Universidad Complutense de Madrid
Facultad de Informática

TRABAJO DE FIN DE GRADO

Daniel Gamo Alonso

Generación de descripciones de imágenes mediante *Deep Learning*

Doble Grado en Ingeniería Informática y Matemáticase

Directores: Alberto Díaz Esteban, Gonzalo Méndez

5 de julio de 2017

TODO - Agradecimientos

Resumen

El objetivo de este trabajo es construir un sistema basado en *deep learning* con el propósito de generar descripciones textuales a partir de las imágenes que se suministren como entrada. Para ello se utilizarán distintas técnicas relacionadas con la visión artificial y el procesamiento del lenguaje natural como redes neuronales y otras metodologías basadas en modelos estadísticos para, a partir de los elementos presentes en las imágenes, predecir y generar descripciones coherentes con su contenido. Entre las posibles aplicaciones podría estar el facilitar la accesibilidad a contenido en forma de imagen a usuarios con algún tipo de discapacidad visual.

Palabras clave: *deep learning*, red neuronal, descripción de imágenes.

TODO: ENGLISH

Lista de contenidos

Capítulo 1

Introducción

A menudo se dice que una imagen vale más que mil palabras, y este dicho no podría ser más acertado. Los humanos nos apoyamos en el sentido de la vista para gran parte de las tareas que realizamos en nuestra vida cotidiana. Esta importancia motiva el contenido de nuestro trabajo, en el que pretendemos construir un modelo usando técnicas de *deep learning* (con las que se han conseguido grandes avances en los últimos años, e incluso meses, dentro de este campo) para estudiar la tarea de analizar y extraer datos de las imágenes. Sin embargo, la importancia de la visión para los humanos no se basa exclusivamente en reconocer objetos, que es la primera aproximación que se hace en este sentido, sino que también contamos con esa poderosa herramienta que es el lenguaje, y que nos permite no solo reconocer, sino describir lo que vemos. Esta idea de conexión entre la vista y el lenguaje es la que constituye el grueso de este trabajo, y nuestro objetivo va a ser estudiar y construir un modelo que relacione el contenido de una imágen con una descripción textual de la misma.

En este proceso de análisis y descripción hay que tener diversos factores en cuenta. Para empezar, hay que definir que vamos a entender por descripciones. Esta tarea es difícil debido al amplio significado que tiene la tarea de describir; no está claro cómo de larga puede ser la descripción, si debe centrarse en todos los detalles o dar una "idea general" del contenido de la imágen, etc. Todas estas razones convierten la tarea de la descripción de imágenes en algo mucho más complejo, alejado del planteamiento algo más sencillo (que no trivial) de extraer elementos de las imágenes y organizarlos en una frase bien estructurada.

1.1 Objetivos

Nuestro objetivo principal en este trabajo es construir un sistema que sea capaz de relacionar el contenido de una imagen con una descripción textual que se acerque a la que daría una persona. Para ello vamos a estudiar, desde el enfoque del *deep learning*, las diferentes técnicas y modelos existentes para el análisis de imágenes y la descripción de su contenido. Destacamos dos conceptos claves para el desarrollo de nuestro trabajo, que son las redes neuronales recurrentes (*Recurrent Neural Network*, RNN) y las redes neuronales convolucionales (*Convolutional Neural Network*, CNN), que han probado su eficacia en las tareas relacionadas con procesamiento del lenguaje natural y análisis de imágenes, respectivamente. Hablaremos en profundidad sobre ello en los capítulos 2 y 3 de esta memoria.

Para esta tarea necesitamos analizar una gran cantidad de datos. Cuando se suministra a una máquina, una imagen queda representada como una matriz de píxeles y una sentencia (descripción) como una lista de palabras (*tokens*). Cada una de estas unidades no da información por si misma; necesita del resto para conformar una unidad con sentido. Además, para inferir las reglas que permitan detectar las relaciones entre los distintos elementos (entre elementos de la imagen, entre elementos de la sentencia y entre elementos de la imagen con su correspondiente sentencia) necesitamos un gran número de imágenes y de sentencias, con lo que la capacidad de computación necesaria para llevarlo a cabo es inmensa. En nuestro caso, contamos con una tarjeta gráfica donada por NVIDIA que será clave en la realización de todos los cálculos involucrados en un tiempo aceptable.

1.2 ¿Qué es una descripción?

Ya hemos comentado la importancia que tiene definir correctamente lo que nuestro modelo va a entender por una descripción. Tenemos un modelo generativo, que necesita descripciones en el entrenamiento con un formato más o menos similar para producir buenos resultados.

Según la RAE, describir se define como:

1. Representar o detallar el aspecto de alguien o algo por medio del lenguaje.
2. Moverse a lo largo de una línea.
3. Definir imperfectamente algo, no por sus cualidades esenciales, sino dando una idea general de sus partes o propiedades.
4. Delinear, dibujar, pintar algo, representándolo de modo que se dé perfecta idea de ello.

En el proceso de descripción de una escena, si nos atenemos a la tercera acepción, no se hace un análisis detallado de todos los objetos y acciones que se reflejan en ella, sino que se resume la información, y se tiende a describir los elementos que más llaman nuestra atención. Ya sea por su importancia o tamaño en la escena, por la impresión subjetiva que nos causan o por el contexto en el que sucede la escena y que los dota de mayor o menor relevancia. Pensemos por ejemplo en una imagen de una persona con el cielo de fondo; un humano destacaría a la persona que aparece en ella y daría menos importancia a otras cosas como el cielo que aparece detrás de la imagen (no es algo que llame la atención, siempre esta ahí), mientras que una máquina podría centrar su atención en ese cielo que aparece de fondo (por ejemplo, porque ocupa un porcentaje de la imagen más alto que la persona).

En los *datasets* que vamos a utilizar, cada imagen va acompañada de cinco frases con una longitud media de ///CALCULAR CON EL CODIGO. La anotación de imágenes se ha realizado utilizando operarios humanos a través de la plataforma Amazon Mechanical Turk. Se pidió a los trabajadores que describiesen el contenido de la imagen con una frase. Se ha probado empíricamente que

en esta colección de datos se suele describir los aspectos más relevantes de la imagen, con especial incapié en descripción de personas, sus acciones, interacciones con la gente o el entorno. [Karpathy (2016)].

