

Instituto Tecnológico Autónomo de México



Deep Learning

---

## Proyecto final: text-based image generation

---

*Profesor:*  
Edgar Francisco Román Rangel

Paola Mejía Domenzaín 157093  
Elizabeth Rodríguez Sánchez 191430  
Juan B. Martínez Parente Castañeda 124458

28 de mayo de 2020

### Resumen

Se presenta una solución para el problema de generación de imágenes a partir de texto. La solución propuesta es una StackGAN que utiliza BERT para generar embeddings. El mejor modelo utilizó Adam como optimizador con una tasa de aprendizaje de 0.0002, LeakyReLU como función de activación y un tamaño de batch de 128. Los resultados fueron satisfactorios, en cuanto a que el modelo logró aprender a reconocer los márgenes negros de las imágenes (de forma horizontal y vertical) y palabras como cielo y agua.

# Índice

<b>1. Introducción</b>	<b>1</b>
<b>2. El problema</b>	<b>2</b>
2.1. Los datos . . . . .	2
2.2. Los retos . . . . .	3
<b>3. La solución</b>	<b>3</b>
3.1. División de entrenamiento y validación . . . . .	4
3.2. Representación latente o <i>embedding</i> . . . . .	4
<b>4. Experimentos y resultados</b>	<b>5</b>
4.1. <i>Embeddings</i> . . . . .	5
4.2. Hiper-parámetros . . . . .	6
4.3. Generación de imágenes sintéticas . . . . .	7
4.4. Generación de imágenes <i>random</i> . . . . .	7
4.5. Stage II . . . . .	8
<b>5. Conclusión</b>	<b>8</b>
<b>A. StackGAN</b>	<b>10</b>
A.1. Arquitectura . . . . .	10
A.2. Resultados del artículo original . . . . .	11
<b>B. Definición de la distancia coseno</b>	<b>12</b>
<b>C. Imágenes generadas con el vector de características de Xception</b>	<b>12</b>
<b>D. Comparaciones</b>	<b>13</b>
D.1. Comparación de <i>embeddings</i> . . . . .	13
D.2. Comparación de tamaños de <i>batch</i> . . . . .	14
D.3. Comparación de optimizadores . . . . .	15
D.4. Comparación de tasa de aprendizaje . . . . .	16
D.5. Comparación de funciones de activación . . . . .	17
D.6. Comparación de épocas . . . . .	18
<b>E. Gráficas de pérdidas</b>	<b>19</b>
<b>F. Código</b>	<b>21</b>
F.1. División en conjuntos de entrenamiento, prueba y validación . . . . .	21
F.2. Creación de <i>embeddings</i> BERT . . . . .	21
F.3. Creación de <i>embeddings</i> Skipgrams . . . . .	21

## 1. Introducción

El problema aquí planteado consiste en generar imágenes a partir de su descripción textual. Se trata de un problema con un auge reciente, para el que se han propuesto ya distintas soluciones. Mencionamos a continuación algunas de ellas, basadas en *generative adversarial networks* (GANs):

- StackGAN (Zhang y col. 2017) es una red neuronal basada en GANs condicionales que logra generar imágenes realistas dividiendo el problema en dos etapas: Stage-I GAN, que genera las formas y colores básicos en imágenes de baja resolución ( $64 \times 64$ ), y Stage-II GAN, que parte de los resultados de Stage-I para añadirles detalles y corregir defectos o distorsiones, generando imágenes de mejor resolución ( $256 \times 256$ ). El diagrama de la arquitectura está en el apéndice A.1.
- AttnGAN (Xu y col. 2017), otra GAN que permite sintetizar detalles en distintas subregiones de la imagen, poniendo atención en las palabras relevantes de la descripción textual. Para ello, se codifican tanto los enunciados como las palabras en vectores. El vector del enunciado se utiliza en una primera etapa para generar una imagen de baja resolución, que sólo contiene las formas y colores de base, sin los detalles descritos por las palabras. Las siguientes etapas rectifican defectos y añaden detalles para generar imágenes de mayor resolución.
- MirrorGAN (Qiao y col. 2019), una GAN que busca mejorar la coincidencia semántica entre el texto de entrada y la imagen generada mediante un proceso texto-imagen-texto, en el que compara el texto original contra una “redescipción” de la imagen generada. La idea de base es que la imagen generada será consistente a nivel semántico si puede ser vuelta a describir correctamente. Genera, progresivamente, imágenes de  $64 \times 64$ ,  $128 \times 128$  y  $256 \times 256$ .

Una de las mayores limitaciones de estas propuestas es que utilizan los mismos tres conjuntos de datos: CUB (Wah y col. 2011), restringido a fotografías de aves; Oxford-102 (Nilsback y Zisserman 2008) que contiene sólo fotografías de flores; y en algunos casos MS COCO (Lin y col. 2014), con imágenes variadas de objetos. Es por esto que no queda claro que se puedan extender fácilmente a cualquier otro conjunto de datos. Asimismo, la complejidad y longitud de las descripciones en las fotografías representaron otra limitación al alcance de nuestra solución.

Los tres modelos antes descritos requieren, para empezar, codificar el texto de entrada. Para ello, se utilizan distintos tipos de *embeddings*. Entre ellos tenemos:

- Char-CNN-RNN (Reed y col. 2016), el cual combina redes convolucionales y recurrentes (CNN-RNN) para aprender “desde cero” un *embedding* a nivel de carácter;
- LSTM bidireccional, usado en AttnGAN para obtener vectores semánticos a partir del texto;
- BERT (Reimers y Gurevych 2019), un modelo de lenguaje contextual, esto es, que genera una representación de cada palabra a partir de otras palabras en la frase. Mientras que muchos modelos de este tipo consideran sólo el contexto que precede a la palabra de interés, BERT es bidireccional, es decir, considera el contexto tanto a la derecha como a la izquierda de la palabra.

Nuestra propuesta es utilizar *embeddings* BERT, combinándolos con la arquitectura de dos fases propuesta para StackGAN.

En la siguiente sección presentamos más detalles acerca del problema a resolver y el conjunto de datos utilizados. En la tercera sección especificamos la solución propuesta y los detalles de su implementación. Los resultados obtenidos se discuten en la cuarta sección, y ésta da paso a nuestras conclusiones en la quinta y última parte del documento.

## 2. El problema

### 2.1. Los datos

Flickr 8K, el conjunto de imágenes utilizadas, fue recopilado por Hodosh, Young y Hockenmaier (2013), quienes mediante *crowdsourcing* obtuvieron varias anotaciones (*captions*) para varios miles de imágenes provenientes de Flickr. La tarea para la que fueron seleccionados fue la creación de un modelo de recuperación de imágenes basado en oraciones. El conjunto de datos final consta de 8,092 imágenes variadas, principalmente de personas y perros realizando diversas actividades, y cinco anotaciones que las describen brevemente. En la figura 1 se muestra un ejemplo.



**Figura 1:** Una imagen de Flickr 8K con sus anotaciones:  
A group of teenage boys on a road jumping joyfully  
Four boys running and jumping  
Four kids jumping on the street with a blue car in the back  
Four young men are running on a street and jumping for joy  
Several young men jumping down the street

Tiempo después, Flickr 8K fue utilizado por un equipo de DataFlair (2020) para la tarea de generación de anotaciones a partir de imágenes. El modelo de aprendizaje profundo que utilizaron fue una conjunción de arquitecturas, particularmente una *convolutional neural network* (CNN) y una red neuronal recurrente *long short-term memory* (LSTM).

Finalmente, como se mencionó antes, el problema que nosotros intentamos resolver es el *contrario* al presentado en DataFlair: queremos generar imágenes a partir de oraciones que las describen. Este reto no es nuevo: Zhang y col. (2017) crearon StackGAN, una red generativa adversarial (GAN por sus siglas en inglés) para generar imágenes sumamente realistas. La solución que proponen sigue un proceso de dos fases: en la primera, se generan imágenes de baja resolución ( $64 \times 64$  pixeles) en las que se esbozan las figuras y colores “primitivos” de las imágenes; en la segunda fase, se toman los resultados anteriores y se refina el resultado, obteniendo imágenes de alta resolución ( $256 \times 256$ ) con mucho mayor lujo de detalle. Los conjuntos de datos que utilizaron para entrenar la red fueron por un lado CUB (Wah y col. 2011), una colección de casi 12 mil fotografías de aves, y por otro Oxford-102 (Nilsback y Zisserman 2008), un conjunto de más de 8 mil fotografías de flores. Por último, utilizaron MS COCO (Lin y col. 2014), una base de más de 80 mil imágenes de diferentes objetos con fondos variados. Los resultados obtenidos en el artículo original de StackGAN son de asombrosa calidad (ver el apéndice A.2).

En cuanto a la estructura de los datos como *input* para la red neuronal, se tienen las 8,092 imágenes en formato .jpg asociadas mediante un *hash* a cinco anotaciones en un archivo de texto con identificador igual al *hash* de su imagen más la terminación #0, #1, #2, #3, y #4. En términos prácticos, el número de imágenes que recibe la red neuronal como entrada es  $8,092 \times 5 = 40,460$ .

