempo de entrenamiento consiste en calcular el promedio del tiempo necesario para entrenar una época. Luego, la **función de costo en el conjunto de validación** consiste en establecer el valor de la función de costo en el conjunto de validación al finalizar el entrenamiento.

La métrica **Average Precision** consiste en calcular los valores AP@k con k = 1 y 10. Para esto se seleccionó de forma aleatoria 100 descripciones del conjunto de prueba y se validó si los resultados eran correctos o no. Ésto se hizo de dos formas: coincidencia exacta y coincidencia parcial. La coincidencia exacta significa que la imagen debe coincidir exactamente con la descripción para ser considerada un resultado correcto. Mientras que la coincidencia parcial es una evaluación menos estricta donde las imágenes que no hacen un match perfecto con la descripción, pero son resultados relevantes también son consideradas correctas.

En la figura 4.5 hay un ejemplo de la diferencia entre la coincidencia exacta y parcial. Como se puede ver la consulta es *Hay un hombre que golpea una pelota de tenis*, entonces solo la primera y la última foto representan esta situación perfectamente, sin embargo, las otras 3 fotos tienen mucha relación con la descripción y son consideradas correctas para la coincidencia parcial.

Hay que destacar que **no** se evaluaron todos los modelos entrenados (que en total son 21), sino que para cada tupla (descriptor de texto, descriptor de imagen) se seleccionó el nivel de Dropout con menor valor en la función de costo. De esta forma se evaluaron 3 modelos para cada versión.

Hay un hombre que golpea una pelota de tenis.

Coincidencia Exacta	Correcto	Incorrecto	Incorrecto	Incorrecto	Correcto
Coincidencia Parcial	Correcto	Correcto	Correcto	Correcto	Correcto

Figura 4.5: Ejemplo de evaluación exacta y parcial

Capítulo 5

Experimentos y Análisis de Resultados

5.1. Texto Inglés - VLAD

5.1.1. Reducción de dimensionalidad

En la figura 5.1 se puede ver que las 128 componentes principales del descriptor VLAD explican cerca del 80 % de la varianza de los datos. Esto quiere decir que hay 7,552 dimensiones que están correlacionadas entre sí, y contienen muy poca información de los datos. Esto puede tener origen en que se usaron muchos centroides para construir el descriptor o que estos no representan la distribución de los datos debido a que no se usaron todos los vectores SIFT para el cálculo de los centroides.

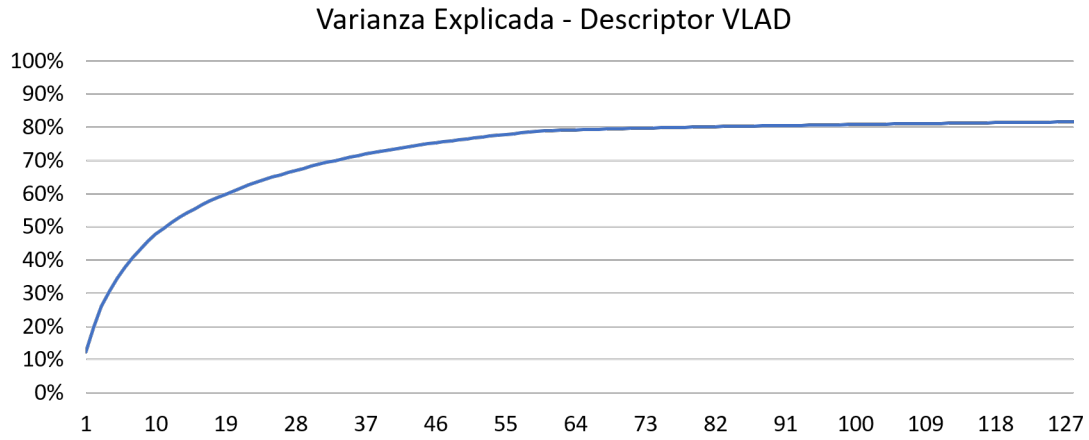


Figura 5.1: Varianza explicada por las componentes principales del descriptor VLAD

5.1.2. Caracterización de Modelos

En la tabla 5.1 se tiene la dimensión de cada uno de los descriptores de texto y la cantidad de parámetros de la red neuronal asociados a dicho descriptor. El descriptor TF-IDF es el que tiene mayor número de dimensiones ya que toma en cuenta todo el vocabulario del corpus que consiste en 369,537 palabras (después de remover las stopwords y aplicar stemming).

Por su parte, el descriptor R-TF-IDF el cual solo considera las palabras que aparecen al menos 3 veces en el corpus, tiene 91,155 dimensiones. Este número es mucho menor que el obtenido con TF-IDF y significa que hay cerca de 280,000 palabras que aparecen menos de 3 veces en el corpus de entrenamiento. Por otro lado, así como el número de dimensiones disminuye, la cantidad de parámetros del modelo disminuye considerablemente, pasando de aproximadamente 55 millones a cerca de 13 millones.

Por último, el descriptor word2vec es de largo fijo con 300 dimensiones y genera un modelo con alrededor de 81 mil parámetros, una cantidad drásticamente menor comparado con los otros dos descriptores.

Variante	Dimensión	Número de Parámetros
TF IDF	369,537	55,467,098
R - TF IDF	91,155	13,709,798
Word2Vec	300	81,548

Tabla 5.1: Dimensión de los descriptores de texto y la cantidad de parámetros de los modelos generados para el primer grupo de experimentos.

5.1.3. Entrenamiento

En la figura 5.2 se tiene la evolución de la función de costo en el conjunto de entrenamiento y validación para el modelo entrenado con la tupla (TF-IDF, VLAD) para distintos valores de Dropout. En ella se puede ver que a medida que se aumenta el Dropout, el gap entre el costo del conjunto de entrenamiento y el de validación se hace cada vez menor, llegando al extremo que al usar Dropout 0.8 las curvas son casi idénticas. Además, se tiene que al aumentar el nivel del Dropout el costo de ambos conjuntos aumenta, empeorando el desempeño de la red en términos de RMSE.

Este efecto del Dropout ocurrió en todos los experimentos, por lo que a partir de este punto, solo se reportan los resultados con un nivel de Dropout 0.2. Los valores de Dropout menores a 0.2 no se evaluaron debido a que el tiempo de entrenamiento aumenta significativamente. Los resultados con Dropout 0.5 y 0.8 de todos los experimentos se pueden ver en apéndice A de Anexos.

Por otra parte, se puede ver que el costo en el conjunto de entrenamiento y validación esta acotado entre 0.0020 y 0.0025, esto quiere decir que la evolución del entrenamiento de esta tupla está muy limitada. Finalmente, se observa que el número de épocas necesarias para entrenar la red oscila entre 4 y 6.

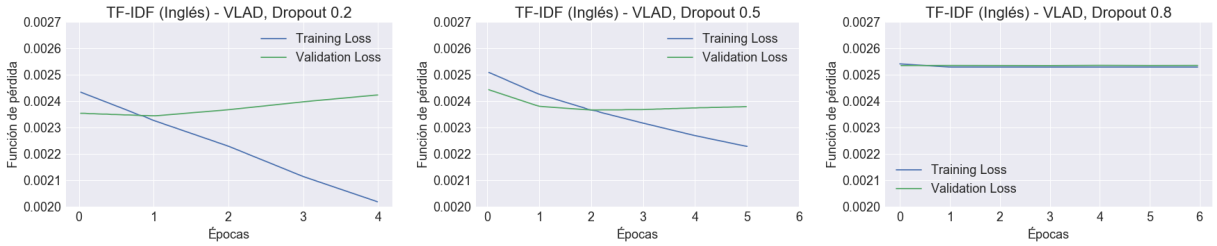


Figura 5.2: Entrenamiento de red neuronal usando TF-IDF, VLAD y distintos valores de Dropout

En la figura 5.3 se puede ver que el entrenamiento de la tupla (R-TF-IDF, VLAD) es muy similar al anterior, pero con una menor diferencia entre el costo en el conjunto de validación y entrenamiento. Esto se debe a que el descriptor R-TF-IDF solo considera las palabras que aparecen al menos 3 veces en el conjunto de entrenamiento, evitando que pequeños detalles en el texto modifiquen la respuesta de la red.