1.3 Problemas

La tarea que nos proponemos presenta una seria de desafíos más allá de la construcción del modelo o de la capacidad de cálculo de la que disponemos.

Cuesta decidir qué es una buena descripción de una imagen y qué no lo es, pues dos personas distintas podrían dar dos descripciones distintas de la misma imagen, ya sea por la importancia que dan a ciertos elementos de la escena o por como describan el mismo elemento. Sin embargo, dos descripciones distintas pueden ser igualmente válidas y esto pone de relieve la importancia de tener buenas métricas para que el sistema sepa cuándo está describiendo algo bien, cuándo está describiendo algo mal, cuando lo hace mejor y cuando lo hace peor. Desarrollaremos en el capítulo 4 con más detalle qué métricas utilizamos, cuál es el razonamiento que hay detrás de ellas y cómo de fiables son.

Otro inconveniente que se nos presenta es la dificultad de obtener imágenes con buenas descripciones asociadas. Aunque hay sitios como Flickr que contienen muchas imágenes con descripciones, a menudo estas últimas no dan información fiable sobre el contenido de la imagen. Cuando subimos una fotografía, no solemos describir su contenido, sino que adjuntamos texto sobre la situación en la que se produce, las personas que nos acompañan o los sentimientos que nos evocan. Por esta razón, no es fácil obtener automáticamente un *dataset* lo bastante bueno como para entrenar al sistema, y precisamos de trabajo humano para acompañar las imágenes (o partes concretas de las imágenes) de descripciones adecuadas. En este sentido, muchos de los *datasets* que se utilizan en este campo han requerido del trabajo de muchas personas, principalmente utilizando la herramienta Amazon Mechanical Turk. Además, en relación con lo que hemos expuesto en el párrafo anterior, necesitamos más de una descripción por imagen para dar perspectiva a nuestro sistema sobre las diferentes formas de describir el mismo contenido. Como en el caso de las métricas, la información acerca de los *datasets* utilizados se desarrollará en el capítulo ??.

Aunque tengamos un conjunto de pares imagen-descripción lo suficientemente grande y bueno para nuestra tarea, todavía queda algo que dificulta nuestra tarea, y es el poder asociar elementos de la descripción con zonas concretas de la imagen y no con toda ella. En este sentido, se han publicado trabajos que cuentan con *datasets* más completos en este sentido, como *Visual Genome* [Krishna et al. (2016)], de manera que las relaciones entre lo descrito y su localización en la imagen son más fáciles de aprender por el sistema.

1.4 Estructura del documento

En el capítulo 1 hemos introducido el tema sobre el que trata esta memoria. El capítulo 2 consiste en un resumen sobre los trabajos más importantes publicados acerca de análisis de imágenes, de sentencias y de relación entre ambas. En el capítulo 3 describiremos nuestro modelo, y hablaremos sobre la teoría de

deep learning que hay detrás del mismo. Además describiremos la estructura y el funcionamiento de la redes neuronales que vamos a implementar. En el capítulo 4 explicamos la metodología que seguimos en nuestros experimentos y analizamos los datos y los resultados, comparándolos con los de trabajos existentes. Se muestran además ejemplos concretos de resultados que nuestro modelo a proporcionado. Por último, en el capítulo 5 exponemos las conclusiones del trabajo y planteamos lineas de trabajo futuras. ///////HASTA CAP 5?

Capítulo 2

Trabajo relacionado

Tanto la tarea de análisis de imágenes como la de generación de sentencias en lenguaje natural mediante *deep learning* está en desarrollo constante, y cada mes aparece un nuevo artículo o tesis sobre este tema.

La idea de utilizar redes neuronales recurrentes para construir modelos relacionados con el procesamiento del lenguaje natural está muy presente en nuestro trabajo. En Kombrink et al. (2011) se propone un modelo que utiliza RNNs con este propósito, y analiza su comportamiento en esta tarea. Existen numerosos estudios sobre la generación de frases, donde nos interesan especialmente aquellos que las construyen en base a unas etiquetas [Bahdanau et al. (2014)], más cercano a la idea de combinar análisis de imágenes y sus descripciones (los elementos que aparecen en ellas se usan para generar las frases).

Análogamente, se ha probado que las redes neuronales convolucionales dan buenos resultados cuando estamos tratando con datos en forma de imagen, como en Krizhevsky et al. (2012). Sin embargo, también se plantea el problema de conseguir buenos tiempos de entrenamiento en estas redes con tantos parámetros que ajustar.

Para trabajar con las técnicas de *deep learning*, han aparecido numerosas APIs que automatizan gran parte de los cálculos necesarios en la red. Entre ellas destacamos Theano (la que usaremos en este proyecto) y TensorFlow, que cuenta con el apoyo de Google.

En el trabajo de Barnard et al. (2003) se explora la conexión entre imágenes (o regiones de imágenes) y palabras mediante diferentes modelos, aún sin utilizar técnicas de *deep learning*, presentes en las publicaciones más actuales.

Los trabajos que combinan ambos enfoques para generar descripciones de imágenes son más recientes, Estos son los que nos interesan, y que vamos a tomar como base para construir nuestro propio modelo. El elemento común entre todos ellos es que se basan en redes neuronales convolucionales para el análisis de imágenes y redes neuronales recurrentes para la generación de frases [Karpathy and Fei-Fei (2015); Vinyals et al. (2015)]. En el trabajo de Karpathy and Fei-Fei (2015) se trabaja con una modificación de las RNNs tradicionales (introduciendo la bidireccionalidad en la red) y de las CNNs, para que ambas trabajen bien en conjunto.

También es importante destacar la importancia de contar con buenos *datsets* y metodologías definidas para la evaluación de parejas imagen-frase. En el trabajo de Hodosh et al. Hodosh et al. (2013) se recopilan anotaciones de 8000

imágenes (5 para cada una de las imágenes) que deben describir las entidades y los eventos. Como en el resto de *datasets* que vamos a estudiar en este trabajo, las anotaciones se han obtenido mediante trabajadores humanos con plataformas de *crowdsourcing*. Este *dataset* se denomina Flickr8k. Mencionamos el trabajo relacionado con otro de los *datsets* que vamos a utilizar: MSCOCO Lin et al. (2014).

