Dog Recognition - Identificación de Perros Convolutional Neural Networks (CNNs)

Este proyecto tiene como misión hacer uso de redes convolucionales para la identificación de perros haciendo uso de redes neuronales profundas convolucionales. Las capas convolucionales actúan como extractores de características.

Con la intención de ahorrar tiempo, se pretende hacer uso de modelos ya entrenados de Deep Learning para la identificación de este tipo de mascotas.

Para ello, se realiza Transfer Learning y Fine-Tuning de los modelos de Oxford VGGFace a través del endopoint de TensorFlow. El código de este repositorio soporta tanto los modelos **VGG16** como **RESNET50** o **SENET50**:

```
from keras_vggface.vggface import VGGFace

# Based on VGG16 architecture -> old paper(2015)
vggface = VGGFace(model='vgg16') # or VGGFace() as default

# Based on RESNET50 architecture -> new paper(2017)
vggface = VGGFace(model='resnet50')

# Based on SENET50 architecture -> new paper(2017)
vggface = VGGFace(model='senet50')
```

Tanto en su versión:

```
include_top=False
```

que permiten quedarte sólo con la parte convolucional de la red para hacer un stacking superior de un clasificador a placer

Como en la versión,

```
include_top=True
```

la cual incluye la parte de clasificación original con todas las capas densas, lo que lo hace más pesado.

Los pesos para los modelos se pueden obtener en el siguiente enlace.

Setup del entorno

Se recomienda hacer uso de Anaconda para la creación de un entorno virtual.

```
$ conda create --name <env> --file requirements.txt
```

Ejemplo de uso

1. Data Cleaning (opcional)

Para evitar la baja varianza entre imágenes se emplea la medida del índice de similitud estructural (SSIM) para medir la similitud entre fotografías. Esto ayuda a evitar datos muy similares (casi idénticos en las particiones de datos de validación y entrenamiento.

En los subdirectorios anidados de `./dataset` se checkea una imagen contra todas las demás y se van eliminando aquellas que sean similares por encima de un valor de similitud (entre `0` y `1`) indicado por el usuario.

```
$ python ssim.py --dir "dir/to/images" --threshold 0.95
```

2. Entrenamiento

Ejemplo del entrenamiento de un dataset de imágenes:

```
$ python training.py --granja test --model resnet50 --epochs 20 --batch_size 30
```

La rutina realiza el entrenamiento, guarda el mejor checkpoint y devuelve un reporte de clasificación sobre el test dataset.

2.1. Customización de arquitectura para Transfer Learning

El script `training.py` se lanza tal y como se muestra arriba. Este script entrena una Red Neuronal convolucional que puede ser `vgg16`, `resnet50` o `senet50` y que acaba en un clasificador que, por defecto, tiene la siguiente implementación `Sequential` de Keras:

```
self.x = self.last_layer
self.x = Dense(self.hidden_dim_1, activation='relu', name='fc1')(self.x)
self.x = Dense(self.hidden_dim_2, activation='relu', name='fc2')(self.x)
self.x = Dense(self.hidden_dim_3, activation='relu', name='fc3')(self.x)
self.out = Dense(self.nb_class, activation='softmax', name='out')(self.x)
```

Clasificador de capas densas totalmente conectadas que se meten tras la capa `flatten` de los modelos convolucionales.

`TODO:` ver si la regularización por Dropout mejora la performance de los modelos.

2.2. Congelación de capas para Fine-Tuning

Normalmente, para Transfer Learning y Fine-Tuning de modelos con dataset pequeños, lo que se hace es congelar la arquitectira transferida y entrenar sólamente el clasificador customizado por nosotros. El número de capas a definir como entrenables se especifica en la función `main()` en la línea `277`:

- `nb_freeze = None` indica que no se congela ninguna capa. Todas las capas son entrenables.
- `nb_freeze = 10` indica que se congelan las 10 primeras capas. Las restantes son entrenables por defecto.
- `nb_freeze = -4` indica que se congelan todas menos las 4 últimas capas. Las restantes son entrenables por defecto.

2.3. Dataset

El dataset sobre el que se desea entrenar debe situarse en la carpeta `./dataset`. Para cada clase, se deben agrupar todas las imágenes en subdirectorios.

Los batches de entrenamiento, validación, así como el núemro de clases a predecir y, por lo tanto, la arquitectura de salida del modelo, están definidas tanto por el generador `ImageDataGenerator()` como por la función `flow_from_directory()`.

Sobre el dataset disponible se hace data augmentation:

- **Rotación aleatoria** de cada imagen de hasta `15 grados`;
- **Desplazamiento en altura y anchura** de hasta el `5%` de la dimensión de la imagen;
- **Horizontal flipping**.

