HostelNet: Clasificador de escenas de hotelería basado en aprendizaje profundo.

Leandro Bugnon (lbugnon@sinc.unl.edu.ar)

20 de julio de 2019

Este informe detalla el análisis y criterios tenidos en cuenta para la solución propuesta al desafío de Despegar en Metadata. El código fuente y el detalle para reproducir los resultados se encuentran disponibles en `http://github.com/lbugnon/hostelnet`.

1. Análisis de los datos

Las imágenes provistas en los conjuntos de entrenamiento representan datos complejos, con gran variedad de tipos de escenas y semánticas, por lo que realizar un procesamiento avanzado de tipo diseño y extracción de características no parece en principio apropiado. Se resolvieron en primer lugar problemas de los archivos, por un lado algunas imágenes contenían el canal *alpha* y otras tenían parte de la imagen corrupta, posiblemente debido al proceso de descarga de las imágenes de la web al momento de crear el conjunto de datos. En este análisis, no se descartó ninguna imagen sino que se resolvieron las incongruencias agregando ceros (o *espacio en blanco*). La resolución de las imágenes es también variable; se tomo un criterio de compromiso entre la calidad de las imágenes y el costo computacional y se normalizaron a tamaño 350×350, posteriormente normalizando los valores de intensidad entre 0 y 1.

A partir de la inspección de las imágenes, se encontró que (lógicamente) están sacadas desde diferentes ángulos, con diferente tipo de iluminación, importantes variaciones de color, etc. Teniendo en cuenta esto, se implementó un método básico de aumentación de datos para entrenar el clasificador que se describe a continuación. Este proceso genera nuevas imágenes a partir de rotaciones, inversiones espejadas y variaciones en color e intensidad. Estas variaciones aleatorias de los datos se generan *sobre la marcha*, a medida que se entrena el modelo.

Se encontró también que la distribución de las clases es altamente desbalanceada, siendo la clase *Bedroom* la más representada con casi 5000 ejemplos en entrenamiento, mientras que la clase *Bar* tiene solo 164 (Fig. 1). A partir de este análisis se optó por asignar a cada imagen una probabilidad inversamente proporcional a la distribución de su clase. De esta forma, cuando se generan mini-batches o pequeños conjuntos de 8 imágenes para entrenar el clasificador en cada iteración, las imágenes que están menos representadas tendrán más probabilidad de ser seleccionadas y se reduce el sesgo mayoritario en el clasificador.

También se analizaron los datos opcionales, que contienen anotaciones de objetos que aparecen en las escenas. Levantando todos los objetos de las estructuras XML, se encontró un total de 58420 objetos distribuidos en 887 clases. Las 100 clases mas abundantes contienen mas del 85 % de los objetos, por lo que el resto se descarta para simplificar. También se realizó un tipo de lematización de las clases, eliminando la variación de los plurales y otras descripciones. Por ejemplo, los términos *plant*, *occluded plant* y *plants* serán tomados como una única clase.

2. Métodos

Para resolver el problema, se desarrolló un método basado en aprendizaje profundo (DL, del inglés *deep learning*). Hoy en día el estado del arte en clasificación de imágenes está dominado por DL, incluso en problemas donde no se tienen volúmenes de datos tan grandes [1]. El conjunto de datos de entrenamiento contiene 13650 imágenes distribuidas en 16 clases con semánticas bastante diferentes. Dada la complejidad y el volumen de los datos, una red neuronal entrenada desde cero probablemente no logre buenas tasas de clasificación, por lo que se optó por partir

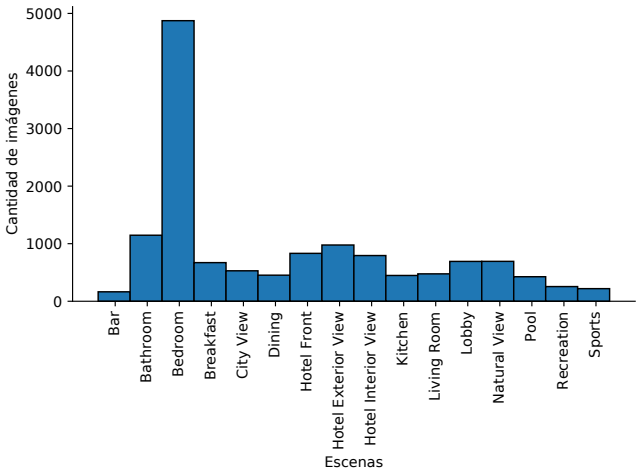

Figura 1: Distribución de las clases de escenas a clasificar en los datos de entrenamiento.