Capítulo 3

Descripción de imágenes mediante deep learning

En este capítulo, presentamos los conceptos más importantes de deep learning que vamos a utilizar para construir nuestro modelo. Tras ello, hablamos primero sobre las redes neuronales convolucionales, cómo funcionan y que caracerísticas las hacen buenas para el tratamiento de imágenes. En la sección (REF) estudiamos las redes neuronales recurrentes que utilizamos en la vertiente de procesamiento del lenguaje natural para la generación de sentencias. Muchas de las cosas de la que aquí hablamos pueden ampliarse con la gran cantidad de textos que existen sobre el tema. En particular, es muy recomendable la lectura del libro *Deep Learning Book*[1] de Goodfellow et al. (2016)

3.1 Machine Learning

Comenzamos hablando de machine learning, campo en el que también se ubica el deep learning. El machine learning consiste en desarrollar sistemas que reciben unos ejemplos de la tarea que deben realizar, y a partir de ellos inferir reglas que permitan a la máquina completar esas tareas automáticamente.

En un sistema de machine learning diferenciamos varios elementos importantes: la tarea a realizar, una medida del rendimiento y la experiencia del aprendizaje.

Algunas de las principales tareas a aprender en machine learning son:

1. **Clasificación.** El sistema dispone de k categorías, de forma que cada entrada $x \in X$ se mapea a una de estas categorías vía una función $f : \mathbb{R} \to \{1, ..., k\}$.

2. **Regresión logística.** Dada una entrada, queremos obtener un valor numérico para la salida correspondiente. En este caso, cada $x \in X$ se mapea a un valor real mediante una función $f : X \to \mathbb{R}$. La principal diferencia con la clasificación es el formato de la salida.

Para cuantificar cómo se comporta el sistema, necesitaremos una medida del rendimiento, que será específica para cada tarea.

[1] Disponible de forma gratuita en http://www.deeplearningbook.org/

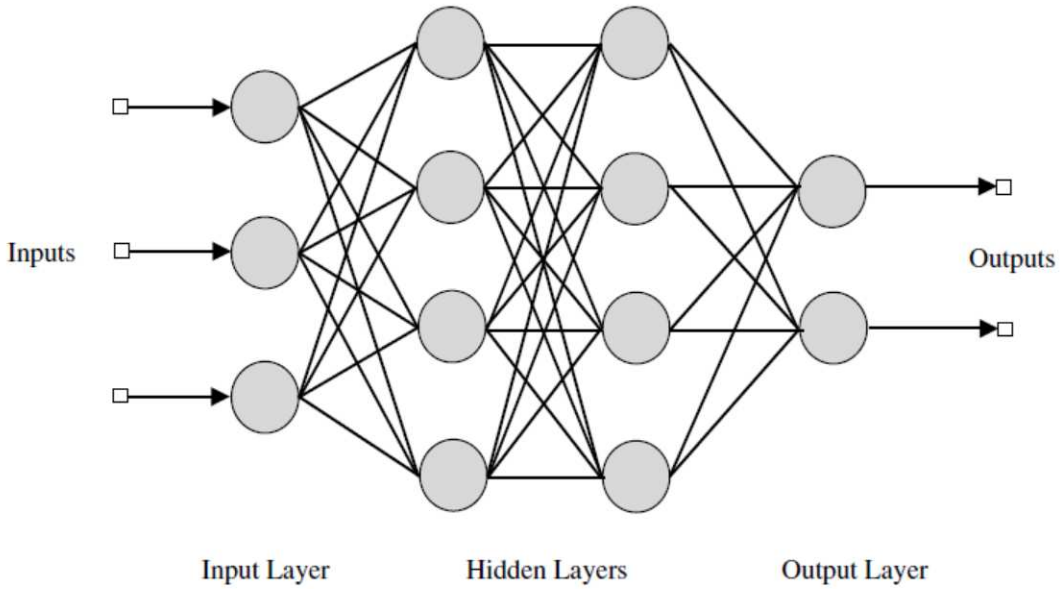


Figura 3.1: Red neuronal con 2 capas ocultas de 4 neuronas cada una. La información fluye de izquierda a derecha. También se utiliza un termino de bias, que se omite por claridad.

Por último, y según cómo sea la experiencia de aprendizaje, hablaremos de aprendizaje supervisado (el sistema es alimentado con ejemplos correctos acompañados de su etiqueta o valor correspondiente) o no supervisado (el sistema es alimentado con ejemplos para que aprenda su estructura).

3.2 Deep Learning

En esta sección explicamos los conceptos de deep learning más importantes que usamos para realizar el trabajo. Comenzamos hablando del esquema más generar de deep learning (redes neuronales *feedforward*, algoritmo de aprendizaje, etc. Tratamos además dos tipos de redes concretas, las redes neuronales convolucionales (para el tratamiento de imágenes) y las redes neuronales recurrentes (para procesamiento del lenguaje natural).

3.2.1 Red Neuronal Feedforward

La red neuronal feedforward constituye la base del deep learning. Esta formado por una capa de entrada (*input layer*), una capa "oculta" (*hidden layer*) y una capa de salida (*output layer*). En estas redes, las neuronas de la entrada estan conectadas a las neuronas de la capa oculta, y las neuronas de la capa oculta a las de salida. Se llaman feedforward porque la información avanza, desde la entrada hasta la salida de la red.

Consideramos la tarea de clasificación planteada anteriormente. Tenemos unos valores de entrada (x, y) y el objetivo es que el modelo aproxime el valor de una función, que en este caso será $y = f^*(x)$ para cada par de entrada (x, y), donde

x sería la entidad a clasificar e y la correspondiente clase. Para realizar esta aproximación, la red neuronal define una función $y = f(x;\theta)$ y modifica los parámetros del modelo, θ, buscando que la función definida por la red neuronal sea una buena aproximación de la función original $f*$. (aprendemos) los valores óptimos de los parámetros θ del sistema de manera que la función $f(x;\theta)$ sea una buena aproximación de $f*$.

Aunque puede resultar sorprendente, esta arquitectura por si sola puede obtener buenos resultados en la aproximación de la función dada (en nuestro casao $f*$). Esta idea queda respaldada por el teorema de aproximación Universal. Este teorema dice que una red neuronal *feedforward* con un número finito de neuronas puede aproximar funciones, siempre que estas sean continuas en subconjuntos compactos de $\mathbb{R}^n$, y la función de activación de la red cumpla ciertos requisitos (la demostración mas conocida del teorema se realiza para la función sigmoide).

