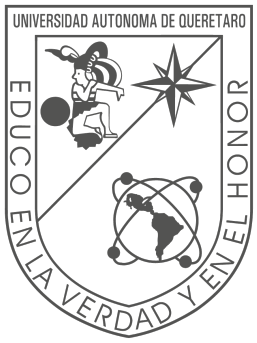
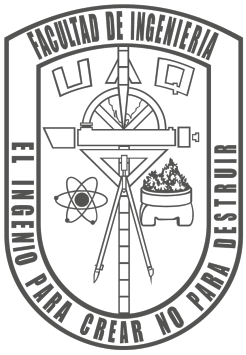
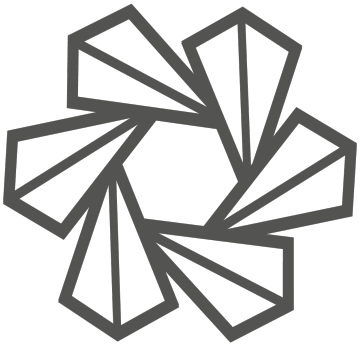
Universidad Autónoma de Querétaro

Facultad de Ingeniería
División de Investigación y Posgrado







Reporte 8

Redes Convolucionales Transfer Learning

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
08 de Noviembre de 2022

Índice

1. Introducción **1**

2. Marco Teórico **1**
2.1. Base de datos . . . 1
2.2. Imagenet . . . 1
2.3. Convoluciones . . . 1
2.4. Pooling . . . 2
2.5. Red Convolucional . . . 2
2.6. Transfer Learning . . . 2
2.6.1. Red LeNet-5 . . . 3
2.7. Red Resnet50 . . . 3
2.8. Red EfficientNetB0 . . . 3
2.9. Red InceptionV3 . . . 4

3. Justificación **4**

4. Resultados **5**

5. Conclusiones **5**

Referencias **6**

6. Anexo: Programa completo en Google Colab **8**

1. Introducción

En la presente práctica se mostrará la aplicación de varios modelos propuestos de redes convolucionales, las cuales han sido previamente entrenadas con una base de datos llamada **Imagenet**, por lo que se transferirán los valores de los pesos para poder preentrenar y partir desde un punto mejor en el aprendizaje de la red. Se probarán 4 redes en distintas configuraciones, LeNet-5, Resnet50 preentrenada y con pesos aleatorios, EfficientNetB0 preentrenada y con pesos aleatorios e InceptrionV3 preentrenada y con pesos aleatorios

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 imagenes contienen una dimensión diferente (véase Figura 1).

Figura 1: Imágenes de la base de datos.

2.2. Imagenet

Es una base de datos que cuenta con cientos de miles de imágenes, teniendo como objetivo avanzar en el mundo de la inteligencia artificial, visión por computadora y Deep Learning [1]. En este caso la base de datos cuenta con imágenes de muchos animales, pudiéndolos clasificar en 1000 categorías.

2.3. 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 2) [2, 3].

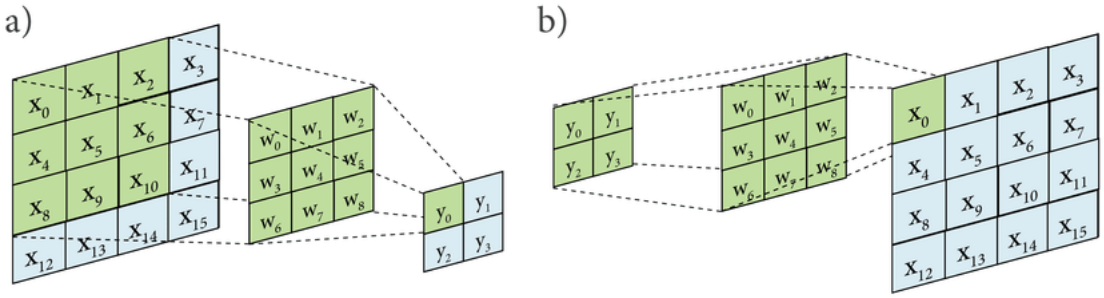


Figura 2: Paleta cúbica de colores RGB.

2.4. 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 [4].
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 minímo 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 3).

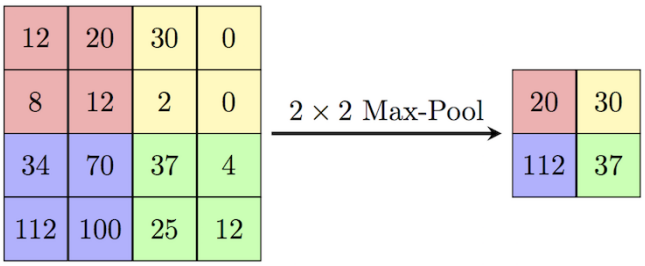


Figura 3: Ejemplo de Maxpool.

2.5. Red Convolucional

Consiste en en 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, lineas, rostros, etc [5].

