CLASIFICACIÓN Y BÚSQUEDA POR SIMILITUD DE SKETCHES USANDO REDES CONVOLUCIONALES

Mario Garrido

Universidad de Chile
Santiago, Chile

ABSTRACT

En el presente trabajo se estudia el desempeño de 2 redes convolucionales (skNet y skResNet) para los problemas de clasificación de sketches del dataset Quickdraw y recuperación de imágenes, utilizando una imagen del dataset como query. El tiempo de entrenamiento de skResNet, pese a su tamaño mucho más grande, es solo el doble de skNet, pero sus resultados para la tarea de búsqueda por similitud son mucho mejores, aunque para lograr de mejor forma este objetivo se debe entrenar la red por pocas épocas, a diferencia del problema de clasificación.

1. INTRODUCCIÓN

Los avances en redes convolucionales han permitido una explosión de aplicaciones en el área de reconocimiento de imágenes, particularmente gracias a la construcción de buenos filtros que permiten extraer información crucial de las imágenes, la que luego es agregada en las capas finales para construir feature vectors. Gracias a esta representación compacta de cada imagen (feature vector) se puede atacar el problema de recuperar contenido relevante respecto a una imagen de query a través de la comparación entre feature vectors, asumiendo que aquellos que se encuentran más cercanos al feature vector de la query corresponden a imágenes parecidas a la query. Hay, sin embargo, una serie de problemas con el supuesto anterior, entre los más graves se encuentran el curse of dimensionality, que en este contexto se manifiesta a través de la incapacidad de utilizar de forma confiable distancias en espacios de alta dimensionalidad, y el hecho de que las representaciones aprendidas tienen como fin caracterizar la imagen para resolver el problema de clasificación y no el problema de minimizar la distancia entre las representaciones de imágenes similares, lo que resulta en que las imágenes de una categoría o clase no tienen por qué, necesariamente, encontrarse cerca en su representación. Si bien, se han desarrollado arquitecturas de redes que atacan el problema de obtener representaciones similares para imágenes similares, vale la pena estudiar el comportamiento de las redes diseñadas para la clasificación con el fin de notar si su desempeño es suficiente para esta tarea.

El dataset a utilizar para la experimentación corresponde a QuickDraw[1] de Google, en su formato de archivos *.bin*, que contiene cerca de 50 millones de imágenes de *256x256* píxeles de sketches en blanco y negro.

Las arquitecturas a utilizar, llamadas *skNet* y *skResNet*, cuentan con 8 y 17 capas convolucionales, respectivamente. Cada capa convolucional usa ReLU como función de activación y aplica batch-normalization. Vale la pena notar que las capas convolucionales van reduciendo la imagen original, al ir aplicando los kernels de 3x3, y que las capas de max-pooling reducen a un poco menos de la mitad el tamaño de los datos, por lo que a medida que se avanza en las redes se va agrupando la información, esto podría parecer, en una primera mirada, como una pérdida de información, y ciertamente hay información que se va ignorando, pero considerando que cada capa convolucional introduce nuevos canales (número de filtros) notamos que esta pérdida no va siendo tan grosera realmente.

En el caso de *skResNet*, que es, también, una red residual, se alimentan algunas capas (las llamadas residuales) con información de capas no inmediatamente anteriores además de la información de la capa anterior, esto implica que es necesario agregar padding. Esto permite 2 comportamientos que terminan siendo beneficiosos: Reconsiderar información que fue posiblemente descartada por los features generados después de la capa anterior y mantener cierta dimensionalidad de mejor forma que tan solo rellenando con valores 0. La forma en que las capas residuales agregan la información es, simplemente, sumando. Las capas de Max-pooling usan kernels de 3x3 y strides de tamaño 2. Las últimas 2 capas de ambas redes corresponden a capas fully-connected.

Una vez que las redes se entrenen para resolver el problema de clasificación se utilizará la penúltima capa de 1024 neuronas para generar feature vectors de las imágenes y, con esto, hacer image retrieval.

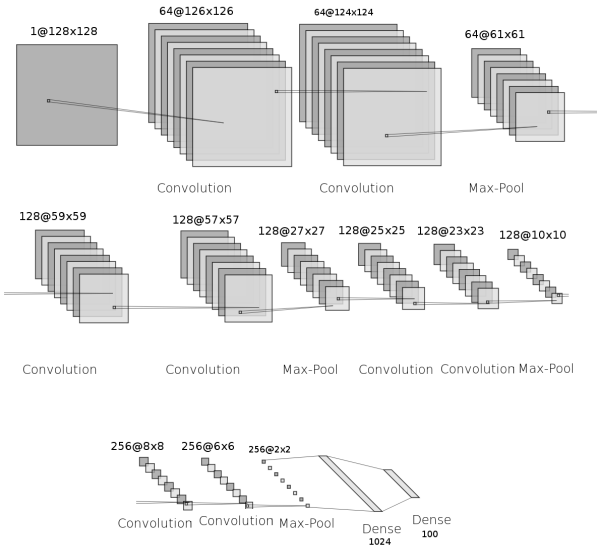

Fig. 1. *Representación de arquitectura de red skNet.*

2. DESARROLLO

Primero es necesario procesar el dataset para obtener los datos para entrenar las redes. El archivo *create_dataset.py* contiene las funciones necesarias para confeccionar, desde una carpeta llamada *files* y un archivo llamado *categories.txt* que contienen los 345 archivos *.bin* y cada uno de los nombres de las categorías posibles, respectivamente, una muestra de 100 clases distintas que cuenta con 1000 imágenes de entrenamiento y 50 de prueba, por cada clase. Estas imágenes se guardan en las carpetas *testing* y *training*, que son creadas por la función. Para iniciar el proceso se debe llamar a la función *construct_from_scratch*. Las imágenes escogidas, de forma aleatoria al igual que las clases, son redimensionadas a *128x128* píxeles.

Una vez que contamos con los datos necesarios podemos utilizar el archivo *train.py* para entrenar ambas arquitecturas, pudiendo establecer el número de épocas a entrenar y el tamaño del batch. El código fue generado utilizando Estimator de Tensorflow 1.5[2]. Las funciones necesarias para entrenar la arquitectura *skNet* se encuentran en el archivo *n1.py*, mientras que las correspondientes a la arquitectura *skResNet* se encuentran en el archivo *n2.py*. Los archivos *n1_features.py* y *n2_features.py* contienen las mismas funciones con una pequeña modificación para utilizar la capa fc1 como generador de features.

Una vez que las redes se encuentran entrenadas se guardan los parámetros de los modelos en las carpetas *checkpoints_n(x)_(y)e*, donde *(x)* corresponde al número de la red

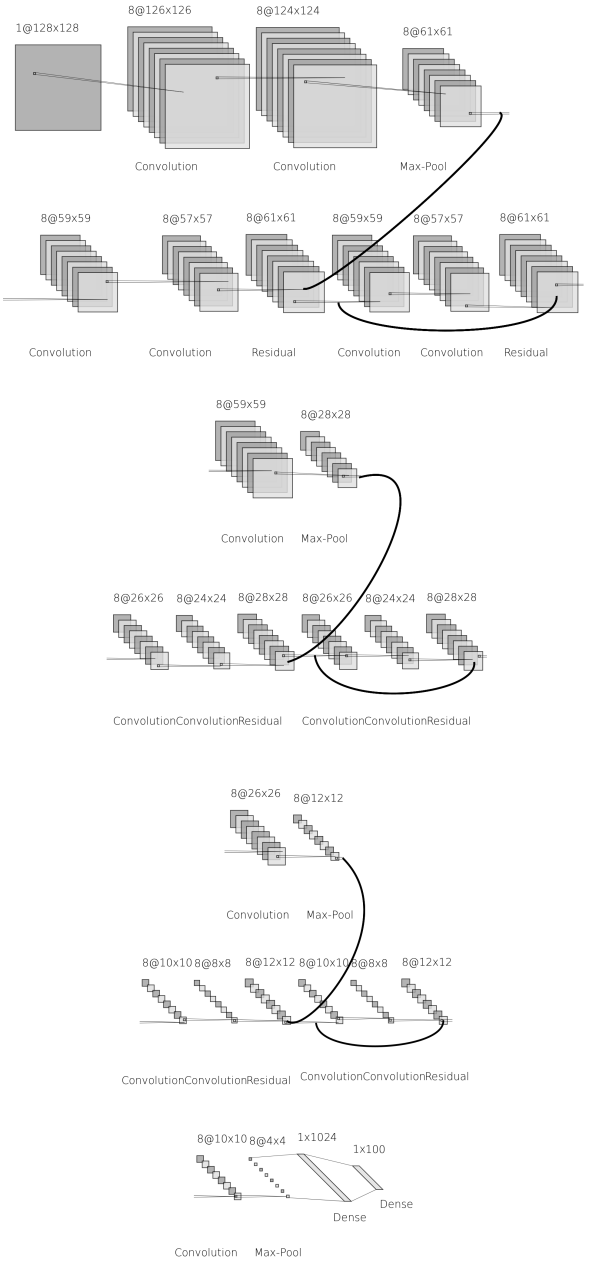

Fig. 2. *Representación de arquitectura de red skResNet.*

(1 si es *skNet* y 2 si es *skResNet*) e *(y)* corresponde al número de épocas.

El archivo *load.py* permite cargar los modelos entrenados y estimar el *accuracy* sobre el conjunto de entrenamiento y el conjunto de prueba, esto es meramente por comodidad en caso de necesitar re-evaluar, ya que en el archivo *train.py* ya se hacen estas estimaciones.

Finalmente, el archivo *mAP.py* permite, una vez que se tienen los modelos entrenados, cargarlos, utilizar sus variantes (*n1_features.py* y *n2_features.py*) para generar los feature vectors de las imágenes y calcular el mAP. Debido al alto costo computacional de esta operación se utiliza un *pool* de 4 threads para agilizar el cálculo. Vale la pena notar que la función *mAP* permite calcular el mAP hasta un valor arbitrario k, si se desea calcular para todo el conjunto de prueba se debe utilizar el valor 4999.

3. RESULTADOS EXPERIMENTALES

De aquí en adelante se llamará la arquitectura *skNet* como n1 y a la arquitectura *skResNet* como n2. Los modelos entrenados se encuentran disponibles en Google Drive[3], ya que para el entrenamiento no se utilizó Colaboratory[4].

Ambas redes fueron entrenadas utilizando el solver Adam, con una tasa de aprendizaje de 0.001 y un tamaño de batch de 128. Los experimentos revelaron que la capacidad de generalización de las redes no aumenta considerablemente al incrementar el número de épocas, esto pareciera ser una señal de sobre-ajuste, pero se aprecia que, en el caso de n1 esta parece oscilar alrededor de un mínimo, como se aprecia en la Figura 3, y a medida que se incrementa el número de épocas se va convergiendo a un mínimo que se ajusta bien al conjunto de entrenamiento (Figura 4) pero con resultados muy parecidos para el conjunto de prueba (Figura 5), de aquí que asumimos que su generalización es pobre, en comparación a n2. Por otro lado, n2 va convergiendo de forma más directa (Figura 3), pero esta convergencia también tiene muy poco impacto en el resultado sobre el conjunto de prueba (Figura 5), aunque en el caso de las 60 y 90 épocas logra superar los resultados de las primeras 20 en el conjunto de prueba.

Dicho esto respecto a la clasificación, notamos que el comportamiento exhibido en la recuperación de imágenes se condice con lo esperado teóricamente (mencionado en la introducción): Mayor accuracy tanto en el conjunto de entrenamiento como el conjunto de prueba no implica una mejor capacidad de generar feature vectors que se encuentren cercanos para imágenes cercanas, y, es más, mientras menos entrenadas se encuentran las redes mayor probabilidad hay de generar representaciones que se presten bien para esta tarea. Al incrementar las épocas hasta 60 y 90 (Figura 7) se oscila alrededor

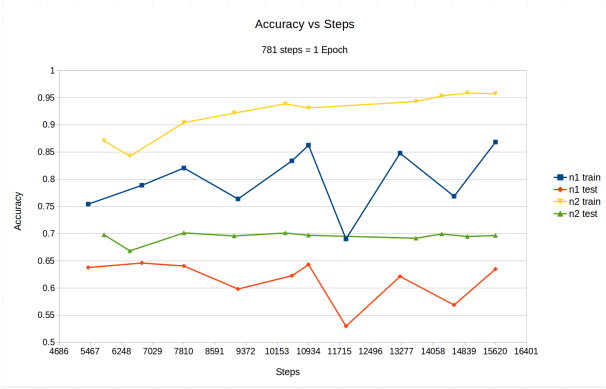

Fig. 3. *Accuracy vs Steps. Los datos van desde 7 épocas hasta 20 épocas.*

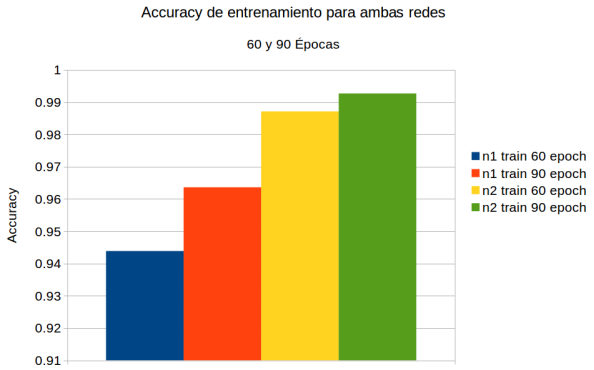

Fig. 4. *Accuracy alcanzada en el conjunto de entrenamiento para altas épocas.*

de valores de mAP peores que aquellos encontrados en las primeras épocas (Figura 6).

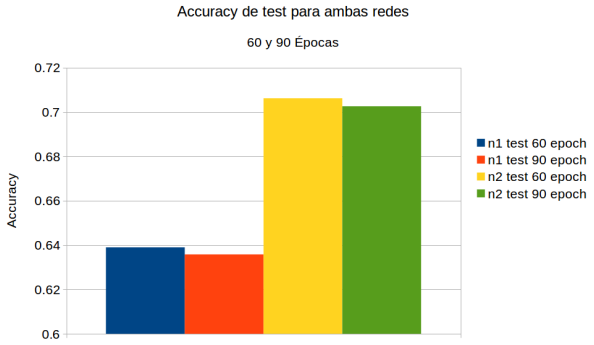

Fig. 5. *Accuracy alcanzada en el conjunto de prueba para altas épocas.*

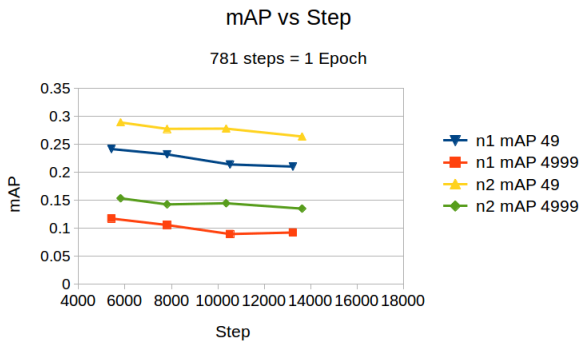

Fig. 6. *mAP hasta k = 49 y k = 4999, épocas 7 hasta 17.*

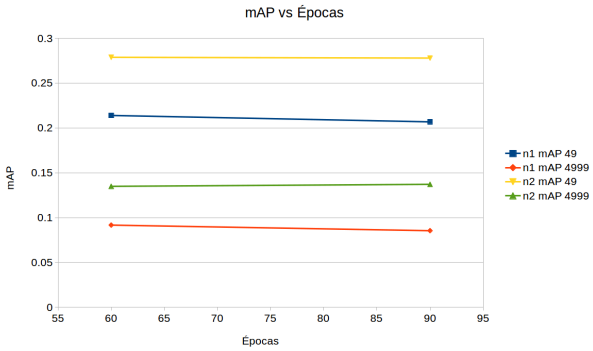

Fig. 7. *mAP hasta k = 49 y k = 4999 para altas épocas.*

4. CONCLUSIONES

Debido a la gran cantidad de clases del dataset este no es un problema trivial de clasificación, sin embargo ambas redes se desempeñan de forma aceptable, aunque un accuracy cercano al 70 % sigue siendo difícil de utilizar en aplicaciones comerciales. Pese a que los costos computacionales aumentarían, quizás sería posible obtener mejores resultados de clasificación utilizando las dimensiones originales de 256x256 píxeles. Respecto al problema de búsqueda por similitud (image retrieval) se tienen resultados poco alentadores, considerando que dentro de las primeras 10 queries, para algunas clases, es raro encontrar una respuesta que corresponda al mismo tipo. Utilizando estas mismas herramientas podría realizarse una clasificación sobre el conjunto ordenado en base a distancias y responder aquellas imágenes que sean clasificadas como del mismo tipo que la query, pero esta configuración no fue probada en el trabajo actual. Vale la pena preguntarse si se obtiene un mejor mAP mientras menos épocas de entrenamiento se realicen simplemente porque tener pesos casi aleatorios produce una representación más compacta (en el espacio de los features y por cada clase) que la red entrenada, lamentablemente esto tampoco fue explorado en el presente trabajo. Para producir buenas representaciones que permitan tener poca distancia entre los feature vectors de imágenes similares es necesario utilizar otras funciones para realizar la minimización e incluso otras arquitecturas como las redes siamesas o redes tripletas, que aprenden los llamados *embeddings* que se prestan mejor para estas tareas.

5. REFERENCES

[1] https://github.com/googlecreativelab/quickdraw-dataset

[2] https://www.tensorflow.org/versions/r1.5/api_docs/python/

[3] 10 Epoch: https://drive.google.com/file/d/1V68BmHMAoX5eX-8_5kyJgJASL0B_Bvpj/view?usp=sharing
15 Epoch: https://drive.google.com/file/d/1UjnX-LuY0gFFfHTi4GPalVzC0bebxrGF/view?usp=sharing
20 Epoch: https://drive.google.com/file/d/1zmvZ0l0oPWCVvjfIhC7Q5nkszoMOsUm1/view?usp=sharing

[4] https://colab.research.google.com/