## 2.2. Los retos

Durante el desarrollo de una solución para el problema planteado, nos enfrentamos a diversos retos. El más grande de ellos fue posiblemente la heterogeneidad de las imágenes de Flickr 8K. Éstas, a diferencia de los datos usados por Zhang y col. (2017), contienen un número muy grande de escenarios diferentes y más complejos que los de CUB u Oxford-102. Esto puede ayudar a explicar la calidad de los resultados obtenidos, presentados en la sección 4.

Otro de los desafíos fue la implementación. Las arquitecturas necesarias para resolver problemas de este tipo son usualmente muy pesadas, por lo que recurrimos a Google Colaboratory para tener acceso a GPUs. Sin duda, el tiempo de ejecución disminuyó considerablemente (de 0.83 épocas por hora utilizando CPUs, a entre 5 y 7 épocas por hora con GPUs). Sin embargo, al tratarse de dicha plataforma, había otra importante limitante: el tiempo máximo de actividad de una instancia de Colab es de 12 horas. La solución particular a este reto fue la de guardar las imágenes de los modelos probados cada cierto número de épocas (100 épocas la mayoría de las veces), para poder cargarlas si la instancia era interrumpida y continuar el entrenamiento desde ese punto.

## 3. La solución

Nuestro mejor método fue una StackGAN con *embeddings* construidos con BERT (*Bidirectional Encoder Representations from Transformers*). Como las GANs, nuestro modelo consta de un generador y un discriminador. La diferencia entre las GANs y StackGAN es que ésta última consta de dos etapas. En la primera de ellas se generan imágenes de baja resolución ( $64 \times 64$  pixeles) y en la segunda la resolución es mejor ( $256 \times 256$  pixeles). Las figuras 2 y 3 muestran las arquitecturas de ambas etapas.

Cabe mencionar que el paper original, *Stackgan: Text to photo-realistic image synthesis with stacked generative adversarial networks* (Zhang y col. 2017), utiliza *embeddings* generados con un modelo char-CNN-RNN (Reed y col. 2016). Como fue mencionado previamente, el modelo char-CNN-RNN hace uso de redes convolucionales y recurrentes para generar un *embedding* a nivel de carácter. Sin embargo, nuestra propuesta es utilizar *embeddings* a nivel de palabra que tomen en cuenta el contexto de la oración. Es decir, procesar las palabras en el contexto de una oración, en lugar de palabra por palabra o carácter por carácter.

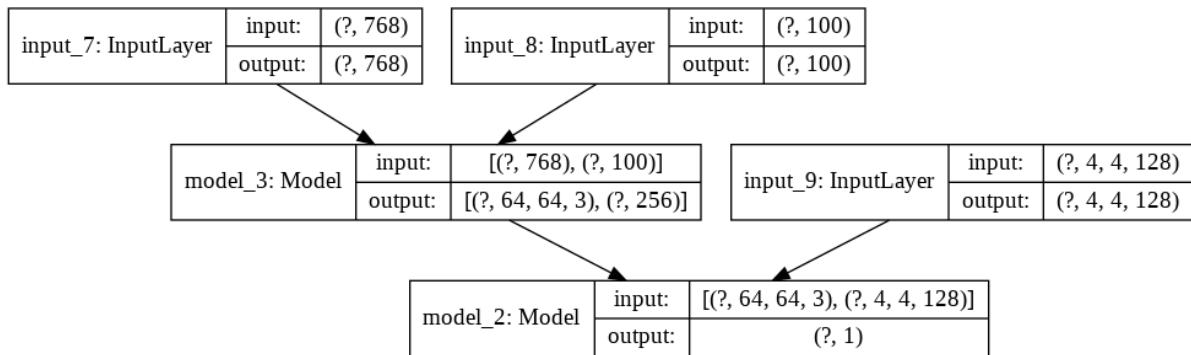
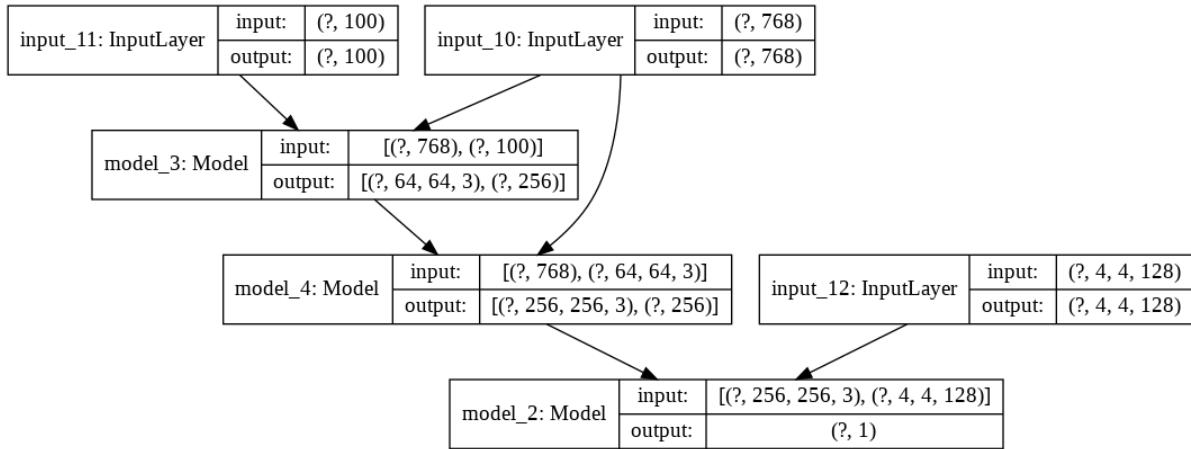


Figura 2: Arquitectura de StackGAN Stage I



**Figura 3:** Arquitectura de StackGAN Stage II

### 3.1. División de entrenamiento y validación

Aprovechando la estructura del nombre de las imágenes, se utilizaron todas las imágenes con terminación #1, #2, #3 y #4 para el conjunto de entrenamiento y las imágenes #0 para el conjunto de prueba y validación. Este a su vez se dividió el 90 % de validación y 10 % de prueba. Es evidente que el conjunto de prueba es mucho menor que el de validación; sin embargo, la decisión se tomó al ver que no había riesgo de sobreentrenamiento, ya que las imágenes no eran claras ni en el entrenamiento ni en las pruebas. El código usado puede ser consultado en el apéndice F.1. Los tamaños de cada conjunto se muestran en la tabla 1.

**Tabla 1:** Tamaño del conjunto de datos

Conjunto	Número de imágenes
Entrenamiento	32,364
Validación	7,281
Prueba	810
Total	40,455

Es importante mencionar que de las 40,460 imágenes posibles solo utilizamos 40,455. Es decir, descartamos cinco nombres de imágenes porque no encontramos la imagen correspondiente en el conjunto de datos. Estos nombres de imágenes tenían terminación “.jpg.1#”.

### 3.2. Representación latente o *embedding*

BERT fue desarrollado por Google en 2018 y es una técnica de procesamiento de lenguaje natural que se entrenó con el Toronto Book Corpus y con Wikipedia. Es importante destacar que BERT sí utiliza el contexto de las palabras (Reimers y Gurevych 2019), pues a diferencia de muchos modelos de este tipo que consideran sólo el contexto que precede a la palabra de interés, BERT es bidireccional, es decir, considera el contexto tanto a la izquierda como a la derecha de la palabra. Existen varios modelos preentrenados de BERT; se decidió tomar “bert-base-nli-mean-tokens” por su buen desempeño histórico (código en el apéndice F.2).<sup>1</sup>

1. Inicialmente se intentó utilizar *transfer learning* para generar el vector de características de las imágenes. Utilizando la red preentrenada de Google, *Xception*, se extrajeron las características y se intentó construir una StackGAN para reconstruir las imágenes. La idea era después construir una red que mapeara alguna representación latente con el vector de características. En el apéndice C se muestran dos imágenes generadas en el Stage II de la StackGAN.

Para evaluar el desempeño de la representación latente, utilizando la división mencionada anteriormente, se realizaron búsquedas semánticas. Se construyó el *corpus* con las frases de entrenamiento y se probó con las frases de validación. Es decir, se usaron las frases de validación para encontrar las frases de entrenamiento más cercanas. La distancia entre las frases se calculó con la distancia coseno. Recordemos que la distancia coseno no toma en cuenta la magnitud de los vectores, sino solamente su dirección (ver detalle en el apéndice B). Algunos ejemplos de la búsqueda semántica se muestran en las tablas 2 y 3. En ellas se puede ver que las frases de entrenamiento con menor distancia a la frase de prueba son muy similares semánticamente. Si entrenamos el modelo con los *embeddings* de las frases de entrenamiento, entonces, en teoría cuando llegue una frase nueva similar a otras frases, las imágenes deberían de ser similares.

**Tabla 2:** Frase de prueba: “A man is cycling on the road”

Frases del entrenamiento	Distancia coseno
A man riding his bike in traffic	0.057
A man riding a bike near traffic	0.062
A man holds an object with his hand while riding his bike down the street	0.074
Man riding a bicycle down a road with clouds overhead	0.075

**Tabla 3:** Frase de prueba: “A child painting a picture”

Frases del entrenamiento	Distancia coseno
A child painting on a piece of spinning paper	0.092
A young girl painting a picture	0.110
A child paints with different colors using brushes	0.11
A child plays with paint	0.151

## 4. Experimentos y resultados

Con el objetivo de contar con distintas alternativas, a lo largo de todo proceso para la generación de imágenes a partir de texto, hicimos variar algunos aspectos importantes de las redes utilizadas. En primer lugar, consideramos cuatro *embeddings* con diferentes niveles de complejidad. Asimismo, probamos varias combinaciones de hiper parámetros, tales como el optimizador utilizado, el tamaño de los *batches*, la tasa de aprendizaje y la función de activación de la última capa. En los siguientes apartados se detallan estos diferentes modelos probados.

### 4.1. Embeddings

Además de BERT, experimentamos con otras representaciones latentes: un *tokenizer simple*, un *one-hot encoder* y *skipgrams*:

- **Tokenizer simple:** los datos de Flickr 8K incluyen, además de las anotaciones *verbatim*, un archivo de texto alternativo con la lematización de las anotaciones. Lo que se hizo fue tomar las anotaciones lematizadas y eliminar las *stopwords*, signos de puntuación y contracciones. Después se le asignó un número a cada palabra del documento y se construyó un vector concatenando los números de cada palabra, agregando ceros al inicio para que todos los vectores tuvieran la misma longitud.

- **One hot encoder:** utilizando el vector generado por el *tokenizer* simple, se construyó un vector ralo de longitud 5,748 (igual al número de palabras en el diccionario). El vector tomaba el valor 1 en las posiciones correspondientes a los números de las palabras de la oración. Por ejemplo, si la representación de la palabra “perro” fuera 3 y la de “corre” fuera 5, entonces “perro corre” sería [00101].
- **Skipgram:** partiendo también aquí de los lemas procesados con el *tokenizer* simple, se codificaron con un *encoder* Sent2Vec (Kiros y col. 2015), en el cual las primeras 2,400 dimensiones pertenecían al modelo *uni-skip* y las últimas 2,400 dimensiones al modelo *bi-skip* (unigramas y bigramas, respectivamente). El código se dispone en el apéndice F3.

Como se ve en las imágenes resultantes del apéndice D.1, el *embedding* que logra capturar más colores y formas diferentes fue BERT. La figura 18 en el apéndice E confirma que la menor pérdida por parte del generador fue con BERT, razón por la cual se decidió continuar con la exploración de parámetros utilizándolo.

## 4.2. Hiper-parámetros

Exploramos los siguientes hiper parámetros:

- **Batch size:** Se experimentó con tamaños de *batch* de 32, 64, 128 y 512. Como se ve en el apéndice D.2, el *batch* de 128 logró capturar más colores y formas diferentes. Sin embargo, en la figura 19 en el apéndice E, es interesante notar que la pérdida del generador es menor con un tamaño de *batch* de 64. También cabe remarcar que la variación de la pérdida es más grande con *batches* de mayor tamaño.
- **Optimizador:** Experimentamos con Adadelta (usa *exponential smoothing* para restringir historia a una ventana temporal), RMSprop (Adadelta con tasa de aprendizaje de 0.001 y gamma de 0.9) y Adam (Adadelta con momentum). Como se ve en el apéndice D.3, el optimizador con mejores resultados visuales fue Adam, y como se ve en la figura 20 del apéndice E, las pérdidas con Adam son menores. Es interesante notar la gran diferencia entre las pérdidas de RMSprop y Adadelta; esto refuerza que los parámetros óptimos para Adadelta son los de RMSprop y la pequeña diferencia entre Adam y RMSprop quizás pueda ser explicada por el momentum de Adam.
- **Tasa de aprendizaje:** Al ver que Adam era la mejor opción de los optimizadores, decidimos explorar diferentes tasas de aprendizaje para el mismo: 0.00002, 0.0002, 0.0001 y 0.002. Como se ve en las imágenes generadas con cada tasa (ver apéndice D.4) la que obtuvo mejores resultados fue 0.002. Sin embargo, es interesante notar que en la figura 20, del apéndice E, la tasa de 0.00002 parece tener menores pérdidas que las otras tasas.
- **Función de activación:** Otro aspecto que hicimos variar fue la función de activación en la última capa. Exploramos las con ReLU (Rectified Linear Units), Leaky ReLU (ayuda a corregir el *vanishing gradient* con un gradiente pequeño en vez de cero cuando la unidad no está activa) y Selu (Scaled Exponential Linear Units). Como se ve en el apéndice D.5, las mejores funciones fueron ReLU y LeakyReLU. Esto también lo confirma la figura 22, del apéndice E. Es interesante notar que aunque Selu tiene menor variación en la pérdida del generador, ReLU y LeakyReLU ambas alcanzan menores pérdidas.
- **Número de épocas:** Finalmente, corrimos 800 iteraciones con los mejores hiper parámetros discutidos arriba. Sin embargo, como se ve en el apéndice D.6, a partir de la época 490 las imágenes fueron “planas”, predominantemente de un solo color.

#### 4.3. Generación de imágenes sintéticas

Como ya se dijo, elegimos el modelo que utiliza BERT con *mini batches* de tamaño 128. El tamaño del error, aunque no fue el menor de entre todos los modelos probados, sí fue de los más pequeños. El motivo por el cual elegimos esta variante fue que las imágenes generadas son cualitativamente mejores que las de otros modelos similares en error.

Del conjunto de prueba, algunas imágenes muestran resultados “exitosos” para anotaciones que tienen en común la palabra *sky* (figura 4a, 4b y 4c) o la palabra *water* (figura 5a, 5b y 5c). Los resultados observados no representan escenas discernibles, pero sí se aprecia que la red logró aprender algunas cosas, como que el cielo suele estar arriba y el agua abajo. Como se mencionó en la sección 2, una posible explicación para estos resultados es que StackGAN fue desarrollado con conjuntos de imágenes más homogéneos que los de Flickr 8K, por lo que a la red le cuesta mucho más trabajo reconstruir imágenes tan variadas.

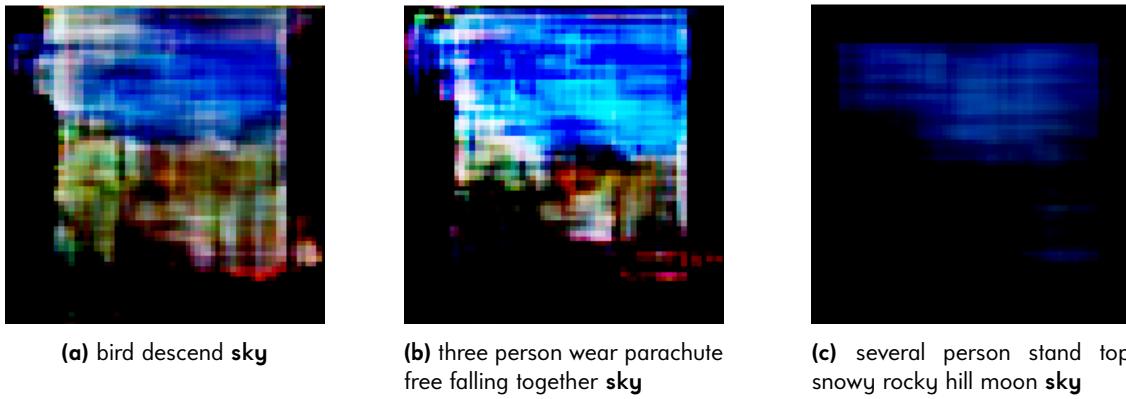


Figura 4: Imágenes del conjunto de entrenamiento con la palabra *sky*

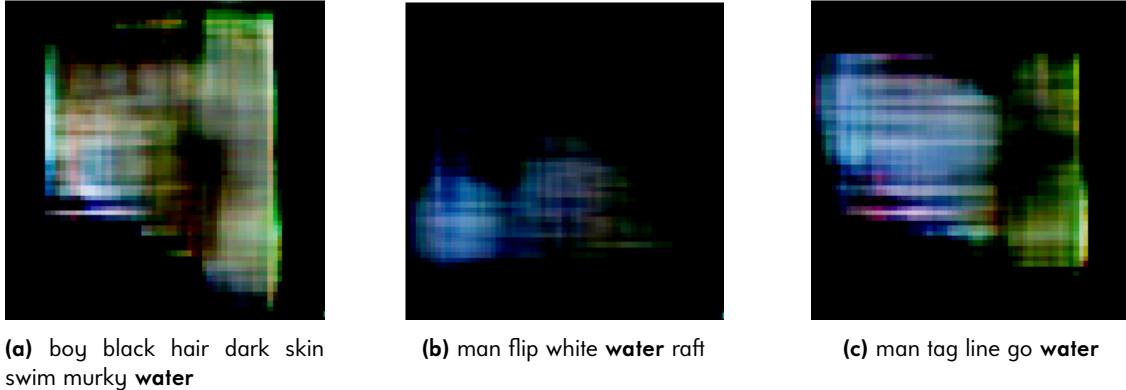
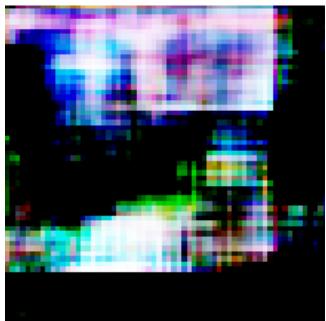


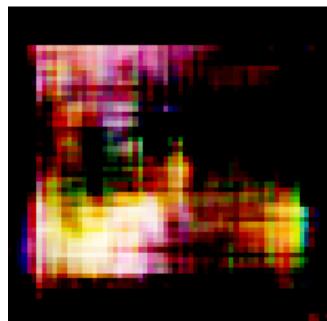
Figura 5: Imágenes del conjunto de prueba con la palabra *water*

#### 4.4. Generación de imágenes *random*

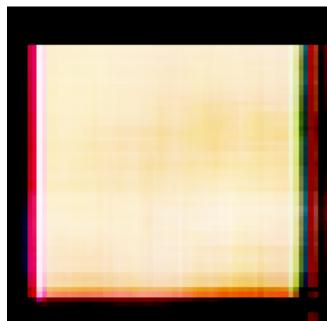
Como una forma adicional de poner a prueba nuestra red, generamos algunas imágenes a partir de texto completamente “nuevo”, es decir, texto que no estaba originalmente en el conjunto de datos. Abajo se muestran imágenes generadas con oraciones *random* que resultaron interesantes. La imagen en 6a permite ver dos áreas azules, una superior, correspondiente al cielo, y otra inferior, correspondiente a un río. La sola palabra “rojo” (figura 6b). Por último, es curioso ver que utilizar palabras que no están dentro de los datos originales genera una imagen (figura 6c).



(a) White clouds above river



(b) Red

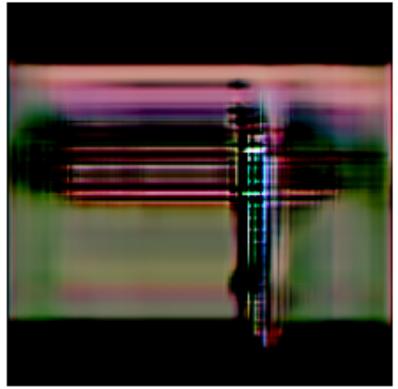


(c) Juan, Ely and Paola finishing the project

**Figura 6:** Imágenes de calidad de Stage I generadas con texto inventado por los autores

#### 4.5. Stage II

Ya hemos explicado que la segunda etapa de una StackGAN busca refinar detalles de las imágenes que se generan en la primera etapa y obtener resultados en mayor resolución. Ante los resultados que obtuvimos en baja resolución, decidimos construir el Stage II utilizando los mismos hiperparámetros del Stage I (*batches* de tamaño 128, Adam con *learning rate* de 0.0002, LeakyReLU y BERT así como, evidentemente, los pesos del generador obtenidos en el Stage I. Las figuras 7 y 8 muestran algunos resultados.

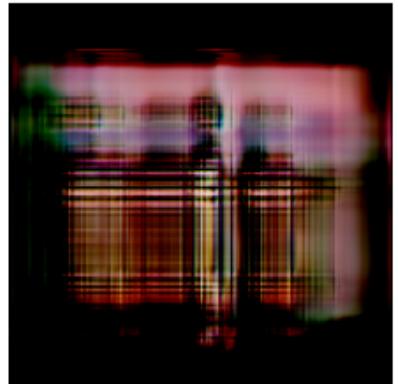


**Figura 7:** black dog jumps into the water holding stick in the mouth

#### 5. Conclusión

En resumen, la solución presentada fue una estructura basada en StackGAN Stage-I utilizando BERT para generar los *embeddings*. El mejor modelo utilizó Adam como optimizador con una tasa de aprendizaje de 0.0002, LeakyReLU como función de activación y un tamaño de *batch* de 128. Los resultados, fueron satisfactorios ya que el modelo logró aprender a reconocer los márgenes negros de las imágenes (de forma horizontal y vertical) y palabras como cielo y agua.

Dado que pasar estas imágenes a la arquitectura Stage-II de StackGAN no dio los resultados esperados, queda la posibilidad de experimentar conectar nuestra red de Stage-I con etapas posteriores de arquitecturas como AttnGAN o MirrorGAN, que afinan detalles basándose en las palabras y la semántica de las descripciones. Otros experimentos futuros pueden incluir el aumento de los datos, por ejemplo, reflejando las imágenes o creando descripciones extendidas concatenando las oraciones para dar más información sobre la imagen.



**Figura 8:** little girl playing in a playground

Las limitaciones del modelo es que todavía no se logran distinguir con claridad las formas concretas como niño, niña, mujer o objetos. Lo que se ven son sombras y manchas.

Durante este proyecto, la mayor lección aprendida fue que hay una multiplicidad de soluciones existentes y muchas por descubrir. Aprendimos que problemas retadores como el aquí presentado requieren mucha creatividad y perseverancia.

## Referencias

- DataFlair. 2020. «Python based Project – Learn to Build Image Caption Generator with CNN & LSTM». *DataFlair Blog*. <https://data-flair.training/blogs/python-based-project-image-caption-generator-cnn/>.
- Hodosh, Micah, Peter Young y Julia Hockenmaier. 2013. «Framing Image Description as a Ranking Task: Data, Models and Evaluation Metrics». *Journal of Artificial Intelligence Research* 47, 853-899. <https://www.jair.org/index.php/jair/article/view/10833/25854>.
- Kiros, Ryan, Yukun Zhu, Ruslan Salakhutdinov, Richard S Zemel, Antonio Torralba, Raquel Urtasun y Sanja Fidler. 2015. «Skip-Thought Vectors». *arXiv preprint arXiv:1506.06726*.
- Lin, Tsung-Yi, Michael Maire, Serge Belongie, Lubomir Bourdev, Ross Girshick, James Hays, Pietro Perona, Deva Ramanan, C. Lawrence Zitnick y Piotr Dollár. 2014. *Microsoft COCO: Common Objects in Context*. *arXiv: 1405.0312 [cs.CV]*.
- Nilsback, M., y A. Zisserman. 2008. «Automated Flower Classification over a Large Number of Classes». En *2008 Sixth Indian Conference on Computer Vision, Graphics Image Processing*, 722-729.
- Qiao, Tingting, Jing Zhang, Duanqing Xu y Dacheng Tao. 2019. «MirrorGAN: Learning Text-to-image Generation by Redescription». *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*.
- Reed, Scott, Zeynep Akata, Honglak Lee y Bernt Schiele. 2016. «Learning deep representations of fine-grained visual descriptions». En *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 49-58.
- Reimers, Nils, e Iryna Gurevych. 2019. «Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks». En *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics. <http://arxiv.org/abs/1908.10084>.
- Wah, C., S. Branson, P. Welinder, P. Perona y S. Belongie. 2011. *The Caltech-UCSD Birds-200-2011 Dataset*. Informe técnico CNS-TR-2011-001. California Institute of Technology.
- Xu, Tao, Pengchuan Zhang, Qiuyuan Huang, Han Zhang, Zhe Gan, Xiaolei Huang y Xiaodong He. 2017. «Attn-GAN: Fine-Grained Text to Image Generation with Attentional Generative Adversarial Networks». *CoRR abs/1711.10485*. *arXiv: 1711.10485*. <http://arxiv.org/abs/1711.10485>.
- Zhang, Han, Tao Xu, Hongsheng Li, Shaoting Zhang, Xiaogang Wang, Xiaolei Huang y Dimitris N Metaxas. 2017. «Stackgan: Text to photo-realistic image synthesis with stacked generative adversarial networks». En *Proceedings of the IEEE international conference on computer vision*, 5907-5915.

## A. StackGAN

### A.1. Arquitectura

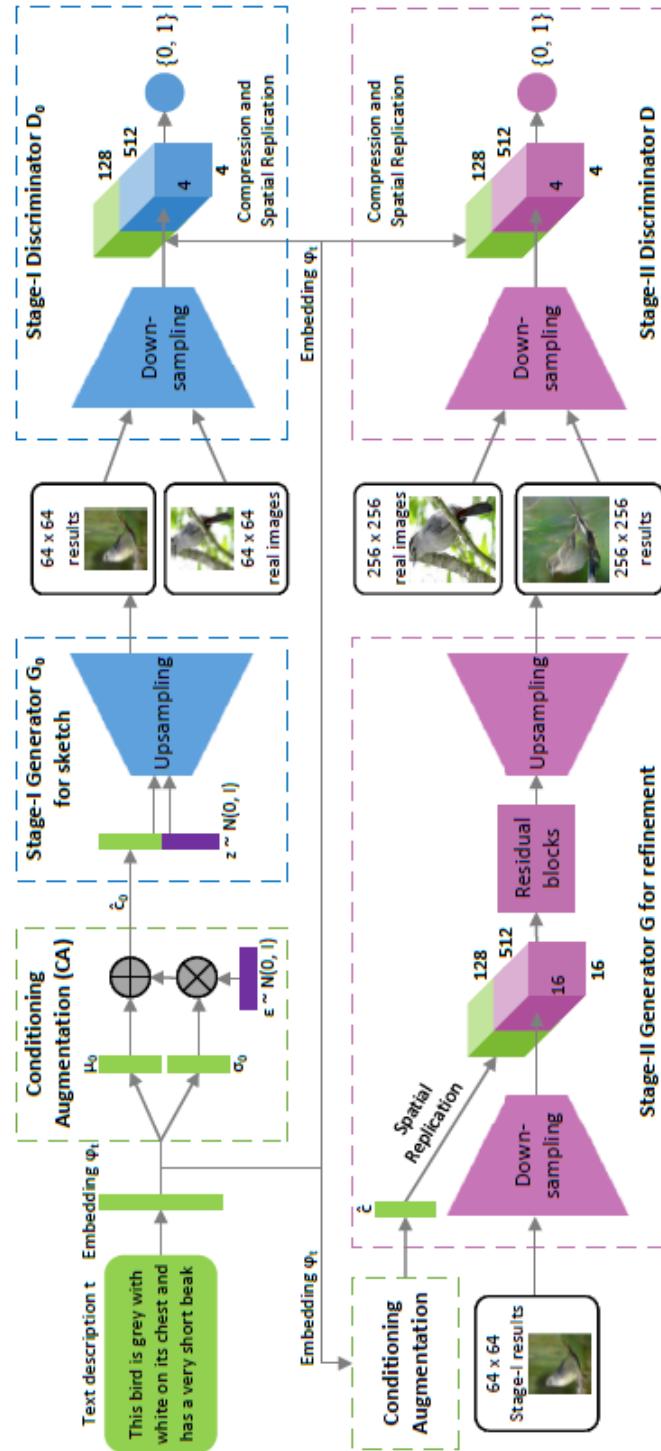


Figura 9: Arquitectura de StackGAN

## A.2. Resultados del artículo original



**Figura 10:** Algunos ejemplos de los resultados obtenidos

## B. Definición de la distancia coseno

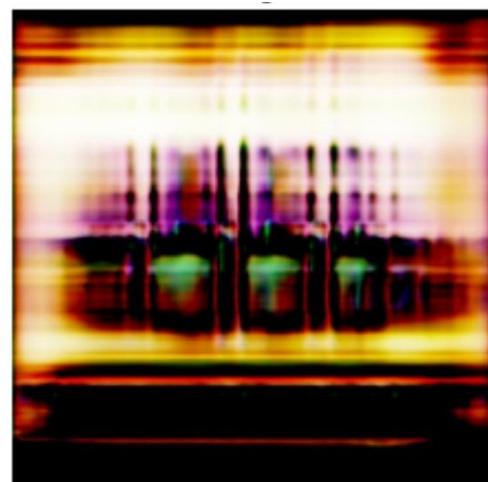
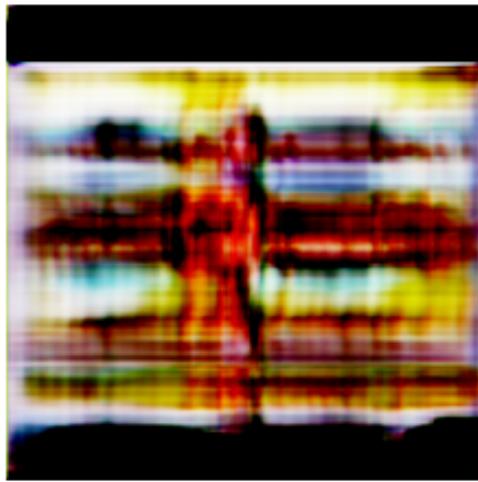
La similitud coseno se define como

$$sim_{cos}(x, y) = \frac{\langle x, y \rangle}{\|x\| \|y\|} = \cos(\theta)$$

donde  $\langle x, y \rangle = \sum_{i=1}^p x_i y_i$  es el producto punto de  $x$  y  $y$ . Esta cantidad es igual al coseno del ángulo entre los vectores  $x$  y  $y$ .

La distancia coseno es entonces  $d_{cos}(x, y) = 1 - sim_{cos}(x, y)$ .

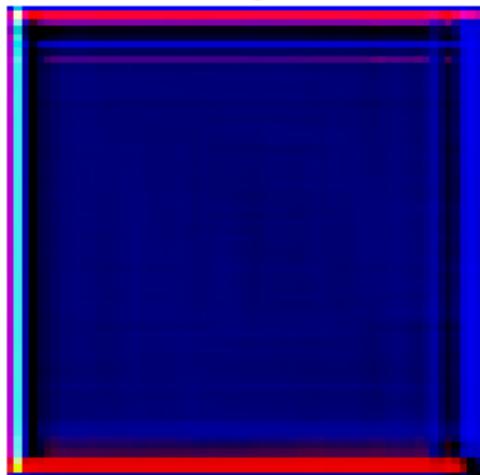
## C. Imágenes generadas con el vector de características de Xception



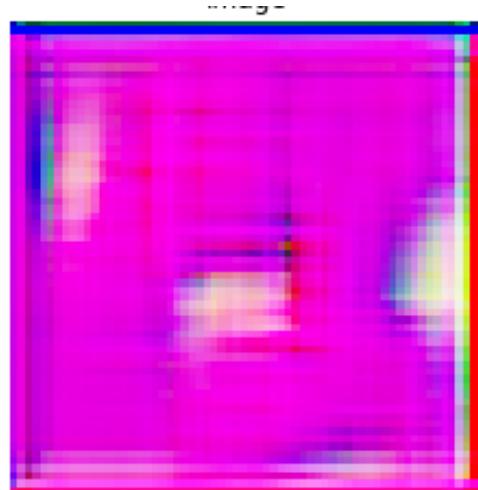
**Figura 11:** Ejemplos de imágenes generadas con el embedding de Xception. Es interesante que una de las imágenes sí capturó el margen negro de algunas de las fotos.

## D. Comparaciones

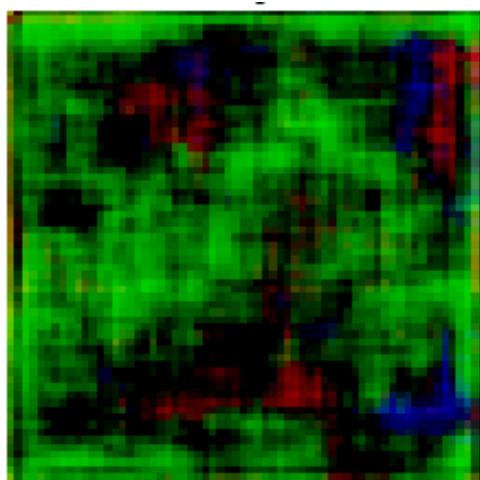
### D.1. Comparación de *embeddings*



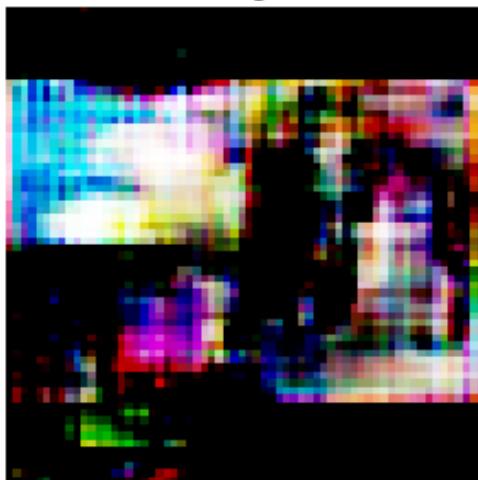
(a) Tokenizer simple



(b) One hot encoding



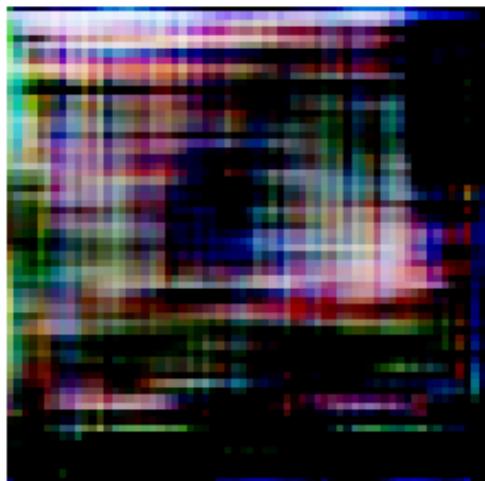
(c) Skipgram



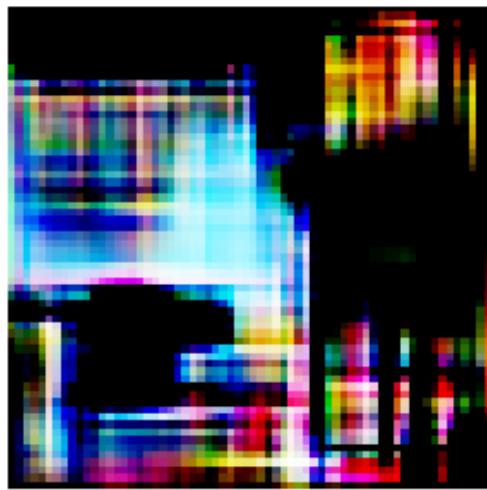
(d) Bert

**Figura 12:** Comparación de *embeddings*

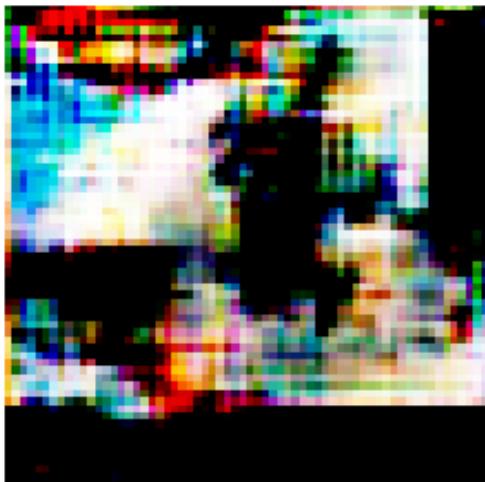
## D.2. Comparación de tamaños de *batch*



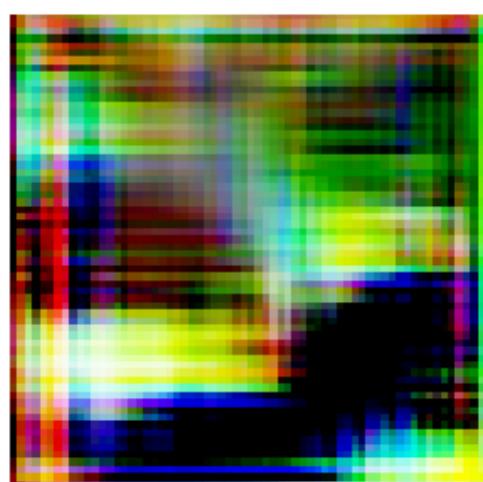
(a) 32



(b) 64



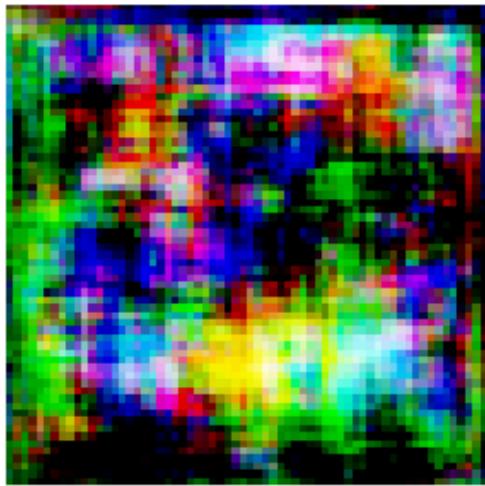
(c) 128



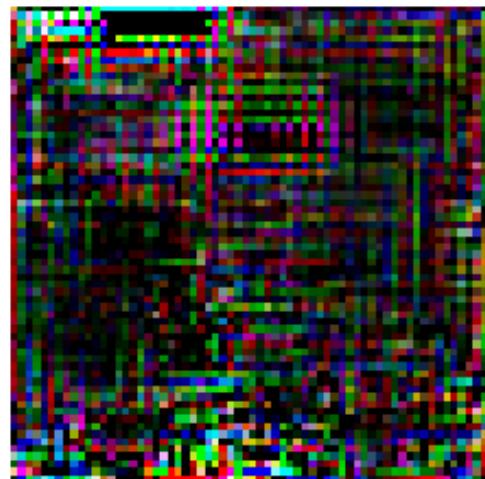
(d) 512

**Figura 13:** Comparación de tamaños de *batch*

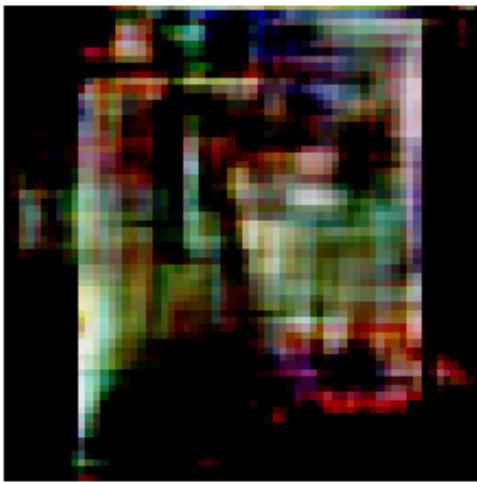
### D.3. Comparación de optimizadores



(a) Adadelta



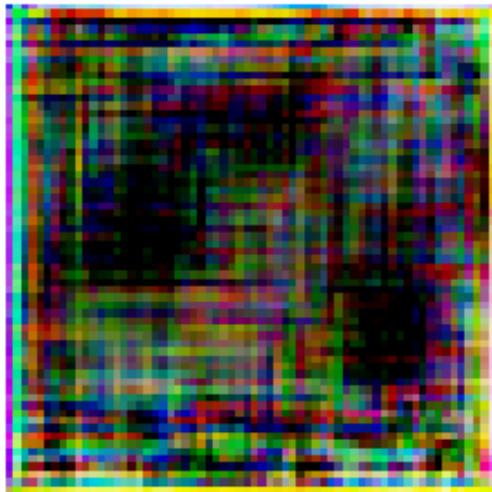
(b) RMSprop



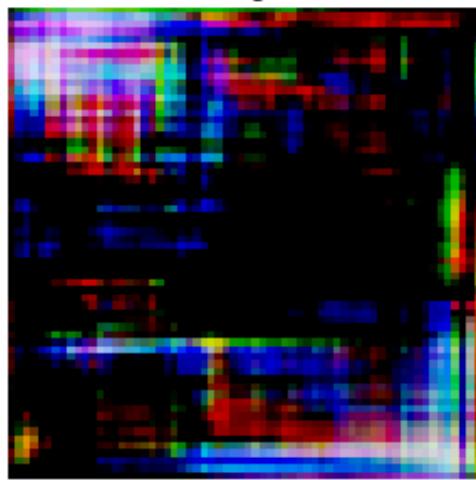
(c) Adam

**Figura 14:** Comparación de optimizadores

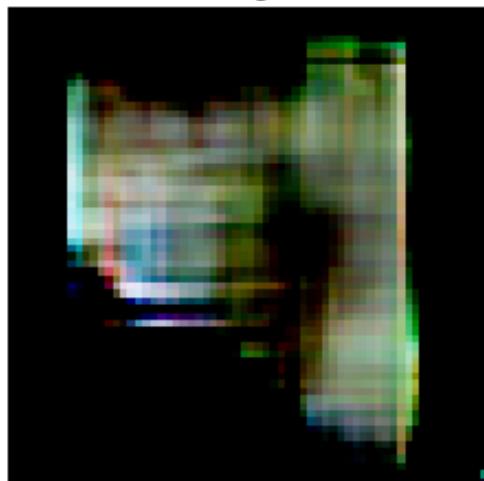
#### D.4. Comparación de tasa de aprendizaje



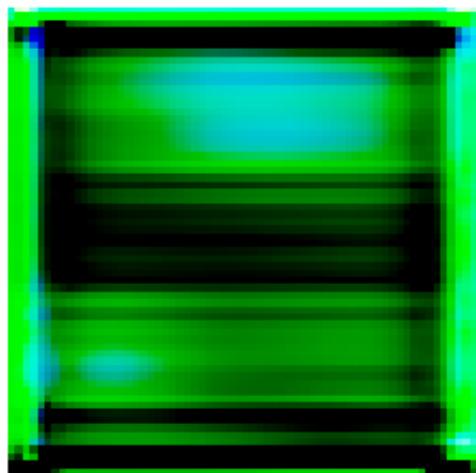
(a) Mas chico: 0.0002



(b) 0.0001



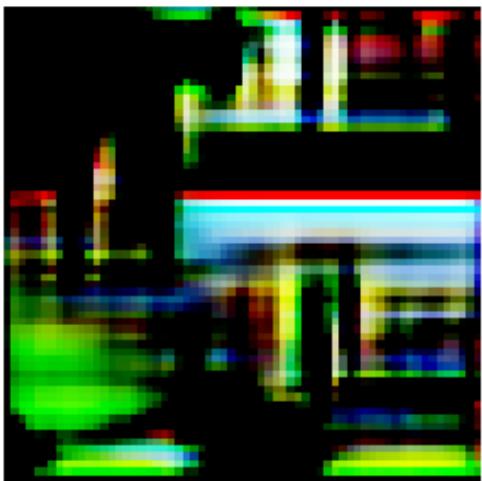
(c) 0.0002



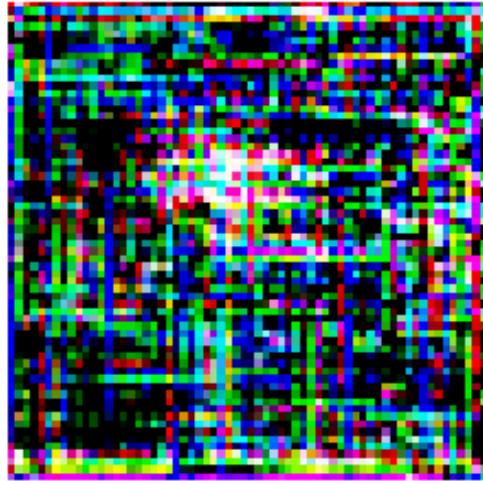
(d) Mas grande: 0.002

**Figura 15:** Comparación de tasa de aprendizaje

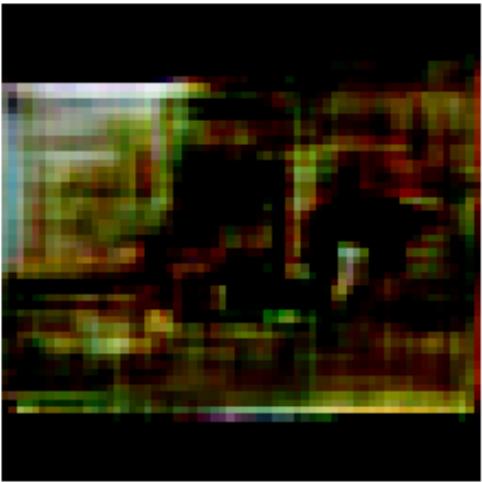
## D.5. Comparación de funciones de activación



(a) ReLU



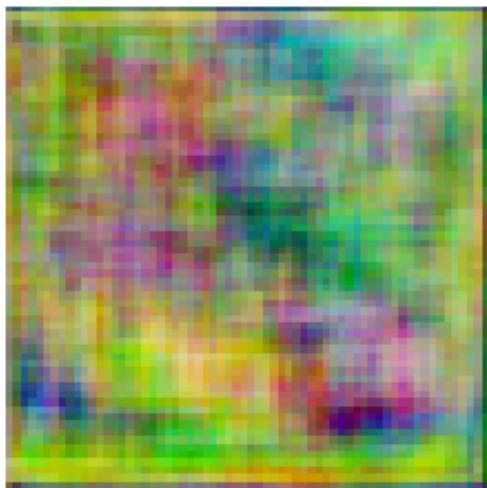
(b) SeLU



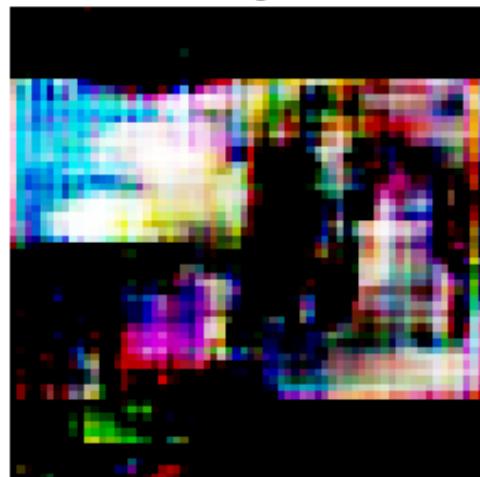
(c) LeakyReLU

**Figura 16:** Comparación de funciones de activación

## D.6. Comparación de épocas



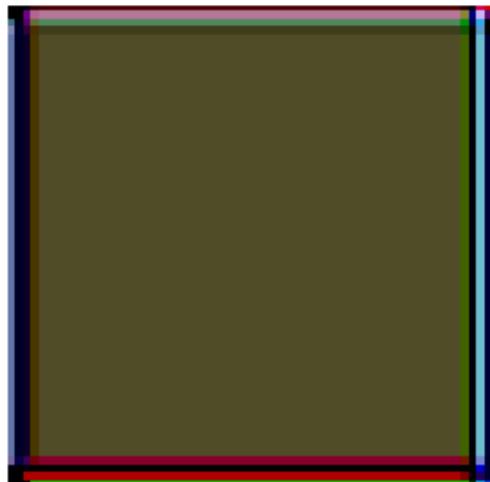
(a) 0



(b) 390



(c) 480

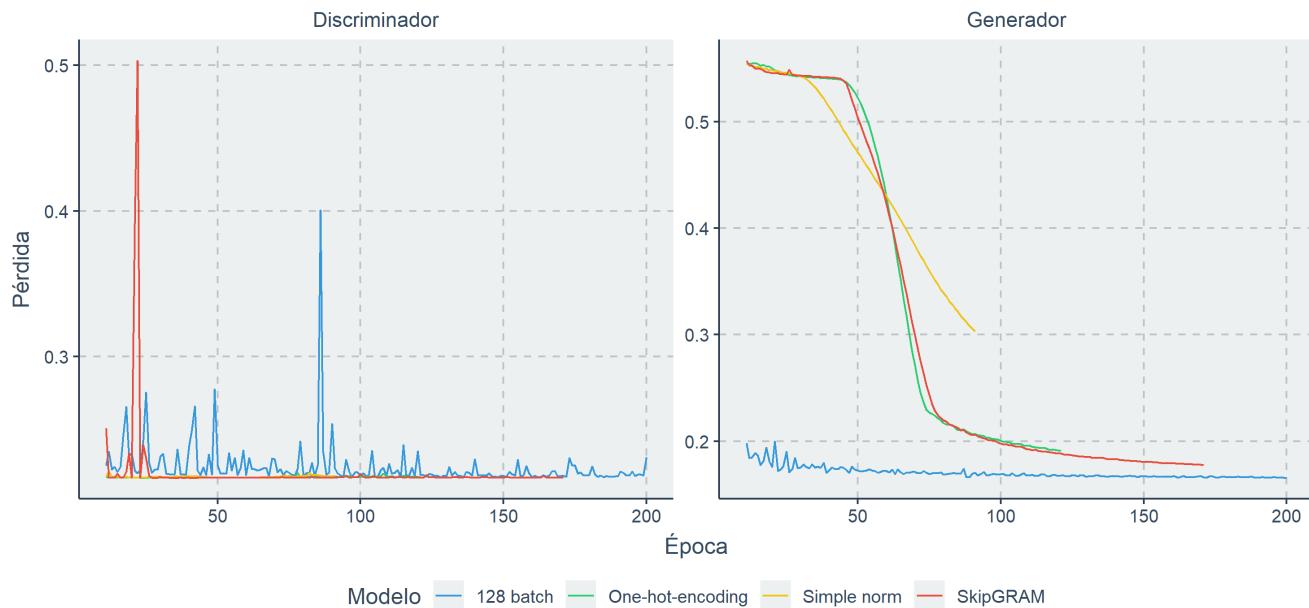


(d) 490

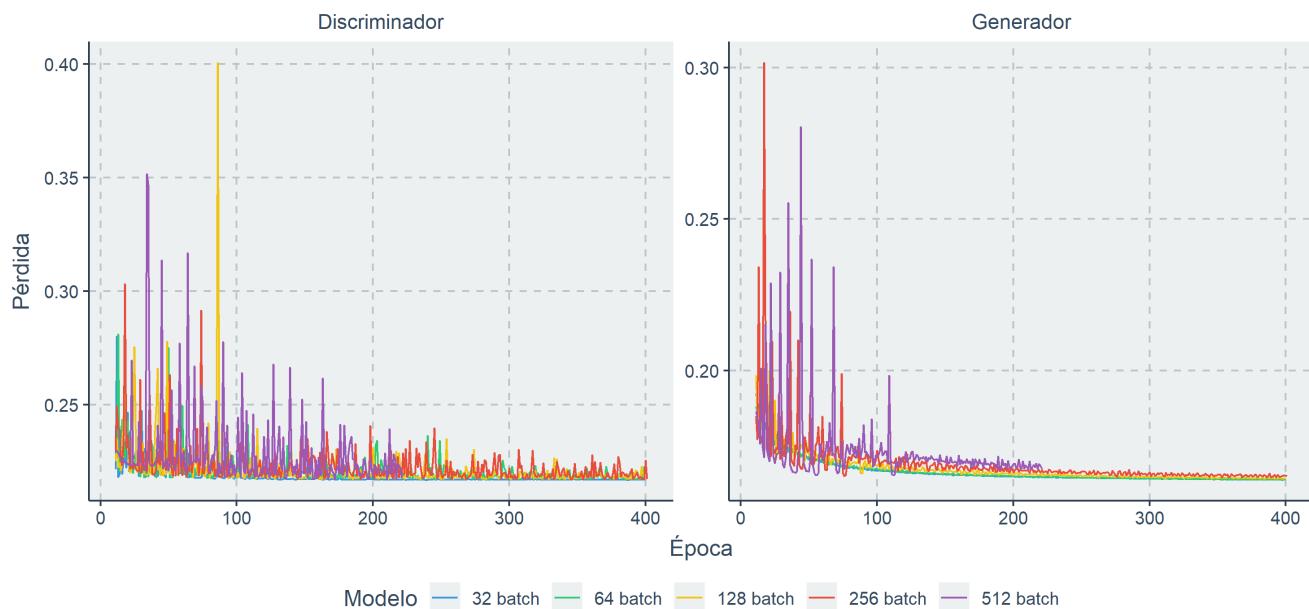
**Figura 17:** Comparación de épocas

## E. Gráficas de pérdidas

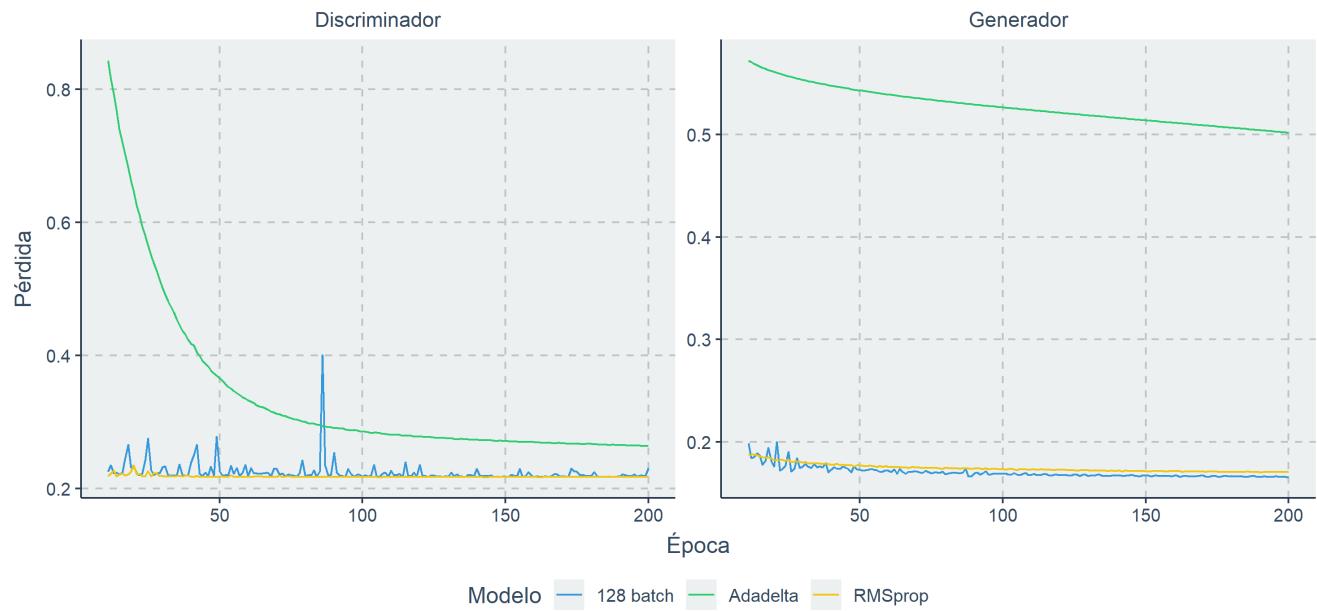
En las siguientes gráficas, el modelo “128 batch” es el *baseline*. Se corrió con un tamaño de *batch* de 128 con el optimizador Adam, una tasa de aprendizaje de 0.002 y la función de activación LeakyReLU.



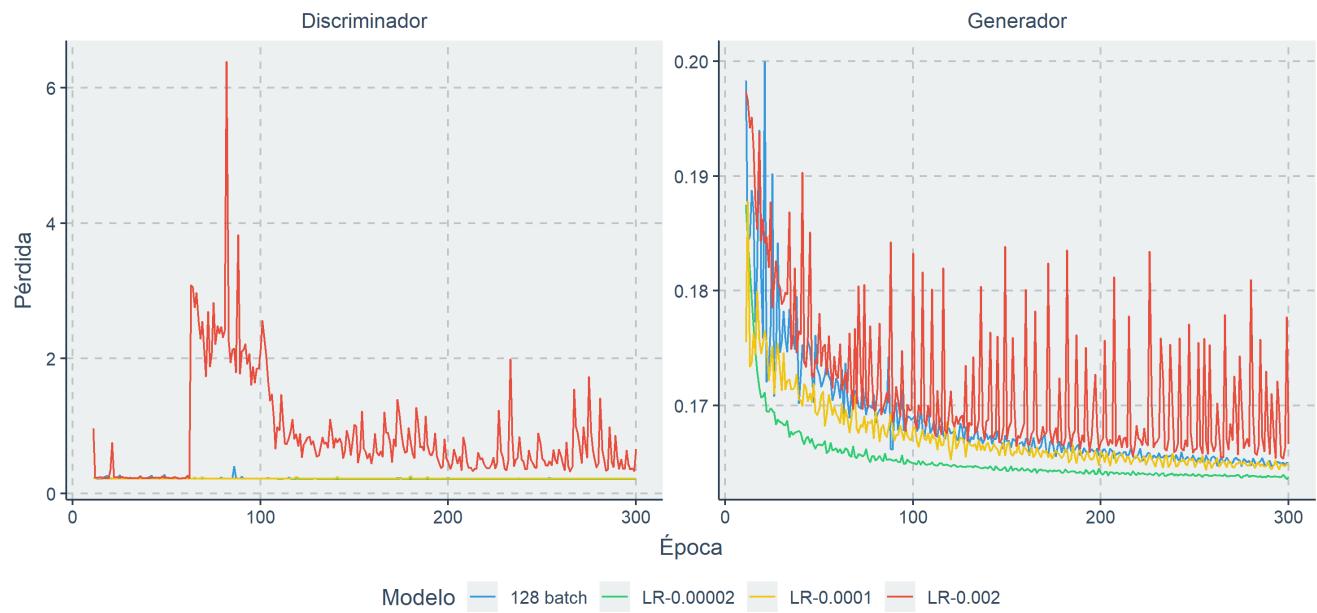
**Figura 18:** Pérdidas con diferentes embeddings: “128 batch” utiliza BERT como embedding



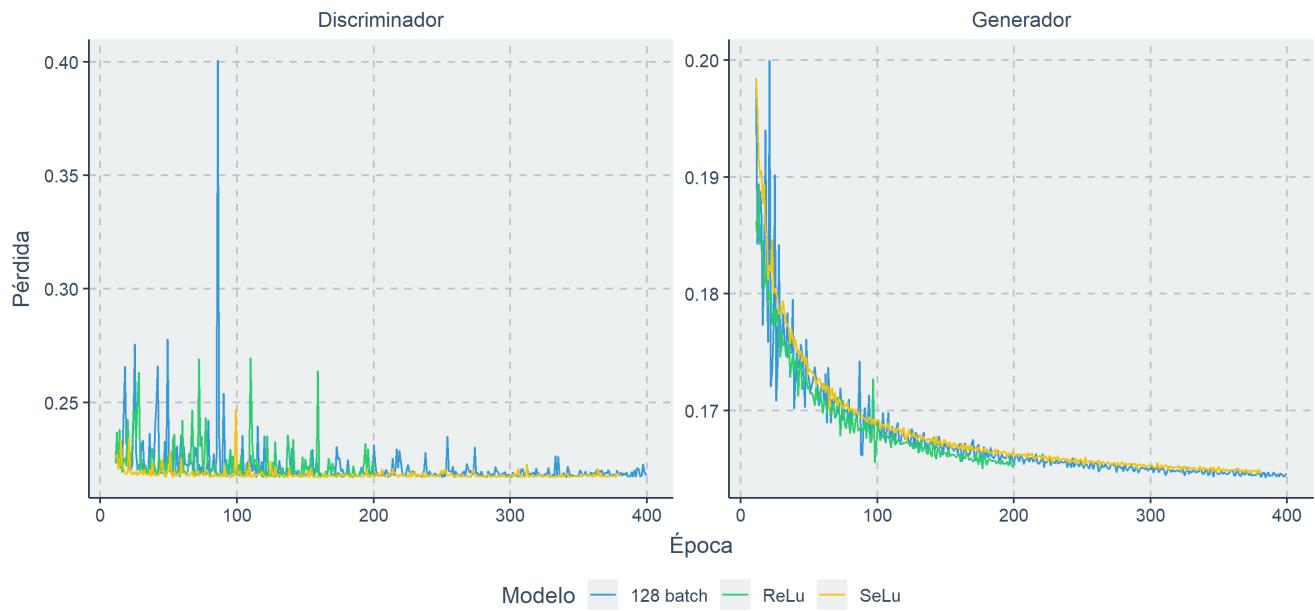
**Figura 19:** Pérdidas con diferentes tamaños de batch



**Figura 20:** Pérdidas con diferentes optimizadores: : “128 batch” utiliza Adam como optimizador



**Figura 21:** Pérdidas con diferentes tasas de aprendizaje: : “128 batch” utiliza una pérdida de 0.0002, “Loss function MORE” utiliza una pérdida de 0.00002



**Figura 22:** Pérdidas con diferentes tasas de aprendizaje: : “128 batch” utiliza LeakyReLU

## F. Código

### F.1. División en conjuntos de entrenamiento, prueba y validación

```

mask_test = (df_corr['ID'].str.endswith('0')).tolist()
mask_train = (~df_corr['ID'].str.endswith('0')).tolist()
df_test_val = df_corr[mask_test]
df_train = df_corr[mask_train]

test_size = len(df_test_val)
val_size = ceil(len(df_test_val)*0.9)

df_val = df_test_val[:val_size]
df_test = df_test_val[val_size:]

```

### F.2. Creación de *embeddings* BERT

```

from sentence_transformers import SentenceTransformer
model = SentenceTransformer('bert-base-nli-mean-tokens')
train_embeddings = model.encode(token_train)

```

### F.3. Creación de *embeddings* Skipgrams

```

import nltk
nltk.download('punkt')
import numpy as np
from skip_thoughts import configuration

```

```

from skip_thoughts import encoder_manager

VOCAB_FILE = "skip_thoughts/pretrained/skip_thoughts_uni_2017_02_02/vocab.txt"
EMBEDDING_MATRIX_FILE =
"skip_thoughts/pretrained/skip_thoughts_uni_2017_02_02/embeddings.npy"
CHECKPOINT_PATH =
"skip_thoughts/pretrained/skip_thoughts_uni_2017_02_02/model.ckpt-501424"

encoder = encoder_manager.EncoderManager()
encoder.load_model(configuration.model_config(bidirectional_encoder=False),
                    vocabulary_file=VOCAB_FILE,
                    embedding_matrix_file=EMBEDDING_MATRIX_FILE,
                    checkpoint_path=CHECKPOINT_PATH)

encodings_train = encoder.encode(synops_train)

encoder.load_model(configuration.model_config(bidirectional_encoder=True),
                    vocabulary_file=VOCAB_FILE,
                    embedding_matrix_file=EMBEDDING_MATRIX_FILE,
                    checkpoint_path=CHECKPOINT_PATH)

bi_encodings_train = encoder.encode(synops_train)

train_encodings = np.concatenate([encodings_train, bi_encodings_train], axis=-1)

```