2.6. Transfer Learning

Es un aprendizaje transferido a partir de experiencias pasadas (entrenamientos anteriores), teniendo 3 tipos: Inductivo, no supervisado y el transductivo [**?**]. Se tiene como objetivo que las redes ya tengan un conocimiento sobre el problema.

2.6.1. Red LeNet-5

Consiste en un arreglo de 7 capas propuesta por Yann LeCun en 1998, tiene como principal objetivo, resolver problemas con imágenes. Dichas capas se van alternando entre convoluciones y Pooling, en donde las capas de convolución son resultado de la multipliación de un kernel por la imagen, por lo que, dichos valores de la matriz del kernel, serán definidas como pesos de la red, el punto es ir disminuyendo el tamaño de la imagen y aplanarla para que pueda entrar a una red neuronal (véase Figura 6).

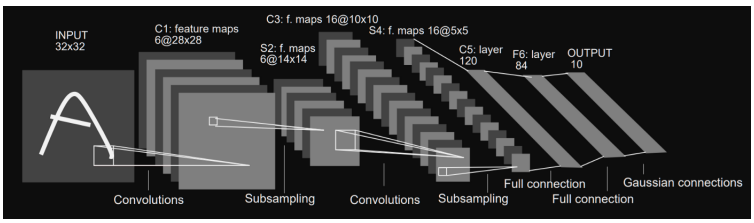


Figura 4: Estructura de red convolucional LeNet5.

2.7. Red Resnet50

ResNet-50 es una red neuronal convolucional con 50 capas de profundidad (véase Figura). Puede cargar una versión preentrenada de la red entrenada en más de un millón de imágenes desde la base de datos de ImageNet. La red preentrenada puede clasificar imágenes en 1000 categorías de objetos (por ejemplo, teclado, ratón, lápiz y animales). Como resultado, la red ha aprendido representaciones ricas en características para una gran gama de imágenes. El tamaño de la entrada de imagen de la red es de 224 por 224 [6, 7].

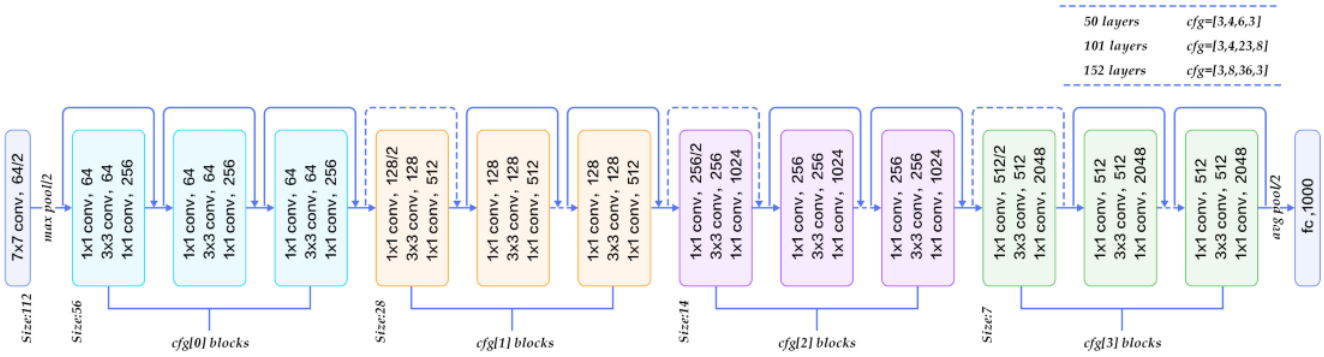


Figura 5: Estructura de red convolucional ResNet50.

2.8. Red EfficientNetB0

Esta red tiene 237 capas divididas en 5 modulos, 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 [8, 9, 10]

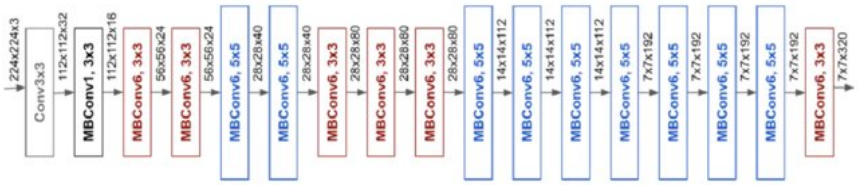


Figura 6: Estructura de red convolucional EfficientNetB0.

2.9. Red InceptionV3

Esta red convolucional esta especializada en la clasificación de imágenes, consistente en la concatenación de filtros en conjunto con funciones de optimización que pueden acelerar su procesamiento en comparación a la primera versión y posee clasificadores auxiliares, siendo esto último, la mayor diferencia entre la versión 1 y la 3 [11].