2.4. Logueo del entrenamiento

Para el entrenamiento se han definido dos callbacks:

- **EarlyStopping** para evitar overfitting o alargar innecesariamente el tiempo de entrenamiento.
- **TensorBoard Callback** que permite logar precisión y funciones de pérdida para su visualización en browser de las curvas de aprendizaje y del grafo creado y entrenado.

```
$ cd logs/folder_tbd/
$ tensorboard --logdir=./ --port 6006
```

De manera que con sólo ir a tu navegador a `http://localhost:6006/` se puede visualizar cómo ha transcurrido el entrenamiento. Ver el siguiente artículo para aclarar dudas.

3. Testeo

De cara al testeo de un modelo ya entrenado con una imagen de muestra, se ejecuta el script `testing.py`:

```
$ python testing.py --granja test  --img "path/to/img"
```

Esta rutina devuelve las predicciones de una imagen en base a todas las clases soportadas por el modelo. La rutina devuelve:

```
$ python testing.py --granja test  --img "path/to/img"
{
    "class 0": score 0,
    "class 1": score 1,
    ...
```

```
    "class N": score N
}
```

4. Grad-CAM Checking

Un inconveniente frecuentemente citado del uso de redes neuronales es que entender exactamente lo que están modelando es muy difícil. Esto se complica aún más utilizando redes profundas. Sin embargo, esto ha comenzado a atraer una gran cantidad de interés de investigación, especialmente para las CNNs para garantizar que la "atención" de la red se centre en las características reales y discriminativas del propio animal, en lugar de otras partes de la imagen que puedan contener información discriminatoria (por ejemplo, una etiqueta de clase, una marca de tiempo impresa en la imagen o un borde sin interés).

Para comprobar que nuestro modelo se centra en partes interesantes de la imagen, se puede elegir una imagen de prueba y comprobar qué regiones son de interés para la red neuronal, se puede ejecutar:

```
$ python grad_CAM.py --granja test --model resnet50 --img "path/to/img"
```

Lo cual te devuelve un mapa de calor sobre la imagen de las regiones de interés.

Entrenamiento y testeo con Flickr-Dog Dataset

1. Obtención del dataset

El primer punto crítico es conseguir fotos de perros. Usamos en primer caso de **Flickr-dog** seleccionando fotos de perros de Flickr disponibles bajo licencias Creative Commons. Se puede descargar el dataset aquí. En este caso, las caras de los perros fueron recortadas y giradas para alinear los ojos horizontalmente y fueron redimensionadas a `250x250 píxeles`.

El dataset está compuesto por dos razas de perros: pugs y huskies. Esas razas fueron seleccionadas para representar los diferentes grados de desafío: los pugs con difíciles de identificar y los huskies son un bastante más fáciles de diferenciar.

Para cada raza hay `21 individuos`, cada uno con al menos 5 fotos. Los individuos fueron etiquetados interpretando los metadatos de las imágenes (usuario, título, descripción, marcas de tiempo, etc.). En total, el conjunto de datos de Flickr-dog tiene `42 clases` y `374 fotos`.

Hemos eliminado 2 perros (un huskie y un pug) debido a que la calidad de las imágenes provista era muy mala y hemos añadido otros dos: `Perrete_1` y `Perrete_2`, de los que se ha sacado un buen número de imágenes.

2. Proceso de Data Augmentation

Como el número de imágenes es reducido para cada clase, primero se ha hecho un Data Augmentation sobre los datos del Flickr-dog dataset. De esta manera se han conseguido `4,876 imágenes`, de las cuales se tomarán el `80%` como training set y el `20%` restante como test/validation set.

3. Entrenamiento

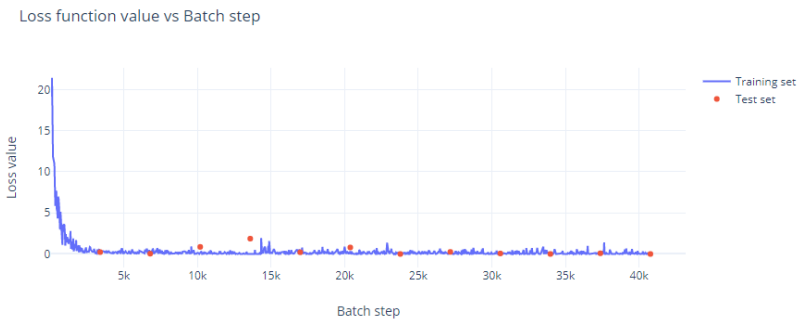
Se ha lanzado un entrenamiento de prueba con el modelo `VGG16` con un tamaño de batch de `32` para un límite de `20 épocas`. Para prevenir overfitting, se ha recurrido a un EarlyStopping de `5 épocas` monitorizando la función de pérdida del conjunto de validación.

```
$ python training.py -ng flickr-dog -m vgg16 -e 20 -bs 32
```

