

# **P.E.T - Pet Enhancement Transition**



## Contenido:

- [1 Justificación y descripción del proyecto](#)
- [2 Obtención de datos](#)
- [3 Descripción y limpieza de los datos](#)
- [4 Exploración y visualización de los datos](#)
- [5 Preparación de los datos para los algoritmos de Machine Learning](#)
- [6 Entrenamiento del modelo y comprobación del rendimiento](#)
  - [6.1 Comparación de los modelos](#)
  - [6.2 Guardar modelo y clases](#)
  - [6.3 Cargar modelo y clases](#)
  - [6.4 Ejemplo de predicción](#)
- [7 Procesamiento de Lenguaje Natural](#)
- [8 Aplicación web](#)
- [9 Fuentes](#)

## Autores:

- Flores Chamizo Serrano
- Diego Valenzuela Perez

# 1 Justificación y descripción del proyecto

Se realizará un modelo para predecir *razas* de mascotas a partir de una *imagen*. Para ello se han utilizado dos *datasets* encontrados en kaggle, uno de *perro* y otro de *gatos*, que se han obtenido por separado pero trabajamos con los dos *mezclados*.

Para comprobar qué modelo se debería utilizar se ha trabajado con tres diferentes: *VGG19*, *ResNet50* e *InceptionV3*.

Una vez se haya comprobado el modelo con mejor rendimiento (*acierto* cerca del 1 y *pérdida* cerca del -1) se guardará el modelo para su posterior uso en una página web.

La *página* web se ha realizado en la plataforma *streamlit* simulando una página real que podría utilizar una protectora. Tendrá varias pestañas con distinta información, pero los apartados a destacar son:

- **Predictor:** Donde se subirá la *imagen* de una mascota con su *nombre* y una *descripción* y se guardaran estos datos junto con la *predicción* de la mascota.
- **Adoptar un animal:** En este apartado se podrá *seleccionar* si queremos adoptar un *perro* o un *gato* y la *raza* de esta.

Se ha decidido buscar las mascotas para adoptar según su raza ya que, a partir de esta, se sobreentiende si es perro o gato y si es de tamaño pequeño, mediano o grande.

Respecto al *NLP* se ha optado por la *traducción* de texto a dos idiomas (*Inglés* y *Alemán*).

Y para el apartado de *Big Data* se ha optado por realizar una base de datos de *voluntarios* en *MongoDB*.

Este proyecto se ha pensado para facilitar el trabajo de identificación a las protectoras de animales y fomentar la adopción.

## 2 Obtención de datos

Las imágenes se han obtenido de kaggle, dataset [70 Dog Breeds-Image Data Set](#) y [Gano Cat Breed Image Collection](#).

Información de los datasets:

- **Dogs:**
  - 70 razas de perros.
  - Imágenes separadas en *train*, *valid* y *test*, listas para entrenar el modelo.
  - Entre 65 y 198 imágenes por raza para entrenar.
  - 10 imágenes para la validación y el test.
  
- **Cats:**
  - 15 razas de gatos.
  - 375 imágenes por raza.
  - Se deberá tratar para mezclarla con el dataset de perros. Se utilizarán las 10 primeras imágenes para *test*, las 10 siguientes para validación (*valid*) y el resto para entrenamiento (*train*).

### 3 Descripción y limpieza de los datos

Para crear los DataFrame se ha recorrido carpeta a carpeta para identificar:

- **Dogs:**
  1. El tipo de la mascota (*Dog*)
  2. El uso del dataset (*train*, *valid* o *test* )
  3. La raza de la mascota.
  4. Nombre de la imagen.
  5. La ruta de la imagen, se excluye la ruta base (donde se encuentra la carpeta principal del dataset).

	path	breed	type	dataset
0	Dog/test/Great Perenees/10.jpg	Great Perenees	Dog	test
1	Dog/test/Great Perenees/09.jpg	Great Perenees	Dog	test
2	Dog/test/Great Perenees/01.jpg	Great Perenees	Dog	test
3	Dog/test/Great Perenees/03.jpg	Great Perenees	Dog	test
4	Dog/test/Great Perenees/08.jpg	Great Perenees	Dog	test
...	...	...	...	...
9341	Dog/train/Shiba Inu/32.jpg	Shiba Inu	Dog	train
9342	Dog/train/Shiba Inu/46.jpg	Shiba Inu	Dog	train
9343	Dog/train/Shiba Inu/14.jpg	Shiba Inu	Dog	train
9344	Dog/train/Shiba Inu/40.jpg	Shiba Inu	Dog	train
9345	Dog/train/Shiba Inu/84.jpg	Shiba Inu	Dog	train

9346 rows × 4 columns

- **Cats:**
  1. El tipo de la mascota (*Cat*)
  2. La raza de la mascota.
  3. Nombre de la imagen.
  4. Se clasifica el dataset para su uso posterior: *Test* (10 primeras), *valid* (Entre 10 y 20) y *train* (Imágenes restantes)
  5. La ruta de la imagen, se excluye la ruta base (donde se encuentra la carpeta principal del dataset).

	path	breed	type	dataset
0	Cat/Birman/Birman-36689172_1935.jpg	Birman	Cat	test
1	Cat/Birman/Birman_20.jpg	Birman	Cat	test
2	Cat/Birman/Birman-34882234_2339.jpg	Birman	Cat	test
3	Cat/Birman/Birman-33559605_264.jpg	Birman	Cat	test
4	Cat/Birman/Birman_77.jpg	Birman	Cat	test
...	...	...	...	...
5620	Cat/Bombay/Bombay-18376775_8377.jpg	Bombay	Cat	train
5621	Cat/Bombay/Bombay-20394160_7978.jpg	Bombay	Cat	train
5622	Cat/Bombay/Bombay-25252179_6811.jpg	Bombay	Cat	train
5623	Cat/Bombay/Bombay_171.jpg	Bombay	Cat	train
5624	Cat/Bombay/Bombay_111.jpg	Bombay	Cat	train

