

Clustering de imágenes con *autoencoders* convolucionales

Cristóbal Loyola Maureira

Link GitHub: https://github.com/cloyolam/clustering_images

1. Descripción del proyecto

El proyecto consiste en llevar a cabo un *clustering* de imágenes (aprendizaje no supervisado). Específicamente, se busca entrenar un modelo que, a partir de una imagen del interior de una casa, sea capaz de determinar a qué lugar de la casa pertenece (baño, cocina, dormitorio, etc.).

Para esto, se cuenta con una base de datos de 11131 imágenes, tanto de interiores de casas como de exteriores. Para el análisis, sólo interesan aquellas imágenes de interiores. Sin embargo, los datos no están etiquetados, por lo tanto no se puede determinar en un principio cuáles son aquellas imágenes que sirven para el estudio. Además, para las imágenes correspondientes a interiores, tampoco se conoce a qué lugar de la casa corresponde.

Dado el tamaño del *dataset*, es impráctico etiquetar los datos manualmente. En vez de eso, se usará dos técnicas vistas durante el curso: redes neuronales convolucionales y *clustering*. La idea es intentar agrupar las fotos similares en un mismo *cluster*, de forma que idealmente las imágenes de dormitorios queden juntas, y lo mismo para el resto de lugares. Sin embargo, dada la alta dimensionalidad de las imágenes, no es posible aplicar técnicas de *clustering* (en particular, *K-means*) directamente. Para llevar a cabo esta tarea, se usará *autoencoders* convolucionales, que permiten generar una representación reducida de la imagen, conservando sólo sus características más importantes.

A continuación, se describe con más detalle las dos etapas del proyecto.

1.1. *Autoencoders* convolucionales

Autoencoders convolucionales es un método de aprendizaje no supervisado. A grandes rasgos, el método consiste en entrenar una red neuronal convolucional, donde el *set* de entrenamiento corresponde a un conjunto de imágenes, y para cada una de ellas, la etiqueta asociada es la misma imagen. La red se divide en dos etapas: en la primera, el input se lleva a capas con un número de neuronas cada vez menor, lo que puede ser pensado como una etapa de codificación; en la segunda, se agregan capas que van creciendo en número de neuronas. Esto

último puede ser pensado como un proceso de decodificación, hasta obtener la imagen original. La etapa importante para el análisis posterior es la primera, ya que permitirá generar una representación reducida de la imagen, con sus características más importantes. Esta versión codificada de las imágenes es la que se utilizará para llevar a cabo el *clustering*.

1.2. *Clustering*

Luego de haber comprimido cada imagen con *autoencoders*, se llevó a cabo un proceso de *clustering* utilizando *K-means*. Este método agrupa de manera iterativa los datos en K *clusters* esféricos, calculando en cada paso la distancia del dato a los centroides de los *clusters* y posteriormente asociándolo al más cercano. Lo que se espera idealmente es que las imágenes correspondientes a un mismo lugar de la casa queden agrupados en un mismo *cluster*.

2. Metodología

Para llevar a cabo los pasos mencionados anteriormente, se usó el lenguaje de programación *python*. Para cargar, manipular y graficar los datos, se utilizó las librerías pandas, numpy, matplotlib y open-cv (para trabajar con imágenes).

Para la etapa de *autoencoders*, se usó Keras (una librería de alto nivel para redes neuronales), utilizando como *backend* la librería Tensorflow. Keras permite definir de manera sencilla la arquitectura de la red, con la opción de agregar al modelo capas convolucionales, de *pooling*, etc.

En la siguiente figura, se muestra la arquitectura de la red utilizada:

Layer (type)	Output Shape	Param #
conv2d_1 (Conv2D)	(None, 240, 320, 32)	896
max_pooling2d_1 (MaxPooling2D)	(None, 120, 160, 32)	0
conv2d_2 (Conv2D)	(None, 120, 160, 16)	4624
max_pooling2d_2 (MaxPooling2D)	(None, 60, 80, 16)	0
conv2d_3 (Conv2D)	(None, 60, 80, 8)	1160
max_pooling2d_3 (MaxPooling2D)	(None, 30, 40, 8)	0
conv2d_4 (Conv2D)	(None, 30, 40, 8)	584
up_sampling2d_1 (UpSampling2D)	(None, 60, 80, 8)	0
conv2d_5 (Conv2D)	(None, 60, 80, 16)	1168
up_sampling2d_2 (UpSampling2D)	(None, 120, 160, 16)	0
conv2d_6 (Conv2D)	(None, 120, 160, 32)	4640
up_sampling2d_3 (UpSampling2D)	(None, 240, 320, 32)	0
conv2d_7 (Conv2D)	(None, 240, 320, 3)	99
<hr/>		
Total params: 13,171		
Trainable params: 13,171		
Non-trainable params: 0		

Figura 1: Arquitectura de la red convolucional

La sexta capa de la red corresponde al vector que se usará para el *clustering*.

Para la etapa de *clustering*, se usó la librería scikit-learn, que cuenta con una implementación optimizada de *K-means*. Se eligió $k = 20$, ya que se consideró que es un número razonable de grupos para encontrar características similares, además de que el algoritmo no toma mucho tiempo (en comparación con valores más altos de k).

Los detalles asociados a cada paso, además del proceso de análisis exploratorio, se encuentran comentados en el *jupyter-notebook* contenido en el repositorio GitHub.

El proyecto se llevó a cabo en un *laptop* con procesador Intel Quad-Core i5-6300HQ 2.3 GHz y 8GB de memoria RAM. Se utilizó un total de 11131 imágenes para entrenar la red convolucional.

3. Resultados

A continuación, se muestra la distribución de las imágenes luego de aplicar *clustering*:

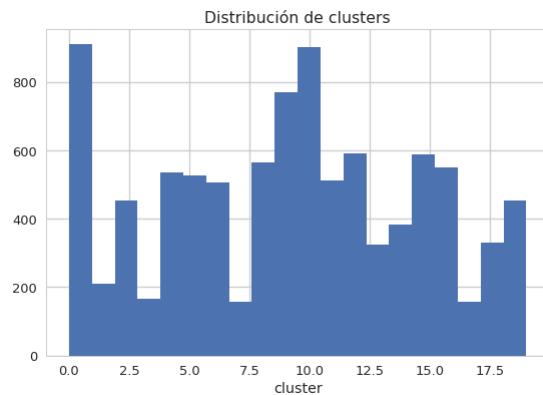


Figura 2: Distribución en los distintos *clusters*

La siguiente imagen muestra 5 imágenes de cada *cluster*, elegidas al azar:

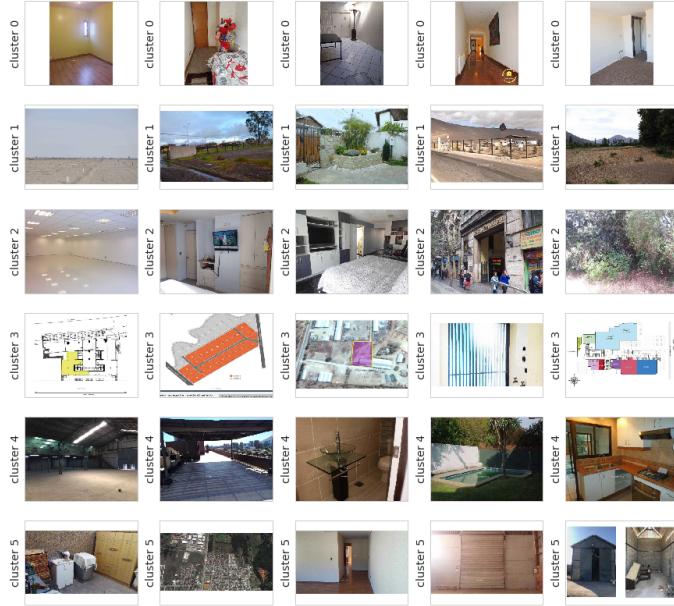


Figura 3: Clusters 0-5

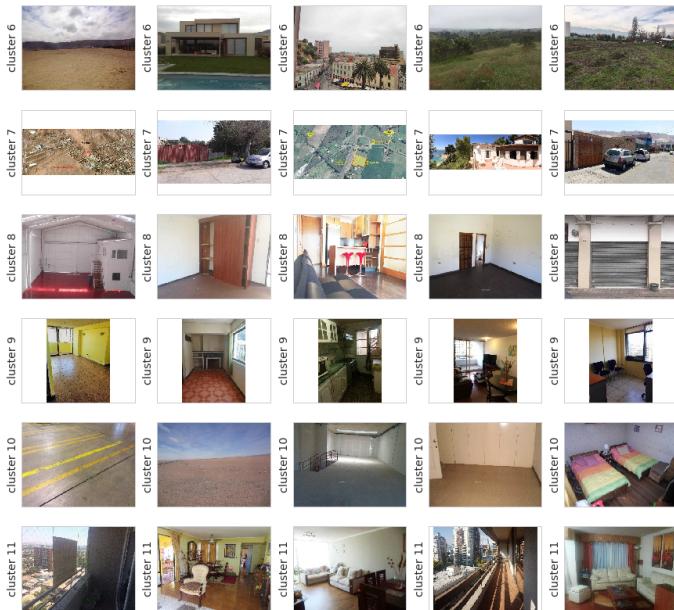


Figura 4: Clusters 6-11

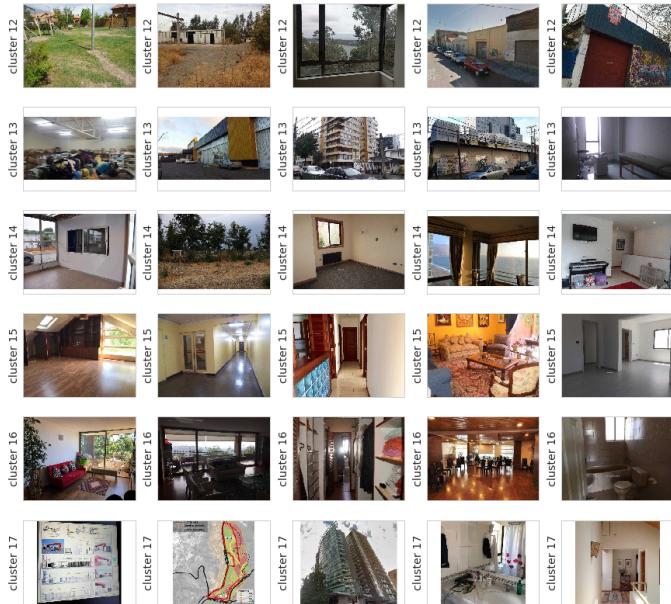


Figura 5: Clusters 12-17



Figura 6: Clusters 18 y 19

Llama la atención el cluster 3, ya que la mayor parte de las imágenes corresponde a planos. Veamos algunas imágenes del cluster:

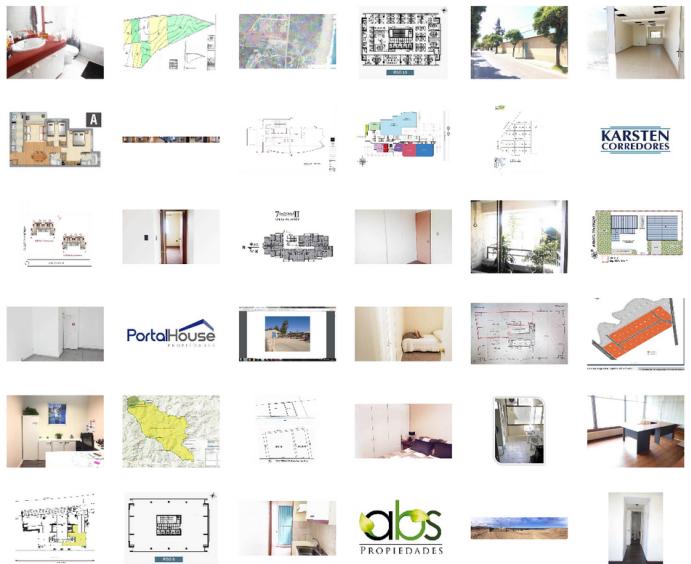


Figura 7: Imágenes del *cluster* 3

Efectivamente, se ve que la mayor parte de las imágenes corresponde a planos, logos de empresas y tablas, todos los cuales tienen en común que su fondo es blanco. Para agrupar estas imágenes, se volvió a hacer *clustering* dentro del *cluster* 3. A continuación se muestra el *cluster* más numeroso resultante:



Figura 8: Cluster con planos

Se puede ver que la totalidad de imágenes corresponde a planos. Esto significa que el *clustering* efectivamente agrupó las imágenes con características similares.

Estudiemos ahora el *cluster* 1 del *clustering* original. En la figura 3 se puede ver que las 5 imágenes de ejemplo corresponden a paisajes y al exterior de una casa. Por este motivo, se volvió a realizar *clustering* dentro del *cluster* 1 para intentar agrupar estas imágenes. A continuación, se muestra las imágenes de uno de los *clusters*:

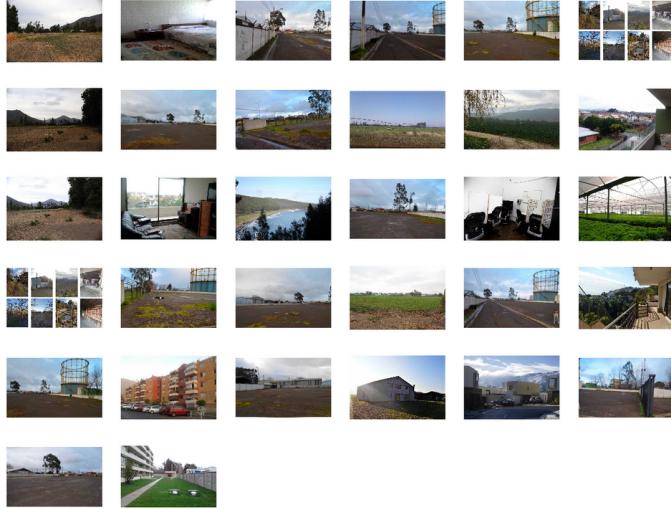


Figura 9: *Cluster* con paisajes

Se puede ver que todas las imágenes del *cluster* (excepto dos de ellas) corresponden a imágenes de paisajes. Nuevamente, el algoritmo agrupó imágenes similares en un mismo *cluster*.

4. Conclusiones

El problema de clasificación de imágenes es costoso, dada la alta dimensionalidad de estas (información de muchos píxeles en tres canales). Si el problema es no supervisado (es decir, si no se conoce a qué clase pertenece cada imagen), una forma de resolverlo es utilizando algoritmos de *clustering*, de forma que las imágenes similares sean agrupadas en un mismo *cluster*. Sin embargo, esto sigue siendo un problema difícil para el caso de imágenes (los algoritmos no funcionan bien si la dimensión es demasiado alta). Una alternativa es utilizar *autoencoders* convolucionales. Estos permiten obtener una versión codificada de las imágenes, que contiene sus características más importantes. Terminado este proceso, el *clustering* puede ser aplicado sobre las imágenes codificadas, que tienen una dimensión mucho más baja que las originales.

Para el *dataset* particular del proyecto, se pudo ver que imágenes similares (por ejemplo, fotos de paisajes) efectivamente fueron agrupadas en un mismo *cluster*. Sin embargo, el objetivo inicial de agrupar las imágenes de distintos lugares de una casa no fue logrado completamente. Esto se debe, por ejemplo, a que al momento de agrupar las imágenes con *K-means*, aquellas con una iluminación similar, pero que no necesariamente contenían los mismos elementos, eran agrupadas en un mismo *cluster*. Para lograr este objetivo, podría llevarse a cabo un proceso de detección de objetos y/o métodos de aprendizaje no super-

visado, además de entrenar una red convolucional más expresiva con la ayuda de GPUs.

5. Instrucciones para ejecutar código

El código se encuentra en formato *jupyter-notebook* en el repositorio de GitHub, con comentarios para la ejecución. Los resultados podrían cambiar al intentar reproducir el resultado, ya que algunos pasos son aleatorios (como el entrenamiento de la red y el algoritmo *K-means*).