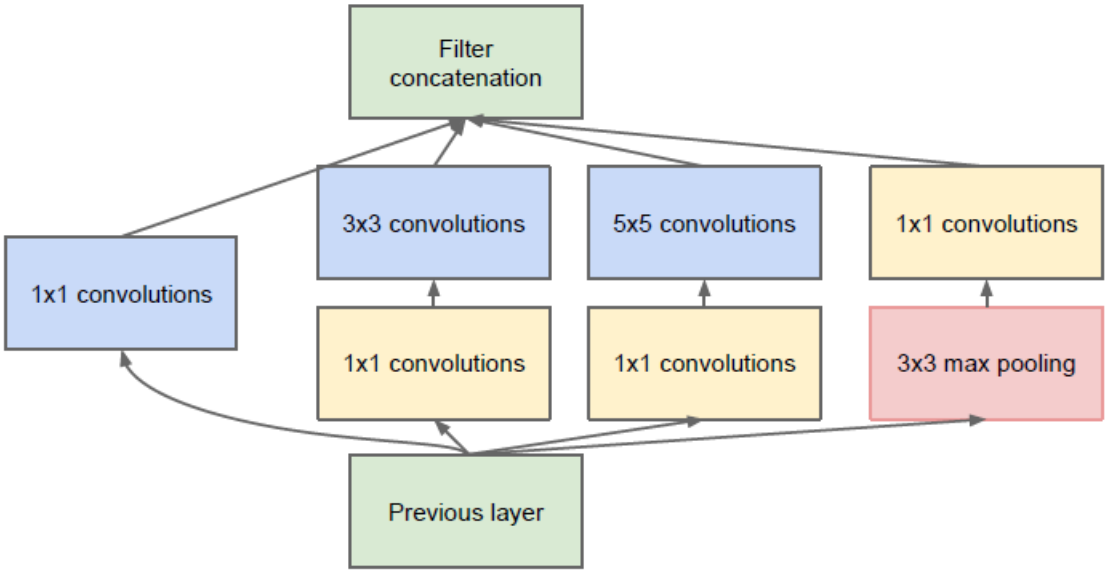


Figura 7: Estructura de red convolucional InceptionV3.

3. Justificación

El uso de redes convolucionales para la clasificación de imágenes, son de gran utilidad para una buena cantidad de aplicaciones practicas. Dentro de la materia, es posible decir que se integran los conocimientos de las anteriores prácticas y se aterrizan topicos de este enorme tema

4. Resultados

Se observa que el uso de Transfer Learning es de gran utilidad para realizar una clasificación de imágenes binaria (perro o gato), esto permite que las redes puedan tener un entrenamiento más rápido y permite que algunos parámetros no sean entrenados, mejorando computacionalmente el tiempo de ejecución. Para el proceso se requirió aplicar una normalización de las imágenes, estableciendo tamaños iguales para todas. La red LeNet5, debido a sus limitaciones debido a su estructura tan sencilla y limitada, obtuvo un overfitting debido a que aprendió totalmente el entrenamiento pero al momento de validar y comprobar en la prueba, los resultados fueron totalmente diferentes.

Por otro lado las redes de *keras.Applications*, permitieron tener más flexibilidad con esto y se pueden tener imágenes mas complejas con mayor resolución, de esta manera es que es posible tener un mejor resultado. Cabe destacar que el uso de Transferlearning fue totalmente útil y necesario para obtener los mejores resultados. Asi como también se observó que cada red, requería un preprocesamiento diferente, por lo que se cuenta con el método *preprocess_input()*.

Tabla 1: Resultados de redes neuronales convolucionales.

Método	Tamaño imagen	Épocas	Batch	% Entrenamiento	% Validación	% Test
Lenet 5	(32,32,1)	100	64	100	63.63	62.625
Resnet50 (Imagenet)	(150,150,3)	10	64	99	96.63	94.75
Resnet50 (pesos aleatorios)	(150,150,3)	10	64	57.71	62.2	55.5
EfficientNetB0 (Imagenet)	(150,150,3)	10	64	95.46	92	92.875
EfficientNetB0 (pesos aleatorios)	(150,150,3)	10	64	54.62	50	50
InceptionV3 (Imagenet)	(150,150,3)	10	64	94.71	85.87	87.125
InceptionV3 (pesos aleatorios)	(150,150,3)	10	64	54.25	56.75	47.5

5. Conclusiones

1. El uso de Transfer Learning es una buena manera de poder tener un preentreno para la red neuronal y de este modo no se requiere entrenar una parte de la red.

2. Cada red neuronal convolucional requiere su propia normalización y es posible realizarla mediante el método *preprocess_input()*.

3. Mediante el uso del método *Include_top=False*, es posible desacoplar la parte de aplanar los datos y adecuarlo a nuestra conveniencia para ajustarse a la configuración de One-Hot.

4. Se observó un tiempo de ejecución mucho mayor en las redes que tenían pesos aleatorios iniciales a pesar de tener la misma cantidad de valores entrenables.

5. El uso de la función de activación *softmax* permitió trabajar con el One Hot y poder categorizar los valores. Por parte de la red neuronal Perceptrón, no se observó mayor diferencia al cambiar su estructura.