Estas redes neuronales suelen ser una composición de n capas, de manera que la función final del modelo puede expresarse como una composición de n funciones:

$$f(x;\theta) = f^{(n)}(f^{(n-1)}(...f^{(1)}(x;\theta)...)) \tag{3.1}$$

donde $f^{(k)}$ denota a la función modelada por la capa k. El término deep learning viene de la profundidad de estas redes, del número de capas que las componen.

3.2.2 Descenso de gradiente estocástico

Hasta ahora hemos hablado de cómo son las redes neuronales en general, y sabemos que son capazas de aproximar la función objetivo. Sin embargo, los resultados teóricos del apartado anterior no nos daban ninguna información sobre cómo guiar al modelo para que llegue a una configuración de parámetros que estime bien la función objetivo.

El algoritmo más utilizado para este propósito es el descenso de gradiente estocástico (SGD, stochastic gradient descent en inglés). Este algoritmo sirve para minimizar una función objetivo descrita como suma de funciones diferenciables. Toma como entrada una tasa de aprendizaje(*learning rate*), los parámetros iniciales del modelo y utlizando los ejemplos que se le suministran devuelve la mejor configuración de parámetros encontrada.

Ya casi tenemos todo para realizar el aprendizaje, pero nos falta por determinar cuál es la función que vamos a minimizar. Esta función se denomina función de coste, y la elección de esta función puede influir en los resultados que obtengamos al aplicar el algoritmo. Lo más común es utilizar la función *negative log-likelihood*, ya que minimizar esta función es equivalente a maximizar la probabilidad del dataset D dados los parámetros del modelo. La función se define como:

$$l(\theta, D) = -\sum_{i=0}^{|D|} log(P(Y = y^{(i)}|x^{(i)}, \theta)) \tag{3.2}$$

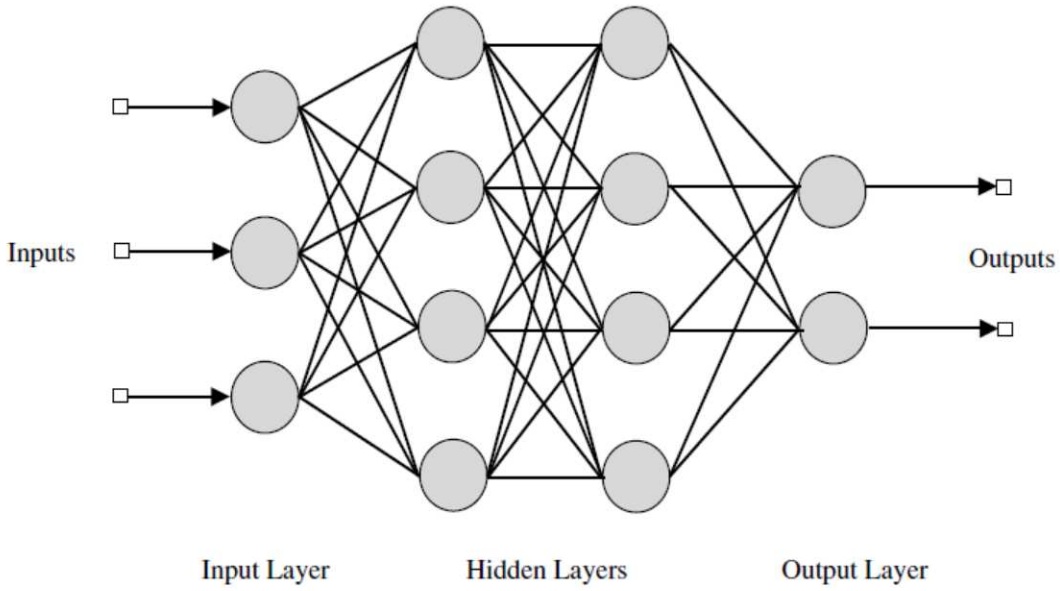


Figura 3.2: Esquema del algoritmo de backpropagation .

En nuestro modelo, usamos una variante del descenso de gradiente estocástico con mini-batches. Esto implica que en cada iteración, se computan varios ejemplos a la vez, lo que permite acelerar los tiempos aplicando computación en paralelo (en nuestro caso usando la GPU).

3.2.3 Backpropagation

Con el algoritmo SGD tenemos un método para optimizar los parámetros de nuestra red, y aun con la técnica de los mini-batches, el cálculo del gradiente es una operación computacionalmente compleja. Para solventar este problema, esta envía mensajes hacia delante en la red, donde se calcula el error comparando la salida del modelo con la salida esperada y se envía hacia atrás para ajustar el peso de cada neurona. El gradiente de la función para la entrada y los parámetros se empieza a calcular en las últimas capas y se propaga hacia las primeras.

Debido a la importancia del cálculo de gradientes, las herramientas que se utilizan para trabajar con redes neuronales (theano en nuestro caso) proveen diferenciación simbólica para computar los mismos de forma automática y eficiente.

3.3 Dataset

Para entrenar nuestro modelo utilizaremos datasets consistentes en una lista de imágenes, cada una de ellas acompañada de 5 descripciones. Los datasets que vamos a utilizar son Flickr8k [Hodosh et al. (2013)], MSCOCO [Lin et al. (2014)] y Flickr30k [Young et al. (2014)]. En deep learning, se suele tener una división de los datasets en tres partes, cada una con un propósito concreto:

1. **Training set.** Son los datos que se utilizan para entrenar al modelo. En el entrenamiento, el sistema recibe estos ejemplos para ajustar una serie de parámetros con el objetivo de mejorar los resultados en la tarea concreta que se esté llevando a cabo.

2. **Validation set.** Al entrenar, el sistema tiende al sobreajuste (*overfitting*): los parámetros de la red se ajustan muy bien a los ejemplos concretos con los que se la entrena, pero ante un dato de entrenamiento diferente ofrece malos resultados. Para evitarlo, se suministran ejemplos al sistema que no forman parte de los datos de entrenamiento. Se calcula la precisión del sistema con datos de entrenamiento y estos últimos de validación, con la intención de actualizar los parámetros tras el entrenamiento sólo si dicha actualización mejora también los resultados sobre los datos de validación.

