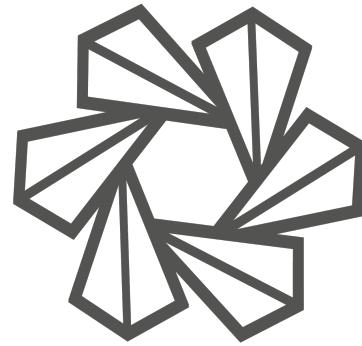
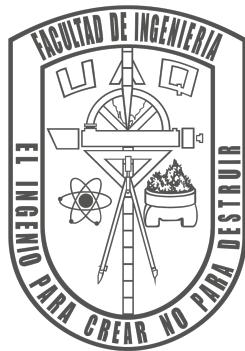
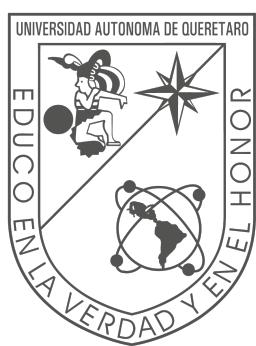


# Universidad Autónoma de Querétaro

Facultad de Ingeniería  
División de Investigación y Posgrado



## Reporte 11 Detector propio y revisión de Grad-Cam

Maestría en Ciencias en Inteligencia Artificial  
Optativa de especialidad II - Deep Learning

Aldo Cervantes Marquez  
Expediente: 262775  
Profesor: Dr. Sebastián Salazar Colores

Santiago de Querétaro, Querétaro, México  
Semestre 2022-2  
29 de Noviembre de 2022

# Índice

|  |           |
|--|-----------|
| <b>1. Introducción</b>                             | <b>1</b>  |
| <b>2. Marco Teórico</b>                            | <b>1</b>  |
| 2.1. Base de datos . . . . .                       | 1         |
| 2.1.1. Imagen de identificación . . . . .          | 1         |
| 2.2. Convoluciones . . . . .                       | 2         |
| 2.3. Pooling . . . . .                             | 2         |
| 2.4. Red Convolucional . . . . .                   | 3         |
| 2.4.1. Red EfficientNetB0 . . . . .                | 3         |
| 2.5. Grad-Cam . . . . .                            | 3         |
| <b>3. Justificación</b>                            | <b>4</b>  |
| <b>4. Resultados</b>                               | <b>4</b>  |
| 4.1. Método 1 . . . . .                            | 5         |
| 4.2. Método 2.1 . . . . .                          | 6         |
| 4.3. Método 2.2 . . . . .                          | 8         |
| <b>5. Conclusiones</b>                             | <b>9</b>  |
| <b>Referencias</b>                                 | <b>10</b> |
| <b>6. Anexo: Programa completo en Google Colab</b> | <b>11</b> |

## 1. Introducción

En la presente práctica, se utilizarán ideas propias para poder generar un detector de perros y gatos en una misma imagen, encuadrando los rostros de los animales según su categoría, aplicando los conocimientos sobre procesamiento de imágenes y estadística para lograrlo.

## 2. Marco Teórico

### 2.1. Base de datos

La base de datos consta de imágenes que contienen perros y gatos, de diferentes razas en diferentes posiciones, paisajes, etc. Por lo que todas las imágenes contienen una dimensión diferente (véase Figura 1).



Figura 1: Imágenes de la base de datos.

#### 2.1.1. Imagen de identificación

Consiste en una imagen que contiene 2 perros y 2 gatos, en donde se observa una imagen limpia con un fondo blanco y un tamaño de imagen de 400x800x3 (véase Figura 2).



Figura 2: Imágenes a detectar.

## 2.2. Convoluciones

Las convoluciones constan de kernels que permiten modificar la imagen píxel a píxel, generalmente dichos kernels representan una matriz de valores, los cuales interactúan con la cantidad de datos en igual forma para realizar la convolución (véase Figura 3) [1, 2].

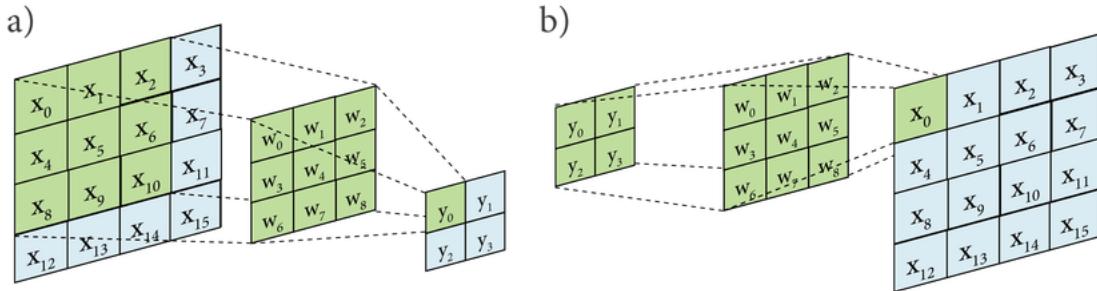


Figura 3: Paleta cúbica de colores RGB.

## 2.3. Pooling

Como sabemos, las convoluciones permiten resaltar partes de la imagen que nos podrían interesar, siempre conservando prácticamente en su totalidad la imagen. Por otro lado, la capa de pooling nos asegura que los patrones detectados en la capa convolucional se mantengan [3]. Además de que las capas de pooling no requieren de ningún parámetro de aprendizaje. Existen principalmente 3 tipos de Pooling: el maxpool, el minpool y el averagepool. El primero indica que

el mínimo de los valores de la sección de la matriz de pooling sera el seleccionado como resultado, mismo caso para el máximo y para el promedio del conjunto de valores (véase Figura 4).

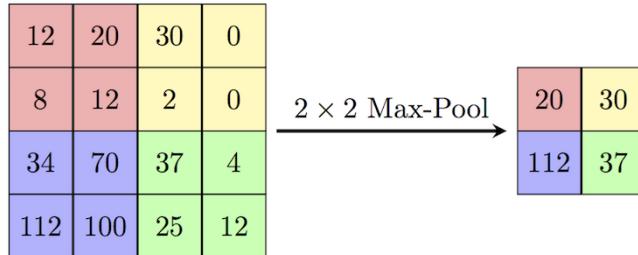


Figura 4: Ejemplo de Maxpool.

## 2.4. Red Convolucional

Consiste en un algoritmo que al igual que una red neuronal, asigna y actualiza pesos a los valores de la función y por lo tanto se optimizan los valores de los kernels (convoluciones) para poder reconocer patrones de siluetas, curvas, líneas, rostros, etc [4].

### 2.4.1. Red EfficientNetB0

Esta red tiene 237 capas divididas en 5 módulos, realizando convoluciones y reducción de dimensiones para poder aplanar la red y finalmente aplicar redes neuronales tipo Perceptrón y por lo tanto funciona de una manera analoga a la LeNet-5 [5, 6, 7]

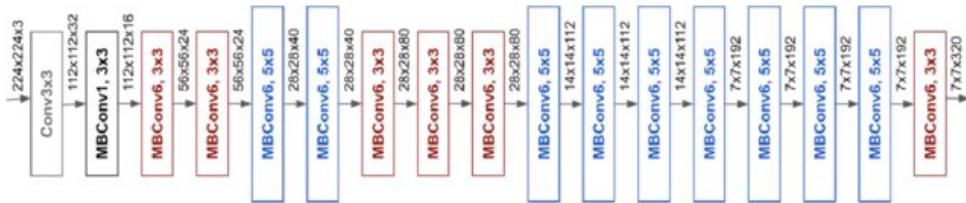


Figura 5: Estructura de red convolucional EfficientNetB0.