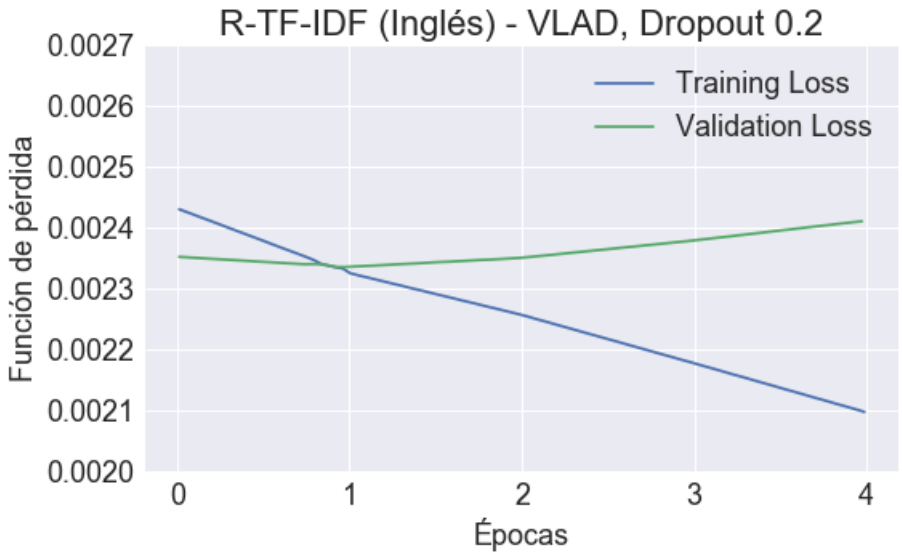


Figura 5.3: Entrenamiento de red neuronal usando R-TF-IDF y VLAD

Por último, en la figura 5.4 está la evolución del entrenamiento con la tupla (word2vec, VLAD). En este caso, las curvas son diferentes a las observadas previamente. Esto tiene sentido ya que la forma de construir el descriptor word2vec es diferente a la del descriptor TF-IDF. También se puede ver que el costo en el conjunto de validación y entrenamiento siguen una curva mucho más plana y la cantidad de épocas necesarias para el entrenamiento es el doble que las obtenidas en los casos anteriores.

En la tabla 5.2 hay un resumen con los resultados finales del entrenamiento (la tabla con los resultados con todos los niveles de Dropout está en el apéndice B de Anexos), donde se puede ver que todos los modelos convergieron a un costo en el conjunto de validación muy similar. Sin embargo, las grandes diferencias están en el tiempo que necesita cada modelo para entrenar una época.

El descriptor TF-IDF necesita en promedio 920 [s] por época, R-TF-IDF 230 [s] y word2vec solo necesita 9 [s]. Esto es completamente esperable ya que el descriptor TF-IDF tiene más de 55 millones de parámetros y word2vec alrededor de 81 mil parámetros. La tabla con todos los niveles de Dropout está en el apéndice B de Anexos.

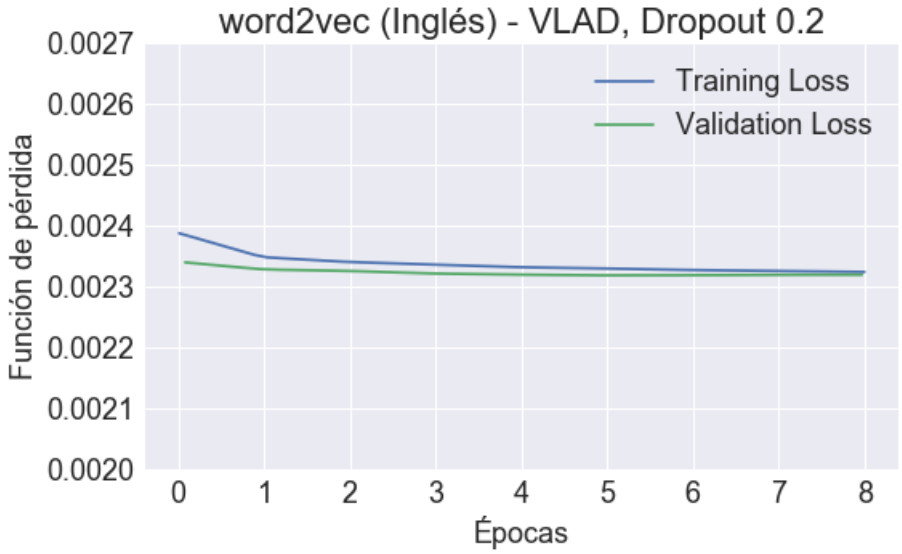


Figura 5.4: Entrenamiento de red neuronal word2vec y VGG16

	Dropout 0.2	
	Seg/Ep	C. Val
TF IDF	920 [s]	0.0024
R-TF IDF	230[s]	0.0024
word2vect	**9 [s]**	**0.0023**

Tabla 5.2: Resultados del entrenamiento del sistema Texto Inglés - VLAD
Seg/Ep es el promedio del tiempo que demora en entrenar con una época y C. Val es el valor de la función de pérdida en el conjunto de validación.

5.1.4. Evaluación AP@k

En el gráfico de la figura 5.5 están las métricas AP@1 y AP@10 con sus respectivos intervalos de confianza de 95 % para las coincidencias exactas. En él se puede ver que en general los resultados están entre 2 % y 4 %, sin diferencias significativas entre los descriptores de texto.

En la figura 5.6 se puede ver que los resultados de la coincidencia parcial son casi 4 veces más altos que los de coincidencia exacta llegando a un máximo de 15 %. Al igual que en el caso anterior, no hay diferencias significativas entre los descriptores de texto. Por otra parte, se puede ver que los resultados decaen un 3 % entre AP@1 y AP@10, indicando que las primeras imágenes retornadas tienen más relación con la consulta que las últimas.