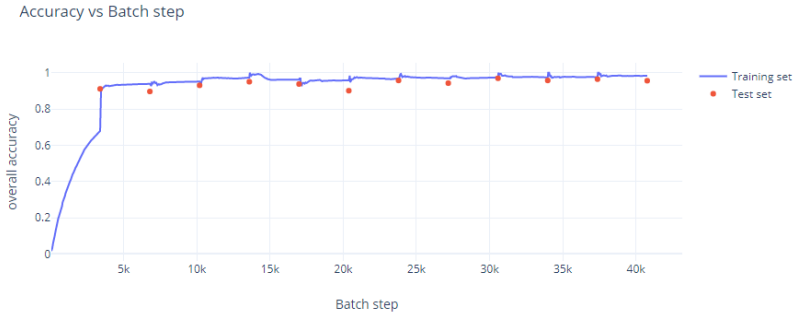
Los resultados del entrenamiento se ofrecen a continuación.

4. Resultados

4.1. Evolución de la función de pérdida durante el entrenamiento



4.2. Evolución de la precisión durante el entrenamiento



4.3. Precision, Recall y F1-Score para cada una de las 42 clases

Pet Name	precision	recall	f1-score	support
Apolo	1.0	1.0	1.0	34.0
Bailey	0.941	1.0	0.969	32.0
Bandit	0.944	1.0	0.971	34.0
Bella	1.0	1.0	1.0	32.0
Bella-Kappel	1.0	0.878	0.935	33.0
Bella-Theresa	1.0	0.848	0.918	33.0
Bu-chan	0.9166	1.0	0.956	33.0
Chloe	1.0	1.0	1.0	32.0
Clara	1.0	0.848	0.918	33.0
Cody	1.0	1.0	1.0	32.0
Dakota	1.0	1.0	1.0	33.0
Dakota-CNC	1.0	1.0	1.0	31.0
Doris	1.0	0.939	0.96875	33.0
Duke	0.911	0.885	0.898	35.0
Eve	1.0	0.939	0.96875	33.0
Gracie	0.785	0.970	0.868	34.0
Injun	0.970	1.0	0.985	33.0
Kira	1.0	0.903	0.949	31.0
Landy	0.970	1.0	0.985	33.0
Lulu	1.0	1.0	1.0	33.0
Meeshka	0.944	1.0	0.971	34.0
Monty	0.935	0.852	0.892	34.0
Newton	0.968	1.0	0.984	31.0
Nikita	0.969	1.0	0.984	32.0
Oliver	0.970	1.0	0.985	33.0
Peaches	0.969	0.914	0.941	35.0
Perrete_1	0.981	0.852	0.912	61.0
Perrete_2	0.848	1.0	0.917	67.0
Princesa	1.0	1.0	1.0	32.0
Purcy	1.0	0.941	0.969	34.0
Rave	1.0	1.0	1.0	32.0
Rico	0.941	0.969	0.955	33.0
Ridley	0.944	0.944	0.944	36.0
Roscoe	0.942	1.0	0.970	33.0
Skipper	1.0	1.0	1.0	32.0
Skye	1.0	0.969	0.984	33.0
Tala	0.888	1.0	0.941	32.0

Pet Name	precision	recall	f1-score	support
Wilco	1.0	1.0	1.0	31.0
Winston	1.0	0.878	0.935	33.0
Yoda	0.888	0.914	0.901	35.0
Zeus-Rekel	1.0	1.0	1.0	32.0
Zowie	1.0	1.0	1.0	34.0
accuracy	0.961	0.961	0.961	0.961
macro avg	0.967	0.963	0.963	1446.0
weighted avg	0.964	0.961	0.961	1446.0

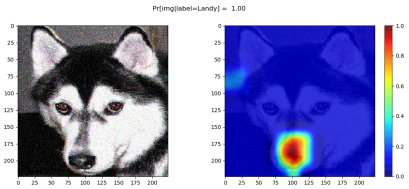
4.4. Matriz de confusión del modelo entrenado