3. **Testing set.** Tras el entrenamiento, estos datos se utilizan para observar la precisión final alcanzada por la red y poder cuantificar los resultados.

Estos datasets consisten en 8000, 123000, y 31000, respectivamente. En nuestros experimentos, tomamos 1000 imágenes para validación, 1000 imágenes para pruebas y el resto para entrenamiento, como se hace en Karpathy (2016).

3.4 Redes neuronales convolucionales

Las redes convolucionales (CNN) son un tipo de red especializada en procesamiento de datos con una topología similar a una cuadrícula (en nuestro caso, una matriz de píxeles que representan la imagen). El nombre de la red viene de la operación de convolución, parte central del funcionamiento de una CNN.

//SPARSE CONNECTIVITY

Pare entender como funciona una red convolucional, vamos a explicar la función de cada una de las capas que la integra. Las más comunes (y que nosotros utilizamos) son la capa de convolución (CONV), la de max pooling (POOL), la de regularización (DROPOUT) y redes fully connected (FC).

- **Convolución.** En esta capa implementa la operación convolución. En ella, un filtro o *kernel* se va desplazando y aplicandose sobre zonas de la imagen, produciendo lo que se denomina mapa de características de la imagen, y que intuitivamente resume la información de esa parte de la imagen. Esta operación se aprovecha de la conexión espacial entre pixeles cercanos, permitiendo analizar la imagen por regiones (y no por pixeles). Los hiperparámetros () para esta arquitectura son el tamaño del kernel, el número de estos que vamos a utilizar, el stride (desplazamiento del kernel a lo largo de cada dimensión) y el padding (completar la imagen con ceros para ajustar la dimensión).

- **No linealidad.** Se aplica una función de activación no lineal operando sobre cada pixel del mapa de activación. Aunque pueden usarse funciones como la sigmoide, se ha probado que la función ReLU (Rectified Linear Unit) da mejores resultados en este tipo de redes neuronales.

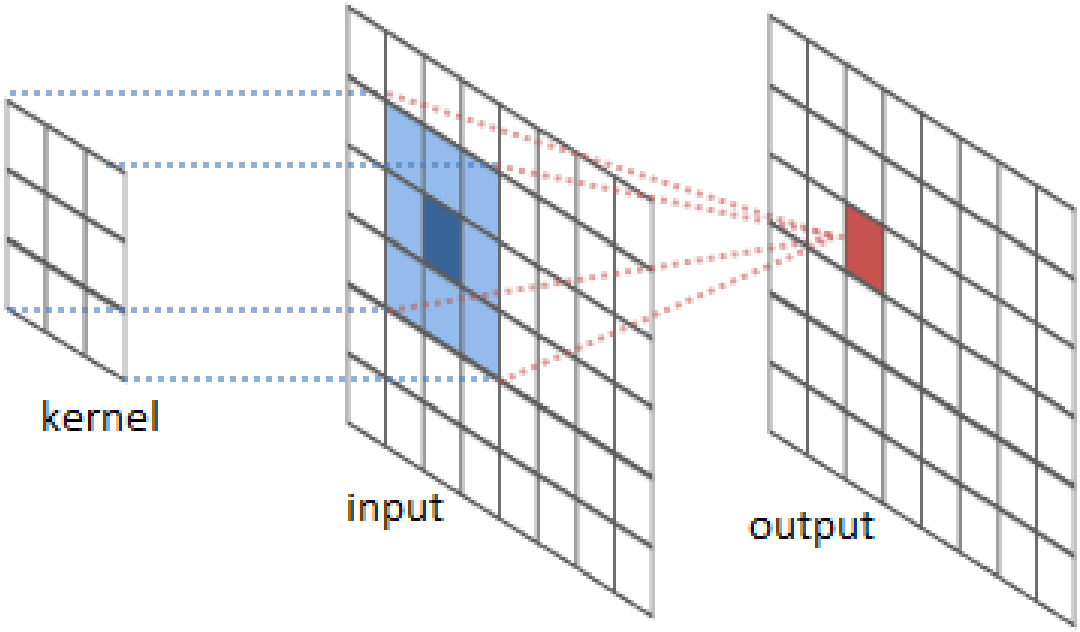


Figura 3.3: Capa convolucional de una CNN. El kernel se aplica sobre una región de la imagen de entrada.

- **Max pooling.** Una capa de pooling reemplaza la salida de la red para una cierta zona de la imagen por un resumen de las salidas cercanas. El método de pooling más común se denomina max pooling, en el que se toma como resumen de la región el valor de salida más alto.

- **Dropout.** Esta regularización se utiliza para evitar los problemas de overfitting. Su funcionamiento es muy simple: para cada salida de la capa anterior, cada neurona se acepta o no (se tiene en cuenta su valor) con una probabilidad de ser usada $1 - P(drop)$.

- **Fully connected layer.** La última capa de la red es una arquitectura fully connected, cuya salida se pasa a un clasificador (softmax) para calcular la clasificación, los erroes y realizar el entrenamiento.

Los parámetros a aprender en la red son los pesos que acompañan a los kernels y los parámetros propios de las capas fully connected.

3.4.1 Ejemplos de redes convolucionales

Se han diseñado numerosas arquitecturas para redes convolucionales, aunque la mayoría realiza las mismas operaciones, quizá en distinto orden y/o un número distinto de veces. Las arquitecturas más comunes son de la forma:

INPUT -¿ [[CONV -¿ RELU]*N -¿ POOL?]*M -¿ [FC -¿ RELU]*K -¿ FC

Algunas de las arquitecturas más importantes son LeNet, GoogLeNet y VGG.

///CITAR Y AMPLIAR//

A la hora de implementar nuestro encoder de imágenes utilizaremos una VGG de 16 capas, que se muestra en la figura.