Se observó que fue posible obtener buenos resultados con Imagenet, sin embargo, considero que iniciando con pesos aleatorios también podría tener éxito si se incrementan las épocas, aunque sea bastante complicado computar tanta información, pudiendo inclusive lograr que la computadora se congele.

Referencias

[1] "Imagenet." https://www.image-net.org/. (Accessed on 11/08/2022).

[2] "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).

[3] "5.2. imágenes rgb — introducción a la programación." https://cupi2-ip.github.io/IPBook/nivel4/seccion4-4.html. (Accessed on 09/17/2022).

[4] "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).

[5] "Intro a las redes neuronales convolucionales — by bootcamp ai — medium." https://bootcampai.medium.com/redes-neuronales-convolucionales-5e0ce960caf8. (Accessed on 09/26/2022).

[6] "Deep residual networks (resnet, resnet50) 2022 guide - viso.ai." https://viso.ai/deep-learning/resnet-residual-neural-network/. (Accessed on 11/08/2022).

[7] "Resnet50. resnet-50 is a convolutional neural... — by aditi rastogi — dev genius." https://blog.devgenius.io/resnet50-6b42934db431. (Accessed on 11/08/2022).

[8] "(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. (Accessed on 11/08/2022).

[9] "Complete architectural details of all efficientnet models — by vardan agarwal — towards data science." https://towardsdatascience.com/complete-architectural-details-of-all-efficientnet-models-5fd5b736142. (Accessed on 11/08/2022).

[10] "[1905.11946] efficientnet: Rethinking model scaling for convolutional neural networks." https://arxiv.org/abs/1905.11946. (Accessed on 11/08/2022).

[11] "Inception v3 model architecture." https://iq.opengenus.org/inception-v3-model-architecture/. (Accessed on 11/08/2022).

Tarea_Practica_8_Aldo_Cervantes

November 9, 2022

1 Uso de archivo zip

uso de zip para descomprimir y obtener los datos

[36]:
```
import h5py
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=[]
```

2 Filtrado de datos en gatos y

[37]:
```
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)
```

```
2000 2000
```

[38]:
```
"""
a=cv2.imread(name)
a=cv2.resize(a,(200,200))
print(a.shape)
img = cv2.cvtColor(a, cv2.COLOR_BGR2RGB)
print(img.shape)
"""
```

[38]:
```
'\na=cv2.imread(name)\na=cv2.resize(a,(200,200))\nprint(a.shape)\nimg =
cv2.cvtColor(a, cv2.COLOR_BGR2RGB)\nprint(img.shape)\n'
```

[ ]:
```

```

3 Normalización de datos

[39]:
```
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)
```

```
<class 'numpy.ndarray'> (2000, 150, 150, 3)
<class 'numpy.ndarray'> (2000, 150, 150, 3)
4 (4000, 150, 150, 3)
[[0]
 [0]
 [0]
 ...
 [0]
 [0]
 [0]]
```

[40]:
```
### 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)
```

```
[1]
[0. 1.]
<class 'numpy.ndarray'> (4000, 2)
(2000, 2)
(2000, 2)
[[1. 0.]
 [1. 0.]
 [1. 0.]
 ...
 [1. 0.]
 [1. 0.]
 [1. 0.]]
```

4 División 60 20 20

[41]:
```
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())
```

```
0.0
```

5 Red LeNet5

5.1 preprocessing y deshabilitar entrenamiento

[42]:
```
"""
import tensorflow as tf
from keras.models import Model, load_model
#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

lr_reduce = ReduceLROnPlateau(monitor='val_accuracy', factor=0.6, patience=8,␣
↪verbose=1, mode='max', min_lr=5e-5)
checkpoint =  ModelCheckpoint('vgg16_finetune.h5', 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=np.vstack((x_cat[:int(xx*0.6),:,:,1:2],x_dog[:int(xx*0.6),:,:,1:2]))
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),:,:,1:2],x_dog[int(xx*0.6):
↪int(xx*0.8),:,:,1:2]))
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):,:,:,1:2],x_dog[int(xx*0.8):,:,:,1:2]))
y_test=np.vstack((y_cat_oh[int(xx*0.8):,:],y_dog_oh[int(xx*0.8):,:]))


model=Sequential()
model.add(tf.keras.layers.
↪Conv2D(6,(5,5),input_shape=(x_size,y_size,1),activation='tanh',padding='valid',strides=1))␣
↪#C1

model.add(tf.keras.layers.AveragePooling2D(pool_size=(2,2))) #S2

model.add(tf.keras.layers.
↪Conv2D(16,(5,5),activation='tanh',padding='valid',strides=1)) #c3

model.add(tf.keras.layers.AveragePooling2D(pool_size=(2,2))) #s4
model.add(tf.keras.layers.Flatten())

model.add(Dense(120,activation='tanh')) #c5
model.add(Dense(84,activation='tanh')) #c6
```

```
from keras.layers import Layer
from keras import backend as K

class RBFLayer(Layer):
    def __init__(self, units, gamma, **kwargs):
        super(RBFLayer, self).__init__(**kwargs)
        self.units = units
        self.gamma = K.cast_to_floatx(gamma)

    def build(self, input_shape):
        self.mu = self.add_weight(name='mu',
                                  shape=(int(input_shape[1]), self.units),
                                  initializer='uniform',
                                  trainable=True)
        super(RBFLayer, self).build(input_shape)

    def call(self, inputs):
        diff = K.expand_dims(inputs) - self.mu
        l2 = K.sum(K.pow(diff,2), axis=1)
        res = K.exp(-1 * self.gamma * l2)
        return res

    def compute_output_shape(self, input_shape):
        return (input_shape[0], self.units)

model.add(RBFLayer(2,0.5)) #c7

model.compile(loss='categorical_crossentropy',optimizer=tf.keras.optimizers.
 ↪SGD(learning_rate=0.25),metrics=['accuracy'])
model.fit(x_train,y_train,verbose=1,␣
 ↪batch_size=64,epochs=100,validation_data=(x_val,y_val))
"""
```