## 2.5. Grad-Cam

El algoritmo consiste en una red convolucional que fue previamente entrenada y tiene las características de un preprocesamiento de la imagen y decodificación [8]. Posteriormente se tiene una función llamada `get_img_array(img_path, size)` la cual permite aplicar el preprocesamiento de la imagen (normalización según el modelo elegido) y se expanden las dimensiones con la función `np.expand_dims` para obtener una dimensión que pueda ser aceptada por el modelo (1x299x299x3). Posteriormente se pasa a la función `make_gradcam_heatmap(...)` en donde se crea una red neuronal con el modelo requerido, después se computa el gradiente con la función `tf.GradientTape()`

normalizando posteriormente el gradiente y reduciendo para obtener la intensidad de la media. Multiplicándolo por cada canal de color del mapa de características [9, 10]. Finalmente con los datos se genera el mapa de calor.

Finalmente se aplica la ultima función para superponer el mapa de calor en la imagen original. Donde básicamente el mapa de calor es ajustado a una escala de colores diferente y se le aplican los 3 canales RGB para finalmente agregarse mediante una suma multiplicada por un factor  $\alpha$  y mostrarla (véase Algoritmo 1).

---

#### Algoritmo 1 Proceso de grad-cam.

**Inicio**

```
imagen=preparar_Imagen(preprocesamiento)
modelo=crear_modelo(red_convolucional_entrenada)
modelo.predecir(imagen)
mapa_calor=make_gradcam_heatmap(imagen, modelo, modelo.ultima_capa)
imagen_nueva=save_and_display_gradcam(imagen, mapa_calor)
imprimir(imagen_nueva)
```

**Fin**


---

### 3. Justificación

El uso de detectores es de gran importancia para muchas aplicaciones en inteligencia artificial como es en el manejo de vehículos autónomos, para supervisión de personal, detección de fallas en algún proceso, etc. Las tecnologías que realizan este tipo de prácticas siguen en desarrollo y es posible crear nuevas ideas sobre el funcionamiento de estos métodos.

### 4. Resultados

Para visualizar de mejor manera los resultados se realizaron 3 métodos, cada uno basado en el anterior, siendo mejorado. Cabe mencionar que la red neuronal convolucional utilizada fue una EfficientNetB0 con las siguientes características basadas en la base de datos de perros y gatos con un tamaño de imagen de 150x150x3 (véase Tabla 1 y Figura 6):

Tabla 1: Características de desempeño de la red EfficientNetB0 personalizada.

| Épocas | Pesos iniciales | Perdida de entrenamiento | % de entrenamiento | Perdida en validación | % de validación | % de prueba |
|--------|-----------------|--------------------------|--------------------|-----------------------|-----------------|-------------|
| 10     | ImageNet        | 0.1107                   | 95.79              | 0.1684                | 92.62           | 93.5        |

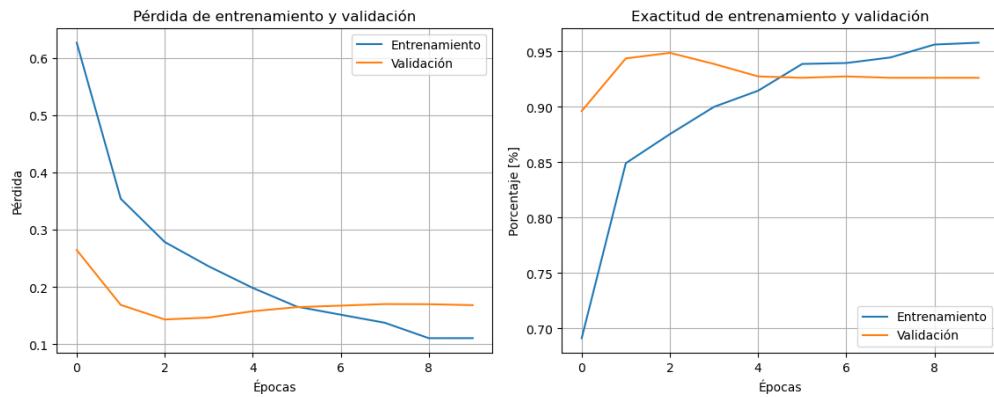


Figura 6: Desempeño de la red neuronal convolucional EfficientNetB0 personalizada.

#### 4.1. Método 1

Este método consiste en el uso de un factor  $P(x)$ ,  $x = \text{perros o gatos}$  en donde se hace una discriminación de las probabilidades segun dicho factor. Teniendo básicamente los siguientes pasos.

1. Preprocesamiento de la imagen para obtener colección de imágenes.
  - a) Normalización de la imagen.
  - b) Segmentación de la imagen en una cantidad definida de cuadros con una resolución que la red neuronal convolucional pueda aceptar de entrada.
2. Importación de la red convolucional personalizada ya entrenada (EfficientNetB0).
3. Predecir los valores de la colección de segmentos obtenidos.
4. Realizar división de valores según el hiperparámetro  $P(x)$  tanto para perros y gatos. Aplicando la siguiente relación lógica para cada probabilidad de la imagen (**Filtro 1**).

$$f_1(Im_n) = \begin{cases} 1 & \text{si } P_n(x) \geq P(x) \\ 0 & \text{si } P_n(x) < P(x) \end{cases} \quad (1)$$

5. Realizar un encuadre de las imágenes que fueron aceptadas.

Obteniendo los siguientes resultados.

Tabla 2: Resultados método 1.

| Prueba | Probabilidad<br>$P(x)$ | Gatos detectados | Perros detectados |
|--------|------------------------|------------------|-------------------|
| 1      | 0.9                    | 28               | 19                |
| 2      | 0.95                   | 21               | 13                |
| 3      | 0.98                   | 13               | 8                 |
| 4      | 0.99                   | 8                | 7                 |
| 5      | 0.992                  | 7                | 5                 |

Como se observa en la Tabla 2, en rojo la peor detección y el verde la mejor, siendo el valor real 2 perros y 2 gatos aproximándose más cuando  $P(x) = 0.992$  de una manera intuitiva es posible observar que mientras más estricto sea la probabilidad, se aproxima más al resultado deseado. Los encuadres de cada prueba se muestran en la Figura 7.

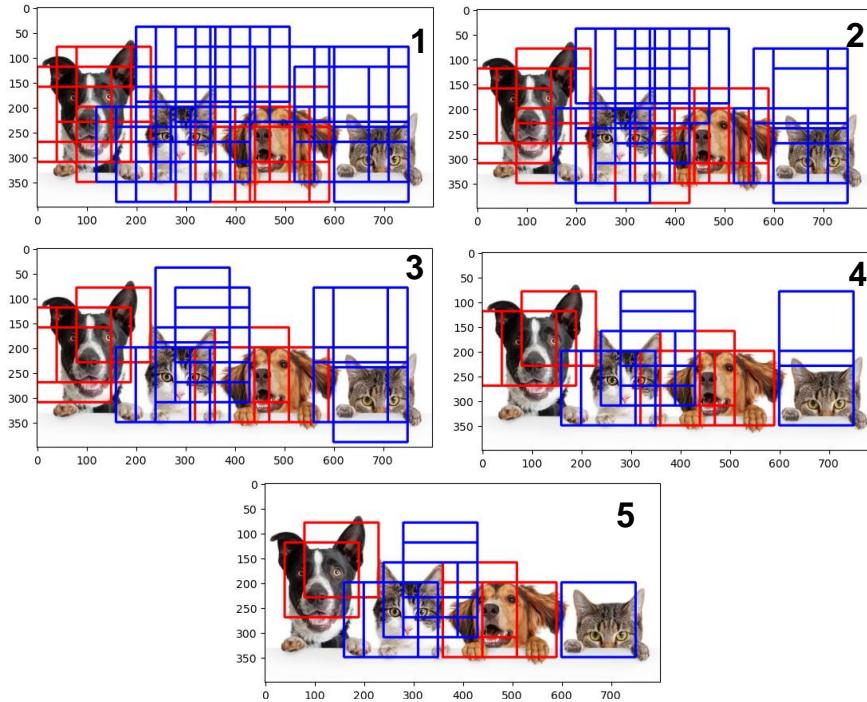


Figura 7: Encuadres aplicando el método 1 para cada prueba realizada (rojo perros y azul gatos).

## 4.2. Método 2.1

Este método agrega unos pasos adicionales al anterior, basándose en los vecinos de cada imagen que ha sido seleccionada. Para posteriormente promediar las probabilidades, generando un nuevo hiperparámetro que es el factor de excepción  $0 \leq f_e(x) \leq 1$  tanto para perro como para gato que aplica cuando no existe una imagen vecina de la seleccionada  $P_{n[x \pm 1, y \pm 1]}(x) \neq$  (véase Figura 8).

Posteriormente se aplica el promedio de las probabilidades de los vecinos  $P_{n_{media}}(x)$  para cada imagen seleccionada en el paso 4 del método 1 (véase Figura 8).

|                     |                   |                     |
|---------------------|-------------------|---------------------|
| $P_{n[x-1,y-1]}(x)$ | $P_{n[x-1,y]}(x)$ | $P_{n[x-1,y+1]}(x)$ |
| $P_{n[x,y-1]}(x)$   | $P_{n[x,y]}(x)$   | $P_{n[x,y+1]}(x)$   |
| $P_{n[x+1,y-1]}(x)$ | $P_{n[x+1,y]}(x)$ | $P_{n[x+1,y+1]}(x)$ |

Figura 8: Aplicación de la probabilidad de los vecinos cercanos.

$$P_{n_{medias}}(a) = \frac{\sum_{x,y}^m P_{n[x,y]}(a)}{m} \quad (2)$$

Finalmente se calcula el promedio de los valores máximos y mínimos de cada probabilidad media recolectada por cada imagen  $\mu_x$ .

$$\begin{aligned} P_{medias}(a) &= [P_{0_{medias}}(a), P_{1_{medias}}(a), \dots, P_{(n-1)_{medias}}(a), P_{n_{medias}}(a)] \\ \mu_a &= \frac{\max(P_{n_{medias}}(a)) + \min(P_{n_{medias}}(a))}{2} \end{aligned} \quad (3)$$

Para finalmente realizar un segundo filtro (**filtro 2**) llamado filtro de media.

$$f_2(P(a)_n) = \begin{cases} 1 & \text{si } P(a)_{n_{media}} \geq \mu_a \\ 0 & \text{si } P(a)_{n_{media}} < \mu_a \end{cases} \quad (4)$$

Cabe mencionar que en este método se utilizó la probabilidad de que NO sea gato es decir  $\bar{P}_n(gato)$ . Por lo que se obtuvieron los siguientes resultados (véase Tabla 3).

Tabla 3: Resultados método 2.1.

| Prueba | Probabilidad $P(x)$ | excepción gato $f_e(gato)$ | excepción perro $f_e(perro)$ | Media Gato $\mu_{gato}$ | Media Perro $\mu_{perro}$ | Cantidad de gatos | Cantidad de perros |
|--------|---------------------|----------------------------|------------------------------|-------------------------|---------------------------|-------------------|--------------------|
| 1      | 0.98                | 0.5                        | 0.5                          | 0.226                   | 0.7033                    | 4                 | 2                  |
| 2      | 0.99                | 0.5                        | 0.4                          | 0.2350                  | 0.7033                    | 5                 | 2                  |
| 3      | 0.99                | 0.5                        | 0.1                          | 0.2350                  | 0.7033                    | 2                 | 2                  |
| 4      | 0.99                | 0.5                        | 1                            | 0.2350                  | 0.7033                    | 2                 | 3                  |

Como se observa, fue posible realizar una detección mucho más limpia, sin embargo, se observó que el mejor resultado fue el de la prueba 4, debido a la posición de los encuadres en la imagen (véase Figura 9).

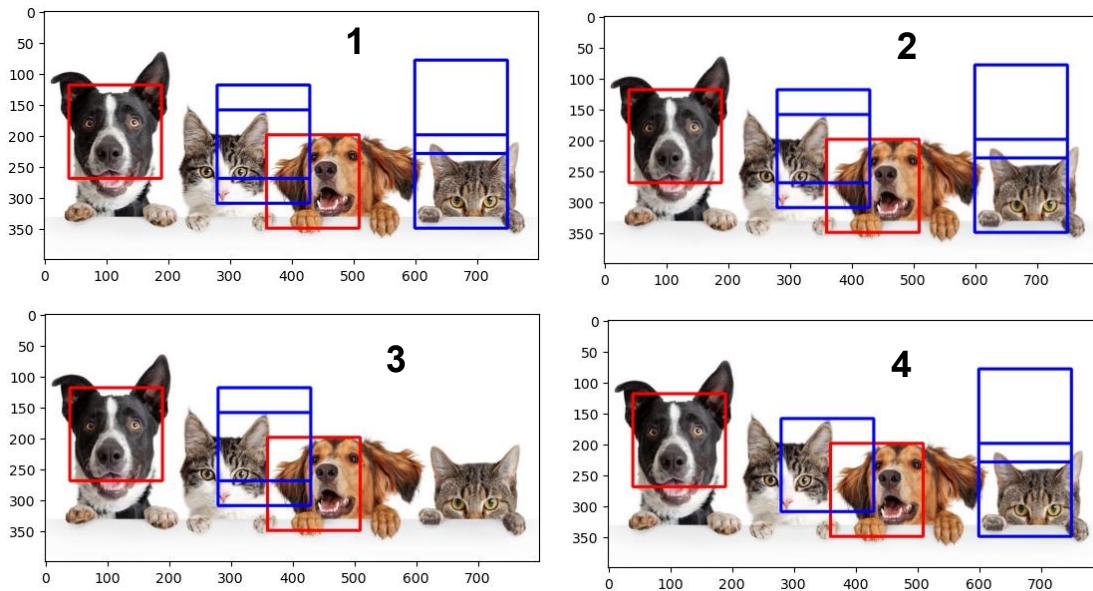


Figura 9: Encuadres aplicando el método 2.1 para cada prueba realizada (rojo perros y azul gatos).

### 4.3. Método 2.2

Este método es exactamente igual al anterior únicamente que en este caso si se utilizará la probabilidad de gatos  $P_n(gato)$ .

Se obtuvieron los siguientes resultados en la Tabla 4.

Tabla 4: Resultados método 2.2.

| Prueba | Probabilidad $P(x)$ | excepción gato $f_e(gato)$ | excepción perro $f_e(perro)$ | Media Gato $\mu_{gato}$ | Media Perro $\mu_{perro}$ | Cantidad de gatos | Cantidad de perros |
|--------|---------------------|----------------------------|------------------------------|-------------------------|---------------------------|-------------------|--------------------|
| 1      | 0.99                | 0                          | 0.5                          | 0.7279                  | 0.7033                    | 5                 | 2                  |
| 2      | 0.99                | 0.5                        | 0.5                          | 0.7649                  | 0.7033                    | 5                 | 2                  |
| 3      | 0.99                | 1                          | 0.5                          | 0.7649                  | 0.7033                    | 6                 | 2                  |
| 4      | 0.992               | 1                          | 0.5                          | 0.7599                  | 0.7033                    | 5                 | 2                  |

Como se observa, fue posible aproximarse a los resultados del método 2.2 mediante una probabilidad un poco más alta, pero se omitió el encuadre del gato de la derecha (véase Figura ), debido a las excepciones y las medias que también se vieron afectadas por esto mismo.

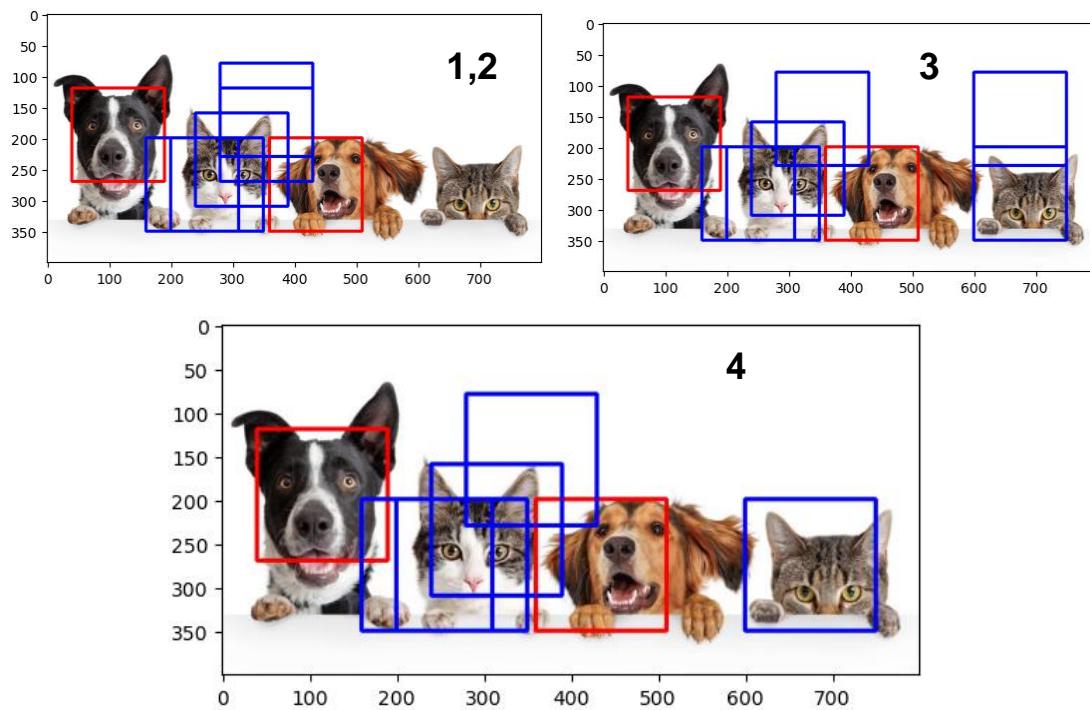


Figura 10: Encuadres aplicando el método 2.2 para cada prueba realizada (rojo perros y azul gatos).

## 5. Conclusiones

1. Fue posible realizar encuadres masomenos precisos con los métodos propuestos. Se obtuvieron 112 segmentos de dicha imagen de 150x150x3.
2. El uso del filtro 2 permitió disminuir considerablemente la cantidad de detecciones debido a su restricción de los promedios.
3. Se propone utilizar una media ponderada o algún otro método para generar un encuadre más certero al objeto deseado.
4. Se propone utilizar en trabajos posteriores otros métodos para obtener  $\mu_a$  como una media ponderada o una media de las medias de los vecinos.
5. Debe considerarse el tamaño de las imágenes proporcionadas para poder generar los segmentos y el paso del segmento al hacer el barrido e integrarlos a la colección de muestras.
6. Aún se desconoce porque al utilizar  $\bar{P}_n(x)$  funcionó mejor que  $P_n(x)$ .

## Referencias

- [1] “2d convolution using python & numpy — by samrat sahoo — analytics vidhya — medium.” <https://medium.com/analytics-vidhya/2d-convolution-using-python-numpy-43442ff5f381>. (Accessed on 09/17/2022).
- [2] “5.2. imágenes rgb — introducción a la programación.” <https://cupi2-ip.github.io/IPBook/nivel4/seccion4-4.html>. (Accessed on 09/17/2022).
- [3] “Cómo crear red convolucional en keras - ander fernández.” <https://anderfernandez.com/blog/que-es-una-red-neuronal-convolucional-y-como-crearlaen-keras/>. (Accessed on 09/26/2022).
- [4] “Intro a las redes neuronales convolucionales — by bootcamp ai — medium.” <https://bootcampai.medium.com/redes-neuronales-convolucionales-5e0ce960caf8>. (Accessed on 09/26/2022).
- [5] “(7) (pdf) empirical analysis of a fine-tuned deep convolutional model in classifying and detecting malaria parasites from blood smears.” [https://www.researchgate.net/publication/348915715\\_Empirical\\_Analysis\\_of\\_a\\_Fine-Tuned\\_Deep\\_Convolutional\\_Model\\_in\\_Classifying\\_and\\_Detecting\\_Malaria\\_Parasites\\_from\\_Blood\\_Smears](https://www.researchgate.net/publication/348915715_Empirical_Analysis_of_a_Fine-Tuned_Deep_Convolutional_Model_in_Classifying_and_Detecting_Malaria_Parasites_from_Blood_Smears). (Accessed on 11/08/2022).
- [6] “Complete architectural details of all efficientnet models — by varadan agarwal — towards data science.” <https://towardsdatascience.com/complete-architectural-details-of-all-efficientnet-models-5fd5b736142>. (Accessed on 11/08/2022).
- [7] “[1905.11946] efficientnet: Rethinking model scaling for convolutional neural networks.” <https://arxiv.org/abs/1905.11946>. (Accessed on 11/08/2022).
- [8] R. Ramprasaath, M. Selvaraju, A. Cogswell, R. Das, D. Vedantam, and D.-B. Parikh, “Iccv 2017 open access repository.” [https://openaccess.thecvf.com/content\\_iccv\\_2017/html/Selvaraju\\_Grad-CAM\\_Visual\\_Explanations\\_ICCV\\_2017\\_paper.html](https://openaccess.thecvf.com/content_iccv_2017/html/Selvaraju_Grad-CAM_Visual_Explanations_ICCV_2017_paper.html), 2017. (Accessed on 11/27/2022).
- [9] Keras, “Grad-cam class activation visualization.” [https://keras.io/examples/vision/grad\\_cam/](https://keras.io/examples/vision/grad_cam/). (Accessed on 11/28/2022).
- [10] M. Rivera, “Grad-cam: Visualizacion de mapas de activación.” [http://personal.cimat.mx:8181/~mrivera/cursos/aprendizaje\\_profundo/GradCAM/GradCAM.html](http://personal.cimat.mx:8181/~mrivera/cursos/aprendizaje_profundo/GradCAM/GradCAM.html), Junio 2020. (Accessed on 11/28/2022).

# P11 detector propio

November 29, 2022

```
[ ]: import h5py
import tensorflow as tf
from tensorflow.keras.utils import to_categorical
#from tensorflow import keras
#import tensorflow.compat
#import keras
import cv2
import numpy as np
import os
import zipfile
from matplotlib import image

files=zipfile.ZipFile('cats_and_dogs_small.zip','r')
files.extractall('')

x_dog=[]
x_cat=[ ]
```

## 1 Filtrado de datos en gatos y perros

```
[ ]: from PIL import Image
x_size=150
y_size=150

for name in files.namelist():
    if '/dogs/' in name and '.jpg' in name:
        a=cv2.imread(name)
        a=cv2.resize(a,(x_size,y_size)) # Dimensión de la imagen
        img = cv2.cvtColor(a, cv2.COLOR_BGR2RGB)
        #img2=img.resize(200,200) # Mobilenet (224,224,3)
        x_dog.append(img)

    elif '/cats/' in name and '.jpg' in name:
        a=cv2.imread(name)
        a=cv2.resize(a,(x_size,y_size)) # Dimensión de la imagen
        img = cv2.cvtColor(a, cv2.COLOR_BGR2RGB)
        x_cat.append(img)
```

```
print(len(x_dog),len(x_cat))
x_dog=np.stack(x_dog, axis=0)
x_cat=np.stack(x_cat, axis=0)
```

## 2 Normalización de datos

```
[ ]: print(type(x_dog),x_dog.shape)
print(type(x_cat),x_cat.shape)
x_dog=x_dog.astype('float32')
x,y,z,w=x_dog.shape
y_dog=np.zeros((x,1),dtype=int)
x_cat=x_cat.astype('float32')
x,y,z,w=x_cat.shape
y_cat=np.ones((x,1),dtype=int)
#x_dog=(x_dog/127.5)-1#x_dog/=255
#x_cat=(x_cat/127.5)-1#x_cat/=255
## Conjunto combinado de perros y gatos
x_comb=np.vstack((x_dog,x_cat))
y_comb=np.vstack((y_dog,y_cat))
print(x_comb.ndim,x_comb.shape)
print(y_dog)
```

```
[ ]: #import keras
### ONE HOT
from tensorflow.keras.utils import to_categorical
y_dog_oh=to_categorical(y_dog,y_dog.max()+2)
y_cat_oh=to_categorical(y_cat,y_cat.max()+1)
print(y_comb[3455])
y_comb_oh=to_categorical(y_comb,y_comb.max()+1)
print(y_comb_oh[3455])
print(type(y_comb_oh),y_comb_oh.shape)
print(y_cat_oh.shape)
print(y_dog_oh.shape)
print(y_dog_oh)
#print(y_cat_oh)
```

```
[ ]: xx,yy,ww,zz=x_cat.shape
x_train=np.vstack((x_cat[:int(xx*0.6),:,:,:],x_dog[:int(xx*0.6),:,:,:]))
y_train=np.vstack((y_cat_oh[:int(xx*0.6),:],y_dog_oh[:int(xx*0.6),:]))
x_val=np.vstack((x_cat[int(xx*0.6):int(xx*0.8),:,:,:],x_dog[int(xx*0.6):int(xx*0.8),:,:,:]))
y_val=np.vstack((y_cat_oh[int(xx*0.6):int(xx*0.8),:],y_dog_oh[int(xx*0.6):int(xx*0.8),:]))
x_test=np.vstack((x_cat[int(xx*0.8):,:,:,:],x_dog[int(xx*0.8):,:,:,:]))
y_test=np.vstack((y_cat_oh[int(xx*0.8):,:,:],y_dog_oh[int(xx*0.8):,:,:]))
```

```
print(x_test.min())
```

### 3 EfficientNet B0

```
[ ]: import tensorflow as tf
from keras.models import Model, load_model
#from tensorflow.keras.applications.efficientnet.EfficientNetB0 import
    ↪EfficientNetB0
#from tensorflow.keras.applications.resnet50 import preprocess_input,
    ↪decode_predictions
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Flatten, Dropout, Input
from tensorflow.keras.callbacks import ModelCheckpoint, EarlyStopping,
    ↪ReduceLROnPlateau
#import h5py
lr_reduce = ReduceLROnPlateau(monitor='val_accuracy', factor=0.6, patience=8,
    ↪verbose=1, mode='max', min_lr=5e-5)
checkpoint = ModelCheckpoint('resnet50.h', monitor='val_accuracy', mode=
    ↪'max', save_best_only = True, verbose= 1)
earlystopper = EarlyStopping(monitor = 'val_loss', min_delta = 0, patience = 10,
    ↪verbose = 1, restore_best_weights = True)

x_train_res=tf.keras.applications.efficientnet.preprocess_input(np.copy(x_train))
x_val_res=tf.keras.applications.efficientnet.preprocess_input(np.copy(x_val))
x_test_res=tf.keras.applications.efficientnet.preprocess_input(np.copy(x_test))

model = tf.keras.applications.efficientnet.
    ↪EfficientNetB0(input_shape=(x_size,y_size,3),weights='imagenet',include_top=False)
    ↪## Colocar otro top
#model.summary()
sal=model.output
sal=Flatten()(sal)
sal=Dense(502,activation='relu')(sal)
sal=Dropout(0.26)(sal)
sal = Dense(256, activation='selu')(sal)
sal=Dense(100,activation='selu')(sal)
sal=Dense(50,activation='relu')(sal)
sal = Dense(2, activation='softmax')(sal)
#Se unen la CNN y el top
efnetb0_custom=Model(inputs=model.input, outputs=sal)
efnetb0_custom.compile(loss="categorical_crossentropy", optimizer=tf.keras.
    ↪optimizers.SGD(learning_rate=0.001), metrics=["accuracy"])
history=efnetb0_custom.fit(x_train_res,y_train,batch_size=64,epochs=10,
    ↪validation_data=(x_val_res,y_val),callbacks=[lr_reduce,earlystopper,checkpoint])
```

```
[ ]: pred=efnetb0_custom.predict(x_test_res)
pred=np.argmax(pred, axis=1)
y1=np.argmax(y_test, axis=1)

#label=np.argmax(yp_oh)
exactitud_test=0
for a in range(len(pred)):
    if pred[a]==y1[a]:
        exactitud_test+=1
print('exactitud de la prueba= ',100*exactitud_test/len(pred), '%')
```

```
[ ]: import matplotlib.pyplot as plt
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.xlabel('Épocas')
plt.ylabel('Pérdida')
plt.legend(['Entrenamiento', 'Validación'])
plt.title('Pérdida de entrenamiento y validación')
plt.grid()
```

```
[ ]: plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('Exactitud de entrenamiento y validación')
plt.xlabel('Épocas')
plt.ylabel('Porcentaje [%]')
plt.legend(['Entrenamiento', 'Validación'])
plt.grid()
```

```
[ ]: ## Salvar el modelo
efnetb0_custom.save('efnetb0.h5')
```

## 4 Primer prueba de imagen

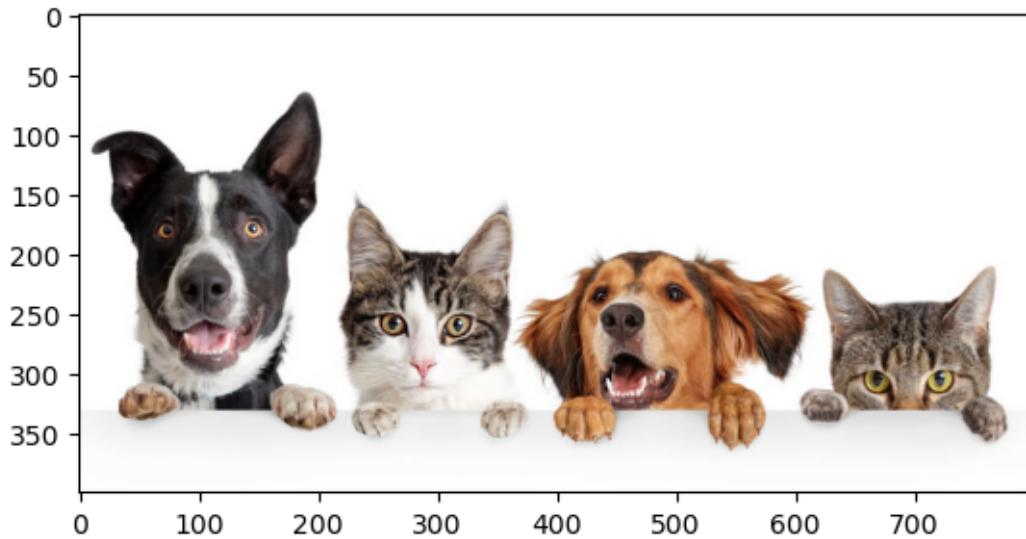
Correr desde aqui

La imagen a probar será una imagen obtenida de internet que contiene 2 perros y 2 gatos

```
[476]: import cv2
import matplotlib.pyplot as plt
import numpy as np
pygs=cv2.imread('muchos perros y gatos.jpg')
#pygs=cv2.resize(pygs,(x_size,y_size)) # Dimensión de la imagen
pygs = cv2.cvtColor(pygs, cv2.COLOR_BGR2RGB)
```

```
[477]: #cv2.rectangle(pygs,(0,150),(340,240),(255,0,0),4) #Prueba de rectangulo
plt.imshow(pygs)
```

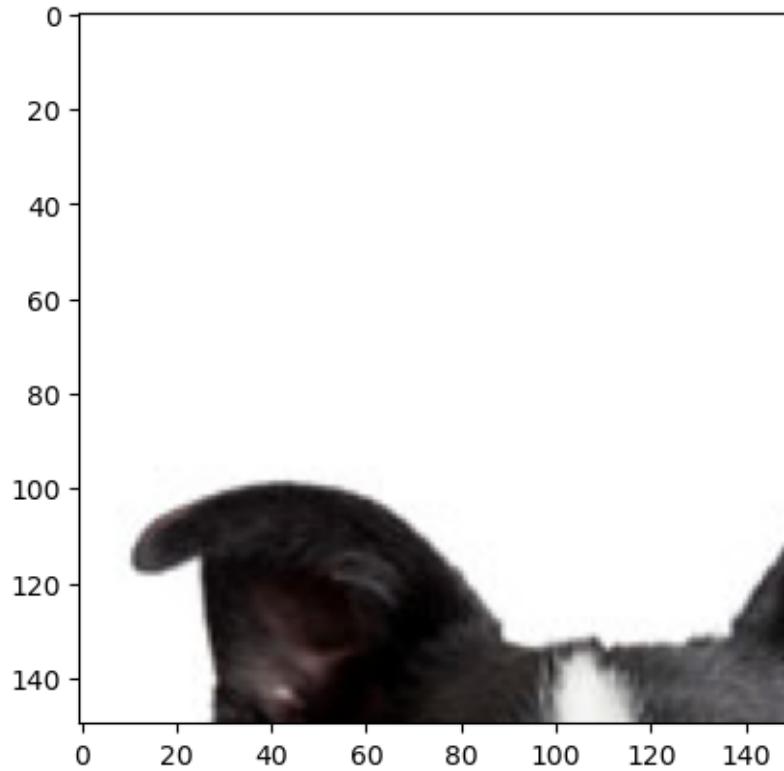
```
[477]: <matplotlib.image.AxesImage at 0x2e04c3faca0>
```



[478]: *#Uso de Crop en la imagen*

```
im1=pygs[0:150,0:150]  
plt.imshow(im1)
```

[478]: <matplotlib.image.AxesImage at 0x2e038f30790>



```
[479]: pygs_segmentos=[]
ren,col,dim=pygs.shape
print(ren,col,dim)
x_size=150
y_size=150
coord_segx=[]
coord_segy=[]
cord_mat=np.arange(7*16)
cord_mat=np.reshape(cord_mat,(7,16))
for c in range(7): #(int(col/y_size)):
    for r in range(16): #(int(ren/x_size)):
        pygs_segmentos.append(pygs[c*40:(c*40)+150,r*40:(r*40)+150])
        coord_segx.append(r*40)
        coord_segy.append(c*40) # se le debe sumar x_shape y y_shape para el
        ↪otro punto

pygs_segmentos=np.stack(pygs_segmentos, axis=0)
print(pygs_segmentos.shape)
```

400 800 3  
(112, 150, 150, 3)

## 5 Llamando al modelo

```
[480]: import tensorflow as tf
import h5py
from keras.models import Model, load_model
efnb0=tf.keras.models.load_model('efnetb0.h5')
```

## 6 Realizando la predicción de los segmentos

```
[481]: # perro 1 0   gato 0 1
prob_pygs=efnb0.predict(pygs_segmentos)
print(prob_pygs.shape)
```

4/4 [=====] - 2s 152ms/step  
(112, 2)

```
[482]: print(prob_pygs[0,:]) # Softmax falla probar con otra función de activación
[0.10102677 0.8989732 ]
```

```
[483]: """
print(perro)
print(prob_pygs[:,0].max())
```

```
plt.imshow(pygs_segmentos[111,:,:,:])
print(coord_segx[86],coord_segy[86])
"""
```

[483]: '\nprint(perro)\nprint(prob\_pygs[:,0].max())\nplt.imshow(pygs\_segmentos[111,:,:,:])\nprint(coord\_segx[86],coord\_segy[86])\n'

## 7 Método #1

Consiste en obtener los valores máximos de las imágenes segmentadas y después compararlas con sus vecinos, para realizar un promedio o una media ponderada para dar una probabilidad y encuadrar los valores del reconocimiento

### 7.1 Prueba 1 con únicamente los valores máximos

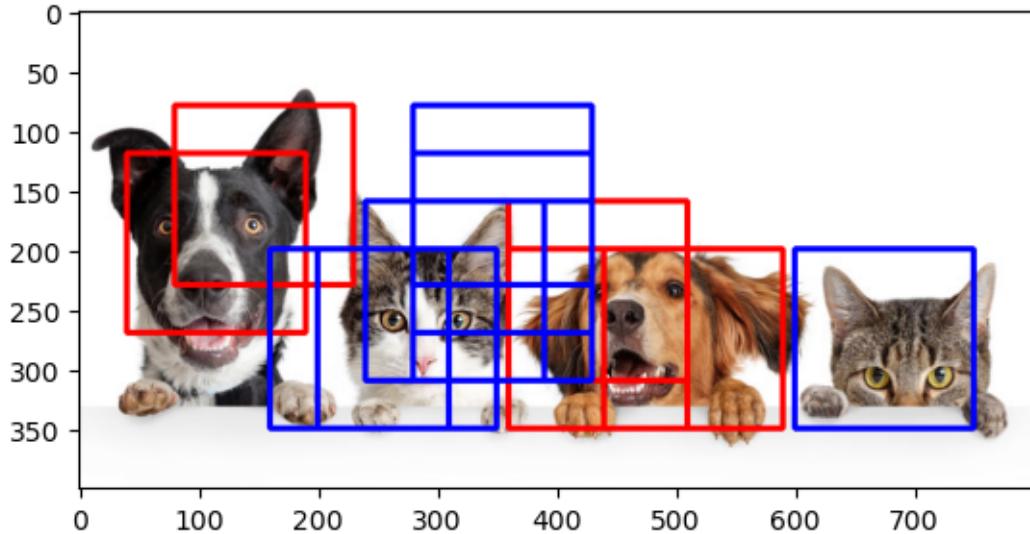
[484]: sensibilidad=0.992  
 perro=np.where(prob\_pygs[:,0]>=sensibilidad)  
 gato=np.where(prob\_pygs[:,1]>=sensibilidad)  
 perro=list(perro[0])  
 gato=list(gato[0])  
 print('valores >=0.98 para perro:',perro)  
 print('valores >=0.98 para gato:',gato)  
 print(cord\_mat)  
 #plt.imshow(pygs\_segmentos[111,:,:,:])  
 print('gatos:',len(gato),'perros:',len(perro))

```
valores >=0.98 para perro: [34, 49, 73, 89, 91]
valores >=0.98 para gato: [39, 55, 70, 71, 84, 85, 95]
[[ 0   1   2   3   4   5   6   7   8   9   10  11  12  13  14  15]
 [ 16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31]
 [ 32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47]
 [ 48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63]
 [ 64  65  66  67  68  69  70  71  72  73  74  75  76  77  78  79]
 [ 80  81  82  83  84  85  86  87  88  89  90  91  92  93  94  95]
 [ 96  97  98  99 100 101 102 103 104 105 106 107 108 109 110 111]]
gatos: 7 perros: 5
```

[485]: im\_prueba1=cv2.imread('muchos\_perros\_y\_gatos.jpg')
im\_prueba1=cv2.cvtColor(im\_prueba1, cv2.COLOR\_BGR2RGB)
# plt.imshow(im\_prueba1)
for aa in perro:
 cv2.rectangle(im\_prueba1,(coord\_segx[aa],coord\_segy[aa]),(coord\_segx[aa]+150,coord\_segy[aa]+150),
 #print(coord\_segx[aa],coord\_segy[aa])
 for bb in gato:
 cv2.rectangle(im\_prueba1,(coord\_segx[bb],coord\_segy[bb]),(coord\_segx[bb]+150,coord\_segy[bb]+150),

```
plt.imshow(im_prueba1)
```

[485]: <matplotlib.image.AxesImage at 0x2e0332b9640>



## 7.2 Para perros:

```
[486]: prom_prob_p=[]
expcion_p=0.5
#####
for p in range(len(perro)):
    xx,yy=np.where(cord_mat==perro[p])
    #print(perro[p])
    xx=int(xx)
    yy=int(yy)
    try:
        p1=prob_pygs[p,0]
    except:
        p1=expcion_p
    try:
        p2=prob_pygs[cord_mat[xx-1][yy],0] #arriba
    except:
        p2=expcion_p
    try:
        p3=prob_pygs[cord_mat[xx-1][yy+1],0] #arr-der
    except:
        p3=expcion_p
    try:
        p4=prob_pygs[cord_mat[xx][yy+1],0] #derecha
```

```

except:
    p4=expcion_p
try:
    p5=prob_pygs[cord_mat[xx+1][yy+1],0] #abajo-der
except:
    p5=expcion_p
try:
    p6=prob_pygs[cord_mat[xx+1][yy],0] # abajo
except:
    p6=expcion_p
try:
    p7=prob_pygs[cord_mat[xx+1][yy-1],0] #ab-izquierda
except:
    p7=expcion_p
try:
    p8=prob_pygs[cord_mat[xx][yy-1],0] # izq
except:
    p8=expcion_p
try:
    p9=prob_pygs[cord_mat[xx-1][yy-1],0] #izq-arr
except:
    p9=expcion_p
prom_prob_p.append(np.mean([p1,p2,p3,p4,p5,p6,p7,p8,p9]))
#####
print(prom_prob_p)

```

[0.53328365, 0.8641559, 0.641704, 0.8584508, 0.69553924]

### 7.3 Para gatos:

```

[487]: prom_prob_g=[]
expcion_g=1
#####
for p in range(len(gato)):
    xx,yy=np.where(cord_mat==gato[p])
    #print(perro[p])
    xx=int(xx)
    yy=int(yy)
    try:
        p1=prob_pygs[p,1]
    except:
        p1=expcion_g
    try:
        p2=prob_pygs[cord_mat[xx-1][yy],1] #arriba
    except:
        p2=expcion_g
    try:

```

```

    p3=prob_pygs[cord_mat[xx-1][yy+1],1] #arr-der
except:
    p3=expcion_g
try:
    p4=prob_pygs[cord_mat[xx][yy+1],1] #derecha
except:
    p4=expcion_g
try:
    p5=prob_pygs[cord_mat[xx+1][yy+1],1] #abajo-der
except:
    p5=expcion_g
try:
    p6=prob_pygs[cord_mat[xx+1][yy],1] # abajo
except:
    p6=expcion_g
try:
    p7=prob_pygs[cord_mat[xx+1][yy-1],1] #ab-izquierda
except:
    p7=expcion_g
try:
    p8=prob_pygs[cord_mat[xx][yy-1],1] # izq
except:
    p8=expcion_g
try:
    p9=prob_pygs[cord_mat[xx-1][yy-1],1] #izq-arr
except:
    p9=expcion_g
prom_prob_g.append(np.mean([p1,p2,p3,p4,p5,p6,p7,p8,p9]))
#####
print(prom_prob_g)

```

[0.8580277, 0.7468798, 0.8975928, 0.60555464, 0.81496173, 0.9143136, 0.8828695813814799]

## 8 Parte 2: Uso de filtro de media

se realiza un filtro de media como se muestra a continuación

$$P_{media}(x) = \frac{P(x)_{max} + P(x)_{min}}{2}$$

Para posteriormente realizar una categorización

$$f(P(x)_n) = \begin{cases} 1 & \text{si } P(x)_n \geq P_{media}(x) \\ 0 & \text{si } P(x)_n < P_{media}(x) \end{cases}$$

```
[488]: m_g=(max(prom_prob_g)+min(prom_prob_g))/2
m_p=(max(prom_prob_p)+min(prom_prob_p))/2
print('media prob gato:',m_g, 'media probabilidad perro:',m_p)
```

media prob gato: 0.75993412733078 media probabilidad perro: 0.6987197399139404

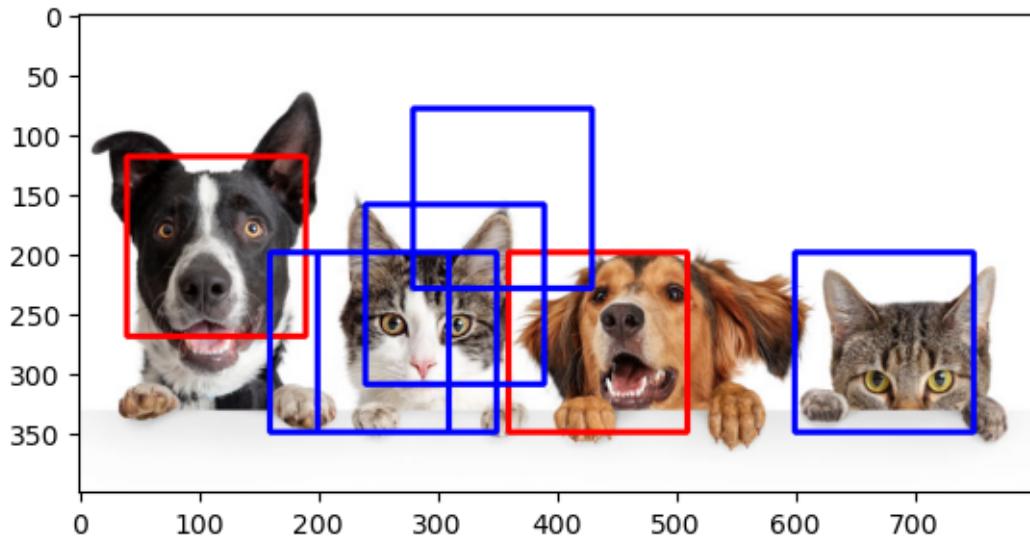
### 8.0.1 Cantidad de perros y gatos

```
[489]: #para perros
p_ac_p=[]
p_ac_g=[]
im_prueba2=cv2.imread('muchos perros y gatos.jpg')
im_prueba2=cv2.cvtColor(im_prueba2, cv2.COLOR_BGR2RGB)
for uu in range(len(prom_prob_p)):
    if prom_prob_p[uu]>=m_p:
        cv2.
    →rectangle(im_prueba2,(coord_segx[perro[uu]],coord_segy[perro[uu]]),(coord_segx[perro[uu]]+150,co
        p_ac_p.append(prom_prob_p[uu])

for ii in range(len(prom_prob_g)):
    if prom_prob_g[ii]>=m_g:
        cv2.
    →rectangle(im_prueba2,(coord_segx[gato[ii]],coord_segy[gato[ii]]),(coord_segx[gato[ii]]+150,co
        print(coord_segx[gato[ii]],coord_segy[gato[ii]])
        p_ac_g.append(prom_prob_g[ii])
print('sensibilidad:',sensibilidad)
print('media prob gato:',m_g, 'media probabilidad perro:',m_p)
print('cantidad de gatos:',len(p_ac_g),'cantidad de perros:',len(p_ac_p))
print('excepción gato:',expcion_g,'excepción perros:',expcion_p)
plt.imshow(im_prueba2)
```

280 80  
240 160  
160 200  
200 200  
600 200  
sensibilidad: 0.992  
media prob gato: 0.75993412733078 media probabilidad perro: 0.6987197399139404  
cantidad de gatos: 5 cantidad de perros: 2  
excepción gato: 1 excepción perros 0.5

[489]: <matplotlib.image.AxesImage at 0x2e03271c460>



[ ]:

# Grad-cam Aldo Cervantes

November 29, 2022

```
[1]: import numpy as np
      import tensorflow as tf
      from tensorflow import keras

      # Display
      from IPython.display import Image, display
      import matplotlib.pyplot as plt
      import matplotlib.cm as cm

[2]: model_builder = keras.applications.Xception
      img_size = (299, 299)
      preprocess_input = keras.applications.Xception.preprocess_input
      decode_predictions = keras.applications.Xception.decode_predictions

      last_conv_layer_name = "block14_sepconv2_act"

      # The local path to our target image
      img_path = keras.utils.get_file(
          "african_elephant.jpg", "https://i.imgur.com/Bvro0YD.png"
      )

      display(Image(img_path))

Downloading data from https://i.imgur.com/Bvro0YD.png
4217496/4217496 [=====] - 8s 2us/step
```



## 1 Algoritmo de Grad-Cam

```
[3]: def get_img_array(img_path, size):
    # `img` is a PIL image of size 299x299
    img = keras.preprocessing.image.load_img(img_path, target_size=size)
    # `array` is a float32 Numpy array of shape (299, 299, 3)
    array = keras.preprocessing.image.img_to_array(img)
    # We add a dimension to transform our array into a "batch"
    # of size (1, 299, 299, 3)
    array = np.expand_dims(array, axis=0)
    return array

def make_gradcam_heatmap(img_array, model, last_conv_layer_name,
                        pred_index=None):
    # First, we create a model that maps the input image to the activations
    # of the last conv layer as well as the output predictions
    grad_model = tf.keras.models.Model(
        [model.inputs], [model.get_layer(last_conv_layer_name).output, model.
    output])
```

```

    )

    # Then, we compute the gradient of the top predicted class for our input
    ↪image
    # with respect to the activations of the last conv layer
    with tf.GradientTape() as tape:
        last_conv_layer_output, preds = grad_model(img_array)
        if pred_index is None:
            pred_index = tf.argmax(preds[0])
        class_channel = preds[:, pred_index]

        # This is the gradient of the output neuron (top predicted or chosen)
        # with regard to the output feature map of the last conv layer
        grads = tape.gradient(class_channel, last_conv_layer_output)

        # This is a vector where each entry is the mean intensity of the gradient
        # over a specific feature map channel
        pooled_grads = tf.reduce_mean(grads, axis=(0, 1, 2))

        # We multiply each channel in the feature map array
        # by "how important this channel is" with regard to the top predicted class
        # then sum all the channels to obtain the heatmap class activation
        last_conv_layer_output = last_conv_layer_output[0]
        heatmap = last_conv_layer_output @ pooled_grads[..., tf.newaxis]
        heatmap = tf.squeeze(heatmap)

        # For visualization purpose, we will also normalize the heatmap between 0 & 1
        heatmap = tf.maximum(heatmap, 0) / tf.math.reduce_max(heatmap)
    return heatmap.numpy()

```

```
[4]: # Prepare image
img_array = preprocess_input(get_img_array(img_path, size=img_size))

# Make model
model = model_builder(weights="imagenet")

# Remove last layer's softmax
model.layers[-1].activation = None

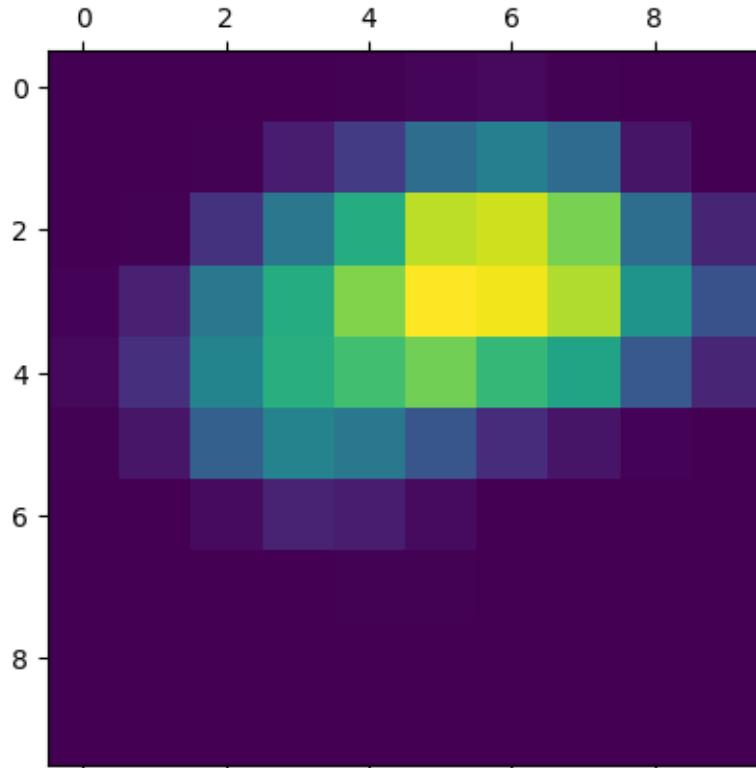
# Print what the top predicted class is
preds = model.predict(img_array)
print("Predicted:", decode_predictions(preds, top=1)[0])

# Generate class activation heatmap
heatmap = make_gradcam_heatmap(img_array, model, last_conv_layer_name)

# Display heatmap
```

```
plt.matshow(heatmap)
plt.show()
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-
applications/xception/xception_weights_tf_dim_ordering_tf_kernels.h5
91884032/91884032 [=====] - 70s 1us/step
1/1 [=====] - 1s 1s/step
Downloading data from https://storage.googleapis.com/download.tensorflow.org/dat
a/imagenet_class_index.json
35363/35363 [=====] - 0s 3us/step
Predicted: [('n02504458', 'African_elephant', 9.862385)]
```



```
[5]: def save_and_display_gradcam(img_path, heatmap, cam_path="cam.jpg", alpha=0.4):
    # Load the original image
    img = keras.preprocessing.image.load_img(img_path)
    img = keras.preprocessing.image.img_to_array(img)

    # Rescale heatmap to a range 0-255
    heatmap = np.uint8(255 * heatmap)

    # Use jet colormap to colorize heatmap
    jet = cm.get_cmap("jet")
```

```
# Use RGB values of the colormap
jet_colors = jet(np.arange(256))[:, :3]
jet_heatmap = jet_colors[heatmap]

# Create an image with RGB colorized heatmap
jet_heatmap = keras.preprocessing.image.array_to_img(jet_heatmap)
jet_heatmap = jet_heatmap.resize((img.shape[1], img.shape[0]))
jet_heatmap = keras.preprocessing.image.img_to_array(jet_heatmap)

# Superimpose the heatmap on original image
superimposed_img = jet_heatmap * alpha + img
superimposed_img = keras.preprocessing.image.array_to_img(superimposed_img)

# Save the superimposed image
superimposed_img.save(cam_path)

# Display Grad CAM
display(Image(cam_path))

save_and_display_gradcam(img_path, heatmap)
```