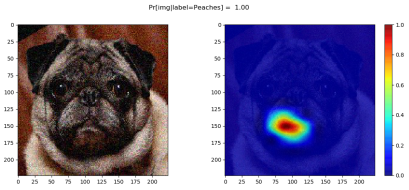
FIELD1	Apolo	Bailey	Bandit	Bella	Bella-Kappel	Bella-Theresa	Bu-chan	Chloe	Clara	Cody	Dakota	Dakota-CNC	Doris	Duke	Eve	Gracie
Apolo	34	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Bailey	0	32	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Bandit	0	0	34	0	0	0	0	0	0	0	0	0	0	0	0	0
Bella	0	0	0	32	0	0	0	0	0	0	0	0	0	0	0	0
Bella-Kappel	0	0	0	0	29	0	0	0	0	0	0	0	0	0	0	0
Bella-Theresa	0	1	1	0	0	28	0	0	0	0	0	0	0	0	0	0
Bu-chan	0	0	0	0	0	0	33	0	0	0	0	0	0	0	0	0
Chloe	0	0	0	0	0	0	0	32	0	0	0	0	0	0	0	0
Clara	0	0	0	0	0	0	0	0	28	0	0	0	0	0	0	1
Cody	0	0	0	0	0	0	0	0	0	32	0	0	0	0	0	0
Dakota	0	0	0	0	0	0	0	0	0	0	33	0	0	0	0	0
Dakota-CNC	0	0	0	0	0	0	0	0	0	0	0	31	0	0	0	0
Doris	0	0	0	0	0	0	0	0	0	0	0	0	31	0	0	1
Duke	0	0	0	0	0	0	0	0	0	0	0	0	0	31	0	3
Eve	0	1	1	0	0	0	0	0	0	0	0	0	0	0	31	0
Gracie	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	33
Injun	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Kira	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Landy	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Lulu	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Meeshka	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Monty	0	0	0	0	0	0	1	0	0	0	0	0	0	1	0	0
Newton	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Nikita	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Oliver	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Peaches	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	2
Perrete_1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Perrete_2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Princesa	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Purcy	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Rave	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Rico	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0
Ridley	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

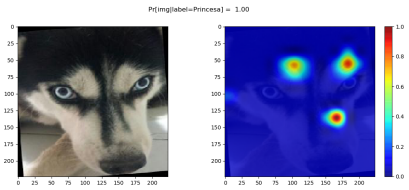
FIELD1	Apolo	Bailey	Bandit	Bella	Bella-Kappel	Bella-Theresa	Bu-chan	Chloe	Clara	Cody	Dakota	Dakota-CNC	Doris	Duke	Eve	Gracie
Roscoe	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Skipper	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Skye	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Tala	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Wilco	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Winston	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2
Yoda	0	0	0	0	0	0	2	0	0	0	0	0	0	0	0	0
Zeus-Rekel	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Zowie	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

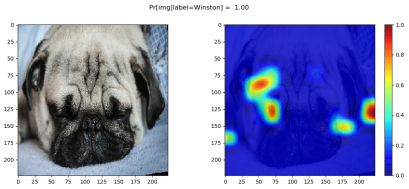
4.5. Grad-CAM visualizations

¿En qué se fija nuestra arquitectura para clasificar a los perros? Veamos algunos ejemplos de **perros correctamente clasificados** por nuestro modelo:





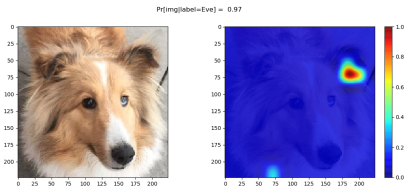




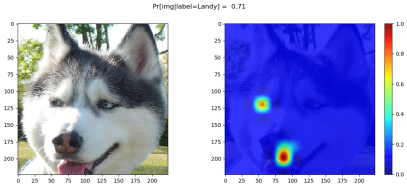
Con estos cuatro ejemplos, parece que nuestro modelo se fija en la fisionomía del rostro de los perros y no en los fondos de las fotografía. Esto quiere decir que nuestro modelo ha aprendido realmente.

Se fija en la zona de la nariz, en el caso de los huskies también en el cambio del color del pelaje alrededor de los ojos. Con los pugs, esas zonas de atención son más dispersas; parece que el gran número de arrugas hace difícil su identificación.

Si le facilitamos al modelo una foto de un perro que no es de las razas que hemos entrenado y que no está en el dataset, obviamente "se perderá".



Se ha fijado en la oreja del perro y, además de decir que es "Eve", un perro cuya matriz de confusión indica que esta mascota no está del todo bien caraterizada, pues si bien se puede confundir con "Bailey" o "Landy" (Huskies) ha habido un caso en el que se ha confundido con "Bandit" (Pug).



De todas maneras, a pesar del fallo, se ha vuelto a fijar principalmente en la zona de la nariz.

Enlaces de Soporte e Interés:

- Keras Framework
- Oxford VGGFace Website
- Arkhi et al.: Deep Face Recognition
- Towards on-farm pig face recognition using convolutional neural networks
- VGGFace implementation with Keras Framework
- Deep Learning For Beginners Using Transfer Learning In Keras
- Transfer learning from pre-trained models
- Transfer Learning using Keras
- Tutorial on using Keras `flow_from_directory()` and generators
- Transfer learning and Image classification using Keras on Kaggle kernels
- A Comprehensive Hands-on Guide to Transfer Learning with Real-World Applications in Deep Learning
- Gradient-weighted Class Activation Mapping - Grad-CAM
- Keras implementation of GradCAM
- Grad-CAM with keras-vis
- A Python module for computing the Structural Similarity Image Metric (SSIM)