[42]:
```
"\nimport tensorflow as tf\nfrom keras.models import Model, load_model\n#from
tensorflow.keras.applications.resnet50 import preprocess_input,
decode_predictions\nfrom tensorflow.keras.models import Sequential\nfrom
tensorflow.keras.layers import Dense,Flatten,Dropout,Input\nfrom
tensorflow.keras.callbacks import ModelCheckpoint, EarlyStopping,
ReduceLROnPlateau\n\nlr_reduce = ReduceLROnPlateau(monitor='val_accuracy',
factor=0.6, patience=8, verbose=1, mode='max', min_lr=5e-5)\ncheckpoint =
ModelCheckpoint('vgg16_finetune.h5', monitor= 'val_accuracy', mode= 'max',
save_best_only = True, verbose= 1)\nearlystopper = EarlyStopping(monitor =
'val_loss', min_delta = 0, patience = 10, verbose = 1, restore_best_weights =
True) \n\nx_train=np.vstack((x_cat[:int(xx*0.6),:,:,1:2],x_dog[:int(xx*0.6),:,:,
1:2]))\ny_train=np.vstack((y_cat_oh[:int(xx*0.6),:],y_dog_oh[:int(xx*0.6),:]))\n
x_val=np.vstack((x_cat[int(xx*0.6):int(xx*0.8),:,:,1:2],x_dog[int(xx*0.6):int(xx
*0.8),:,:,1:2]))\ny_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),:]))\nx_test=np.vstack((x_cat[int(xx*0.8):,:,:,1:2],x_do
g[int(xx*0.8):,:,:,1:2]))\ny_test=np.vstack((y_cat_oh[int(xx*0.8):,:],y_dog_oh[i
nt(xx*0.8):,:]))\n\n\nmodel=Sequential()\nmodel.add(tf.keras.layers.Conv2D(6,(5,
5),input_shape=(x_size,y_size,1),activation='tanh',padding='valid',strides=1))
#C1\n\nmodel.add(tf.keras.layers.AveragePooling2D(pool_size=(2,2))) #S2\n\nmodel
.add(tf.keras.layers.Conv2D(16,(5,5),activation='tanh',padding='valid',strides=1
)) #c3\n\nmodel.add(tf.keras.layers.AveragePooling2D(pool_size=(2,2))) #s4\nmode
l.add(tf.keras.layers.Flatten())\n\nmodel.add(Dense(120,activation='tanh'))
#c5\nmodel.add(Dense(84,activation='tanh')) #c6\n\nfrom keras.layers import
Layer\nfrom keras import backend as K\n\nclass RBFLayer(Layer):\n    def
__init__(self, units, gamma, **kwargs):\n        super(RBFLayer,
self).__init__(**kwargs)\n        self.units = units\n        self.gamma =
K.cast_to_floatx(gamma)\n\n    def build(self, input_shape):\n        self.mu =
self.add_weight(name='mu',\n
shape=(int(input_shape[1]), self.units),\n
initializer='uniform',\n                                    trainable=True)\n
super(RBFLayer, self).build(input_shape)\n\n    def call(self, inputs):\n
diff = K.expand_dims(inputs) - self.mu\n        l2 = K.sum(K.pow(diff,2),
axis=1)\n        res = K.exp(-1 * self.gamma * l2)\n        return res\n\n
def compute_output_shape(self, input_shape):\n        return (input_shape[0],
self.units)\n    \nmodel.add(RBFLayer(2,0.5)) #c7\n\nmodel.compile(loss='categor
ical_crossentropy',optimizer=tf.keras.optimizers.SGD(learning_rate=0.25),metrics
=['accuracy'])\nmodel.fit(x_train,y_train,verbose=1,
batch_size=64,epochs=100,validation_data=(x_val,y_val))\n"
```

[43]:
```
"""
pred=model.predict(x_test)
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),'%')
"""
```