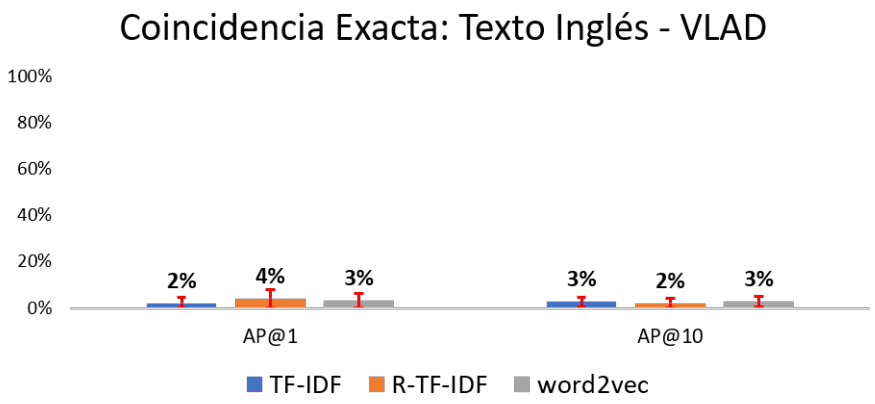


Figura 5.5: Resultados de Average Precision bajo una coincidencia exacta

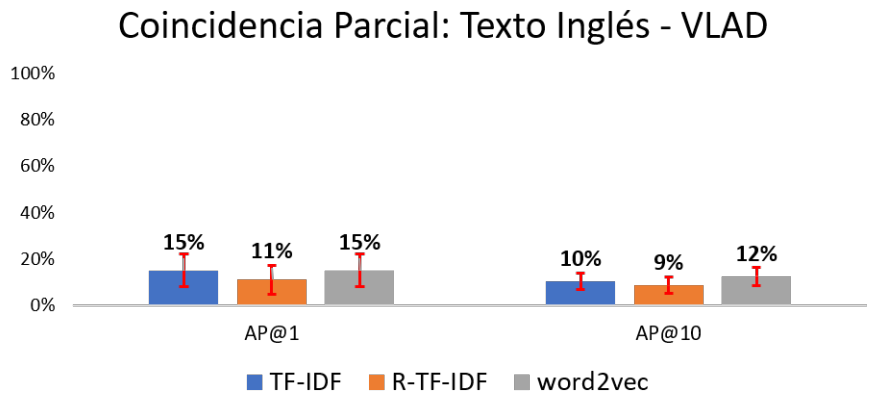


Figura 5.6: Resultados de Average Precision bajo una coincidencia parcial

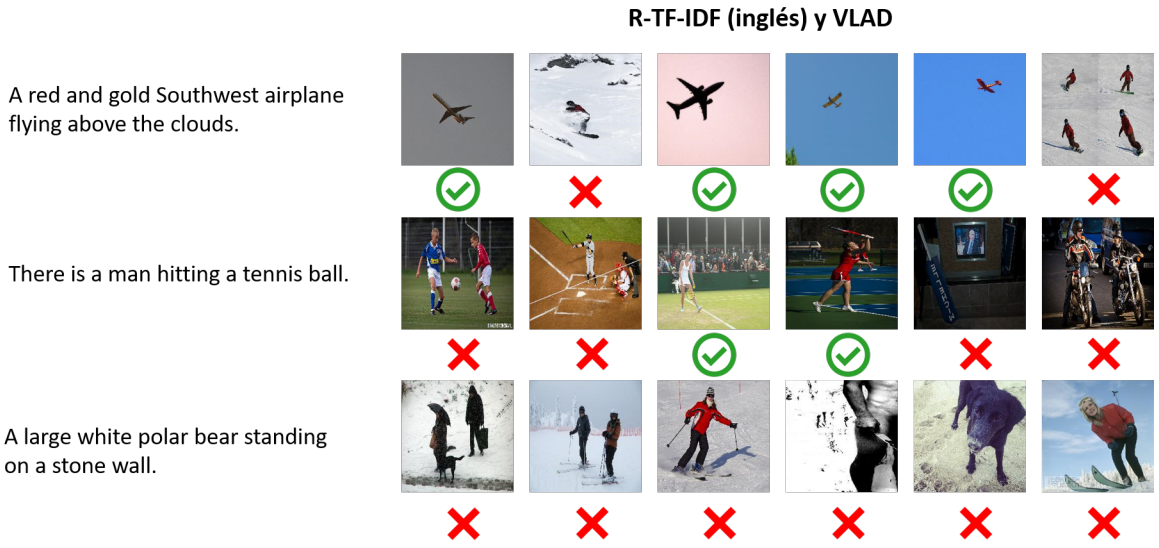


Figura 5.7: Ejemplos de las imágenes retornadas por la red entrenada con R-TF-IDF (inglés) y VLAD

En la figura 5.7 se pueden ver los resultados de 3 consultas. La primera, es sobre aviones y si bien esta versión encuentra algunos aviones, también confunde este concepto con personas practicando snowboard. Algo similar ocurre con la segunda consulta donde se confunde el tenis con fútbol y béisbol. En la última consulta, se desea buscar un oso polar pero el modelo no encuentra ningún oso polar, en cambio, retorna personas practicando snowboard, concepto que de alguna manera está relacionado con osos polares ya que ambos se encuentran comúnmente en un ambiente dominado por la nieve.

5.2. Texto Inglés - VGG16

5.2.1. Reducción de dimensionalidad

En la figura 5.8 se puede ver que las 128 componentes principales del descriptor VGG16 explican cerca del 50 % de la varianza de los datos. Al comparar esto con lo visto en la versión 1 (ver 5.1) se puede decir que el descriptor VGG16 perdió un 50 % de la información del descriptor original vs el 20 % que perdió el descriptor VLAD al aplicar PCA. Esto significa que la varianza del descriptor VGG16 está distribuida en más dimensiones que en el descriptor VLAD, sobre todo si se considera que el descriptor original de VGG16 tiene 4,096 dimensiones versus las 7,552 de VLAD.

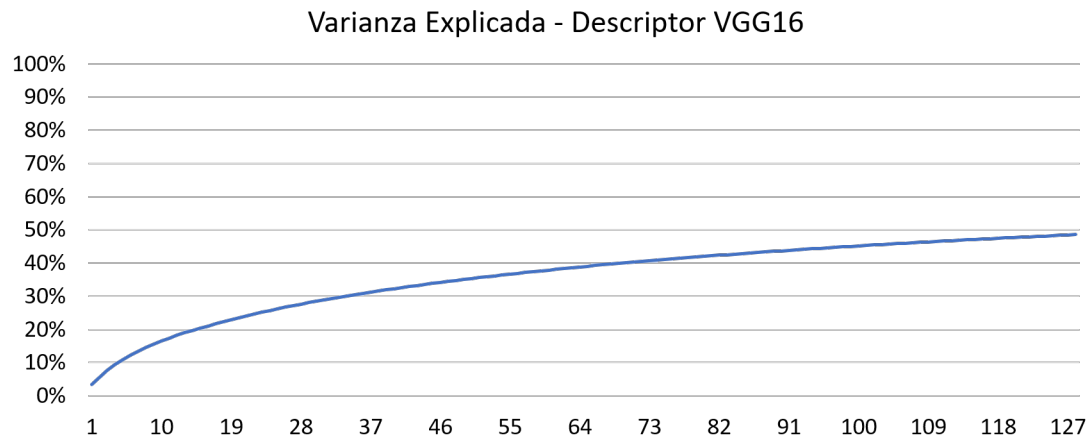


Figura 5.8: Varianza explicada por las componentes principales del descriptor VGG16

5.2.2. Caracterización de Modelos

En la tabla 5.3 se tiene la dimensión de cada uno de los descriptores de texto y la cantidad de parámetros de la red neuronal asociados a dicho descriptor. Como el corpus que se usa en esta versión es en inglés, los números son idénticos a lo observado en el grupo 1 (ver 5.1).

A modo de resumen el descriptor con mayor dimensionalidad es TF-IDF que tiene aproximadamente 370 mil dimensiones con un modelo de 55 millones de parámetros, mientras

que word2vec tiene 300 dimensiones y el modelo 81 mil parámetros. R-TF-IDF está en el punto medio y tiene un vocabulario mucho menor que TF-IDF solo al filtrar las palabras que aparecen menos de 3 veces en el corpus.

Variante	Dimensión	Número de Parámetros
TF IDF	369,537	55,467,098
R - TF IDF	91,155	13,709,798
Word2Vec	300	81,548

Tabla 5.3: Dimensión de los descriptores de texto y la cantidad de parámetros de los modelos generados para el segundo grupo de experimentos.

5.2.3. Entrenamiento

En la figura 5.9 se tiene la evolución de la función de costo en el conjunto de entrenamiento y validación para el modelo entrenado con la tupla (TF-IDF, VGG16) para Dropout 0.2. En ella se puede ver que la curva es mucho más dinámica que la observada en 5.2 ya que la función de costo decae de 200 a 140 y la cantidad de épocas de entrenamiento asciende a 8.

En la figura 5.10 se puede ver lo que sucede al entrenar con el descriptor R-TF-IDF. En términos generales, la curva es muy parecida al entrenamiento con TF-IDF con la diferencia de que el gap entre el costo de validación y entrenamiento es menor. La razón de esto es que el descriptor R-TF-IDF se calcula de la misma forma que el descriptor TF-IDF, por lo tanto, la evolución de la función de costo es similar y al mismo tiempo el riesgo de overfitting es menor ya que R-TF-IDF solo usa el vocabulario más relevante desechando más de 250,000 palabras.

En la figura 5.11 está la evolución del entrenamiento con la tupla (word2vec, VGG16). En este caso el comportamiento de las curvas es diferente a lo observado previamente. Primero se puede ver que el entrenamiento dura más épocas llegando a 35, pero también se puede ver que la función de perdida se estanca en la época 10, y desde ahí la curva se vuelve plana hasta terminar el entrenamiento.

En la tabla 5.4 hay un resumen con los resultados finales del entrenamiento (la tabla con los resultados con todos los niveles de Dropout está en el apéndice B de Anexos), donde se tiene que el modelo entrenado con TF-IDF es el que necesita más tiempo de entrenamiento por época, seguido por el descriptor R-TF-IDF y word2vec, siguiendo la misma línea que el grupo Texto Inglés - VLAD. Por otra parte, el descriptor word2vec es el que tiene menor costo en el conjunto de validación.

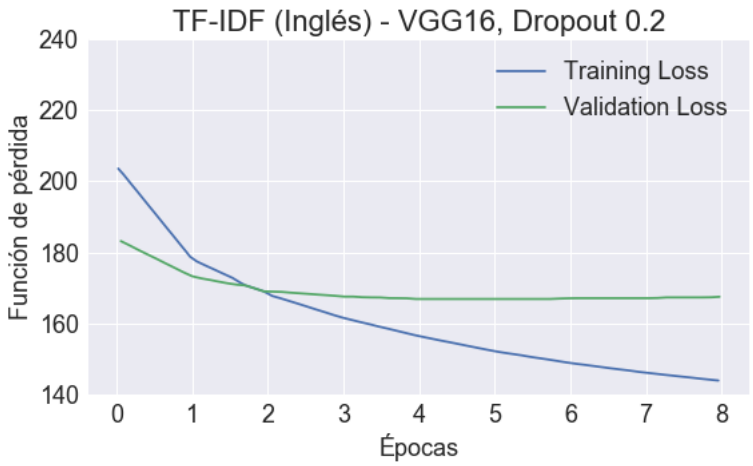


Figura 5.9: Entrenamiento de red neuronal usando TF-IDF y VGG16

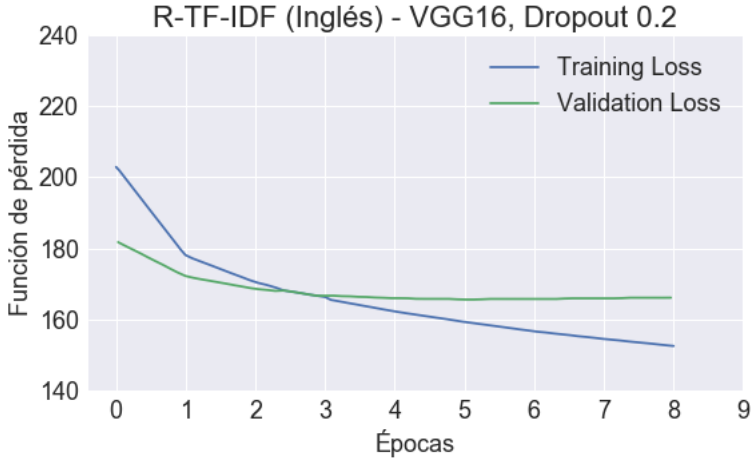


Figura 5.10: Entrenamiento de red neuronal usando R-TF-IDF y VGG16

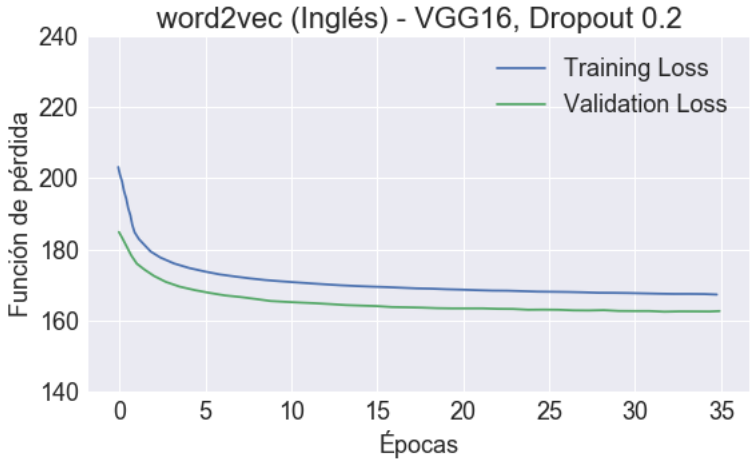


Figura 5.11: Entrenamiento de red neuronal usando word2vec y VGG16

	Dropout 0.2	
	Seg/Ep	C. Val
TF IDF	890 [s]	169.24
R-TF IDF	220 [s]	166.14
Word2vec	**9 [s]**	**162.61**

Tabla 5.4: Resultados del entrenamiento del sistema Texto Inglés - VGG16
Seg/Ep son los segundos que demora el modelo en entrenar una época y C. Val es el valor de la función de pérdida en el conjunto de validación.

5.2.4. Evaluación AP@k

En la figura 5.12 se tiene el desempeño de los modelos bajo una coincidencia exacta está cercano al 19 %, valor mucho más alto que el 3 % alcanzado por la tupla (Texto Inglés - VLAD). También se puede ver que el valor de AP@k disminuye al pasar de $k = 1$ a $k = 10$, lo que significa que nuevamente, las imágenes relevantes están concentradas en las primeras posiciones. Por último, los intervalos de confianza tienen traslape, lo que indica que no hay diferencias significativas entre los descriptores de texto.

En la figura 5.13 se puede ver el desempeño de los modelos bajo una coincidencia parcial, donde el descriptor R-TF-IDF saca una ventaja significativa para AP@1. Para AP@10 la hay translape entre los intervalos de confianza, por lo tanto, no se puede concluir que el descriptor R-TF-IDF es mejor que los demás en términos de AP@10.

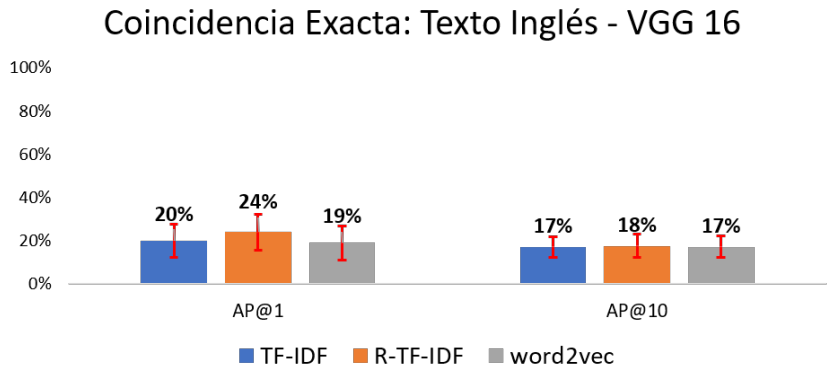


Figura 5.12: Resultados de Average Precision bajo una coincidencia exacta

A partir de aquí se desprenden dos aspectos clave: que el descriptor VGG16 tiene un desempeño cinco veces mejor que el descriptor VLAD. Y que un menor costo en el conjunto de validación (durante el entrenamiento) no garantiza los mejores resultados en la búsqueda de imágenes, ya que el modelo con menor costo en el conjunto de validación fue con el descriptor word2vec el cual no supero en ningún caso a los resultados alcanzados por R-TF-IDF en Average Precision.

En la figura 5.14 se pueden ver consultas que son representativas de la superioridad de esta versión frente a la primera. Respecto a la primera consulta se puede ver que el concepto de *avión volando* no es confundido con otras descripciones, aunque no todos los resultados

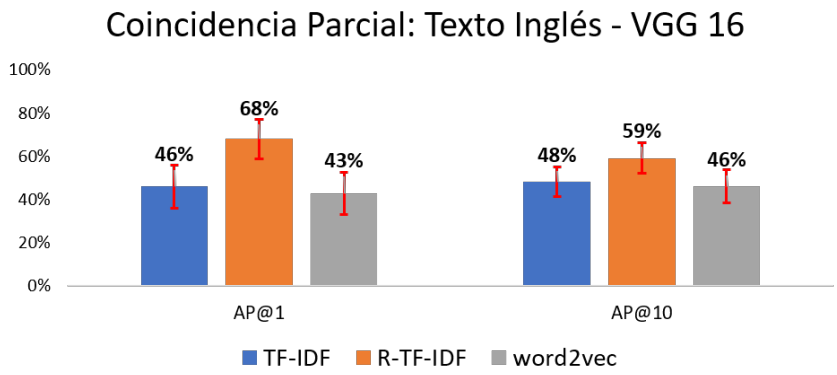


Figura 5.13: Resultados del retrieval bajo una coincidencia parcial

coinciden exactamente con la consulta ya que no son rojo con dorado, sin embargo, todos son resultados relevantes a la consulta realizada. Lo mismo ocurre con la segunda consulta, donde no todas las imágenes contienen la escena exacta de un hombre pegándole a una pelota de tenis, pero todas contienen una escena de un jugador de tenis jugando. Sin embargo, en la última consulta se puede ver que esta versión todavía necesita ser mejorada ya que no fue capaz de encontrar los osos polares contenidos en la base de datos.

Figura 5.14: Ejemplos de las imágenes retornadas por la red entrenada con R-TF-IDF (inglés) y VGG16

5.3. Texto Español - VGG16

5.3.1. Reducción de dimensionalidad

El descriptor de imagen usado en esta versión es el VGG16, por lo tanto, la reducción de dimensionalidad es la misma que la mostrada en la figura 5.8. A modo de recordatorio, las 128 componentes principales del descriptor VGG16 explican cerca del 50 %. En contraste con lo que ocurre la versión 1, esto indica que la información del descriptor VGG16 está distribuida en más dimensiones que en el descriptor VLAD.

5.3.2. Caracterización de Modelos

En la tabla 5.5 se tiene la dimensión de cada descriptor de texto y la cantidad de parámetros de la red neuronal asociados a su respectivo descriptor. Al igual que en las versiones anteriores es el descriptor TF-IDF el que tiene mayor dimensión seguido por R-TF-IDF y word2vec. Lo mismo pasa con la cantidad de parámetros de la red neuronal. Sin embargo, se puede ver que la dimensión del descriptor TF-IDF tiene casi 30,000 dimensiones más, lo que indica que el corpus en español tiene 30,000 palabras más que el corpus en inglés. Esto también se refleja en el descriptor R-TF-IDF el cual tiene cerca de 3,000 dimensiones más que en las versiones 1 y 2, lo que indica que hay 3,000 palabras más que aparecen más de 3 veces en el corpus en español que en inglés.

Variante	Dimensión	Número de Parámetros
TF IDF	395,479	59,358,398
R - TF IDF	94,412	14,198,348
Word2Vec	300	81,548

Tabla 5.5: Dimensión de los descriptores de texto y la cantidad de parámetros de los modelos generados para el tercer grupo de experimentos.

5.3.3. Entrenamiento

En la figura 5.15, 5.16 y 5.17 se tiene la evolución de la función de costo para los modelos entrenados con las tuplas (TF-IDF, VGG16); (R-TF-IDF, VGG16) y (word2vec, VGG16) donde todos los descriptores de texto están construidos en español. Las curvas de entrenamiento son muy similares a las encontradas en el grupo 2, donde el descriptor R-TF-IDF tiene menor gap entre el costo en el conjunto de entrenamiento y validación. El modelo entrenado con word2vec tiene un comportamiento diferente ya que su entrenamiento dura muchas más épocas pero se satura en la época número 15.

En la tabla 5.6 se tiene un resumen del entrenamiento de los 3 modelos. Para los descriptores TF-IDF y R-TF-IDF el entrenamiento de una época es mayor a los observados en el grupo 1 y 2 ya que la cantidad de parámetros de los modelos en español es mayor a los

entrenados en inglés, excepto por el descriptor word2vec que tiene un tamaño fijo. Una vez más, el descriptor que menor costo en el conjunto de validación tuvo es el word2vec, seguido por R-TF-IDF y finalmente TF-IDF. Al comparar el costo de validación con los obtenidos en el grupo 2 se puede ver que en todos los casos los modelos en español tienen un costo en el conjunto de validación mayor a su homólogo en inglés. Este aumento en el costo en el conjunto de validación puede tener origen en errores de traducción.

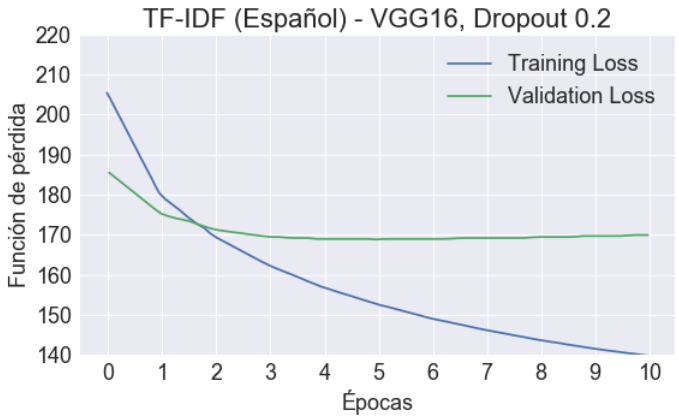


Figura 5.15: Entrenamiento de red neuronal usando TF-IDF, descriptor VGG16 y Dropout 0.2

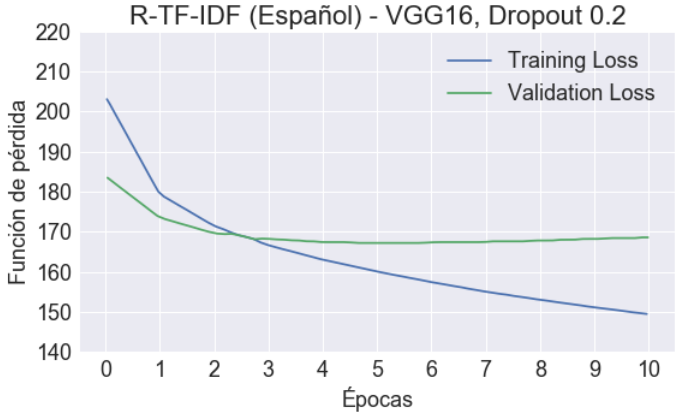


Figura 5.16: Entrenamiento de red neuronal usando R-TF-IDF, descriptor VGG16 y Dropout 0.2

	Dropout 0.2	
	Seg/Ep	C. Val
TF IDF	1000 [s]	169.63
R-TF IDF	260 [s]	168.263
word2vect	10 [s]	164.84

Tabla 5.6: Resultados del entrenamiento del sistema Texto Español - VGG16
Seg/Ep es el promedio del tiempo que demora en entrenar con una época y C. Val es el valor de la función de pérdida en el conjunto de validación.

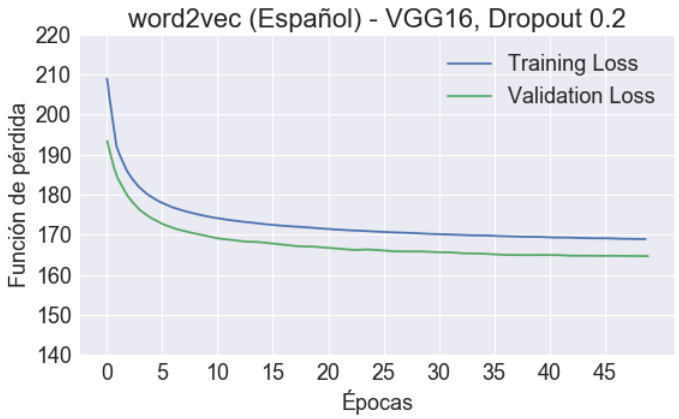


Figura 5.17: Entrenamiento de red neuronal usando word2vec, descriptor VGG16 y Dropout 0.2

5.3.4. Evaluación AP@k

En la figura 5.18 se puede ver los resultados de Average Precision bajo una coincidencia exacta. Aquí se puede ver una clara tendencia donde el descriptor TF-IDF es el que tiene mejores resultados alcanzando una efectividad de 33 %, seguido por R-TF-IDF con una efectividad de 27 % y word2vec con una efectividad de 19 % para AP@1 (para AP@10 la tendencia es similar), sin embargo, el traslape entre los intervalos de confianza no permite concluir que TF-IDF siempre será mejor que los otros descriptores.

En la figura 5.19 están los resultados de Average Precision bajo una coincidencia parcial. Estos resultados son casi tres veces mejores que los obtenidos por la coincidencia exacta, indicando que hay más resultados relevantes que solo aquellos que coinciden perfectamente con la descripción. Al igual que en el caso anterior, no se pueden establecer diferencias claras entre los descriptores de texto, debido al traslape en los intervalos de confianza.

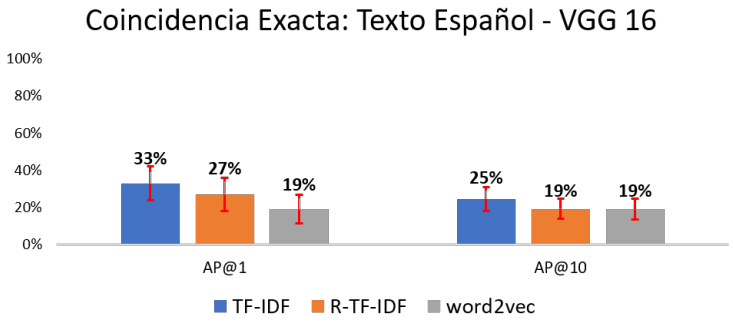


Figura 5.18: Resultados del retrieval bajo una coincidencia exacta

Con estos números los resultados en español fueron levemente peores que en inglés. Esto puede tener origen en varios factores, como por ejemplo, que las traducciones automáticas no sean de la mejor calidad, o errores en las descripciones en inglés conllevan a errores de traducción, etc.

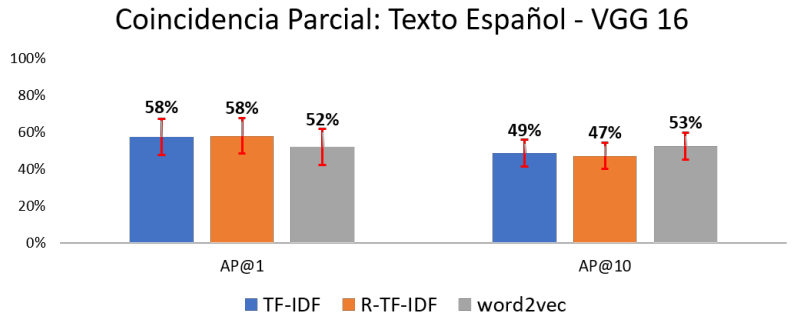


Figura 5.19: Resultados del retrieval bajo una coincidencia parcial

Sin embargo, hay casos donde el modelo en español encontró resultados relevantes cuando el modelo en inglés no lo hizo. Esto se puede ver en la figura 5.20 donde la tercera consulta que no funcionó en "Texto Inglés - VLAD"(5.7) ni en "Texto Inglés - VGG16"(5.14) funciona en "Texto Español - VGG16". Este resultado podría tener origen en la semántica del idioma ya que en el caso de inglés *bear* no significa solo *oso*, sino que también está asociado a otros términos como *aguantar*, *sostener*, etc.

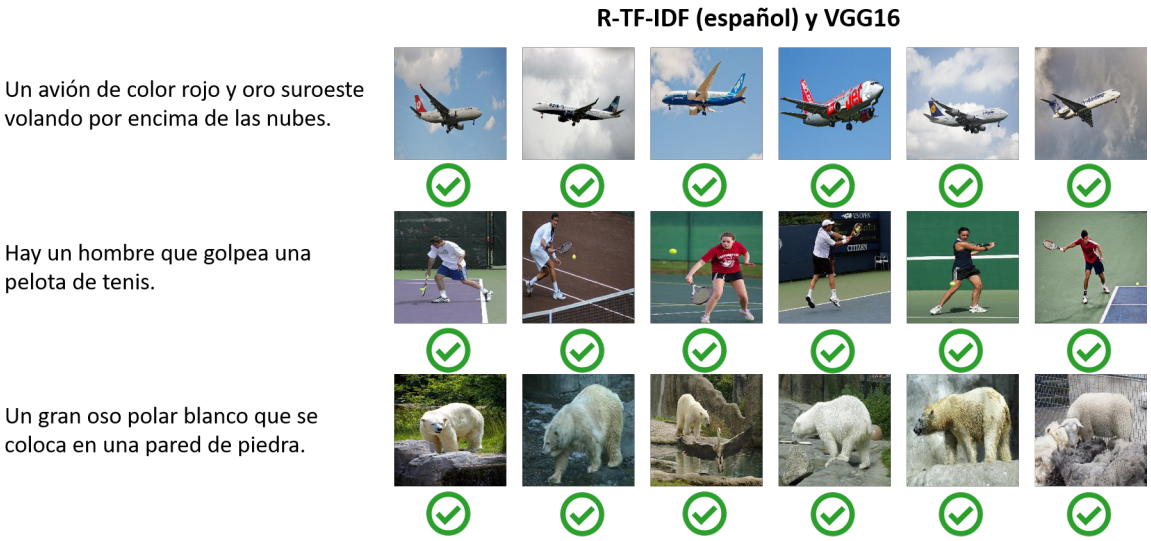


Figura 5.20: Ejemplos de las imágenes retornadas por la red entrenada con R-TF-IDF (español) y VGG16

Para finalizar, en la figura 5.14 también se puede ver que el concepto de *avión volando* no es confundido con otras descripciones, aunque no todos los resultados coinciden exactamente con la descripción ya que no son rojo con dorado, sin embargo, todos son resultados relevantes a la consulta realizada. Lo mismo ocurre con la segunda consulta, donde no todas las imágenes contienen la escena exacta de un hombre pegándole a una pelota de tenis, pero todas contienen una escena de un jugador jugando tenis.

5.3.5. DEMO: Imágenes Propias

Como parte del trabajo se decidió probar el mejor modelo de inglés y español en imágenes propias. Esto significa usar la versión 2 con Dropout 0.2 con la tupla (R-TF-IDF (inglés), VGG16) y la versión 3 con (R-TF-IDF (español), VGG16). En la figura 5.21 se pueden ver los resultados para 3 consultas en inglés y español. Los resultados son bastante buenos y responden al comportamiento visto durante la evaluación de los modelos.

También se tiene que en general los resultados para la consulta en inglés y español comparten varios resultados, pero en un orden diferente. Por otra parte, se puede notar como la estructura propia del idioma altera los resultados, por ejemplo *Snowman* es una sola palabra que tiene un solo significado, sin embargo para lograr el mismo significado semántico en español se necesitan tres palabras *Hombre de Nieve*, pero hay que considerar que la palabra *de* es una stopword y es eliminada de la descripción, por lo que el modelo en español solo recibe la descripción *Hombre nieve* lo que puede inducir errores y retornar resultados de personas en la nieve en vez de un hombre de nieve.

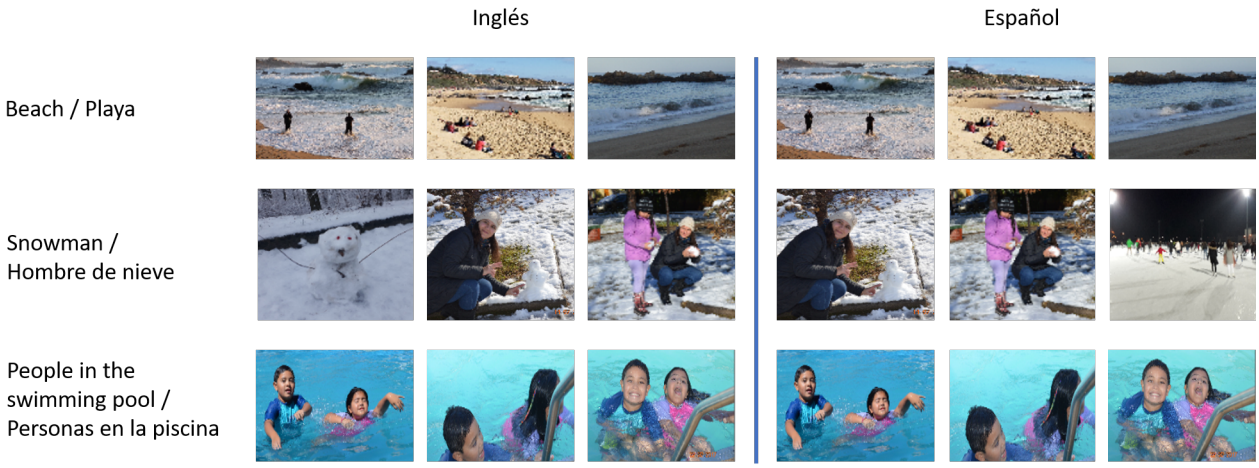


Figura 5.21: Ejemplo del búsqueda en imágenes propias

5.4. Problemas Encontrados y Posibles Soluciones

Durante el desarrollo de este trabajo, se encontraron diversos problemas. Por ejemplo, errores ortográficos en las descripciones de las imágenes o incluso errores gramaticales. Esto impacta directamente en la capacidad del modelo por aprender conceptos de manera precisa. Para solucionar esto se debe hacer un esfuerzo para supervisar y asegurar la calidad de las descripciones que crean las personas.

Otro problema que surgió fue durante la traducción masiva de texto. Para hacer esto no hay un servicio disponible de forma gratuita. Una de las alternativas es usar *Translation API* de Google, pero el costo estimado para traducir todo el dataset es de 500USD, monto que esta fuera del alcance en el contexto de este trabajo. La solución fue usar *Google Spread Sheets* ya que contiene una función de traducción, pero de menor calidad que la API pagada.

Esto es suficientemente bueno, sin embargo, si se desea mejorar la calidad de los modelos en español, lo mejor es invertir en crear dataset en español desde el comienzo.

Por último, se tiene que el dataset MSCOCO no tiene la misma cantidad de ejemplos para cada situación. Esto provoca que el modelo aprenda mejor aquellos conceptos que se repiten en el dataset. Una solución para esto no es fácil ya que requeriría tener todas las posibles situaciones que puede capturar una imagen en la misma cantidad. Es por esto, que se debe definir bien cuál es el objetivo del modelo, así se puede crear un dataset más especializado y balanceado permitiendo que el modelo aprenda de manera correcta todos los conceptos importantes.

Capítulo 6

Conclusiones y Trabajo Futuro

6.1. Conclusiones

En este trabajo se implemento un método que permite transformar descriptores de texto en descriptores visuales, para buscar imágenes relevantes a partir de una consulta de texto cumpliendo con el objetivo general de este trabajo.

En particular, se implementaron los descriptores de texto TF-IDF, R-TF-IDF y word2vec tanto en inglés como en español, cumpliendo con el primer objetivo específico. De la misma forma, se implementaron los descriptores VLAD y Deep Features para las imágenes, lo que está en línea con el segundo objetivo específico.

Luego, se entrenaron 21 redes neuronales usando los descriptores de texto en inglés y español, y los descriptores de imágenes VLAD y Deep Features. cumpliendo con el tercer objetivo específico . Finalmente, se usó el Average Precision para evaluar la relevancia de los resultados entregados por los modelos entrenados, de manera que se cumplió con el último objetivo específico. En definitiva, todos los objetivos propuestos al inicio del trabajo fueron cumplidos.

De acuerdo a los resultados obtenidos, los descriptores de texto tienen un tiempo de entrenamiento muy distintos entre sí, siendo word2vec el más rápido de entrenar con 9 [s] por época, seguido por R-TF-IDF con alrededor de 220 [s] por época y por último TF-IDF con un tiempo por época entre 850-900 [s].

En cuanto al costo en el conjunto de validación, si bien se puede ver que el descriptor word2vec es el que tiene el menor valor en todos los experimentos, la diferencia entre este descriptor y los demás no es muy grande. Ésto podría tener origen en que las representaciones de word2vec usadas son modelos pre-entrenados con el corpus de Wikipedia el cual tiene una estructura semántica diferente a las descripciones del MSCOCO, sin embargo, su principal ventaja es un entrenamiento muy rápido. Por otra parte, vale la pena mencionar que aumentar el valor de Dropout por sobre 0.2 reduce el gap entre el costo de entrenamiento y validación, pero no disminuye el valor final del costo de validación.

De los resultados también se desprende que hay grandes diferencias en Average Precision dependiendo de la combinación de descriptor de texto y descriptor de imagen usado durante el entrenamiento. Frente a esto, el descriptor VGG16 obtiene resultados casi tres veces mejores que el descriptor VLAD, y es la tupla (R-TF-IDF, VGG16) la que logro el mejor resultado de los 21 modelos entrenados, alcanzando un AP@k cercano al 60 % para k = 1 y 10. Esto quiere decir que 6 de cada 10 resultados son relevantes y coinciden parcialmente con la consulta realizada.

Otro de los aportes de este trabajo es haber entrenado un modelo completamente en español y reportar resultados que son comparables a los alcanzados por los modelos en inglés. Esto es vital, ya que significa que esta solución para resolver la búsqueda de imágenes sin etiquetar no está sujeta a un idioma, sino que se puede expandir a otros lenguajes siempre y cuando se cuente con datos de entrenamiento.

Por último, este trabajo no solo reporta los resultados obtenidos en el dataset MSCOCO si no que demuestra el funcionamiento de los modelos entrenados en imágenes propias que no están relacionadas con este dataset, donde se puede ver que tanto en inglés como español los resultados efectivamente coinciden con lo que se buscaba. Esto conduce a las aplicaciones que se le puede dar a este tipo de sistemas que le permitirían a potenciales usuarios realizar búsquedas por texto de cualquier tipo en sus fotos personales como carpetas de computador, redes sociales o incluso esto se podría integrar en buscadores de páginas de retail para que los clientes puedan hacer búsquedas que van más allá de la marca o el tipo de producto.

6.2. Trabajo Futuro

Existen varias líneas que se pueden seguir tomando este trabajo como base para mejorar los resultados obtenidos. La primera es enfocarse directamente en los Deep Features como descriptores de imágenes y hacer pruebas con otro tipo de redes convolucionales como la ResNet 50 (He et al. [17]) o la Inception (Szegedy et al. [41]). Además, se debe hacer una evaluación masiva (más de mil consultas) para establecer si existen diferencias entre los descriptores de texto TF-IDF, R-TF-IDF y word2vec.

La segunda es eliminar PCA como paso intermedio y usar técnicas de Transfer Learning (Pan et al. [33]) para que la red neuronal aprenda como hacer la reducción de dimensionalidad que mejor se adapte a los datos de entrenamiento. O incluso usar redes neuronales duales [Wang et al. [42]] que creen un espacio intermedio entre los descriptores de imágenes y texto para realizar las búsquedas.

Por último, en este trabajo solo se evaluaron los resultados frente a un número fijo de vecinos más cercanos, pero en el caso ideal, el sistema CBIR con QBD debe ser capaz de identificar todos y solo los resultados que son relevantes para la consulta realizada, esto significa, estudiar la distribución de distancia de los vecinos cuyo significado semántico efectivamente coincide con la consulta realizada.

Bibliografía

[1] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. Tensorflow: a system for large-scale machine learning. In *OSDI*, volume 16, pages 265–283, 2016.

[2] Jurandy Almeida, Ricardo da S Torres, and Siome Goldenstein. Sift applied to cbir. *Revista de Sistemas de Informacao da FSMA n*, 4:41–48, 2009.

[3] Suraya Abu Bakar, Muhammad Suzuri Hitam, and Wan Nural Jawahir Hj Wan Yussof. Content-based image retrieval using sift for binary and greyscale images. In *Signal and Image Processing Applications (ICSIPA), 2013 IEEE International Conference on*, pages 83–88. IEEE, 2013.

[4] Steven Bird and Edward Loper. Nltk: the natural language toolkit. In *Proceedings of the ACL 2004 on Interactive poster and demonstration sessions*, page 31. Association for Computational Linguistics, 2004.

[5] Gary Bradski and Adrian Kaehler. *Learning OpenCV: Computer vision with the OpenCV library*. .º'Reilly Media, Inc.", 2008.

[6] Fernando Sancho Caparrini. Redes neuronales: una visión superficial, 2017. URL `http://www.cs.us.es/~fsancho/?e=72`.

[7] Rich Caruana, Steve Lawrence, and C Lee Giles. Overfitting in neural nets: Backpropagation, conjugate gradient, and early stopping. In *Advances in neural information processing systems*, pages 402–408, 2001.

[8] Francois Chollet. *Deep learning with python*. Manning Publications Co., 2017.

[9] François Chollet et al. Keras, 2015.

[10] Visual networking Index Cisco. Forecast and methodology, 2016-2021, white paper. *San Jose, CA, USA*, 1, 2016.

[11] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, pages 248–255. Ieee, 2009.

[12] Jianfeng Dong, Xirong Li, and Cees GM Snoek. Word2visualvec: Image and video to sentence matching by visual feature prediction. *CoRR, abs/1604.06838*, 2, 2016.

[13] Jianfeng Dong, Xirong Li, and Cees GM Snoek. Predicting visual features from text for image and video caption retrieval. *IEEE Transactions on Multimedia*, 2018.

[14] Yunchao Gong, Qifa Ke, Michael Isard, and Svetlana Lazebnik. A multi-view embedding space for modeling internet images, tags, and their semantics. *International journal of computer vision*, 106(2):210–233, 2014.

[15] Yunchao Gong, Liwei Wang, Micah Hodosh, Julia Hockenmaier, and Svetlana Lazebnik. Improving image-sentence embeddings using large weakly annotated photo collections. In *European Conference on Computer Vision*, pages 529–545. Springer, 2014.

[16] David R Hardoon, Sandor Szedmak, and John Shawe-Taylor. Canonical correlation analysis: An overview with application to learning methods. *Neural computation*, 16 (12):2639–2664, 2004.

[17] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[18] Eric H Huang, Richard Socher, Christopher D Manning, and Andrew Y Ng. Improving word representations via global context and multiple word prototypes. In *Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics: Long Papers-Volume 1*, pages 873–882. Association for Computational Linguistics, 2012.

[19] Sung Ju Hwang and Kristen Grauman. Learning the relative importance of objects from tagged images for retrieval and cross-modal search. *International journal of computer vision*, 100(2):134–153, 2012.

[20] Sadegh Bafandeh Imandoust and Mohammad Bolandraftar. Application of k-nearest neighbor (knn) approach for predicting economic events: Theoretical background. *International Journal of Engineering Research and Applications*, 3(5):605–610, 2013.

[21] Herve Jegou, Florent Perronnin, Matthijs Douze, Jorge Sánchez, Patrick Perez, and Cordelia Schmid. Aggregating local image descriptors into compact codes. *IEEE transactions on pattern analysis and machine intelligence*, 34(9):1704–1716, 2012.

[22] Justin Johnson, Ranjay Krishna, Michael Stark, Li-Jia Li, David Shamma, Michael Bernstein, and Li Fei-Fei. Image retrieval using scene graphs. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3668–3678, 2015.

[23] Andrej Karpathy. Cs231n: Convolutional neural networks for visual recognition., 2018. URL `http://cs231n.github.io/convolutional-networks/`.

[24] Andrej Karpathy, Armand Joulin, and Li F Fei-Fei. Deep fragment embeddings for bidirectional image sentence mapping. In *Advances in neural information processing systems*, pages 1889–1897, 2014.

[25] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[26] Benjamin Klein, Guy Lev, Gil Sadeh, and Lior Wolf. Fisher vectors derived from hybrid gaussian-laplacian mixture models for image annotation. *arXiv preprint arXiv:1411.7399*, 2014.

[27] 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.

[28] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. Microsoft coco: Common objects in context. In *European conference on computer vision*, pages 740–755. Springer, 2014.

[29] David G Lowe. Distinctive image features from scale-invariant keypoints. *International journal of computer vision*, 60(2):91–110, 2004.

[30] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*, pages 3111–3119, 2013.

[31] Tomas Mikolov, Edouard Grave, Piotr Bojanowski, Christian Puhrsch, and Armand Joulin. Advances in pre-training distributed word representations. In *Proceedings of the International Conference on Language Resources and Evaluation (LREC 2018)*, 2018.

[32] Marcos Orchard. *Apuntes SSII*. 2016.

[33] Sinno Jialin Pan, Qiang Yang, et al. A survey on transfer learning. *IEEE Transactions on knowledge and data engineering*, 22(10):1345–1359, 2010.

[34] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, et al. Scikit-learn: Machine learning in python. *Journal of machine learning research*, 12(Oct):2825–2830, 2011.

[35] Martin F Porter. An algorithm for suffix stripping. *Program*, 14(3):130–137, 1980.

[36] Martin F Porter. Snowball: A language for stemming algorithms, 2001.

[37] Max H Quinn, Erik Conser, Jordan M Witte, and Melanie Mitchell. Semantic image retrieval via active grounding of visual situations. In *Semantic Computing (ICSC), 2018 IEEE 12th International Conference on*, pages 172–179. IEEE, 2018.

[38] Gerard Salton and Christopher Buckley. Term-weighting approaches in automatic text retrieval. *Information processing & management*, 24(5):513–523, 1988.

[39] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[40] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, 15(1):1929–1958, 2014.

[41] 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.

[42] Liwei Wang, Yin Li, and Svetlana Lazebnik. Learning deep structure-preserving image-text embeddings. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 5005–5013, 2016.

[43] Matthew D Zeiler and Rob Fergus. Visualizing and understanding convolutional networks. In *European conference on computer vision*, pages 818–833. Springer, 2014.

Apéndice A

Evolución de Entrenamientos

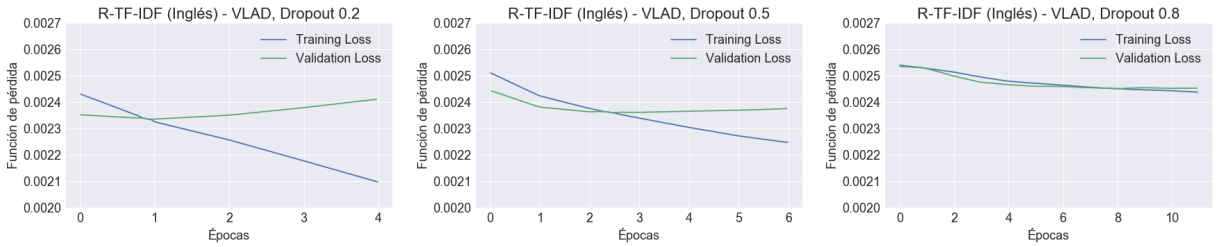


Figura A.1: Entrenamiento de red neuronal usando R-TF-IDF - VLAD y distintos valores de Dropout

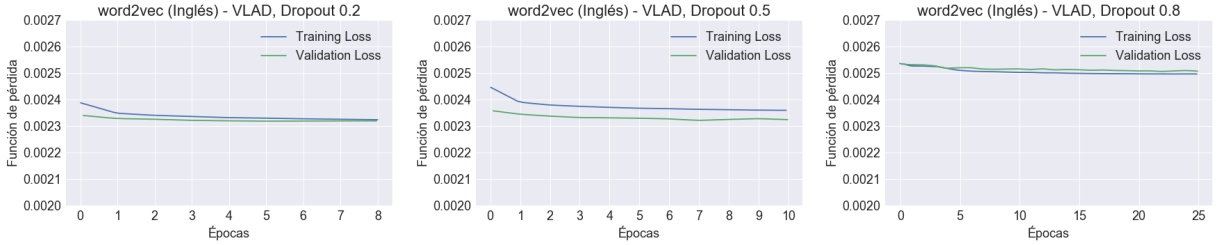


Figura A.2: Entrenamiento de red neuronal usando word2vec - VLAD y distintos valores de Dropout

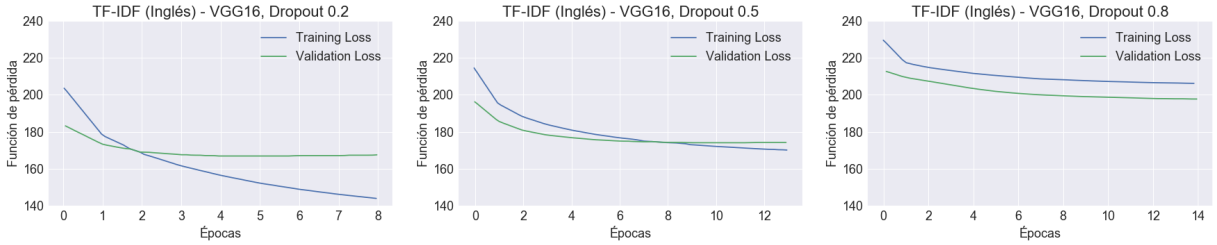


Figura A.3: Entrenamiento de red neuronal usando TF-IDF - VGG16y distintos valores de Dropout

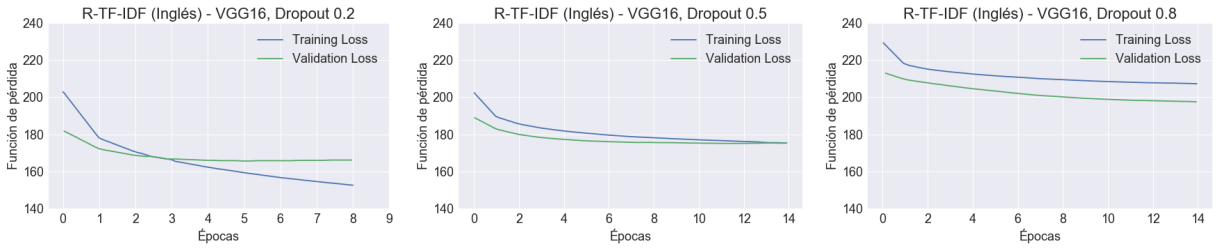


Figura A.4: Entrenamiento de red neuronal usando R-TF-IDF - VGG16 y distintos valores de Dropout

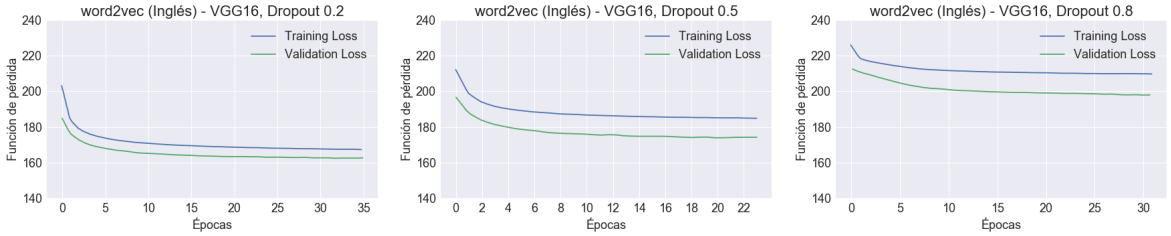


Figura A.5: Entrenamiento de red neuronal usando word2vec - VGG16 y distintos valores de Dropout

Apéndice B

Resultado de Entrenamientos

	Dropout 0.2		Dropout 0.5		Dropout 0.8	
	Seg/Ep	C. Val	Seg/Ep	C. Val	Seg/Ep	C. Val
TF IDF	920 [s]	0.0024	920 [s]	0.0024	920 [s]	0.0025
R-TF IDF	230[s]	0.0024	230 [s]	0.0024	230 [s]	0.0025
word2vect	**9 [s]**	**0.0023**	**9 [s]**	**0.0023**	**9 [s]**	**0.0025**

Tabla B.1: Resultados del entrenamiento del sistema Texto Inglés - VLAD
Seg/Ep es el promedio del tiempo que demora en entrenar con una época y C. Val es el valor de la función de pérdida en el conjunto de validación.

	Dropout 0.2		Dropout 0.5		Dropout 0.8	
	Seg/Ep	C. Val	Seg/Ep	C. Val	Seg/Ep	C. Val
TF IDF	890 [s]	169.24	860 [s]	174.16	850 [s]	197.59
R-TF IDF	220 [s]	166.14	220 [s]	**172.78**	220 [s]	**197.39**
Word2vec	**9 [s]**	**162.61**	**9 [s]**	174.05	**9[s]**	197.53

Tabla B.2: Resultados del entrenamiento del sistema Texto Inglés - VGG16
Seg/Ep son los segundos que demora el modelo en entrenar una época y C. Val es el valor de la función de pérdida en el conjunto de validación.