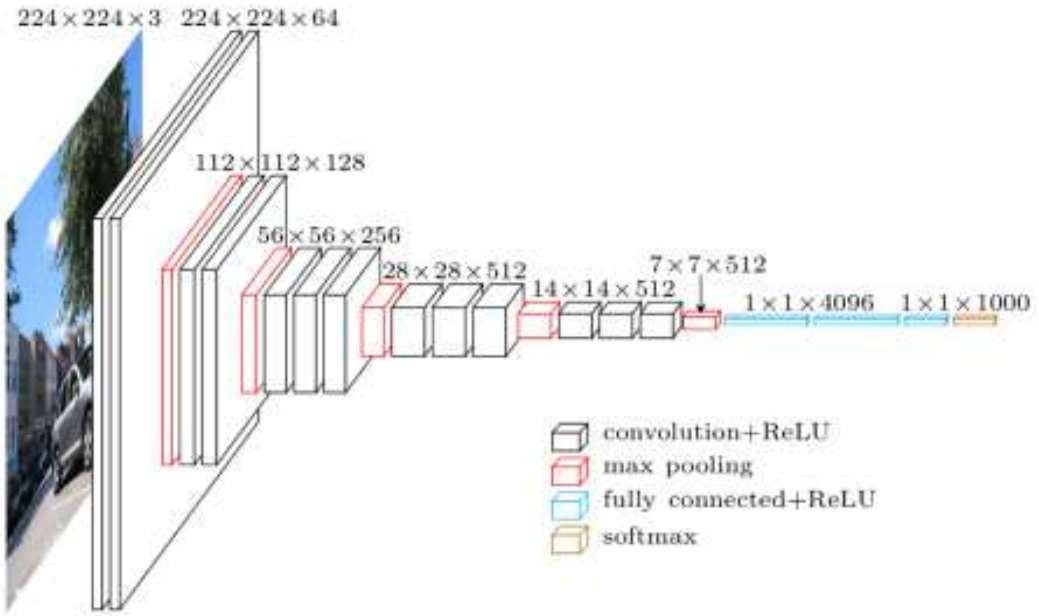


Figura 3.4: Arquitectura VGG de 16 capas.

3.5 Redes neuronales recurrentes

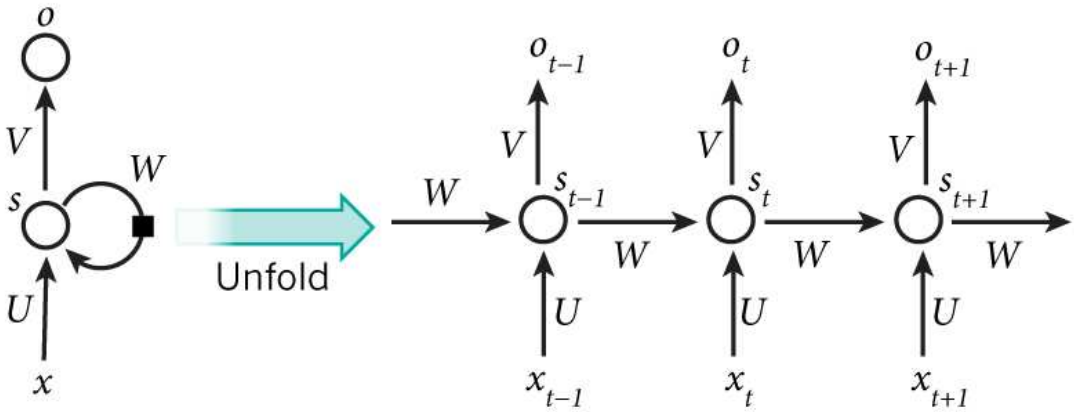
A diferencia de las redes neuronales tradicionales, en las redes neuronales recurrentes se considera que los datos de entrada (y salida) no son independientes entre sí. Esta propiedad es adecuada para nuestra tarea de procesar sentencias, pues nos interesa ver la relación existente entre las palabras, y no su representación de forma individual.

En base a una arquitectura podemos construir modelos que nos permitan, entre otras cosas:

1. Clasificar sentencias de acuerdo a su probabilidad de aparecer en una situación real, dándonos una medida de su corrección sintáctica y/o gramática.

2. Generar texto nuevo (original) tras entrenar el sistema con frases de prueba.

Observamos la importancia de considerar dependencias entre las entradas y las salidas de la red: en el caso de las frases, si queremos generar una nueva palabra, tendremos que tener en cuenta la parte de la frase ya generada, pues esta influirá en el resto de la sentencia.



En este caso, x_t representa la entrada de la red, s_t el estado oculto y o_t la salida en el paso t. En la figura vemos que el estado s_t se calcula como función del estado anterior s_{t-1}, la entrada en el paso actual x_t. La red posee "memoria" en el sentido en que los estados anteriores condicionan el estado actual.

Sin embargo, esta memoria no se mantiene durante muchas fases. Existe un tipo concreto de RNN, las conocidas como *long short-term memory* (LSTM) que favorece la persistencia de los datos de los estados anteriores durante un número de mayor de fases, lo que las hace especialmente indicadas para comprensión de lenguaje natural, análisis de textos manuscritos y reconocimiento de voz.

3.6 Asociando imágenes y descripciones

Ya hemos planteado el contexto teórico y los conceptos de deep learning más importantes a la hora de realizar nuestro trabajo. En esta sección vamos a partir de todo esto para llegar hasta la construcción final de nuestro modelo.

La tarea principal de nuestro trabajo consiste en diseñar un modelo que, dada una imagen como entrada, sea capaz de generar (predecir) una sentencia que la describa. Además, con el objetivo de ver la relación entre las imágenes y las sentencias, necesitamos una manera de compararlas, digamos asignando una puntuación a una pareja (imagen, sentencia) que indique lo relacionados que están entre sí. Así, una pareja en la que la imagen quede bien descrita por la sentencia tendrá mayor puntuación que una pareja en la que la imagen y su sentencia tengan poca relación entre sí. Tanto una imagen como una sentencia son objetos de alta dimensión, de manera que asociar unos con otros no es una tarea inmediata, y requiere de una arquitectura más compleja que la de las redes neuronales clásicas.

3.6.1 Representación de imágenes

Para representar las imágenes, podemos hacerlo de dos formas: codificar cada imagen como un vector (global image encoding) o codificar cada imagen como un conjunto de vectores (fragment-level image encoding), cada uno asociado a una parte de esta. Para simplificar nuestro modelo y mantener tiempos de entrenamiento admisibles, usaremos la primera aproximación.