[43]:
```
"\npred=model.predict(x_test)\npred=np.argmax(pred,axis=1)\ny1=np.argmax(y_test,
axis=1)\n\n#label=np.argmax(yp_oh)\nexactitud_test=0\nfor a in
range(len(pred)):\n    if pred[a]==y1[a]:\n
exactitud_test+=1\nprint('exactitud de la prueba=
',100*exactitud_test/len(pred),'%')\n"
```

Red ResNet50

[48]:
```
import tensorflow as tf
from keras.models import Model, load_model
from tensorflow.keras.applications.resnet50 import ResNet50
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

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.resnet50.preprocess_input(np.copy(x_train))
x_val_res=tf.keras.applications.resnet50.preprocess_input(np.copy(x_val))
x_test_res=tf.keras.applications.resnet50.preprocess_input(np.copy(x_test))
model = ResNet50(input_shape=(x_size,y_size,3),weights=None,include_top=False)␣
↪## Colocar otro top
for i in range(165):
    model.layers[i].trainable=False
#model.summary()
sal=model.output
sal=Flatten()(sal)
sal=Dense(502,activation='selu')(sal)
sal=Dropout(0.26)(sal)
sal = Dense(256, activation='relu')(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
resnet50_custom=Model(inputs=model.input, outputs=sal)
resnet50_custom.compile(loss="categorical_crossentropy", optimizer=tf.keras.
↪optimizers.SGD(learning_rate=0.001), metrics=["accuracy"])
history=resnet50_custom.fit(x_train_res,y_train,batch_size=64,epochs=10,␣
↪validation_data=(x_val_res,y_val),callbacks=[lr_reduce,earlystopper,checkpoint])
#model.summary()
#preds=model.predict(x_comb)
#model.add(tf.keras.layers.Flaten())
"""
tf.keras.applications.MobileNet(
    input_shape=(200,200,3),
    alpha=1.0,
    depth_multiplier=1,
```

```
    dropout=0.001,
    include_top=True,
    weights="imagenet",
    input_tensor=None,
    pooling=None,
    classes=1000,
    classifier_activation="softmax",
    **kwargs
)
"""
```

```
Epoch 1/10
38/38 [==============================] - 57s 1s/step - loss: 1.3805 - accuracy:
0.4900 - val_loss: 0.7172 - val_accuracy: 0.5075

Epoch 00001: val_accuracy improved from -inf to 0.50750, saving model to
resnet50.h
INFO:tensorflow:Assets written to: resnet50.h\assets

C:\Users\aldoa\anaconda3\envs\env2\lib\site-
packages\keras\utils\generic_utils.py:494: CustomMaskWarning: Custom mask layers
require a config and must override get_config. When loading, the custom mask
layer must be passed to the custom_objects argument.
  warnings.warn('Custom mask layers require a config and must override '

Epoch 2/10
38/38 [==============================] - 51s 1s/step - loss: 0.7683 - accuracy:
0.5046 - val_loss: 0.7103 - val_accuracy: 0.5100

Epoch 00002: val_accuracy improved from 0.50750 to 0.51000, saving model to
resnet50.h
INFO:tensorflow:Assets written to: resnet50.h\assets

C:\Users\aldoa\anaconda3\envs\env2\lib\site-
packages\keras\utils\generic_utils.py:494: CustomMaskWarning: Custom mask layers
require a config and must override get_config. When loading, the custom mask
layer must be passed to the custom_objects argument.
  warnings.warn('Custom mask layers require a config and must override '

Epoch 3/10
38/38 [==============================] - 52s 1s/step - loss: 0.7437 - accuracy:
0.5113 - val_loss: 0.7002 - val_accuracy: 0.5013

Epoch 00003: val_accuracy did not improve from 0.51000
Epoch 4/10
38/38 [==============================] - 53s 1s/step - loss: 0.7219 - accuracy:
0.5258 - val_loss: 0.6868 - val_accuracy: 0.5512

Epoch 00004: val_accuracy improved from 0.51000 to 0.55125, saving model to
resnet50.h
```

```
INFO:tensorflow:Assets written to: resnet50.h\assets

C:\Users\aldoa\anaconda3\envs\env2\lib\site-
packages\keras\utils\generic_utils.py:494: CustomMaskWarning: Custom mask layers
require a config and must override get_config. When loading, the custom mask
layer must be passed to the custom_objects argument.
  warnings.warn('Custom mask layers require a config and must override '

Epoch 5/10
38/38 [==============================] - 51s 1s/step - loss: 0.7222 - accuracy:
0.5179 - val_loss: 0.6867 - val_accuracy: 0.5375

Epoch 00005: val_accuracy did not improve from 0.55125
Epoch 6/10
38/38 [==============================] - 53s 1s/step - loss: 0.7098 - accuracy:
0.5192 - val_loss: 0.6827 - val_accuracy: 0.5437

Epoch 00006: val_accuracy did not improve from 0.55125
Epoch 7/10
38/38 [==============================] - 54s 1s/step - loss: 0.7025 - accuracy:
0.5242 - val_loss: 0.6769 - val_accuracy: 0.5875

Epoch 00007: val_accuracy improved from 0.55125 to 0.58750, saving model to
resnet50.h
INFO:tensorflow:Assets written to: resnet50.h\assets

C:\Users\aldoa\anaconda3\envs\env2\lib\site-
packages\keras\utils\generic_utils.py:494: CustomMaskWarning: Custom mask layers
require a config and must override get_config. When loading, the custom mask
layer must be passed to the custom_objects argument.
  warnings.warn('Custom mask layers require a config and must override '

Epoch 8/10
38/38 [==============================] - 52s 1s/step - loss: 0.6857 - accuracy:
0.5558 - val_loss: 0.6722 - val_accuracy: 0.5838

Epoch 00008: val_accuracy did not improve from 0.58750
Epoch 9/10
38/38 [==============================] - 53s 1s/step - loss: 0.6897 - accuracy:
0.5617 - val_loss: 0.6840 - val_accuracy: 0.5437

Epoch 00009: val_accuracy did not improve from 0.58750
Epoch 10/10
38/38 [==============================] - 53s 1s/step - loss: 0.6804 - accuracy:
0.5771 - val_loss: 0.6654 - val_accuracy: 0.6200

Epoch 00010: val_accuracy improved from 0.58750 to 0.62000, saving model to
resnet50.h
INFO:tensorflow:Assets written to: resnet50.h\assets
```

```
C:\Users\aldoa\anaconda3\envs\env2\lib\site-
packages\keras\utils\generic_utils.py:494: CustomMaskWarning: Custom mask layers
require a config and must override get_config. When loading, the custom mask
layer must be passed to the custom_objects argument.
  warnings.warn('Custom mask layers require a config and must override '
```