de una red neuronal pre-entrenada. Esta técnica, denominada aprendizaje por transferencia, permite partir de un modelo ya ajustado intensivamente con millones de imágenes de otros dominios (en este caso del conjunto de datos ImageNet [2]), y llevarlo al dominio de las escenas de hoteles mediante una nueva etapa de entrenamiento.

Uno de los modelos que mejores resultados han obtenido son las redes residuales (ResNet) [3]. El método propuesto en estas redes, el uso de bloques residuales, ha permitido entrenar mayor profundidad o cantidad de capas manteniendo un costo computacional razonable y mejorando la capacidad de generalización. El bloque residual consiste basicamente en un agregado de una conexión lineal entre dos capas permite la transmisión de la información a lo largo de la red, salteando capas y mitigando el problema de los gradientes que se desvanecen o explotan. Incluso se reduce la importancia de elegir el número correcto de capas, ya que la red puede aprender a *saltearse* algunos bloques residuales, optimizando la cantidad de parámetros que son utilizados realmente. En esta arquitectura, la primer capa, que consiste en una red convolucional de 64 filtros, en general aprende características básicas de las imágenes, como la detección de bordes. Las capas siguientes aprenden a extraer características cada vez mas complejas y mas descriptivas de los objetos en la imagen, llegando a la capa final donde se genera la salida correspondiente a la clase detectada. Partiendo de una red pre-entrenada, se inicializó de cero la última capa de la red, definiendo ahora las 16 salidas esperadas. Para realizar un entrenamiento fino de la red completa con las imágenes de hoteles, se definió una tasa de aprendizaje diferente según la profundidad de la red. La primer capa se dejó inmutable, ya que genera características comunes a cualquier fotografía. La tasa de aprendizaje de las siguientes capas se fijaron en 10^{-5} y la capa final en 10^{-4}. El orden de magnitud de diferencia lleva a que se modifiquen con mayor velocidad los pesos de la ultima capa (que comienzan con valores aleatorios y son más específicos al problema a resolver) y se va ajustando muy lentamente los pesos de las capas intermedias que generan las características significativas de las imágenes, adaptándolas lentamente a este nuevo dominio y aprovechando la información que ya fue aprendida en el entrenamiento con ImageNet.

Para incluir la información de los objetos en la imagen, se diseñó la arquitectura representada en la Figura 2. Se utilizó la ResNet descripta anteriormente como base, que genera 2048 características a partir de cada imagen. Luego, hay dos redes de capas completamente conectadas que toman estas características como entradas. Una es la red detectora de objetos (ObjNet), que tiene 100 salidas y es entrenada para predecir cuales de esos 100 objetos se encuentran en la imagen, teniendo en cuenta que es una clasificación multiclase (puede haber más de una clase correcta). La otra, es la red clasificadora de escenas (HostelNet) y se entrena para predecir la etiqueta esperada de cada imagen. El entrenamiento de este modelo se realizó en dos etapas. Para cada época (un paso completo por el conjunto de entrenamiento), se entrenó primero la ObjNet con las etiquetas provistas en los datos optativos, y luego la red clasificadora de escenas con todos los datos de entrenamiento, continuando tantas épocas como sean necesarias hasta que converge la HostelNet. Luego para la predicción final, la ObjNet no se utiliza, pero se espera que la ResNet de base haya aprendido mejores características para describir las imágenes, necesarias para identificar los objetos. Se evaluó también otra variante de esta arquitectura que consiste en concatenar la salida de red detectora

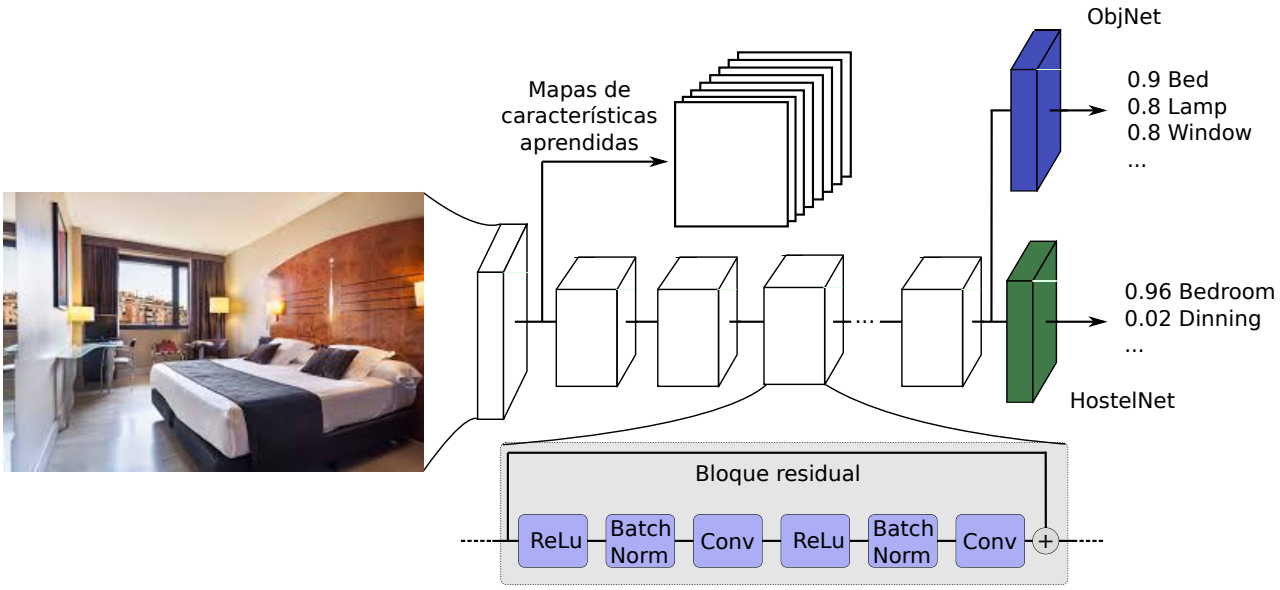

Figura 2: Arquitectura de red propuesta. El último bloque en azul es una etapa para realizar clasificación de los objetos en las escenas, y el bloque verde realiza la clasificación de la escena en las 16 clases provistas.

de objetos a las características de la ResNet y utilizarla como entrada a la HostelNet, repitiendo el mismo esquema de entrenamiento.

La etapa de aumentación de datos genera deferentes representaciones de los mismos objetos, por ejemplo de una cama, en diversas orientaciones y variaciones de color y tamaño. Esto permite que el clasificador pueda aprender cuales son las características que describen mejor un objeto, y no aprenda las características de algunos ejemplos en particular, mejorando de esta forma la capacidad de generalización del modelo.

3. Validación y resultados

Se utilizó un esquema de validación cruzada de 10 particiones para tener una estimación del rendimiento de cada propuesta, utilizando particiones de entrenamiento (80 %), optimización (10 %) y test (10 %). Los modelos superadores en el valor promedio de las 10 particiones, se evaluaron en test definitivo. Para las pruebas finales, se promedió la salida de los modelos obtenidos en cada partición, eliminando de esta forma fallas de predicciones puntuales. Si bien se utiliza la métrica de la tasa de acierto balanceada para medir el resultado final, para la optimización los gradientes se calculan utilizando la función de pérdida entropía cruzada (*cross-entropy* [4]; esto también es útil a la hora de promediar la salida de los modelos, ya que esta función de pérdida permite medir mejor qué tan acertada fue una predicción considerando las 16 salidas de la red, y no solo si fue un acierto o un error en la clase correcta. Se implementó además un método de finalización temprana (*early stop*) para seguir la evolución de la función de pérdida en la partición de optimización y detener el entrenamiento antes de que el modelo comience a sobreajustarse.

Se evaluaron redes con diferente profundidad (cantidad de capas), y se encontró que las de 152 capas lograban los mejores resultados, manteniendo un costo computacional relativamente bajo gracias a los bloques residuales. Para los primeros resultados se encontró en la matriz de confusion de la salida que muchos ejemplos se clasificaban erroneamente como *Bedrooom*, la clase mayoritaria. Los resultados mejoraron sustancialmente al incorporar un vector de pesos en el cálculo de la función de pérdida, dando mas peso relativo a las clases minoritarias (es decir, las clases minoritarias tienen mayor efecto en la retropropagación del error).

Se realizaron algunas pruebas preliminares sobre la arquitectura que combina ObjNet-HostelNet. Se observó que la red efectivamente pudo aprender a reconocer objetos (es decir, identificar qué objetos aparecen en la imagen) siguiendo la función de pérdida. Aún así, los métodos propuestos llegaron a igualar pero no superar la implementación de solamente la HostelNet, es decir, sin utilizar los datos opcionales, por lo que la implementación definitiva fue

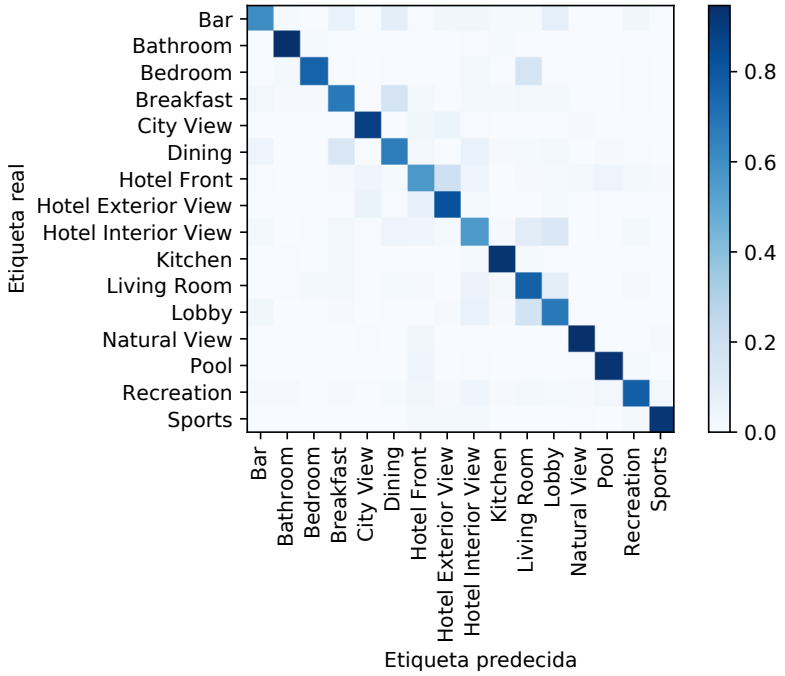

Figura 3: Matriz de confusión acumulada para los casos de test en la validación cruzada. En azul más oscuro se ven las distribuciones más altas.

basada en esta red más simple.

Finalmente, se puede realizar un análisis sobre las estimaciones del clasificador. En primer lugar, en la Figura 3 se puede ver la matriz de confusión de clases predecidas versus las etiquetas reales. Se puede ver que las clases que más se confunden entre sí son aquellas que están más cercanas en cuanto a su significado y elementos en común, por ejemplo *hotel interior view* junto a *lobby* y *living room*; mientras que *breakfast* se confunde con *dinning* y *bar*. En este sentido, se pueden analizar imágenes para identificar elementos sobre los que se puedan mejorar las soluciones. En la Figura 4 se muestra una pequeña selección de las imágenes que, presuntamente, están mal clasificadoas dentro del conjunto de las primeras 300 imágenes de la partición de evaluación final. Las primeras dos filas corresponden a lo que parecen ser dormitorios de tipo ejecutivo, clasificados como *living room*, posiblemente debido a la presencia de televisor, escritorio y sillas. En particular, es interesante como el ID 121 se confunde con *Lobny* e ID 286 por *living room*, posiblemente por la carga de decoraciones. En los siguientes casos la red parece identificar elementos que no terminan de definir la clase correspondiente, por ejemplo en el caso 107 posiblemente está tomando el techo como *exterior*, mientras que el 142 confunde un escritorio de computación con un dispositivo deportivo, o el 163 confunde un desayuno por cena. Para resolver estos casos, donde parece que el clasificador no está funcionando pero aún así se encuentran elementos que tienen sentido, es necesario etiquetar más ejemplos o implementar otros métodos de entrenamiento, por ejemplo incorporando técnicas de aprendizaje por refuerzo.

Referencias

[1] C. Sun, A. Shrivastava, S. Singh, and A. Gupta, "Revisiting Unreasonable Effectiveness of Data in Deep Learning Era," *Proceedings of the IEEE International Conference on Computer Vision*, vol. 2017-Octob, pp. 843–852, 2017.

[2] Jia Deng, Wei Dong, R. Socher, Li-Jia Li, Kai Li, and Li Fei-Fei, "ImageNet: A large-scale hierarchical image database," *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pp. 248–255, 2009.

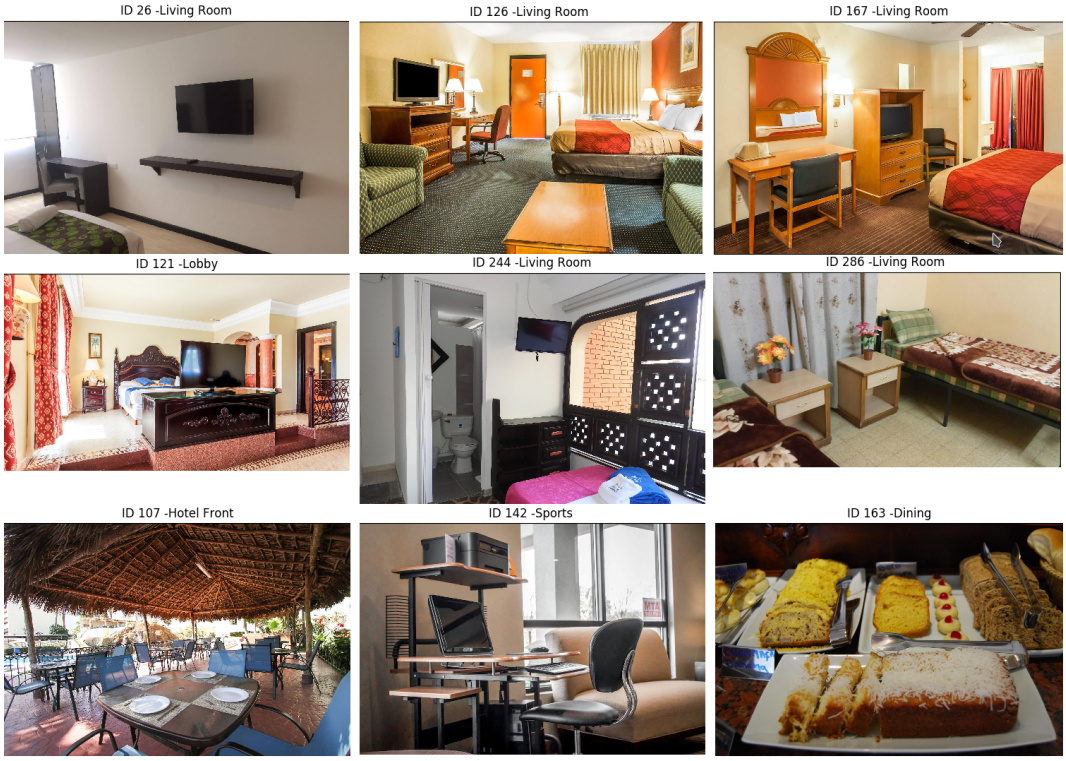

Figura 4: Contra-ejemplos: imágenes que posiblemente se hayan clasificado de forma incorrecta en el conjunto de datos final. Cada una lleva el ID de identificación de la imagen seguido de la clase estimada por la red.

[3] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.

[4] Z. Zhang and M. R. Sabuncu, "Generalized Cross Entropy Loss for Training Deep Neural Networks with Noisy Labels," *NeurIPS*, 2018.