Para transformar nuestra imagen de entrada en su representación vectorial utilizaremos una red neuronal convolucional, que toma la imagen I (matriz de tamaño $Height * Weight * Depth$) y aplica una serie de transformaciones hasta convertirla en un vector v. Representamos esta transformación mediante una función CNN_θ (donde θ representa los parámetros de la red). Este vector representa las características de la imagen, y será de utilidad al modelar las secuencias asociadas a imágenes.

Para realizar esta transformación de la imagen, como se hace en numerosos trabajos [Karpathy (2016)], se preentrena una CNN con el reto de clasficación de ImageNet [2], consistente en una serie de imágenes y 1000 categorías diferentes para su clasificación.

Las imágenes atraviesan las diferentes capas de la CNN (descritas en la sección SEC) y tras atravesar la última capa fully connected, obtenemos un vector de

[2]Disponible para su descarga en http://image-net.org/download

Figura 3.5: Mejores 5 resultados para cada imagen usando la red neuronal convolucional.

dimensión 4098. En la última capa se encuentra el clasificador (softmax), que para cada una de las 1000 categorías del desafío de clasificación de ImageNet, da un valor entre 0 y 1 indicando la probabilidad de que el objeto de la categoría aparezca en la imagen.

Todo este proceso da lugar a una CNN con parámetros aprendidos (denotados por θ_0) que usaremos para realizar una transformación fija de la imagen en un vector, sobre el que se aplica una transformación afín para obtener el encoding de la imagen:

$$v = W[CNN_{\theta_0}(I)] + b \tag{3.3}$$

Para ahorrarnos tiempo de entrenamiento, y puesto que hay muchos modelos preentrenados disponibles, nosotros utilizamos la red VGG-16 [Simonyan and Zisserman (2014)] ya entrenada en el desafío de ImageNet [3].

En la figura podemos ver varios ejemplos de clasificación de imágenes usando la CNN. Aunque la clasificacion es buena, observamos que en la primera imagen la red detecta que aparece un violín, pero no la persona que lo toca. Esto se debe a que el desafio de ImageNet, aunque resulte sorprendente, no cuenta con categorías como "hombre", "mujer" o similares.

3.6.2 Representación de sentencias

Ya sabemos como codificar las imágenes, y ahora queremos hacer lo mismo con las sentencias. Así, transformaremos una sentencia, dada como una secuencia de palabras $(s_1, s_2, ..., s_T)$, en un vector, de manera que podamos asociarlo con la imagen y entrenar la red en consecuencia.

Nuestro objetivo es obtener una representación de la sentencia como un vector s. Cadap palabra de la entrada puede interpretarse como un vector one-hot de dimensión nuestro vocabulario, donde el índice asociado a la palabra es 1 y el resto son 0. De cara a la implementación, y como estamos trabajando con un lenguaje de tamaño fijo, todas aquellas palabras que queden fuera de nuestro vocabulario se mapearan a la palabra especial UNK (desconocido), que será a todos los efectos una palabra más de nuestro vocabulario.

La forma más inmediata de codificar es usando la técnica de bag of words, donde cada palabra se considera como un elemento individual y se obtiene su codificación proyectando el vector one-hot con una transformación lineal. La rep-

[3] Modelos preentrenados para theano + lasagne disponibles en https://github.com/Lasagne/Recipes

resentación final puede obtenerse como suma de cada representación individual:

$$s = \sum_{i=1}^{T} s_i \quad \text{donde} \quad s_i = W_w \mathbb{I}_i \tag{3.4}$$

Sim embargo, esta codificación no posee propiedades atractivas para nuestra tarea. En particular, no conserva las relaciones espaciales entre las palabras, como la que hay entre un adjetivo y el objeto al que se refiere.

Para superar las limitaciones de la codificación mediante bag of words, utilizamos las redes neuronales recurrentes. El comportamiento de nuestro encoder viene dado por la siguiente recurrencia para $t = 1, ...T$:

$$\begin{aligned} h_0 &= \vec{0} \\ e_t &= W_w \mathbb{I}_w \\ h_t &= f(W_h h h_{t-1} + W_x h e_t + b) \end{aligned} \tag{3.5}$$

En estas ecuaciones, los parámetros $W_w, W_h h, W_x h, b$ son aprendidos, y la no linealidad f que utlizamos es una ReLU. La codificación de la secuencia puede interpretarse como el último valor h_T o la suma de los estados ocultos ($s = \sum_{t=1}^{T} h_t$).

3.6.3 Comparación imagen-sentencia

Con lo expuesto anteiormente, podemos tomar un par de entrada (imagen, sentencia) y codificarlo como vectores (v, s). Usando estos vectores, queremos buscar una función de coste que relacione estas representaciones. Podemos interpretar el producto $S = v^T s$ entre los vectores imagen y sentencia como una puntuación de lo cercanos que están entre sí en el espacio vectorial común.

EXPLICACION DE LA ELECCION

En Karpathy (2016), se utiliza una función de coste para el entrenamiento con buenas propiedades para esta tarea:

$$L(\theta) = \sum_k \left[\sum_l max(0, S_{kl} - S_{kk} + 1) + \sum_l max(0, S_{lk} - S_{kk} + 1) \right] + \lambda \|\theta\|^2 \tag{3.6}$$

Esta será la función de coste que utilizaremos en nuestro entrenamiento.

EJEMPLOS DE SCORE

3.7 Generación de descripciones

ULTIMA PARTE, ACOMPAÑANDO AL CODIGO

Capítulo 4

Implementación del modelo

Para entrenar un modelo de deep learning no solo basta contar con tener una forma de implementar todas las técnicas y algoritmos vistos en las secciones anteriores. Estas técnicas, aunque válidas, pueden llegar a ser muy lentas según el tamaño del dataset. Además, al tratar con imágenes, las dimensiones de los datos crecen notablemente, aumentando más aún la complejidad.

Para solucionar este problema, existen muchas herramientas para deep learning que ofrecen cálculo simbólico para poder ejecutar operaciones en la GPU. Estos entornos representan la arquitectura como un grafo computacional y bajo esta perspectiva, instancian las variables simbólicas en tiempo de ejecución con datos concretos. Esta característica es fundamental, pues en los algoritmos de aprendizaje aplicamos una serie de transformaciones a los datos de entrada, y utilizando la GPU podemos hacer estos cálculos para cada entrada de forma paralela. Aunque cargar los datos en la GPU es más costoso que hacerlo en memoria princpial, una vez cargados, las operaciones de nuestra red se realizan mucho mas rápido que en la CPU.

4.1 Theano

En los últimos años han aparecido numerosos framework de trabajo con redes neuronales. Los más utilizados son Tensorflow Abadi et al. (2016), Theano Bergstra et al. (2010) y Torch Collobert et al. (2011)

Theano[1] es un framework escrito sobre Python que ofrece, entre otras, las siguientes funcionalidades:

- **Integración con numpy**.
- **Uso transparente de la GPU**.
- **Diferenciación simbólica eficiente**.
- **Integración con numpy**.

Theano crea un grafo computacional para nuestro modelo, y en ejecución, ese grafo es alimenatado con los valores concretos de las variables.

[1] www.deeplearning.net

4.1.1 Tensor

El objeto fundamental en Theano es el tensor, que representa una variable u expresión simbólica.

//CREACION DE TENSORES //ALGUNAS OPERACIONES //COMPILACION DE FUNCIONES //UPDATES //(SCAN PARA RECURRENCIA)

4.1.2 Modelo preentrenado con lasagne

Para generar el modelo de la arquitectura CNN hemos utilizado una implementación de lasagne, una librería sobre theano para abstraer la creación de capas y ajuste de parámetros). Para la codificación de imágenes usamos una red tipo VGG de 16 capas preentrenada ya en el desafío de clasificación de ImageNet.

4.2 GPU

Como ya hemos comentado en la introducción, Theano permite ejecutar operaciones sobre la GPU, resultando en mayores tiempos de carga y menores tiempos de ejecución. Para realizar nuestros experimentos necesitamos además que la GPU tenga una cantidad de memoria alta, de manera que pueda almacenar en ella los datos del modelo y poder acceder a ellos para ejecutar el grafo descrito.

En nuestro caso, contamos con una tarjeta gráfica TITAN X PASCAL de 12 Gb de memoria que fue donada por NVIDIA para la realización de nuestro proyecto.

4.3 CUDA

Para trabajar sobre la GPU, NVIDIA ofrece unas herramientas de desarrollo y un compilador que permiten usar una variante del lenguaje C para codificar algoritmos en tarjetas gráfica NVIDIA compatibles.

CUDA[2] (Compute Unified Device Architecture) aprovecha los núcleos de la tarjeta gráfica para lanzar hilos de ejecución simultáneos. En el caso de las redes neuronales, la evaluación de cada ejemplo en el minibatch (la unidad que se carga en la GPU cada iteración) se realiza de forma independiente al resto de ejemplos, luego estos cálculos pueden ser paralelizados y sacar provecho de la GPU.

Theano ofrece integración con CUDA (actualmente en su versión 8).

4.3.1 CUDnn

CUDnn[3] es una librería de CUDA que ofrece primitivas para redes neuronales, con operaciones aceleradas en la GPU. Esta librería se utiliza en theano para implementar la operación de convolución

En nuestros experimentos hemos trabajado con la versión 5.1 de CUDnn.

[2] http://www.nvidia.es/object/cuda-parallel-computing-es.html
[3] https://developer.nvidia.com/cudnn

Capítulo 5

Experimentos

En este capítulo presentamos los resultados obtenidos por nuestro modelo para las distintas tareas, los comparamos con otros trabajos y discutimos las elecciones de hiperparámetros y algoritmos que hemos utilizado en los experimentos. Describimos también el preprocesamiento de los datos que hacemos antes del entrenamiento y comentamos los tiempos obtenidos en cada uno de los apartados.

5.1 Clasificación de imágenes

5.1.1 Preprocesamiento

5.1.2 Resultados cuantitativos

5.1.3 Resultados cualitativos

5.2 Correspondencia imagen-sentencia

5.3 Generación de descripciones

Capítulo 6

Conclusiones y trabajo futuro

///LO ULTIMO

Bibliografía

Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, et al. Tensorflow: Large-scale machine learning on heterogeneous distributed systems. *arXiv preprint arXiv:1603.04467*, 2016.

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*, 2014.

Kobus Barnard, Pinar Duygulu, David Forsyth, Nando de Freitas, David M Blei, and Michael I Jordan. Matching words and pictures. *Journal of machine learning research*, 3(Feb):1107–1135, 2003.

James Bergstra, Olivier Breuleux, Frédéric Bastien, Pascal Lamblin, Razvan Pascanu, Guillaume Desjardins, Joseph Turian, David Warde-Farley, and Yoshua Bengio. Theano: A cpu and gpu math compiler in python. In *Proc. 9th Python in Science Conf*, pages 1–7, 2010.

Ronan Collobert, Koray Kavukcuoglu, and Clément Farabet. Torch7: A matlab-like environment for machine learning. In *BigLearn, NIPS Workshop*, number EPFL-CONF-192376, 2011.

Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.

Micah Hodosh, Peter Young, and Julia Hockenmaier. Framing image description as a ranking task: Data, models and evaluation metrics. *Journal of Artificial Intelligence Research*, 47:853–899, 2013.

Andrej Karpathy. *Connecting Images and Natural Language*. PhD thesis, Stanford University, 2016.

Andrej Karpathy and Li Fei-Fei. Deep visual-semantic alignments for generating image descriptions. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3128–3137, 2015.

Stefan Kombrink, Tomáš Mikolov, Martin Karafiát, and Lukáš Burget. Recurrent neural network based language modeling in meeting recognition. In *Twelfth Annual Conference of the International Speech Communication Association*, 2011.

Ranjay Krishna, Yuke Zhu, Oliver Groth, Justin Johnson, Kenji Hata, Joshua Kravitz, Stephanie Chen, Yannis Kalantidis, Li-Jia Li, David A Shamma, Michael Bernstein, and Li Fei-Fei. Visual genome: Connecting language and vision using crowdsourced dense image annotations. 2016. URL https://arxiv.org/abs/1602.07332.

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

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.

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

Oriol Vinyals, Alexander Toshev, Samy Bengio, and Dumitru Erhan. Show and tell: A neural image caption generator. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3156–3164, 2015.

Peter Young, Alice Lai, Micah Hodosh, and Julia Hockenmaier. From image descriptions to visual denotations: New similarity metrics for semantic inference over event descriptions. *Transactions of the Association for Computational Linguistics*, 2:67–78, 2014.