[48]:
```
'\ntf.keras.applications.MobileNet(\n    input_shape=(200,200,3),\n
alpha=1.0,\n    depth_multiplier=1,\n    dropout=0.001,\n    include_top=True,\n
weights="imagenet",\n    input_tensor=None,\n    pooling=None,\n
classes=1000,\n    classifier_activation="softmax",\n    **kwargs\n)\n'
```

[49]:
```
#preds2=np.argmax(preds,axis=1)
## Sale 111
pred=resnet50_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),'%')
```

```
exactitud de la prueba=  55.5 %
```

6 EfficientNetB0

[53]:
```
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

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=None,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])
```

```
Epoch 1/10
38/38 [==============================] - 114s 3s/step - loss: 0.7817 - accuracy:
0.5121 - val_loss: 0.6955 - val_accuracy: 0.5000

Epoch 00001: val_accuracy improved from -inf to 0.50000, saving model to
resnet50.h
INFO:tensorflow:Assets written to: resnet50.h\assets

C:\Users\aldoa\anaconda3\envs\env2\lib\site-
packages\keras\utils\generic_utils.py:494: CustomMaskWarning: Custom mask layers
require a config and must override get_config. When loading, the custom mask
layer must be passed to the custom_objects argument.
  warnings.warn('Custom mask layers require a config and must override '

Epoch 2/10
38/38 [==============================] - 108s 3s/step - loss: 0.7432 - accuracy:
0.5217 - val_loss: 0.7003 - val_accuracy: 0.5000

Epoch 00002: val_accuracy did not improve from 0.50000
Epoch 3/10
38/38 [==============================] - 106s 3s/step - loss: 0.7194 - accuracy:
0.5454 - val_loss: 0.6995 - val_accuracy: 0.5000

Epoch 00003: val_accuracy did not improve from 0.50000
Epoch 4/10
38/38 [==============================] - 103s 3s/step - loss: 0.7187 - accuracy:
```

```
0.5354 - val_loss: 0.7034 - val_accuracy: 0.5000

Epoch 00004: val_accuracy did not improve from 0.50000
Epoch 5/10
38/38 [==============================] - 106s 3s/step - loss: 0.7089 - accuracy:
0.5279 - val_loss: 0.7149 - val_accuracy: 0.5000

Epoch 00005: val_accuracy did not improve from 0.50000
Epoch 6/10
38/38 [==============================] - 113s 3s/step - loss: 0.7175 - accuracy:
0.5312 - val_loss: 0.7100 - val_accuracy: 0.5000

Epoch 00006: val_accuracy did not improve from 0.50000
Epoch 7/10
38/38 [==============================] - 111s 3s/step - loss: 0.7047 - accuracy:
0.5417 - val_loss: 0.7087 - val_accuracy: 0.5000

Epoch 00007: val_accuracy did not improve from 0.50000
Epoch 8/10
38/38 [==============================] - 113s 3s/step - loss: 0.6963 - accuracy:
0.5500 - val_loss: 0.7104 - val_accuracy: 0.5000

Epoch 00008: val_accuracy did not improve from 0.50000
Epoch 9/10
38/38 [==============================] - 122s 3s/step - loss: 0.6931 - accuracy:
0.5567 - val_loss: 0.7237 - val_accuracy: 0.5000

Epoch 00009: ReduceLROnPlateau reducing learning rate to 0.0006000000284984708.

Epoch 00009: val_accuracy did not improve from 0.50000
Epoch 10/10
38/38 [==============================] - 117s 3s/step - loss: 0.6919 - accuracy:
0.5462 - val_loss: 0.7225 - val_accuracy: 0.5000

Epoch 00010: val_accuracy did not improve from 0.50000
```