5625 rows × 4 columns

Una vez identificada toda la información, se añade al DataFrame cuatro columnas:

- **path**: La ruta donde se encuentra la imagen.
- **breed**: Indica la raza de la mascota.
  - **Dogs**: *Chinese Crested, Chihuahua, Collie, Cairn, Bull Terrier, Cocker, Cockapoo, Clumber, Chow, Corgi, Dhole, Dalmation, Coyote, Elk Hound, Doberman, Dingo, Great Dane, Golden Retriever, German Sheperd, French Bulldog, Groenendael, Greyhound, Great Perenees, Japanese Spaniel, Irish Wolfhound, Irish Spaniel, Labrador, Labradoodle, Komondor, Maltese, Malinois, Lhasa, Pekinese, Newfoundland, Mex Hairless, Poodle, Pomeranian, Pit Bull, Rottweiler, Rhodesian, Pug, Scotch Terrier, Schnauzer, Saint Bernard, Shih-Tzu, Shiba Inu, Shar\_Pei, Yorkie, Vizsla, Siberian Husky, African Wild Dog, Basenji, American Spaniel, Afghan, Basset, Bearded Collie, Beagle, Bernaise, American Hairless, Airedale, Border Collie, Borzoi, Bloodhound, Bluetick, Bull Mastiff, Blenheim, Boxer, Boston Terrier, Bichon Frise, Bulldog*
  - **Cats**: *Birman, Russian Blue, Sphynx, American Shorthair, Ragdoll, British Shorthair, Egyptian Mau, Bengal, American Bobtail, Abyssinian, Siamese, Maine Coon, Persian, Tuxedo, Bombay*
- **type**: El tipo de mascota que es, *cat* (gato) o *dog* (perro).
- **dataset**: Indica donde se utilizarán los datos *train* (entrenamiento), *valid* (validación) o *test* (prueba de predicción).

También se ha realizado:

- **df.info()**: Para saber si hay valores nulos en alguna columna de los dos dataframes.
- **df.unique()**: Se utiliza la columna de la raza para sacar los valores únicos que predecirá el modelo (clases).

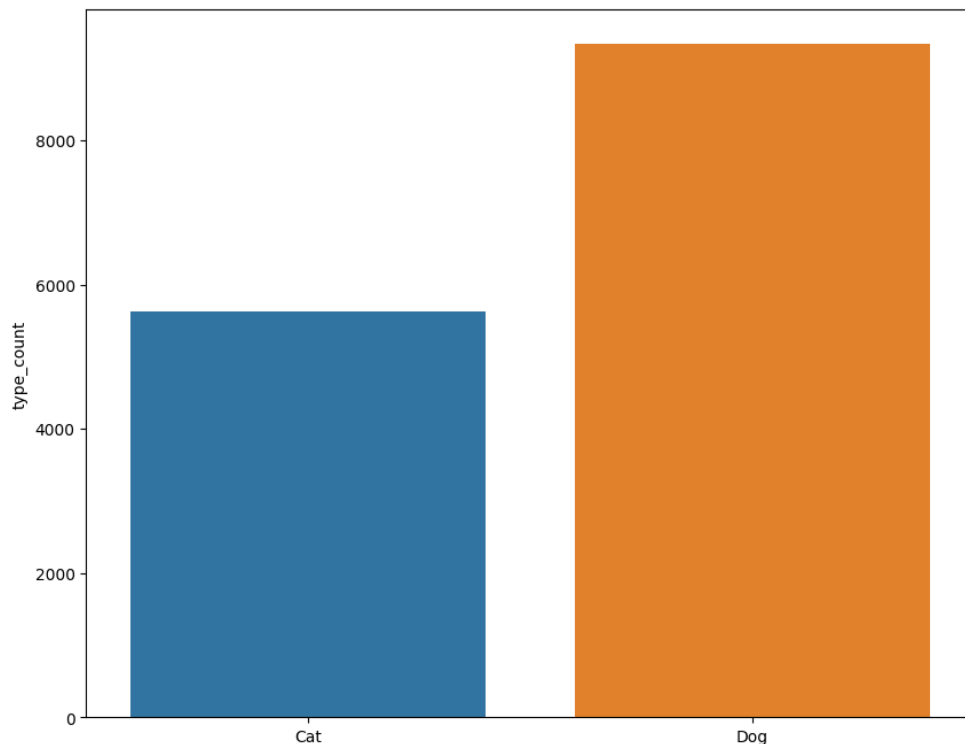
## 4 Exploración y visualización de los datos

Antes de realizar las gráficas, junto los dataframe en uno solo llamado **pets** mediante la función **concat** de pandas.

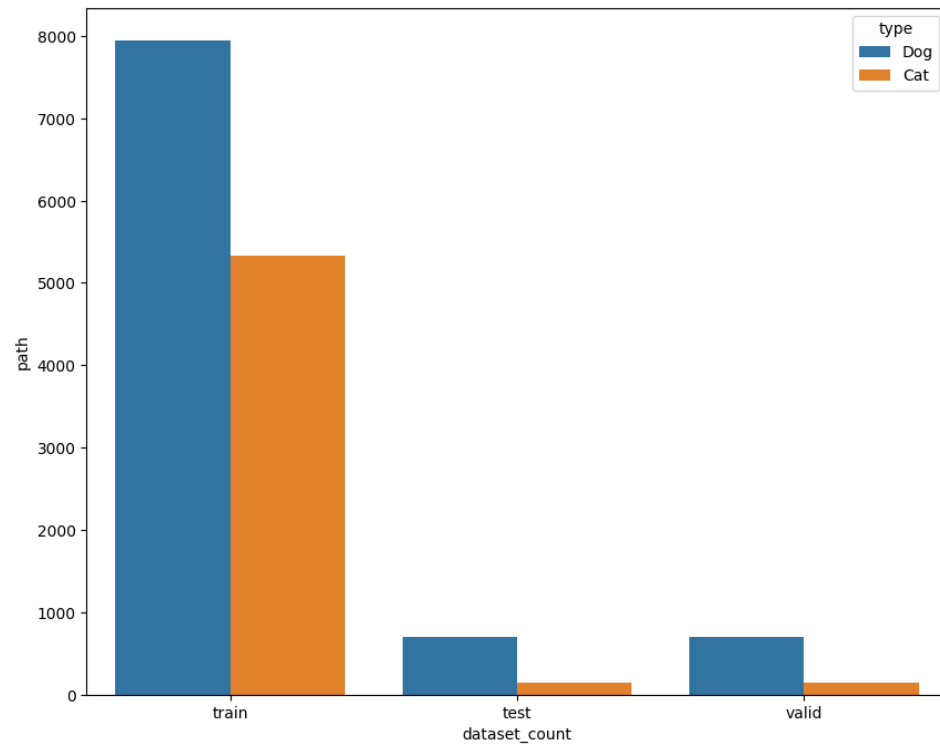
	path	breed	type	dataset
0	Dog/test/Great Perenees/10.jpg	Great Perenees	Dog	test
1	Dog/test/Great Perenees/09.jpg	Great Perenees	Dog	test
2	Dog/test/Great Perenees/01.jpg	Great Perenees	Dog	test
3	Dog/test/Great Perenees/03.jpg	Great Perenees	Dog	test
4	Dog/test/Great Perenees/08.jpg	Great Perenees	Dog	test
...	...	...	...	...
5620	Cat/Bombay/Bombay-18376775_8377.jpg	Bombay	Cat	train
5621	Cat/Bombay/Bombay-20394160_7978.jpg	Bombay	Cat	train
5622	Cat/Bombay/Bombay-25252179_6811.jpg	Bombay	Cat	train
5623	Cat/Bombay/Bombay_171.jpg	Bombay	Cat	train
5624	Cat/Bombay/Bombay_111.jpg	Bombay	Cat	train
14971 rows x 4 columns				

Se ha querido mostrar la cantidad de imágenes que tenemos:

1. **Gatos y perros:** Se puede ver como hay más imágenes de perro que de gatos.

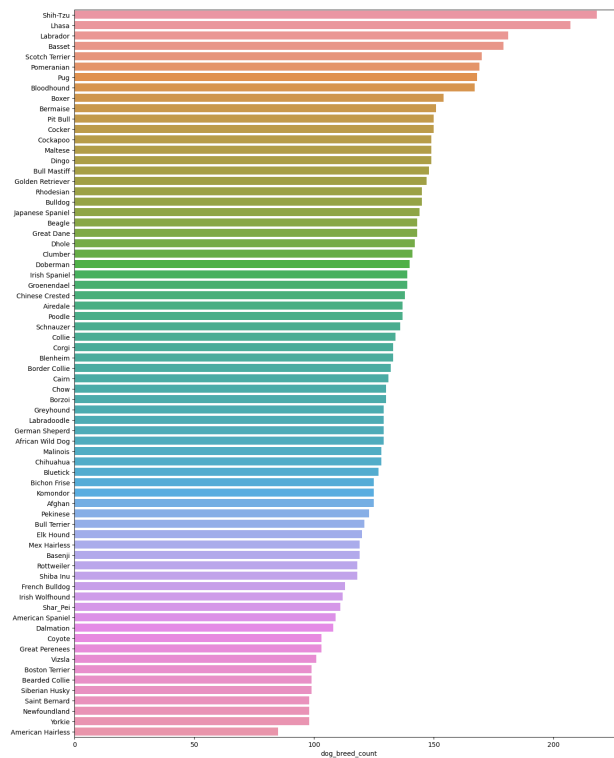


2. **Uso de datos:** Utilizamos una gráfica de doble barra para ver la cantidad de datos que luego usará cada tipo de mascota en el entrenamiento, validación y test del modelo. Se aprecia que hay más imágenes en train que en las otras dos acciones, ya que cuanto más imágenes tengamos para el entrenamiento más exactas podrán ser las predicciones.

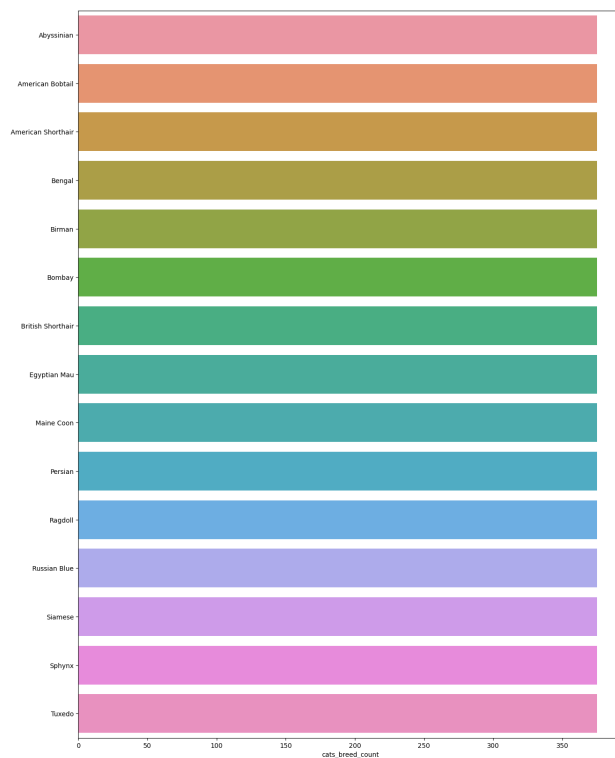


3. **Raza de mascota:** Como se indicó, en la obtención de los datos, las razas de *perros* tienen entre 85 y 218 imágenes y las de *gatos* 375 en cada una. Para esta gráfica se ha preferido utilizar los dataframe por separado para su mejor visualización.

- **Perros:**

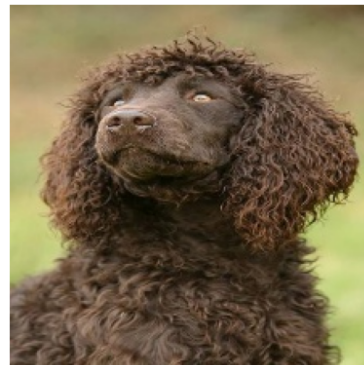


- **Gatos:**



4. **Muestra de imágenes:** Se han elegido cuatro imágenes aleatorias, dos de perro y otras dos de gato, para visualizar algún ejemplo de lo que nos podemos encontrar entre las imágenes.

○ **Perros:**



○ **Gatos:**





## 5 Preparación de los datos para los algoritmos de Machine Learning

Antes de preparar los datos se han generado cuatro variables a partir del dataframe *pets* generado en el apartado de visualización:

- **pets\_b\_uniq**: Lista de razas únicas de todas las mascotas.
- **pets\_train**: Agrupación de la columna dataset que contiene la palabra *train*.
- **pets\_valid**: Agrupación de la columna por *valid*.
- **pets\_test**: Agrupación de la columna por *test*.

Los datos se han preparado mediante la función **ImageDataGenerator** de la librería *keras.preprocessing.image*.

Se ha utilizado cuatro variables:

- **img\_gen**: Esta variable se ha utilizado para inicializar la función con un parámetros:
  - **rescale**: Usado para poner todas las imágenes al mismo tamaño. Se le ha indicado *1./255.* para que permita números decimales al escalar.
- **pets\_gen\_train**: En esta variable se guardan los datos separados en **x** e **y** para el entrenamiento del modelo, se ha utilizado la variable anterior junto a la función *flow\_from\_dataframe* para indicarle que los datos serán obtenidos a partir de un DataFrame. Dentro de esta se utilizan varios parámetros:
  - **dataframe**: Es el DataFrame *pets\_train*.
  - **directory**: Ruta base donde se encuentra el dataset. Si no hubiera, los datos de la columna **x** deberán ser rutas absolutas.
  - **x\_col**: Columna del DataFrame donde se encuentran los datos pasados al modelo (*path*).
  - **y\_col**: Columna del DataFrame donde se encuentra los datos que deberá predecir el modelo (*breed*).
  - **classes**: Lista de etiquetas únicas para la clasificación. Se encuentra en la variable *pets\_b\_uniq* que contiene un *groupby.unique* de las razas de las mascotas.
  - **shuffle**: Se le indica *True* para que los datos estén mezclados y el modelo no pueda aprender secuencialmente.
  - **class\_mode**: Se le indicará *categorical* ya que permite la salida de varias etiquetas.
  - **target\_size**: Indicará el alto y ancho de todas las imágenes (256 x 256).

- **batch\_size**: Tamaño del conjunto de imágenes con el que irá aprendiendo el modelo. Se le ha indicado 64.
- **pets\_gen\_valid**: Utiliza lo mismo que la variable anterior, los únicos parámetros que cambian son:
  - **dataframe**: Uso del dataframe *pets\_valid*.
  - **batch\_size**: Se ha indicado 128 para que vaya un poco más rápido.
- **pets\_gen\_test**: Esta variable se utiliza para la evaluación del modelo. Utiliza los mismos parámetros que las anteriores y algunos que cambian, estos son:
  - **dataframe**: Se utilizan los datos del DataFrame *pets\_test*.
  - **shuffle**: En este caso se indicará *False* porque no es necesario mezclar los datos.
  - **batch\_size**: Su valor será 1 para que se tome su tiempo evaluando las imágenes para el test.

La salida de las variables generadores sería estas:

```
Found 13271 validated image filenames belonging to 85 classes.  
Found 850 validated image filenames belonging to 85 classes.  
Found 850 validated image filenames belonging to 85 classes.
```

## 6 Entrenamiento del modelo y comprobación del rendimiento

Se han utilizado tres modelos: *VGG19*, *RestNet50* e *InceptionV3*.

Para entrenar con los modelos se ha realizado los siguientes pasos:

- **Carga del modelo:** Se llama al modelo con tres parámetros:
  - **weights:** Como los modelos trabajan con imágenes se le indica *imagenet*.
  - **include\_top:** El valor será *False* para que el modelo cargado no utilice la última capa como salida.
  - **input\_shape:** Tamaño de la imagen que utilizará. Se le indica el alto y ancho más el canal de colores donde 3 es a color (256, 256, 3).
- **Detener el entrenamiento del modelo:** Se utiliza la función *trainable* del modelo y se le indica *False* para que no entrene las capas que tiene el modelo al descargarse. Estas capas se utilizarán como base de entrenamiento.
- **Visualización del modelo:**
  - **summary:** Resumen de las capas que componen el modelo, la cantidad de parámetros que utiliza y los que serán entrenados o no. En este caso, al ser la base, no debería aparecer parámetros entrenables.
  - **plot\_model:** Es una función de *tensorflow.keras.utils* para mostrar la estructura que sigue el modelo.
- **Creación de las capa no convolucionales:** Se creará una *Sequential* a la que se le añade con la clase *add* los tensores:
  - **Modelo base:** Modelo cargado en el paso anterior, modificado para que no sea sobreentrenado.
  - **GlobalAveragePooling2D:** Agrupa los datos que recibe. No se le pasa ningún parámetro.
  - **Dense:** Se le indica que se reducirán los datos a 128 salidas en un principio.
  - **Dropout:** Esta capa se utiliza para evitar el *sobreajuste*, se le ha indicado un 0,2 como valor *inicial*, ya que durante el entrenamiento irá cambiando aleatoriamente.
  - **Dense:** Para este último tensor, se indica como salida el total de clases que tenemos para predecir, se ha utilizado *len(pets\_b\_uniq)* para que las extraiga automáticamente. También se le pasa la función de activación (*softmax*).

Una vez añadida las capas, se realiza dos acciones:

- **build:** Llamado patrón de construcción retrasa, es decir, se construye manualmente el modelo. Para ello se le indica *None* (Para que no coja las imagenes por lote) y 256, 256, 3 (Alto, ancho y canal de color de la imagen)
- **summary:** Resumen de los parámetros que componen el modelo, cantidad utilizados, los entrenados y los que no. Aquí sí hay parámetros entrenables, así se vería la salida de cada modelo:

#### ■ ResNet50:

```
Model: "sequential_1"
```

Layer (type)	Output Shape	Param #
resnet50 (Functional)	(None, 8, 8, 2048)	23587712
global_average_pooling2d_1 (GlobalAveragePooling2D)	(None, 2048)	0
dense_2 (Dense)	(None, 128)	262272
dropout_1 (Dropout)	(None, 128)	0
dense_3 (Dense)	(None, 85)	10965

```

=====
Total params: 23,860,949
Trainable params: 273,237
Non-trainable params: 23,587,712
=====

```

#### ■ VGG19:

```
Model: "sequential"
```

Layer (type)	Output Shape	Param #
vgg19 (Functional)	(None, 8, 8, 512)	20024384
global_average_pooling2d_1 (GlobalAveragePooling2D)	(None, 512)	0
dense (Dense)	(None, 128)	65664
dropout (Dropout)	(None, 128)	0
dense_1 (Dense)	(None, 85)	10965

```

=====
Total params: 20,101,013
Trainable params: 76,629
Non-trainable params: 20,024,384
=====

```

#### ■ InceptionV3:

```
Model: "sequential_2"
```

Layer (type)	Output Shape	Param #
inception_v3 (Functional)	(None, 6, 6, 2048)	21802784
global_average_pooling2d_2 (GlobalAveragePooling2D)	(None, 2048)	0
dense_4 (Dense)	(None, 128)	262272
dropout_2 (Dropout)	(None, 128)	0
dense_5 (Dense)	(None, 85)	10965

```

=====
Total params: 22,076,021
Trainable params: 273,237
Non-trainable params: 21,802,784
=====

```

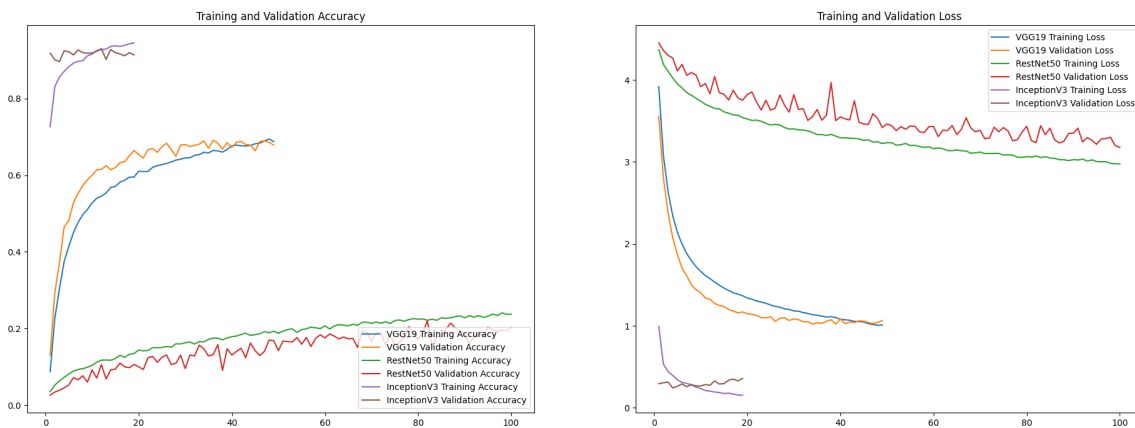
- **Compilación:** Para compilar el modelo se utilizan tres parámetros:
  - **Optimizer:** Utiliza como optimizador el método Adam. Utiliza el descenso de gradiente estocástico, es decir, la estimación adaptativa de momentos de primer y segundo orden.
  - **Loss:** Se utiliza como función de pérdida *categorical\_crossentropy* para que evalúe la pérdida cruzada entre las etiquetas indicadas (dos o más) y las predicciones realizadas. Esto devuelve un dígito decimal.
  - **Metrics:** Lista de métricas para evaluar el modelo durante el entrenamiento y los test. Normalmente se utiliza *accuracy* que es la frecuencia de acierto en las predicciones con respecto a las etiquetas.
- **Creación de punto de parada:** Para ello se utiliza la función *EarlyStopping* de la librería *tensorflow.python.keras.callbacks* y utiliza dos parámetros:
  - **Monitor:** Este parámetro irá mirando el resultado de la variable *val\_loss* al entrenar el modelo.
  - **Patience:** Indica el número de épocas sin mejora y luego se detendrá. Si después de 15 épocas no ha cambiado el *val\_loss* el modelo se detendrá, sin embargo, si cambia un poco volverá a empezar la cuenta.
- **Entrenamiento del modelo:** Para el entrenamiento usamos la función *fit* con unos determinados parámetros:
  - **Datos de entrenamiento:** Se le pasa el generador *pets\_gen\_train* realizado en la preparación de los datos (Esto contiene la x y la y en una misma variable).
  - **Validation\_data:** Utiliza el *pets\_gen\_valid* creado en la preparación de los datos para que vaya evaluando la pérdida y las métricas del modelo al final de cada época.
  - **callbacks:** Esto se realiza para que deje de entrar si no está mejorando el modelo o para que no sobreentrene y cree *overfitting*.
  - **epoch:** Cantidad de vueltas dadas a todos los datos proporcionados de entrenamiento (*pets\_gen\_train*).
- **Evaluación del test:** Se le pasa el generador para el test (*pets\_gen\_test*) a la función *evaluate*, para ver cuánto porcentaje de acierto tiene el modelo al predecir las imágenes guardadas para este paso.
- **Crear fichero.csv con las métricas del modelo:** Se realizará un fichero.csv para guardar cada época de entrenamiento con sus métricas obtenidas (*loss*, *val\_loss*, *accuracy* y *val\_accuracy*). Este documento nos puede ayudar en caso de falla de luz o suspensión del entorno y así no perder el proceso.

## 6.1 Comparación de los modelos

Para comparar los modelos se ha utilizado el historial de entrenamiento y las métricas obtenidas durante cada época.

Para mostrar las métricas, se han agrupado en dos gráficas:

1. **Training and Validation Accuracy:** Va mostrando el acierto que ha ido teniendo cada modelo en cada época de entrenamiento y evaluación.
2. **Training and Validation Loss:** Va mostrando la función de pérdida obtenida en cada época de entrenamiento y evaluación de los modelos.



En las gráficas se puede ver que no todos han entrenado las mismas épocas, ni obtenido los mismos resultados:

- **ResNet50:** Es el modelo que más épocas ha entrenado pero no el que tiene mejor resultado, ya que en 100 épocas el acierto es de 0.202 y su pérdida de 3.177 en la evaluación.
- **VGG19:** Este modelo se quedaría a mitad de los otros dos, ya que en 49 épocas el acierto es de 0.678 y su pérdida de 1.066 en la evaluación.
- **InceptionV3:** Es el modelo con mejor resultado de los entrenados. Ha realizado 19 épocas, y a pesar de ser el que menos épocas ha realizado, obtiene un 0.914 de acierto y 0.358 de pérdida en la evaluación.

## 6.2 Guardar modelo y clases

Según los resultados observados en el anterior apartado, se ha guardado para su posterior uso el modelo *InceptionV3*.

Para ello se ha utilizado la función *save* del propio modelo. Utiliza como parámetro un string para indicar el nombre que obtendrá el fichero.

Como queremos guardar también las clases como las ve el modelo, extraemos un *diccionario* de las clases y su índice con *class\_indices*. Esto guardaría como clave el

nombre de la raza y como valor el dígito, como nos interesa que esté al revés, le damos la vuelta mediante un bucle for.

El diccionario será guardado en un *fichero.json* con la función *dump* de la librería *json*.

### 6.3 Cargar modelo y clases

Al momento de utilizar el modelo se deberá cargar en la página con la función *load\_model* al que se le pasará como parámetro el *fichero.h5* del modelo.

Y para cargar el diccionario de las clases, se utiliza la función *load* de *json* y lo recorremos con un *bucle* para convertir el índice de *string* a *integer*, ya que lo guarda como un string.

### 6.4 Ejemplo de predicción

Se utilizan distintas funciones para realizar el tratado de la imagen hasta su predicción.

Estas son:

- **keras.preprocessing.image.load\_img**: Carga la imagen con el tamaño (256, 256).
- **keras.utils.img\_to\_array**: Se convierte la imagen a *array*, ya que al cargar la imagen es una instancia de imagen.
- **np.expand\_dims**: Se realizará un nuevo *tamaño* para el array, para ello se le indica como parámetro *axis=0*.
- **preprocess\_input**: Esta función pertenece al modelo guardado (InceptionV3), esto permitirá *codificar* la imagen para facilitar la predicción del modelo.

Una vez realizado el tratado de la imagen, se predice con *model.predict(img)* y luego se indica *pred.argmax()* para sacar el dígito correspondiente al diccionario de razas. Con ese dígito se extrae del diccionario el nombre de la raza a la que corresponde.

```
image = 'Pets_Breeds/Dog/test/Borzoi/01.jpg'
img = keras.preprocessing.image.load_img(image, target_size=(256, 256))
img_array = keras.utils.img_to_array(img)
img_batch = np.expand_dims(img_array, axis=0)
img_preprocessed = preprocess_input(img_batch)
pred = incep.predict(img_preprocessed)
pred = pred.argmax()
print(pred)
inv_map[pred]

1/1 [=====] - 1s 757ms/step
25
'Borzoi'
```

## 7 Procesamiento de Lenguaje Natural

Hemos implementado *DialogFlow* en nuestra aplicación y le hemos puesto frases típicas de *chatbot* de una protectora como, hálame de tu protectora, quiero ser voluntario, donaciones, etc.

El agente entiende las frases en *español*, *inglés* y *alemán*, aquí muestro unos ejemplos de las intenciones y las estancias puestas y del agente respondiendo:

- **Intenciones de preguntas generales:**

” Add user expression

” ¿Cómo puedo ayudar?

” ¿Teneis perros y gatos para adoptar?

” Háblame de tu protectora

- **Frases similares que también detecta:**

Háblame de la protectora	Háblame de la protectora, que podrías decirme de la protectora, Que me cuentas sobre vosotros, Háblame sobre vosotros
¿Teneis perros y gatos para adoptar?	Qué teneis para adoptar?, Qué hay disponible?
¿Qué necesitan las mascotas que están en tu protectora?	¿Qué necesitan las mascotas que están en tu protectora?
¿Cómo puedo ayudar?	¿Cómo puedo ayudar?, podría ayudar?, se puede ayudar en la protectora?

[Click here to edit entry](#)

- **Prueba del agente hablando en español:**

puedo ser voluntario?

Antes de nada agradecerte que quieras dar ese paso y segundo tienes un formulario en la pagina de Voluntariado donde puedes hacer ese proceso :)

Diga algo...

- **En inglés:**



What's the cost to adopt a pet?

The adoption fee varies depending on the animal and its needs. Please visit our website or contact us for more information about our adoption fees.

Diga algo...

- **Y en alemán:**

"Was kann ich tun, um zu helfen?"

Es gibt viele Möglichkeiten, unserem Tierheim zu helfen. Sie können eine Spende machen, Ihre Zeit als Freiwilliger zur Verfügung stellen oder sogar eines unserer Tiere adoptieren. Besuchen Sie unsere Website oder kontaktieren Sie uns für weitere Informationen.

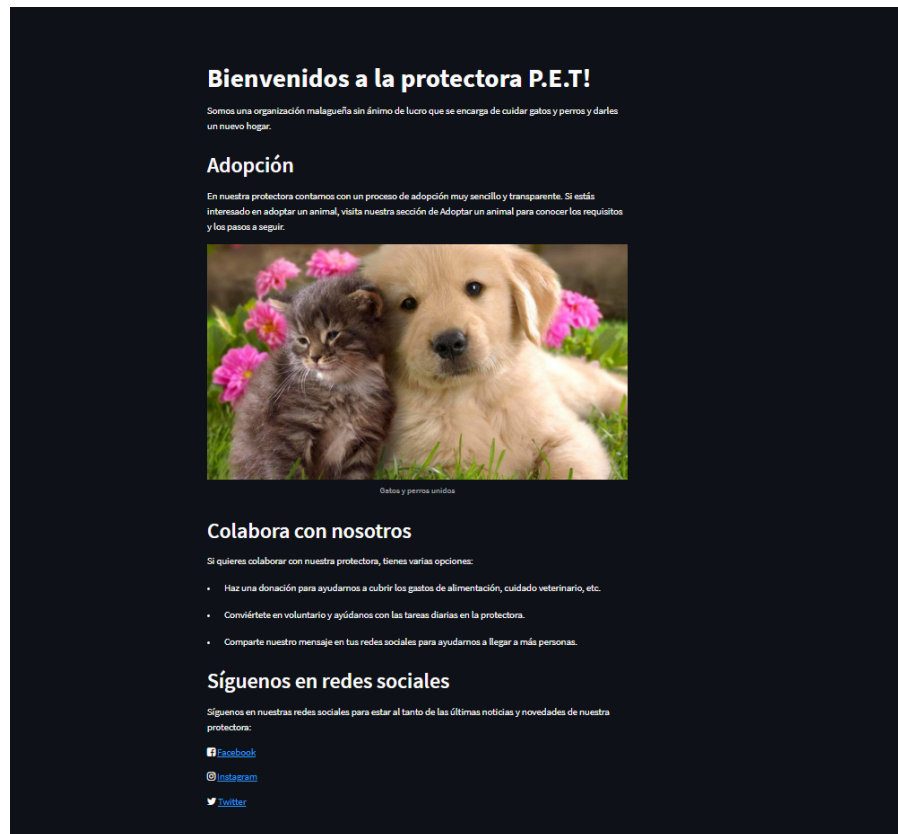
Diga algo...

## 8 Aplicación web

URL: <https://pet-animal-shelter.streamlit.app/>

La aplicación se ha realizado en *streamlit* y voy a mostrar el contenido de ellas y que hacen:

1. **Página principal:** Muestra un poco pues de qué va nuestra organización, formas de ayudarnos y nuestras redes sociales para que nos conozcan mejor.



2. **Adopción de mascotas:** En este apartado tenemos dos selectbox para que el usuario elija si quiere perro o gato y la raza. Sale del apartado de predicción, el nombre, la descripción y la imagen usada que se muestra en la pantalla para el cliente.

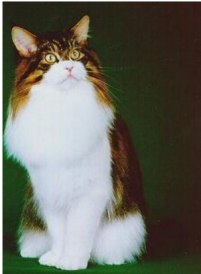
Gato

¿De qué raza te gustaría adoptar?

Maine Coon

Has seleccionado adoptar un Maine Coon

Aquí hay alguna foto de los Maine Coon disponibles para su adopción:



Maine Coon imagen

Nombre: Toby

Descripción: es un animal muy cariñoso y peludito

3. **Donaciones:** Se trata de todo aquel que le apetezca donar lo pueda hacer desde aquí.

### Donar:

Gracias por considerar una donación a nuestro refugio de animales.

Todas las donaciones serán utilizadas para:

- Proporcionar atención médica a los animales
- Comprar alimentos y suministros
- Construir nuevas instalaciones en el refugio

Si prefieres donar en especie, estas son algunas opciones:

- Alimentos para animales
- Mantas
- Juguetes

Por favor, póngase en contacto con nosotros si desea donar alguno de estos elementos.

¿Cuánto te gustaría donar?

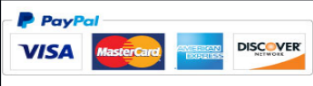
0

— +

Donar

Has seleccionado donar \$0

Si lo prefieres, puedes donar con Paypal:



Además, ¡nos encantaría contar con tu ayuda como voluntario!

Si estás interesado en ser voluntario, por favor ponte en contacto con nosotros.

4. **Contacto:** Desde aquí pueden enviarnos un correo, se enviará al gmail de la protectora.

### Contacto:

Para cualquier consulta, por favor contáctenos en el siguiente formulario:

Nombre

Email

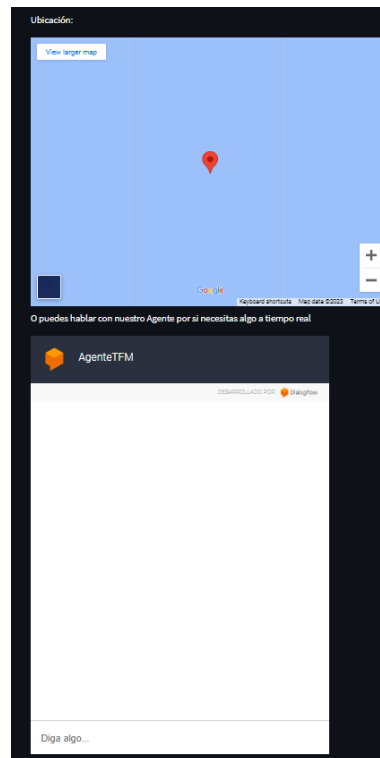
Mensaje

Enviar mensaje

O puedes contactarnos directamente:

- Teléfono: 555-1234
- Dirección: Calle Principal 123
- Horario: Lunes a Viernes de 9am a 5pm

5. **Ubicación y chatbot:** La localización de la protectora y un agente para que pueda resolver las preguntas lo mejor que pueda



6. **Voluntariado:** Un formulario donde te puedes registrar como voluntario, se guardará en la base de datos de mongodb y se enviará un email al correo de la protectora.

- **Formulario:**

**Voluntariado**

¡Únete a nuestro equipo de voluntarios y ayúdanos a mejorar el mundo!

Rellena el siguiente formulario y nos pondremos en contacto contigo pronto.

Nombre completo

Email

Teléfono

Dirección

Ciudad

Provincia

País

Áreas de interés como voluntario

Disponibilidad

Tiempo completo

Enviar formulario

- Al rellenar todo el formulario saldrá este mensaje:

Gracias por tu interés en ser voluntario. Nos pondremos en contacto contigo pronto.

- Guardado de datos en MongoDB:

```
_id: ObjectId('6408c5ce18a75163af5bd038')
nombre: "Pepito el de las puertas"
email: "prueba@gmail.com"
telefono: "123456789"
direccion: "avenida de la piruleta"
ciudad: "Narnia"
provincia: "La comarca"
pais: "La ciudad de las nubes"
intereses: "No hacer nada"
disponibilidad: "Tiempo completo"
```

- Correo enviado:



pruebapet262@gmail.com <pruebapet262@gmail.com>  
para mí ▾

Hola encargado del voluntariado,

Te informamos que Pepito el de las puertas se ha registrado como voluntario en nuestra organización.

Su dirección de correo electrónico es: [prueba@gmail.com](mailto:prueba@gmail.com)

¡Gracias!

7. **Administración:** Esta página solo está habilitada para los administradores de la página, es donde se harán las predicciones. Tendrá dos apartados:
  - **Inicio de sesión:** Requerirá una contraseña para acceder al apartado de predicción. Si no se introduce una contraseña correcta, no se mostrará.

**Iniciar sesión**

Contraseña

- **Predictor:** Una vez iniciada la sesión correctamente, permitirá subir una *imagen*, indicarle un *nombre* y una breve *descripción* de la mascota. Una vez se guarden estos datos, mostrará la raza predicha.


Subir imagen

Drag and drop file here  
Limit 200MB per file • JPG, JPEG, PNG

Browse files

Maine\_Coon\_05.jpg 9.0KB

X



Toby

Descripción

es un animal cariñoso y peludito

Guardar

Prediction accuracy  
7 days prediction is  
**73%**  
accuracy

La raza es Maine Coon

Datos guardados

---

# 9 Fuentes

## 1. Repositorio:

- [GitHub](#)

## 2. Presentación:

- [Genial.ly](#)

## 3. Entrenamiento del modelo:

- [Dataset de perros](#)
- [Dataset de gatos](#)
- [Visualización de imagen](#)
- [Image Data Generator](#)
- [Modelo VGG 19](#)
- [Modelo ResNet 50](#)
- [Modelo Inception V3](#)
- [Función plot model](#)
- [Capas de keras](#)
- [Modelo Sequential](#)
- [Callbacks Early Stopping](#)
- [Optimizador Adam](#)
- [Función de pérdida Categorical Crossentropy](#)
- [Metrica Accuracy](#)
- [Guardado y carga del modelo](#)

## 4. Página web:

- [Enviar correos con Python](#)
- [Chatbot en Streamlit](#)
- [MongoDB en Streamlit](#)
- [Documentación Streamlit](#)