

CC66T-1

Reconocimiento Visual con Deep Learning

Tarea 1: Evaluación de características aprendidas por redes convolucionales en el contexto de búsqueda por similitud

Profesor de Cátedra: José M. Saavedra Rondo

Profesor Auxiliar: Cristobal Loyola

Integrantes:

Sebastián Alday

Álvaro Neira

14 de Noviembre de 2021

Índice

Índice	2
Abstract	3
Introducción	4
Desarrollo	5
Entrenamiento de los modelos	5
Preparación de los datos	5
Generación de TFRecords	6
Entrenamiento AlexNet	7
Entrenamiento ResNet-34	8
Búsqueda por similitud	9
Preparación del catálogo	10
AlexNet - capa fc7	10
ResNet-34 - capa avg-pool	11
Cálculo de Métricas para Búsqueda por Similitud	12
Métricas a utilizar	12
Implementación cálculo de métricas	14
Implementación leave-one-out y distancias	15
Resultados Experimentales y Discusión	17
Entrenamiento de los modelos	17
AlexNet	17
ResNet-34	17
Ejemplos de predicciones de los modelos	17
Búsqueda por similitud	18
Ejemplos de búsqueda por similitud	18
Métricas de evaluación	19
Discusión	22
Conclusiones	23
Referencias	24

1. Abstract

El presente informe detalla el trabajo realizado para resolver el problema de búsqueda por similitud en un dataset de sketches de animales llamado QuickDraw-Animals. Para realizar la búsqueda por similitud se entrenaron dos redes neuronales convolucionales con diferentes arquitecturas base: AlexNet y ResNet-34. Los resultados obtenidos en el entrenamiento de las redes es un accuracy de 75% en ambas redes neuronales. La búsqueda por similitud se evaluó con tres métricas distintas: mean average precision (mAP), precision@1 y gráfico recall-precision. Los resultados obtenidos en la red AlexNet fueron 37,8% y 43,91% de mAP utilizando distancia euclidiana y distancia coseno, respectivamente. En el caso de la red ResNet-34 se obtuvo 52% y 57% de mAP en distancia euclidiana y coseno. Luego, en precision@1 se obtuvo 73% y 75% en la red AlexNet para la distancia euclidiana y coseno, mientras que en la red ResNet-34 se obtuvo 75% y 76% en distancia euclidiana y coseno, respectivamente. Finalmente, el gráfico recall-precision muestra que la red ResNet-34 tiene un mejor comportamiento para realizar la búsqueda por similitud, teniendo una precisión cercana al 50% para un recall de 80%, mientras que en la red AlexNet la precisión es cercana a 30% para el recall de 80%.

2. Introducción

El presente informe detalla la Tarea N°1 realizada para el curso código CC66T - 1 Reconocimiento Visual con Deep Learning del Departamento de Ciencias de la Computación de la Universidad de Chile, perteneciente al Diplomado en Inteligencia Artificial del Programa de Educación Continua.

El objetivo general de la tarea es entender y evaluar la capacidad de las redes convolucionales como extractores de características visuales en el contexto de búsqueda por similitud. Para ello, en esta tarea se evalúan las características aprendidas por redes convolucionales neuronales en un dataset de dibujos (sketches) de animales. En particular se entrenan dos modelos convolucionales: AlexNet y ResNet-34. En cada uno de ellos se obtienen los pesos entrenados para luego ser utilizados para realizar una búsqueda por similitud dentro del dataset mencionado anteriormente.

Luego de realizar la búsqueda por similitud, se calculan las métricas Mean Average Precision (mAP), Precision@1 y el gráfico Recall-Precision, para de esta manera evaluar las redes entrenadas. También se prueban las dos distancias entre elementos que se revisaron en las clases teóricas del curso Reconocimiento Visual con Deep Learning. Estas distancias son la distancia euclidiana y la distancia coseno, con las que se comparan los resultados de las redes convolucionales neuronales entrenadas.

Finalmente, se realiza una discusión acerca de los resultados obtenidos en el entrenamiento y en las métricas de búsqueda por similitud de las redes AlexNet y ResNet-34.

3. Desarrollo

La presente sección contiene los principales aspectos del desarrollo de la tarea. Se explican los pasos a seguir para resolver el problema. En particular, se incluye el entrenamiento de los modelos neuronales, la búsqueda por similitud y el cálculo de las métricas para evaluar los resultados obtenidos.

a. Entrenamiento de los modelos

Los modelos a entrenar se obtuvieron del repositorio **convnet2** (<https://github.com/jmsaavedrar/convnet2>), perteneciente al profesor José Manuel Saavedra, que corresponde a la implementación de una red convolucional neuronal (CNN) basada en tensorflow 2.3+. A continuación se presenta la preparación de los datos a entrenar, la generación de TFRecords y el entrenamiento de AlexNet y ResNet-34.

i. Preparación de los datos

Los datos de entrenamiento y validación (que son utilizados también como conjunto de test) utilizados en el trabajo están disponibles en: <https://www.dropbox.com/s/3xx71kxw0jmwvqd/QuickDraw-Animals.zip>. El dataset QuickDraw - Animals corresponde a un conjunto de sketches de animales categorizados en 12 clases que se incluyen en el archivo 'mapping.txt' (Tabla 1). Los sketches de entrenamiento son 12.000 imágenes, mientras que las de test son 2.399 imágenes.

Tabla 1: Categorías de los sketches de dataset QuickDraw-Animals.

Nombre de la clase	Número de la clase
sheep	0
bear	1
bee	2
cat	3
camel	4
cow	5
crab	6
crocodile	7
duck	8
elephant	9
dog	10
giraffe	11

Una vez que se ha descargado y descomprimido el dataset, se deben generar los archivos de texto para comenzar el entrenamiento y testeo. Esto se realizó siguiendo las instrucciones de la Figura 1.

Figura 1: Generación de archivos de texto para entrenamiento y testeo.

```
# Generar archivos de train y test
import os
import pandas as pd

def generate_annotations(base_dir, datasettype='test'):
    filenames = []
    labels = []
    data_prefix = f"data/{datasettype}_images/"
    data_dir = os.path.join(base_dir, data_prefix)
    print(f"buscando en el directorio: {data_dir}")
    # r=root, d=directories, f = files
    for r, d, f in os.walk(data_dir):
        for file in f:
            category_name = r.split('/')[-1]
            file = f"{category_name}/{file}"
            filenames.append(os.path.join(data_prefix, file))
            labels.append(mapping_dict[category_name])
    print(f"total de imágenes: {len(filenames)}")

    df = pd.DataFrame()
    df['filename'] = filenames
    df['cat_id'] = labels
    df = df.sample(frac=1)

    df.to_csv(f"{datasettype}.txt", sep='\t', header=False, index=False)

!cd /content/convnet2/data
generate_annotations("/content/convnet2/", 'test')
generate_annotations("/content/convnet2/", 'train')

lls
```

ii. Generación de TFRecords

El paquete TensorFlow está optimizado para trabajar con archivos binarios tipo TFRecords. Para la construcción de los archivos TFRecords de entrenamiento y prueba, se creó el archivo 'sketch.config' en el repositorio convnet2, a partir de una modificación del archivo 'sbir_cl.config'. El detalle se muestra en la Figura 2.

Figura 2: Archivo 'sketch.config'.

```
[SKETCH]
NUM_EPOCHS = 20
NUM_CLASSES = 12
BATCH_SIZE = 128
VALIDATION_STEPS = 19
LEARNING_RATE = 0.01
DECAY_STEPS = 10000
SNAPSHOT_DIR = /content/convnet2/snapshots
DATA_DIR = /content/convnet2/data
CHANNELS = 3
IMAGE_TYPE = SKETCH
IMAGE_WIDTH = 224
IMAGE_HEIGHT = 224
USE_MULTITHREADS = False
#CKPFILE = /content/convnet2/snapshots/alexnet_020.h5
```

Posteriormente, se crearon los archivos TFRecords utilizando el código 'create_tfrecords.py' del repositorio convnet2 (Figura 3).

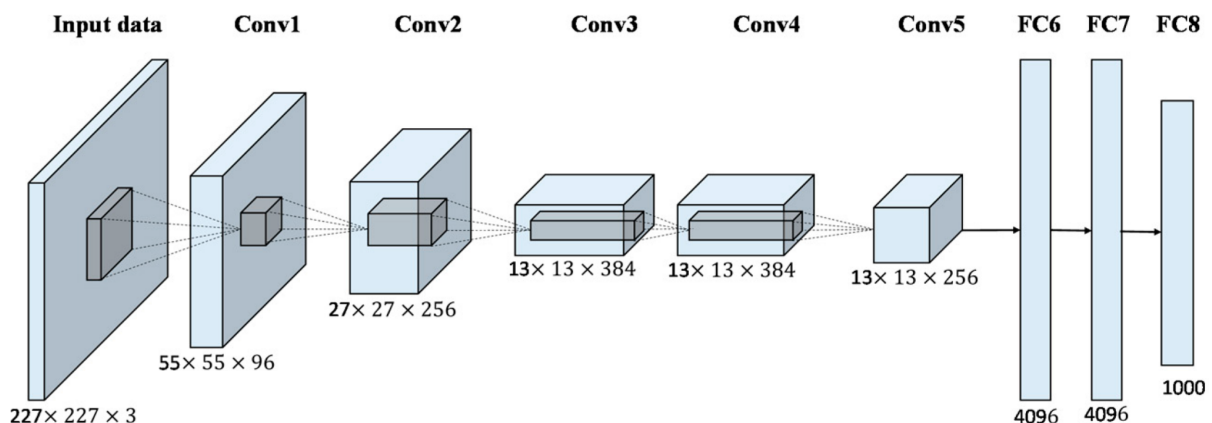
Figura 3: Creación de TFRecords mediante el uso de 'create_tfrecords.py'.

```
# Creamos los tfrecords
%cd /content/convnet2
!python datasets/create_tfrecords.py -type all -config configs/sketch.config -name SKETCH
```

iii. Entrenamiento AlexNet

La arquitectura de la red convolucional neuronal AlexNet se presenta en la Figura 4.

Figura 4: Arquitectura AlexNet.



Fuente: <https://www.mdpi.com/2072-4292/9/8/848>.

El entrenamiento con la arquitectura de la red AlexNet se realizó utilizando el código ‘train_simple.py’ del repositorio convnet2. Se indicaron las opciones de modo entrenamiento, archivo de configuración ‘sketch.config’ y el nombre del modelo SKETCH (Figura 5).

Figura 5: Entrenamiento red AlexNet.

```
# Entrenamos AlexNet
%cd /content/convnet2
!python train_simple.py -mode train -config configs/sketch.config -name SKETCH
```

iv. Entrenamiento ResNet-34

La arquitectura de la red convolucional ResNet-34 se presenta en la Figura 6.

Figura 6: Arquitectura ResNet-34.

layer name	output size	ResNet-34	
conv1	112×112	7×7, 64, <i>stride</i> 2	
conv2_x	56×56	3x3 max pool, stride 2	
		<div><div>3 × 3, 64</div><div>3 × 3, 64</div></div>	×3
conv3_x	28×28	<div><div>3 × 3, 128</div><div>3 × 3, 128</div></div>	×4
		<div><div>3 × 3, 256</div><div>3 × 3, 256</div></div>	×6
conv4_x	14×14	<div><div>3 × 3, 256</div><div>3 × 3, 256</div></div>	×6
		<div><div>3 × 3, 512</div><div>3 × 3, 512</div></div>	×3
	1×1	average pool, 1000-d fc, softmax	
FLOPs		3.6×10 ⁹	

Para utilizar ResNet-34, se hizo un *git fork* del repositorio convnet2 en <https://github.com/alvaro-neira/convnet2>. Con el propósito de utilizar múltiples *branches git* dependiendo de cada tarea (tfrecords, entrenamiento, búsqueda, metricas, graficos, etc.).

Se utilizó un archivo de configuración muy similar al mostrado anteriormente (Figura 7). Notar que la última línea de dicho archivo se comenta o descomenta dependiendo de la tarea que se quiera realizar.

Figura 7: aneira_tfr.config

```
[SKETCH]
NUM_EPOCHS = 20
NUM_CLASSES = 12
BATCH_SIZE = 128
VALIDATION_STEPS = 19
LEARNING_RATE = 0.01
DECAY_STEPS = 10000
SNAPSHOT_DIR = /content/convnet2/snapshots
DATA_DIR = /content/convnet2/data/sketch_folder
CHANNELS = 3
IMAGE_TYPE = SKETCH
IMAGE_WIDTH = 224
IMAGE_HEIGHT = 224
USE_MULTITHREADS = False
#NUM_THREADS = 10
CKPFILE = /content/convnet2/snapshots/020.h5
```

Se crearon los TFRecords de manera muy similar a la indicada anteriormente. Luego, el entrenamiento se realizó utilizando el código ‘train_simple.py’ del repositorio convnet2. El código se modificó de la siguiente manera para poder entrenar esta arquitectura (según se vio en las clases prácticas):

- En la sección de importación de paquetes se agrega la línea “from models import resnet”.
- La línea 86: “model = resnet.ResNet(...)” se descomenta y se cambia el valor de use_bottleneck por False.
- Se dejan estos cambios en la rama git *ssearch_tfr_resnet34*

Posteriormente, se corre en colab, las instrucciones que se muestran en la Figura 8.

Figura 8: Entrenamiento de la red ResNet-34.

```
#Entrenamiento ResNet-34 ~12 minutos
%cd /content/convnet2
!git checkout ssearch_tfr_resnet34
!rm -rf snapshots/*
!python train_simple.py -mode train -config configs/aneira_tfr.config -name SKETCH
```

b. Búsqueda por similitud

La búsqueda por similitud se realiza utilizando los pesos de los modelos entrenados anteriormente y el archivo ‘ssearch.py’ del repositorio convnet2. Para realizar la búsqueda por similitud se utiliza la última capa convolucional de cada uno de los modelos entrenados.

i. Preparación del catálogo

La preparación del catálogo de búsqueda se realizó con las instrucciones de la Figura 9.

Figura 9: Creación de archivo 'catalog.txt'.

```
%cd /content/convnet2/data

# Crear directorio para la búsqueda por similitud
mkdir ssearch

# Eliminar la columna de clasificación del archivo test.txt
awk 'NF{NF-=1};1' <test.txt >catalog.txt
mv catalog.txt ssearch/
```

ii. AlexNet - capa fc7

El vector de características de AlexNet se extrae con un cambio en el archivo 'alexnet.py' del repositorio convnet2. Se comenta la línea 72 donde se crea la capa fc8, entonces se obtiene la capa fc7, que es la capa que se utilizará como vector de características.

Por otro lado, se agrega la línea "CKPFILE = /content/convnet2/snapshots/alexnet_020.h5" en el archivo de configuración 'sketch.config' creado anteriormente. De esta forma se cargan los pesos de la última época entrenada con AlexNet.

Finalmente, se modifica el archivo 'ssearch.py' del repositorio convnet2. Primero, se agrega la línea "from models import alexnet" en la sección de importación de paquetes. Luego, se define el modelo AlexNet en el constructor de la clase SSearch como se muestra en la Figura 10.

Figura 10: Definición modelo AlexNet en constructor clase SSearch.

```
#loading classifier model
model = alexnet.AlexNetModel(self.configuration.get_number_of_classes())
input_image = tf.keras.Input([self.input_shape[0], self.input_shape[1], self.input_shape[2]],
                               name = 'input_image'])
model(input_image)
model.summary()
model.load_weights(self.configuration.get_checkpoint_file(), by_name = True, skip_mismatch = True)
self.sim_model = model
print('sim_model was loaded OK')
# defining process arch
self.process_fun = imgproc.process_sketch
```

Se realiza el cálculo de los feature vectors donde se calculan las relaciones de distancia entre los sketches. Para esto, se ejecuta el código 'ssearch.py' en modo *compute* (Figura 11).

Figura 11: Cálculo de feature vectors.

```
# Calculamos los feature vectors - AlexNet
%cd /content/convnet2/
!python ssearch.py -config configs/sketch.config -name SKETCH -mode compute
```

Luego de las modificaciones mencionadas y el cálculo de los feature vectors, se ejecuta 'ssearch.py' en modo *search* (Figura 12).

Figura 12: Búsqueda por similitud en modo *search*.

```
# Hacemos la búsqueda por similitud - AlexNet
!python ssearch.py -config configs/sketch.config -name SKETCH -mode search
```

iii. ResNet-34 - capa avg-pool

El vector de características de ResNet-34 se extrae con un cambio en el archivo 'resnet.py' del repositorio convnet2. Se comentan las líneas 234 y 235 donde se crea la última capa de clasificación, entonces se obtiene la capa de average pooling (avg-pool), que es la capa que se utilizará como vector de características.

Por otro lado, se agrega la línea "CKPFILE = /content/convnet2/snapshots/020.h5" en el archivo de configuración 'aneira_tfr.config' creado anteriormente. De esta forma se cargan los pesos de la última época entrenada con ResNet-34.

Finalmente, se modifica el archivo 'ssearch.py' del repositorio convnet2. Primero, se agrega la línea "from models import resnet" en la sección de importación de paquetes. Luego, se define el modelo ResNet-34 en el constructor de la clase SSearch como se muestra en la Figura 13.

Figura 13: Definición modelo ResNet-34 en constructor clase SSearch.

```
# loading classifier model
model = resnet.ResNet([3,4,6,3],[64,128,256,512], self.configuration.get_number_of_classes(), se_factor = 0)
input_image = tf.keras.Input((self.input_shape[0], self.input_shape[1], self.input_shape[2]),
                               name = 'input_image')
model(input_image)
model.summary()
model.load_weights(self.configuration.get_checkpoint_file(), by_name = True, skip_mismatch = True)
self.sim_model = model
print('sim_model was loaded OK')
# defining process arch
self.process_fun = imgproc.process_sketch
```

Todos estos cambios (indicados en las clases prácticas) se registran en la rama git *ssearch*. Posteriormente, en colab se realiza el cálculo de feature vectors y la búsqueda por similitud, como indican las figuras 14 y 15 respectivamente.

Figura 14: Cálculo de feature vectors ResNet-34.

```
# Calculamos los feature vectors ResNet-34 <1 minuto
%cd /content/convnet2/
!git checkout ssearch
!python ssearch.py -config configs/aneira_tfr.config -name SKETCH -mode compute
```

Figura 15: Búsqueda por similitud en modo *search*. ResNet-34.

```
# Hacemos la búsqueda por similitud ResNet-34
%cd /content/convnet2/
!git checkout ssearch
!python ssearch.py -config configs/aneira_tfr.config -name SKETCH -mode search
```

c. Cálculo de Métricas para Búsqueda por Similitud

En la presente sección se muestran las métricas que se utilizan para evaluar el rendimiento de los modelos, cómo se definió el cálculo de las métricas en el código, así como la implementación de la técnica leave-one-out y las distancias utilizada para comparar los sketches.

i. Métricas a utilizar

Las métricas que se utilizan para evaluar los resultados de los modelos en búsqueda por similitud serán las siguientes:

- **Mean Average Precision (mAP)**

La definición de Average Precision corresponde al promedio de la precisión de todos los valores relevantes. Formalmente se puede escribir de la siguiente manera:

$$f_{\Gamma_q}(x) = \{1, \text{ si } x \text{ es relevante}; 0, \text{ si } x \text{ no es relevante}\}$$

$$pr_q(i; R) = \frac{\sum_{k=1}^i f_{\Gamma_q}(r_k)}{i} f_{\Gamma_q}(r_i)$$

$$AP_q(R) = \frac{\sum_{i=1}^{|R|} pr_q(i; R)}{|\Gamma_q|}$$

Donde:

q: Solicitud hecha al modelo o query.

Γ_q : Conjunto de valores relevantes para la query q.

R: Ránking de la query q.

pr_q : precisión obtenida en la query q.

i: número de recuperados.

AP_q : Average Precision de la query q.

Al obtener el average precision para todas las queries q , se puede calcular el mean Average Precision (mAP):

$$mAP = \frac{\sum_{q \in Q} AP_q(R)}{|Q|}$$

Donde Q corresponde a todos los queries sobre los cuales se quiere calcular la métrica mAP.

La métrica mAP se calcula por clase de animal y para el total del conjunto de test.

- Precision@1

Precision@1 corresponde a la precisión (definida en el punto anterior) del modelo para el primer relevante. Es decir, sólo nos interesa el primer relevante del ranking y la posición en que aparece. Para calcular el precision@1 de la clase y el total se promedian los valores obtenidos para cada query q .

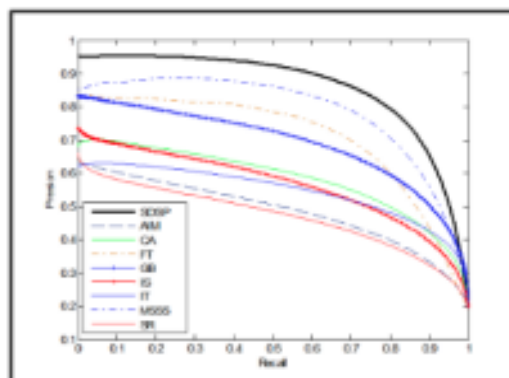
La métrica precision@1 se calcula por clase y para el total del conjunto de test.

- Gráfico Recall-Precision:

El gráfico de recall-precision, como su nombre lo indica, muestra la relación que existe entre la precisión y el recall. Se utiliza la definición de precisión que se mostró en los puntos anteriores. Recall corresponde a un porcentaje de recuperación de objetos relevantes. En este trabajo se obtienen 10 valores de recall correspondientes al porcentaje de recuperados desde 10% a 100%, de forma equiespaciada. Un ejemplo de gráfico recall-precision se muestra en la Figura 16.

El gráfico recall-precision se calcula para el total de las consultas incluidas en el conjunto de test.

Figura 16: Ejemplo de gráfico Recall-Precision.



Fuente: Clase 4-5 Profesor José Saavedra.


```

# Agregar a una lista el average precision,
# suma de precision sobre los relevantes
AP_animal.append(prec/relevantes)
#print(AP[q])
q += 1

# Para obtener recall y precision en cada valor
relevantes_total = relevantes
relevantes_recall = [math.floor(i*relevantes_total) for i in recall]
for fquery in filenames:
    cat_fquery = fquery.split("/")[2]
    # Repetir cálculo de la precisión por recall
    if animal == cat_fquery:
        im_query = ssearch.read_image(fquery)
        idx = ssearch.search(im_query)
        r_filenames = ssearch.get_filenames(idx)
        #r_filenames.insert(0, fquery)
        relev_recall_fquery = 0
        recall_count = 1
        #nqueries_animal +=1
        #prec_recall = 0
        for ix,cat in enumerate(r_filenames):
            rank_pos = ix + 1

            if cat.split("/")[2] == cat_fquery:
                relev_recall_fquery += 1
                if relev_recall_fquery == relevantes_recall[recall_count]:
                    prec_recall = relev_recall_fquery/rank_pos
                    precision_recall[recall_count] += prec_recall
                    recall_count += 1

prec_at_1_animal = sum(at_1)/len(at_1)
mAP_animal = sum(AP_animal)/len(AP_animal)
print('mAP: ', animal, mAP_animal)
print('p@1: ', animal, prec_at_1_animal)
prec_at_1.append(prec_at_1_animal)
mAP.append(mAP_animal)

prec_at_1_total = sum(prec_at_1)/len(prec_at_1)
mAP_total = sum(mAP)/len(mAP)
print('mAP total: ', mAP_total)
print('p@1 total: ', prec_at_1_total)
precision_recall = [i/len(filenames) for i in precision_recall]
precision_recall[0] = 1
print('precision: ', precision_recall)
print('recall: ', recall)

```

iii. Implementación leave-one-out y distancias

La implementación de la técnica leave-one-out se realizó en el archivo ‘ssearch.py’ del repositorio convnet2. En específico se implementó en el retorno de los índices de la función search. Se observa que se retorna desde el segundo elemento del ranking en adelante, ya que el primer elemento siempre será la misma query (la distancia entre una query y sí misma es cero). Por otro lado, en la misma función search se incluye la distancia a utilizar: euclidiana o coseno. Las dos configuraciones se muestran en la Figura 15. En esta figura la distancia utilizada es la distancia euclidiana, mientras que para ocupar la distancia coseno se deben descomentar las líneas que comienzan con “sim = ...” y la línea “idx_sorted = np.argsort(-sim)”, así como comentar la línea que empieza con “d = ...” y la línea “idx_sorted = np.argsort(d)”.

Figura 18: Función search de la clase SSearch, descrita en 'sssearch.py'.

```
def search(self, im_query):
    assert self.enable_search, 'search is not allowed'
    q_fv = self.compute_features(im_query, expand_dims = True)
    #sim = np.matmul(self.normalize(self.features), np.transpose(self.normalize(q_fv)))
    #sim = np.reshape(sim, (-1))
    #it seems that Euclidean performs better than cosine
    d = np.sqrt(np.sum(np.square(self.features - q_fv[0]), axis = 1))
    #idx_sorted = np.argsort(-sim)
    idx_sorted = np.argsort(d)
    #return idx_sorted[:90]
    return idx_sorted[1:]
```


4. Resultados Experimentales y Discusión

La presente sección muestra los resultados experimentales obtenidos con ambas redes neuronales entrenadas, AlexNet y ResNet-34. Con el objetivo de facilitar la comparación entre los modelos, se presentan los resultados de entrenamiento, búsqueda por similitud y cada métrica para ambas redes y posteriormente se realiza la discusión de los resultados obtenidos.

a. Entrenamiento de los modelos

i. AlexNet

El entrenamiento de la red AlexNet obtuvo un accuracy de **74.83%** y un valor de pérdida igual a **1.0628** en el conjunto de validación.





ii. ResNet-34

El entrenamiento de la red ResNet-34 obtuvo un accuracy de **75.08%** y un valor de pérdida igual a **1.1707** en el conjunto de validación.

iii. Ejemplos de predicciones de los modelos

Se realiza una inspección de los resultados del entrenamiento de los modelos, verificando las predicciones que se obtienen ante distintas queries.

Tabla 2: Resultado de predicción para 4 ejemplos del conjunto de prueba.

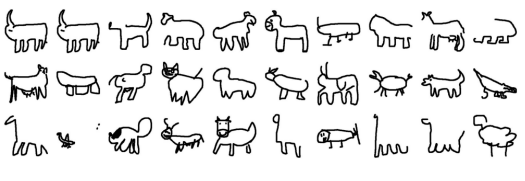
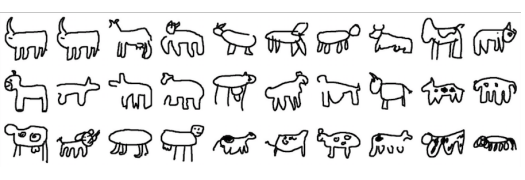
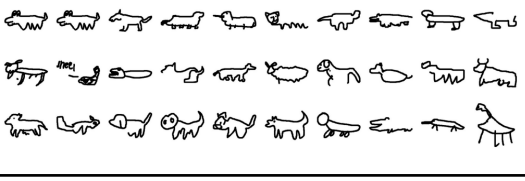
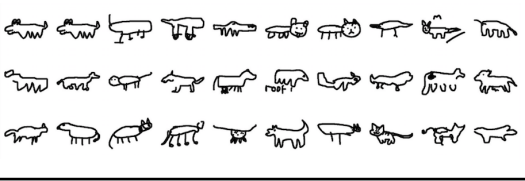
 camel	 crab	 elephant	 giraffe
AlexNet: camel [0.998]	AlexNet: crab [0.999]	AlexNet: elephant [0.882]	AlexNet: giraffe [0.986]
ResNet-34: camel [0.999]	ResNet-34: crab [0.999]	ResNet-34: elephant [0.995]	ResNet-34: giraffe [0.926]

b. Búsqueda por similitud

i. Ejemplos de búsqueda por similitud

En la Tabla 3 se presentan dos ejemplos que se obtienen de la consulta en ambas redes entrenadas. Se observa que, en general, los primeros sketches del ranking corresponden a la clase de la búsqueda, pero a medida que se avanza en los resultados de la búsqueda, se observan sketches de otras clases. En particular, la clase dog tiene más problemas, ya que existen otras clases que son similares, como cat o bear.

Tabla 3: Resultado de búsqueda por similitud para 2 ejemplos del conjunto de prueba.

cow	AlexNet	
	ResNet-34	
dog	AlexNet	
	ResNet-34	

ii. Métricas de evaluación

- Mean Average Precision

La Tabla 4 presenta la métrica mean average precision para cada clase y para el total de elementos de entrenamiento. Se muestran los resultados de los dos modelos entrenados, ResNet-34 y AlexNet, donde se hicieron pruebas con distancia euclidiana y coseno.

Tabla 4: Mean Average Precision (mAP) por clase y total.

	mAP (%)			
Red:	ResNet-34		AlexNet	
Distancia:	euclidiana	coseno	euclidiana	coseno
sheep	59.71	68.62	37.98	51.43
bear	36.18	38.28	22.69	24.77
bee	68.79	72.74	52.15	61.64
cat	42.15	53.16	31.33	34.10
camel	66.19	68.47	48.93	57.75
cow	38.3	44.18	26.51	29.73
crab	47.36	54.42	33.11	42.37
crocodile	68.38	74.66	67.74	71.93
duck	56.00	60.8	26.89	34.57
elephant	39.09	43.52	23.04	28.77
dog	20.78	20.77	15.39	14.21
giraffe	80.75	85.28	67.87	75.63
total	51.96	57.06	37.8	43.91

- Precision@1

La Tabla 5 presenta la métrica Precision@1 para cada clase y para el total de elementos de entrenamiento. Se muestran los resultados de los dos modelos entrenados, ResNet-34 y AlexNet, donde se hicieron pruebas con distancia euclidiana y coseno.

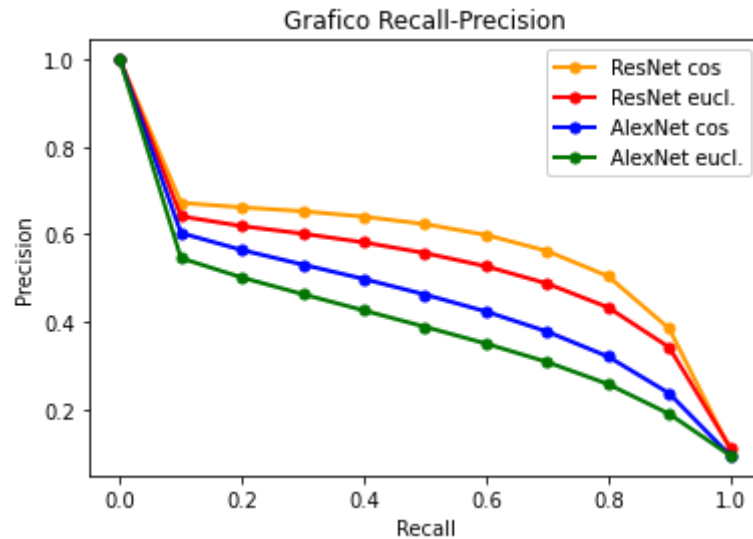
Tabla 5: Precision@1 por clase y total.

	Precision@1 (%)			
Red:	ResNet-34		AlexNet	
Distancia:	eucl.	cos	eucl.	cos
sheep	84.43	82.00	79.14	83.37
bear	66.08	62.38	63.37	62.85
bee	87.57	85.89	81.11	85.78
cat	71.10	78.48	68.47	68.98
camel	82.80	82.39	82.86	86.11
cow	64.47	68.96	65.24	66.20
crab	77.53	78.76	72.77	77.27
crocodile	83.18	84.83	84.87	86.88
duck	78.73	80.05	75.55	78.61
elephant	66.88	69.26	61.78	66.24
dog	47.38	47.44	49.44	46.05
giraffe	91.49	90.71	91.82	94.08
total	75.13	75.93	73.03	75.21

- Gráfico Recall-Precision

El gráfico de la Figura 18 presenta las curvas recall-precision para las redes entrenadas ResNet-34 y AlexNet, considerando distancia coseno y distancia euclidiana.

Figura 18: Gráfico Recall-Precision de modelos ResNet-34 y AlexNet, distancia euclidiana y coseno.



c. Discusión

El entrenamiento en ambas redes neuronales convolucionales, AlexNet y ResNet-34, es muy similar, con un accuracy cercano al 75%.

La principal observación que se puede obtener de los resultados anteriores, específicamente del gráfico recall-precision, es que el modelo ResNet-34 tiene un mejor rendimiento que AlexNet para el dataset QuickDraw-Animals. Además, a modo general, para los sketches es mejor la distancia coseno que la distancia euclidiana.

En el gráfico de la Figura 18, se puede observar que el comportamiento de la curva recall-precision de la red ResNet-34 evidencia un mejor ordenamiento de los relevantes, teniendo una baja importante de la precisión a partir del 80% de recall. En cambio, la red AlexNet presenta un descenso lineal de la precisión en función del recall, lo que indica que tiene un peor rendimiento en las búsquedas por similitud. En este contexto, ResNet-34 (especialmente con distancia coseno) funcionaría mejor en búsquedas por similitud con ranking.

El mAP de ResNet-34 es aproximadamente un 14% mayor que el mAP obtenido con la red AlexNet, en el caso de la comparación utilizando distancia euclidiana, y un 12% mayor utilizando distancia coseno. La comparación entre las distancias euclidiana y coseno, considerando una misma red neuronal, indica que la distancia coseno es mejor que la distancia euclidiana para este dataset.

La métrica Precision@1 muestra que ambos modelos tienen una precisión al primer elemento relevante cercano al 75% promedio para las dos distancias probadas.

El resultado de Precision@1 indica que no existe una gran diferencia entre modelos si lo que queremos es obtener sólo un sketch relevante (de la misma clase) similar a la query solicitada. Sin embargo, los valores de mAP y el gráfico recall-precision muestran que la red ResNet tiene un mejor comportamiento para obtener todos los relevantes, por lo que serviría en aplicaciones donde se necesite una gran variedad de sketches similares al de la query.

Por otro lado, es importante la naturaleza de los dibujos que se están utilizando para hacer la comparación. Las clases de animales que tienen características muy distintivas, como la jirafa, camello, abeja y cocodrilo, tienen resultados de mAP sobre el 60% y de precision@1 sobre el 80%, esto quiere decir que son más fáciles de distinguir para la red. Por el contrario, las clases de animales que no tienen características tan distintivas, que son muy similares a otras clases o que tienen una formas muy diversas de ser representados, no logran buenos resultados en la búsqueda por similitud, como perro, oso, vaca o gato.

5. Conclusiones

El entrenamiento y utilización de los modelos AlexNet y ResNet-34 para la búsqueda por similitud en un dataset de sketches de animales, permite algunas realizar conclusiones importantes acerca del uso de las redes neuronales convolucionales con datasets complejos como el utilizado en esta tarea.

Una primera observación es que los valores obtenidos en el accuracy de entrenamiento no reflejan el desempeño que tendrá la red para realizar una búsqueda por solicitud. En el presente trabajo los valores de entrenamiento fueron similares, sin embargo, las métricas para evaluar la búsqueda por similitud, como el mAP o el gráfico recall-precision son distintos.

Dependiendo del problema que se necesite resolver o la aplicación que se quiera desarrollar, será la métrica que se utilizará para evaluar los modelos o arquitecturas que se tengan como opción. Por ejemplo, en el trabajo desarrollado, el Precision@1 es muy similar en ambos modelos, por lo que si esta es la métrica importante, entonces se puede escoger por otros factores, como puede ser la velocidad de procesamiento. Por otro lado, la red ResNet-34 tiene un mejor rendimiento evaluado mediante la métrica mAP o con el gráfico precision-recall, esto significa que obtiene los relevantes de cada clase antes que la red AlexNet.

6. Referencias

El código utilizado está disponible en:

Alday, Sebastián. n.d. "Tareal: Reconocimiento Visual." Tareal:

Reconocimiento Visual.

<https://colab.research.google.com/drive/1x0-m3ThiDlRT6Stp1bYaq3MxvBUZ5jil>.

Neira, Alvaro. n.d. "convnet2." convnet2.

<https://github.com/alvaro-neira/convnet2.git>.

Neira, Alvaro. n.d. "euclidian.ipynb." euclidian.ipynb.

<https://github.com/alvaro-neira/image-processing-homework1/blob/main/euclidian.ipynb>.

Neira, Alvaro. n.d. "full_cosine.ipynb." full_cosine.ipynb.

https://github.com/alvaro-neira/image-processing-homework1/blob/main/full_cosine.ipynb.