[54]:
```python
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),'%')
```

```
exactitud de la prueba=  50.0 %
```

7 InceptionV3

[57]:
```
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.inception_v3.preprocess_input(np.copy(x_train))
x_val_res=tf.keras.applications.inception_v3.preprocess_input(np.copy(x_val))
x_test_res=tf.keras.applications.inception_v3.preprocess_input(np.copy(x_test))

model = tf.keras.applications.
 ↪InceptionV3(input_shape=(x_size,y_size,3),weights=None,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
incv3_custom=Model(inputs=model.input, outputs=sal)
incv3_custom.compile(loss="categorical_crossentropy", optimizer=tf.keras.
 ↪optimizers.SGD(learning_rate=0.001), metrics=["accuracy"])
history=incv3_custom.fit(x_train_res,y_train,batch_size=64,epochs=10,␣
 ↪validation_data=(x_val_res,y_val),callbacks=[lr_reduce,earlystopper,checkpoint])
```

```
Epoch 1/10
38/38 [==============================] - 66s 2s/step - loss: 0.7774 - accuracy:
0.4929 - val_loss: 0.6936 - val_accuracy: 0.4613

Epoch 00001: val_accuracy improved from -inf to 0.46125, saving model to
resnet50.h
INFO:tensorflow:Assets written to: resnet50.h\assets
Epoch 2/10
38/38 [==============================] - 60s 2s/step - loss: 0.7294 - accuracy:
0.5183 - val_loss: 0.6953 - val_accuracy: 0.5000

Epoch 00002: val_accuracy improved from 0.46125 to 0.50000, saving model to
resnet50.h
INFO:tensorflow:Assets written to: resnet50.h\assets
Epoch 3/10
```

```
38/38 [==============================] - 59s 2s/step - loss: 0.7275 - accuracy:
0.5083 - val_loss: 0.6953 - val_accuracy: 0.4712

Epoch 00003: val_accuracy did not improve from 0.50000
Epoch 4/10
38/38 [==============================] - 60s 2s/step - loss: 0.7169 - accuracy:
0.5146 - val_loss: 0.6920 - val_accuracy: 0.5125

Epoch 00004: val_accuracy improved from 0.50000 to 0.51250, saving model to
resnet50.h
INFO:tensorflow:Assets written to: resnet50.h\assets
Epoch 5/10
38/38 [==============================] - 60s 2s/step - loss: 0.7096 - accuracy:
0.5158 - val_loss: 0.6931 - val_accuracy: 0.5200

Epoch 00005: val_accuracy improved from 0.51250 to 0.52000, saving model to
resnet50.h
INFO:tensorflow:Assets written to: resnet50.h\assets
Epoch 6/10
38/38 [==============================] - 59s 2s/step - loss: 0.7122 - accuracy:
0.5029 - val_loss: 0.6966 - val_accuracy: 0.4975

Epoch 00006: val_accuracy did not improve from 0.52000
Epoch 7/10
38/38 [==============================] - 59s 2s/step - loss: 0.7072 - accuracy:
0.5138 - val_loss: 0.6998 - val_accuracy: 0.4900

Epoch 00007: val_accuracy did not improve from 0.52000
Epoch 8/10
38/38 [==============================] - 61s 2s/step - loss: 0.7111 - accuracy:
0.5129 - val_loss: 0.6946 - val_accuracy: 0.5213

Epoch 00008: val_accuracy improved from 0.52000 to 0.52125, saving model to
resnet50.h
INFO:tensorflow:Assets written to: resnet50.h\assets
Epoch 9/10
38/38 [==============================] - 60s 2s/step - loss: 0.7062 - accuracy:
0.5133 - val_loss: 0.6957 - val_accuracy: 0.5387

Epoch 00009: val_accuracy improved from 0.52125 to 0.53875, saving model to
resnet50.h
INFO:tensorflow:Assets written to: resnet50.h\assets
Epoch 10/10
38/38 [==============================] - 60s 2s/step - loss: 0.6973 - accuracy:
0.5425 - val_loss: 0.7054 - val_accuracy: 0.5675

Epoch 00010: val_accuracy improved from 0.53875 to 0.56750, saving model to
resnet50.h
```

```
INFO:tensorflow:Assets written to: resnet50.h\assets
```

[58]:
```
pred=incv3_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),'%')
```

```
exactitud de la prueba=  47.5 %
```

[ ]: