

PROYECTO FINAL

”WATCHPET”

SISTEMAS INTELIGENTES SIS-341

Weimar Adalid Condori Yupanqui
Universidad Católica San Pablo
Ingeniería Mecatrónica
La Paz - Bolivia
Email: weimar.condori@ucb.edu.bo

Brayan Gerson Durán Toconás
Universidad Católica San Pablo
Ingeniería Mecatrónica
La Paz - Bolivia
Email: brayan.duran.t@ucb.edu.bo

Jhonn Santa Cruz
Universidad Católica San Pablo
Ingeniería Mecatrónica
La Paz - Bolivia
Email: jhonn.santacruz@ucb.edu.bo

Belen
Universidad Católica San Pablo
Ingeniería Mecatrónica
La Paz - Bolivia
Email: jhonn.santacruz@ucb.edu.bo

Abstract—El presente proyecto se muestra el desarrollo de un programa en Python para la clasificación de perros y gatos. Dicha clasificación está basada en modelos de predicción (Denso,CNN) entrenados mediante un Dataset de, 23262 imágenes de perros y gatos, y testeado con imágenes externas al Dataset original.

1. Introducción

En la actualidad el empleo del deep learned o aprendizaje supervisado es muy recurrente, estos aparecen en nuestra vida cotidiana para mejorarla, como objetivo académico, este documento tiene la intención de hacer posible la aplicación de un programa de aprendizaje supervisado capaz de predecir una etiqueta mediante una imagen.

Para introducirse al proyecto, se presentarán los siguientes conceptos.

1.1. Aprendizaje supervisado

Es un método para inferir una función a partir de datos de entrenamiento en aprendizaje automático y minería de datos. Dos objetos por par componen los datos de entrenamiento.

1.2. Red convolucional

La red neuronal convolucional, también conocida como convnets o CNN, es un método bien conocido en aplicaciones de visión por ordenador. Este tipo de arquitectura es dominante para reconocer objetos de una imagen o video.

1.3. TensorFlow

Google creó una biblioteca de aprendizaje automático llamada TensorFlow para abordar las necesidades actuales.

2. Objetivos

2.1. Objetivo general

Implementar modelos computacionales para la identificación de perros y gatos mediante técnicas de procesamiento de imágenes basadas en contenido y aprendizaje profundo.

2.2. Objetivos específicos

- Construir un conjunto de datos de imágenes de perros y gatos para su procesamiento y extracción de características utilizando diferentes fuentes de recolección.
- Diseñar una estrategia metodológica para la identificación de perros y gatos en fotos a partir de técnicas de recuperación de imágenes y aprendizaje profundo.
- Desarrollar modelos computacionales para la identificación de perros y gatos en fotos a partir del entrenamiento de los algoritmos seleccionados.
- Evaluar el rendimiento de los modelos computacionales propuestos para su implementación, empleando las métricas de desempeño precisión (accuracy, perdida).

3. Código

En esta sección se presentará el código para el funcionamiento del proyecto. El código se divide en 3 grandes partes (Preparación de datos, Entrenamiento, Predicción), todas estas partes, están encargadas de cumplir con la clasificación adecuada de las imágenes.

3.1. Preparación de los datos

Para el presente proyecto se empleó un Dataset oficial de TensorFlow. Dicho Dataset tiene el nombre de "gatos_vs_perros" y se compone de imágenes de perros y gatos, además de una etiqueta para identificar la especie de cada imagen.

```
[7]: #Creo la variable que contendrá todos los pares de los datos (imagen y
...etiquetas) ya modificadas (blanco y negro). 100x100
datos_entrenamiento = []

[8]: #Guardamos las imágenes en un tamón de TAMANO_IMG (100x100) y lo convertimos a,
...blanco y negro
for i, (imagen, etiqueta) in enumerate(datos['train']):
    #Doblo los datos
    imagen = cv2.resize(imagen, (TAMANO_IMG, TAMANO_IMG))
    imagen = cv2.cvtColor(imagen, cv2.COLOR_BGR2GRAY)
    imagen = imagen.reshape(TAMANO_IMG, TAMANO_IMG, 1) #Cadastrar tamaño a 100,100,1
    datos_entrenamiento.append((imagen, etiqueta)) #Guardamos en la maza,
...variable creada anteriormente
```

main

December 12, 2022

```
[1]: import tensorflow as tf
import tensorflow_datasets as tfds
#Cargar datos de archivo "kagglecatsanddogs_332q.zip"
datos, metadatos = tfds.load('cats_vs_dogs', as_supervised=True, with_info=True)

[2]: #Imprimir los metadatos para revisarlos
metadatos

[3]: tfds.core.DatasetInfo(
    name='cats_vs_dogs',
    version='2018_04_04',
    description='A large set of images of cats and dogs. There are 1738 corrupted images that are dropped.
homepage: https://www.microsoft.com/en-us/download/details.aspx?id=84765',
    data_path='C:/Users/V9440\TensorFlow\datasets\cats_vs_dogs\4.0.0',
    features=FeaturesDict({
        'image': Feature(shape=(None, None, 3), dtype=tf.uint8),
        'label': Feature(shape=(), dtype=tf.string),
        'label_binarized': Feature(shape=(), dtype=tf.int64, num_classes=2),
    }),
    supervised_key='image',
    label_key='label',
    disable_shuffling=False,
    splits={
        'train': tfds.SplitInfo(num_examples=23262, num_shards=0),
    },
    citations='''@InProceedings{asirra-a-captcha-that-exploits-interest-aligned-manual-image-categorization,
author={Elson, Jersey and Doucer, John (JD) and Howell, Jon and Saul, Jared},
title={Asirra: A CAPTCHA that Exploits Interest-Aligned Manual Image Categorization},
booktitle={Proceedings of 14th ACM Conference on Computer and Communications Security (CCS)},
year = {2007},
month = {October},
publisher = {Association for Computing Machinery, Inc.},
url = {https://www.microsoft.com/en-us/research/publication/asirra-a-captcha-that-exploits-interest-aligned-manual-image-categorization/},
edition = {Proceedings of 14th ACM Conference on Computer and Communications Security (CCS)},
jyear = {2007}
}''',
    #Otro ejemplo de los datos de entrenamiento del dataset
    tfds_as_dataframe(datos['train']).take(10), metadatos)
```

```
[5]: #Mostrar un ejemplo de los datos de entrenamiento del dataset
tfds_as_dataframe(datos['train']).take(10), metadatos

[6]: #Lo pasamos a TAMANO_IMG (100x100) y lo convertimos a blanco y negro *(solo
...píxeles visibles)*
import tensorflow as tf
import cv2
plt.figure(figsize=(20,20))

TAMANO_IMG=100

for i, (imagen, etiqueta) in enumerate(datos['train'].take(25)):
    imagen = cv2.resize(imagen, (TAMANO_IMG, TAMANO_IMG))
    imagen = cv2.cvtColor(imagen, cv2.COLOR_BGR2GRAY)
    plt.imshow(imagen, cmap='gray')
    plt.xticks([])
    plt.yticks([])
    plt.show(imagen, cmap='gray')
```



```
[7]: #Ver cuantos datos tengo en la variable
len(datos_entrenamiento)

[10]: 23262

[11]: #Separar los datos
#Separar mis variables X (entradad) y y (etiquetas) separadas
X = np.array([x for x, y in datos_entrenamiento])
y = np.array([y for x, y in datos_entrenamiento])

for imagen, etiqueta in datos_entrenamiento:
    X.append(imagen)
    y.append(etiqueta)
```

```
[13]: #Normalizar los datos de los X (imágenes). Se pasan a numero flotante y dividien
...entre 255 para pedir de 0-1 en lugar de 0-255
#Para poder entrenar de manera optima
import numpy as np
X = np.array(X).astype(float) / 255
```

```
[16]: #Convertir etiquetas en arreglo simple
y = np.array(y)
```

```
[17]: #Datos tipo array de numpy
y
```

```
[17]: array([1., 1., ..., 0., 1., 0.], dtype=int64)
```

```
[18]: #Los tamaños de "X" y "y"
print(X.shape)
print(y.shape)
```

```
(23262, 100, 100, 1)
(23262,)
```

```
[19]: #Ver los modelos iniciales
#Nos sirven como salida (en lugar de softmax) para mostrar como podría
...funcionar con dicha función de activación.
#Seguiremos probando diferentes datos y ver si mejoran. Realizamos el entrenamiento para el
...final considerar que si la respuesta es
#cerco a 0, es un gato, y si se acerca a 1, es un perro.
```

```
modeloDense = tf.keras.models.Sequential([
    tf.keras.layers.Flatten(input_shape=(100, 100, 1)),
    tf.keras.layers.Dense(100, activation='relu'),
    tf.keras.layers.Dense(1, activation='relu'),
```

```
4
```

Figure 1. Código preparación de los datos

3.2. Entrenamiento

Para el entrenamiento se decidió entrenar al modelo con nuevos datos generados y con los originales del dataset, por otra parte, se escogieron 3 arquitecturas, una con una red neuronal densa y dos con una red neuronal convolucional ideal para imágenes. La configuración para cada arquitectura se presenta en el código a continuación.

```

[10]: #Ver cuantos datos tengo en la variable
len(datos_entrenamiento)

[10]: 23262

[11]: #separamos datos
#Preparar mis variables X (entradas) y y (etiquetas) separadas
X = [] #imagenes de entrada (pixeles / fotos)
y = [] #etiquetas (perro o gato / indice)

for imagen, etiqueta in datos_entrenamiento:
    X.append(imagen)
    y.append(etiqueta)

[13]: #Normalizar los datos de las X (imagenes). Se pasan a numero flotante y dividen
#entre 255 para quedar de 0-1 en lugar de 0-255
#Para poder entrenar de manera optima
import numpy as np

X = np.array(X).astype(float) / 255

[16]: #Convertir etiquetas en arreglo simple
y = np.array(y)

[17]: #Datos tipo array de numpy
y

[17]: array([1, 1, 1, ..., 0, 1, 0], dtype=int64)

[18]: #Los tamaños de "X" y "y"
print(X.shape)
print(y.shape)

(23262, 100, 100, 1)
(23262,)

[19]: #Crear los modelos iniciales
#Usan sigmoid como salida (en lugar de softmax) para mostrar como podria
#funcionar con dicha funcion de activacion.
#Sigmoid regresa siempre datos entre 0 y 1. Realizamos el entrenamiento para al
#final considerar que si la respuesta se
#acerca a 0, es un gato, y si se acerca a 1, es un perro.

modeloDenso = tf.keras.models.Sequential([
    tf.keras.layers.Flatten(input_shape=(100, 100, 1)),
    tf.keras.layers.Dense(150, activation='relu'),
    tf.keras.layers.Dense(150, activation='relu'),
    tf.keras.layers.Dense(1, activation='sigmoid'),
])

[20]: #Compilar modelos. Usar crossentropy binario ya que tenemos solo 2 opciones
#(perro o gato)
modeloDenso.compile(optimizer='adam',
                     loss='binary_crossentropy',
                     metrics=['accuracy'])

modeloCNN.compile(optimizer='adam',
                   loss='binary_crossentropy',
                   metrics=['accuracy'])

modeloCNN2.compile(optimizer='adam',
                    loss='binary_crossentropy',
                    metrics=['accuracy'])

[21]: from tensorflow.keras.callbacks import TensorBoard

```

4

```

[22]: #La variable de tensorboard se envia en el arreglo de "callbacks" (hay otros)
#tipos de callbacks soportados)
#En este caso guarda datos en la carpeta indicada en cada epoca, de manera que
#despues
#Tensorboard los lee para hacer graficas
tensorboardDenso = TensorBoard(log_dir='logs/denso')
modeloDenso.fit(X, y, batch_size=32,
                 validation_split=0.15,
                 epochs=30,
                 callbacks=[tensorboardDenso])

Epoch 1/30
618/618 [=====] - 9s 13ms/step - loss: 0.7133 -
accuracy: 0.5408 - val_loss: 0.6738 - val_accuracy: 0.5825
Epoch 2/30
618/618 [=====] - 8s 12ms/step - loss: 0.6755 -
accuracy: 0.5756 - val_loss: 0.6757 - val_accuracy: 0.5897
Epoch 3/30
618/618 [=====] - 8s 13ms/step - loss: 0.6667 -
accuracy: 0.5939 - val_loss: 0.6708 - val_accuracy: 0.5914
Epoch 4/30
618/618 [=====] - 8s 12ms/step - loss: 0.6702 -
accuracy: 0.5820 - val_loss: 0.6701 - val_accuracy: 0.5940
Epoch 5/30
618/618 [=====] - 8s 13ms/step - loss: 0.6630 -
accuracy: 0.5971 - val_loss: 0.6686 - val_accuracy: 0.5968
Epoch 6/30
618/618 [=====] - 8s 12ms/step - loss: 0.6637 -
accuracy: 0.6001 - val_loss: 0.6940 - val_accuracy: 0.5562
Epoch 7/30
618/618 [=====] - 8s 12ms/step - loss: 0.6598 -
accuracy: 0.6039 - val_loss: 0.6662 - val_accuracy: 0.5954
Epoch 8/30
618/618 [=====] - 8s 13ms/step - loss: 0.6609 -
accuracy: 0.6012 - val_loss: 0.6693 - val_accuracy: 0.5854
Epoch 9/30
618/618 [=====] - 8s 13ms/step - loss: 0.6577 -
accuracy: 0.6104 - val_loss: 0.6773 - val_accuracy: 0.5931
Epoch 10/30
618/618 [=====] - 8s 12ms/step - loss: 0.6570 -
accuracy: 0.6107 - val_loss: 0.6749 - val_accuracy: 0.5797
Epoch 11/30
618/618 [=====] - 8s 13ms/step - loss: 0.6517 -
accuracy: 0.6180 - val_loss: 0.6669 - val_accuracy: 0.5980
Epoch 12/30
618/618 [=====] - 8s 12ms/step - loss: 0.6560 -
accuracy: 0.6112 - val_loss: 0.6689 - val_accuracy: 0.5963

```

6

```

tf.keras.layers.Dense(1, activation='sigmoid')

])

modeloCNN = tf.keras.models.Sequential([
    tf.keras.layers.Conv2D(32, (3,3), activation='relu', input_shape=(100, 100, 1)),
    tf.keras.layers.MaxPooling2D(2, 2),
    tf.keras.layers.Conv2D(64, (3,3), activation='relu'),
    tf.keras.layers.MaxPooling2D(2, 2),
    tf.keras.layers.Conv2D(128, (3,3), activation='relu'),
    tf.keras.layers.MaxPooling2D(2, 2),

    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(100, activation='relu'),
    tf.keras.layers.Dense(1, activation='sigmoid')
])

modeloCNN2 = tf.keras.models.Sequential([
    tf.keras.layers.Conv2D(32, (3,3), activation='relu', input_shape=(100, 100, 1)),
    tf.keras.layers.MaxPooling2D(2, 2),
    tf.keras.layers.Conv2D(64, (3,3), activation='relu'),
    tf.keras.layers.MaxPooling2D(2, 2),
    tf.keras.layers.Conv2D(128, (3,3), activation='relu'),
    tf.keras.layers.MaxPooling2D(2, 2),

    tf.keras.layers.Dropout(0.5),
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(250, activation='relu'),
    tf.keras.layers.Dense(1, activation='sigmoid')
])

[20]: #Compilar modelos. Usar crossentropy binario ya que tenemos solo 2 opciones
#(perro o gato)
modeloDenso.compile(optimizer='adam',
                     loss='binary_crossentropy',
                     metrics=['accuracy'])

modeloCNN.compile(optimizer='adam',
                   loss='binary_crossentropy',
                   metrics=['accuracy'])

modeloCNN2.compile(optimizer='adam',
                    loss='binary_crossentropy',
                    metrics=['accuracy'])

[21]: from tensorflow.keras.callbacks import TensorBoard

```

5

```

Epoch 29/30
618/618 [=====] - 8s 12ms/step - loss: 0.6403 -
accuracy: 0.6349 - val_loss: 0.6716 - val_accuracy: 0.5954
Epoch 30/30
618/618 [=====] - 7s 12ms/step - loss: 0.6411 -
accuracy: 0.6316 - val_loss: 0.6755 - val_accuracy: 0.5946
[22]: <keras.callbacks.History at 0x2975aeaf7d90>
[23]: tensorboardCNN = TensorBoard(log_dir='logs/cnn')
modeloCNN.fit(X, y, batch_size=32,
               validation_split=0.15,
               epochs=30,
               callbacks=[tensorboardCNN])

Epoch 1/30
618/618 [=====] - 78s 125ms/step - loss: 0.6314 -
accuracy: 0.6308 - val_loss: 0.5563 - val_accuracy: 0.7140
Epoch 2/30
618/618 [=====] - 76s 122ms/step - loss: 0.4965 -
accuracy: 0.7598 - val_loss: 0.4585 - val_accuracy: 0.7802
Epoch 3/30
618/618 [=====] - 76s 123ms/step - loss: 0.4262 -
accuracy: 0.8010 - val_loss: 0.4391 - val_accuracy: 0.7883
Epoch 4/30
618/618 [=====] - 78s 126ms/step - loss: 0.3765 -
accuracy: 0.8275 - val_loss: 0.4107 - val_accuracy: 0.8188
Epoch 5/30
618/618 [=====] - 77s 124ms/step - loss: 0.3197 -
accuracy: 0.8602 - val_loss: 0.4030 - val_accuracy: 0.8203
Epoch 6/30
618/618 [=====] - 77s 125ms/step - loss: 0.2714 -
accuracy: 0.8845 - val_loss: 0.3995 - val_accuracy: 0.8312
Epoch 7/30
618/618 [=====] - 76s 123ms/step - loss: 0.2201 -
accuracy: 0.9068 - val_loss: 0.4259 - val_accuracy: 0.8226
Epoch 8/30
618/618 [=====] - 77s 125ms/step - loss: 0.1679 -
accuracy: 0.9317 - val_loss: 0.4432 - val_accuracy: 0.8364
Epoch 9/30
618/618 [=====] - 78s 126ms/step - loss: 0.1191 -
accuracy: 0.9532 - val_loss: 0.5787 - val_accuracy: 0.8206
Epoch 10/30
618/618 [=====] - 79s 127ms/step - loss: 0.0864 -
accuracy: 0.9676 - val_loss: 0.5446 - val_accuracy: 0.8281
Epoch 11/30
618/618 [=====] - 80s 130ms/step - loss: 0.0632 -
accuracy: 0.9769 - val_loss: 0.6150 - val_accuracy: 0.8304

```

8

```

Epoch 28/30
618/618 [=====] - 78s 126ms/step - loss: 0.0159 -
accuracy: 0.9948 - val_loss: 1.2669 - val_accuracy: 0.8183
Epoch 29/30
618/618 [=====] - 78s 127ms/step - loss: 0.0153 -
accuracy: 0.9952 - val_loss: 1.3214 - val_accuracy: 0.8066
Epoch 30/30
618/618 [=====] - 78s 126ms/step - loss: 0.0126 -
accuracy: 0.9957 - val_loss: 1.1977 - val_accuracy: 0.8209
[23]: <keras.callbacks.History at 0x2975b5055e0>
[24]: tensorboardCNN2 = TensorBoard(log_dir='logs/cnn2')
modeloCNN2.fit(X, y, batch_size=32,
                validation_split=0.15,
                epochs=30,
                callbacks=[tensorboardCNN2])

```

Epoch 1/30
618/618 [=====] - 85s 136ms/step - loss: 0.6528 -
accuracy: 0.5984 - val_loss: 0.6007 - val_accuracy: 0.6802
Epoch 2/30
618/618 [=====] - 85s 138ms/step - loss: 0.5314 -
accuracy: 0.7332 - val_loss: 0.4944 - val_accuracy: 0.7582
Epoch 3/30
618/618 [=====] - 85s 138ms/step - loss: 0.4653 -
accuracy: 0.7797 - val_loss: 0.4367 - val_accuracy: 0.7974
Epoch 4/30
618/618 [=====] - 86s 140ms/step - loss: 0.4284 -
accuracy: 0.7990 - val_loss: 0.4026 - val_accuracy: 0.8143
Epoch 5/30
618/618 [=====] - 86s 138ms/step - loss: 0.3892 -
accuracy: 0.8240 - val_loss: 0.3602 - val_accuracy: 0.8298
Epoch 6/30
618/618 [=====] - 86s 139ms/step - loss: 0.3545 -
accuracy: 0.8413 - val_loss: 0.3875 - val_accuracy: 0.8298
Epoch 7/30
618/618 [=====] - 86s 139ms/step - loss: 0.3219 -
accuracy: 0.8571 - val_loss: 0.3778 - val_accuracy: 0.8358
Epoch 8/30
618/618 [=====] - 86s 139ms/step - loss: 0.2916 -
accuracy: 0.8742 - val_loss: 0.3629 - val_accuracy: 0.8413
Epoch 9/30
618/618 [=====] - 86s 139ms/step - loss: 0.2591 -
accuracy: 0.8859 - val_loss: 0.3677 - val_accuracy: 0.8441
Epoch 10/30
618/618 [=====] - 87s 140ms/step - loss: 0.2322 -
accuracy: 0.9032 - val_loss: 0.3697 - val_accuracy: 0.8521

10

```

        zoom_range=[0.7, 1.4],
        horizontal_flip=True,
        vertical_flip=True
    )
)
datagen.fit(X)

plt.figure(figsize=(20,8))

for imagen, etiqueta in datagen.flow(X, y, batch_size=10, shuffle=False):
    for i in range(10):
        plt.subplot(2, 5, i+1)
        plt.xticks([])
        plt.yticks([])
        plt.imshow(imagen[i].reshape(100, 100), cmap="gray")
    break

```



```

[27]: modeloDenso_AD = tf.keras.models.Sequential([
    tf.keras.layers.Flatten(input_shape=(100, 100, 1)),
    tf.keras.layers.Dense(150, activation='relu'),
    tf.keras.layers.Dense(150, activation='relu'),
    tf.keras.layers.Dense(1, activation='sigmoid')
])

modeloCNN_AD = tf.keras.models.Sequential([
    tf.keras.layers.Conv2D(32, (3,3), activation='relu', input_shape=(100, 100, 1)),
    tf.keras.layers.MaxPooling2D(2, 2),
    tf.keras.layers.Conv2D(64, (3,3), activation='relu'),
    tf.keras.layers.MaxPooling2D(2, 2),
    tf.keras.layers.Conv2D(128, (3,3), activation='relu'),

```

13

```

Epoch 27/30
618/618 [=====] - 84s 137ms/step - loss: 0.0481 -
accuracy: 0.9829 - val_loss: 0.5886 - val_accuracy: 0.8473
Epoch 28/30
618/618 [=====] - 84s 137ms/step - loss: 0.0462 -
accuracy: 0.9837 - val_loss: 0.6158 - val_accuracy: 0.8578
Epoch 29/30
618/618 [=====] - 84s 136ms/step - loss: 0.0473 -
accuracy: 0.9843 - val_loss: 0.5764 - val_accuracy: 0.8430
Epoch 30/30
618/618 [=====] - 84s 136ms/step - loss: 0.0438 -
accuracy: 0.9851 - val_loss: 0.7596 - val_accuracy: 0.8327
[24]: <keras.callbacks.History at 0x2975b446f10>
[25]: #ver las imagenes de la variable X sin modificaciones por aumento de datos
plt.figure(figsize=(20, 8))
for i in range(10):
    plt.subplot(2, 5, i+1)
    plt.xticks([])
    plt.yticks([])
    plt.imshow(X[i].reshape(100, 100), cmap="gray")

```



```

[26]: #Realizar el aumento de datos con varias transformaciones. Al final, grafican
       #10 como ejemplo
from tensorflow.keras.preprocessing.image import ImageDataGenerator

datagen = ImageDataGenerator(
    rotation_range=30,
    width_shift_range=0.2,
    height_shift_range=0.2,
    shear_range=15,

```

```

    tf.keras.layers.MaxPooling2D(2, 2),
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(100, activation='relu'),
    tf.keras.layers.Dense(1, activation='sigmoid')
))

modeloCNN2_AD = tf.keras.models.Sequential([
    tf.keras.layers.Conv2D(32, (3,3), activation='relu', input_shape=(100, 100, 1)),
    tf.keras.layers.MaxPooling2D(2, 2),
    tf.keras.layers.Conv2D(64, (3,3), activation='relu'),
    tf.keras.layers.MaxPooling2D(2, 2),
    tf.keras.layers.Conv2D(128, (3,3), activation='relu'),
    tf.keras.layers.MaxPooling2D(2, 2),

    tf.keras.layers.Dropout(0.5),
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(250, activation='relu'),
    tf.keras.layers.Dense(1, activation='sigmoid')
])

modeloDenso_AD.compile(optimizer='adam',
                       loss='binary_crossentropy',
                       metrics=['accuracy'])

modeloCNN_AD.compile(optimizer='adam',
                      loss='binary_crossentropy',
                      metrics=['accuracy'])

modeloCNN2_AD.compile(optimizer='adam',
                      loss='binary_crossentropy',
                      metrics=['accuracy'])

```

```

[29]: #Separar los datos de entrenamiento y los datos de pruebas en variables
       #diferentes
len(X) * .85 #19700
len(X) - 19700 #5562

X_entrenamiento = X[:19700]
X_validacion = X[19700:]

y_entrenamiento = y[:19700]
y_validacion = y[19700:]

```

14

```
[30]: #Usar la función flow del generador para crear un iterador que podamos enviarlo
...como entrenamiento a la función FIT del modelo
data_gen_entrenamiento = datagen.flow(X_entrenamiento, y_entrenamiento,...)
...batch_size=32)
```

```
[31]: tensorboardDenso_AD = TensorBoard(log_dir='logs/denso_AD')
```

```
modeloDenso_AD.fit(
    data_gen_entrenamiento,
    epochs=25, batch_size=32,
    validation_data=(X_validacion, y_validacion),
    steps_per_epoch=int(np.ceil(len(X_entrenamiento) / float(32))), 
    validation_steps=int(np.ceil(len(X_validacion) / float(32))),
    callbacks=[tensorboardDenso_AD]
)
```

```
Epoch 1/25
616/616 [=====] - 18s 29ms/step - loss: 0.7347 - accuracy: 0.5080 - val_loss: 0.6912 - val_accuracy: 0.5022
Epoch 2/25
616/616 [=====] - 18s 29ms/step - loss: 0.6933 - accuracy: 0.5025 - val_loss: 0.6924 - val_accuracy: 0.5157
Epoch 3/25
616/616 [=====] - 18s 29ms/step - loss: 0.6935 - accuracy: 0.5053 - val_loss: 0.6926 - val_accuracy: 0.5073
Epoch 4/25
616/616 [=====] - 18s 29ms/step - loss: 0.6926 - accuracy: 0.5027 - val_loss: 0.6931 - val_accuracy: 0.4986
Epoch 5/25
616/616 [=====] - 18s 29ms/step - loss: 0.6928 - accuracy: 0.4985 - val_loss: 0.6932 - val_accuracy: 0.4989
Epoch 6/25
616/616 [=====] - 18s 29ms/step - loss: 0.6925 - accuracy: 0.4992 - val_loss: 0.6931 - val_accuracy: 0.4997
Epoch 7/25
616/616 [=====] - 18s 29ms/step - loss: 0.6922 - accuracy: 0.5056 - val_loss: 0.6925 - val_accuracy: 0.5020
Epoch 8/25
616/616 [=====] - 18s 28ms/step - loss: 0.6924 - accuracy: 0.5090 - val_loss: 0.6931 - val_accuracy: 0.4994
Epoch 9/25
616/616 [=====] - 18s 28ms/step - loss: 0.6930 - accuracy: 0.5019 - val_loss: 0.6920 - val_accuracy: 0.5059
Epoch 10/25
616/616 [=====] - 18s 29ms/step - loss: 0.6927 - accuracy: 0.5042 - val_loss: 0.6916 - val_accuracy: 0.5059
Epoch 11/25
```

```
Epoch 29/40
616/616 [=====] - 73s 118ms/step - loss: 0.4420 - accuracy: 0.7948 - val_loss: 0.4361 - val_accuracy: 0.7987
Epoch 30/40
616/616 [=====] - 73s 118ms/step - loss: 0.4430 - accuracy: 0.7935 - val_loss: 0.3731 - val_accuracy: 0.8321
Epoch 31/40
616/616 [=====] - 73s 119ms/step - loss: 0.4441 - accuracy: 0.7874 - val_loss: 0.4080 - val_accuracy: 0.8105
Epoch 32/40
616/616 [=====] - 73s 118ms/step - loss: 0.4331 - accuracy: 0.7981 - val_loss: 0.4907 - val_accuracy: 0.7777
Epoch 33/40
616/616 [=====] - 73s 118ms/step - loss: 0.4365 - accuracy: 0.7958 - val_loss: 0.4181 - val_accuracy: 0.8086
Epoch 34/40
616/616 [=====] - 73s 118ms/step - loss: 0.4304 - accuracy: 0.7995 - val_loss: 0.3641 - val_accuracy: 0.8352
Epoch 35/40
616/616 [=====] - 73s 118ms/step - loss: 0.4274 - accuracy: 0.8012 - val_loss: 0.3998 - val_accuracy: 0.8150
Epoch 36/40
616/616 [=====] - 72s 117ms/step - loss: 0.4175 - accuracy: 0.8089 - val_loss: 0.3819 - val_accuracy: 0.8243
Epoch 37/40
616/616 [=====] - 72s 117ms/step - loss: 0.4227 - accuracy: 0.8035 - val_loss: 0.5207 - val_accuracy: 0.7661
Epoch 38/40
616/616 [=====] - 72s 118ms/step - loss: 0.4189 - accuracy: 0.8084 - val_loss: 0.3683 - val_accuracy: 0.8316
Epoch 39/40
616/616 [=====] - 73s 119ms/step - loss: 0.4166 - accuracy: 0.8070 - val_loss: 0.3445 - val_accuracy: 0.8467
Epoch 40/40
616/616 [=====] - 73s 119ms/step - loss: 0.4104 - accuracy: 0.8085 - val_loss: 0.3727 - val_accuracy: 0.8394
```

```
[32]: <keras.callbacks.History at 0x29760d2b6e0>
```

```
[33]: tensorboardCNN2_AD = TensorBoard(log_dir='logs/cnn2_AD')

modeloCNN2_AD.fit(
    data_gen_entrenamiento,
    epochs=40, batch_size=32,
    validation_data=(X_validacion, y_validacion),
    steps_per_epoch=int(np.ceil(len(X_entrenamiento) / float(32))), 
    validation_steps=int(np.ceil(len(X_validacion) / float(32))),
```

```
[32]: tensorboardCNN_AD = TensorBoard(log_dir='logs-new/cnn_AD')

modeloCNN_AD.fit(
    data_gen_entrenamiento,
    epochs=40, batch_size=32,
    validation_data=(X_validacion, y_validacion),
    steps_per_epoch=int(np.ceil(len(X_entrenamiento) / float(32))), 
    validation_steps=int(np.ceil(len(X_validacion) / float(32))),
```

```
) callbacks=[tensorboardCNN_AD]

)
Epoch 1/40
616/616 [=====] - 79s 127ms/step - loss: 0.6014 - accuracy: 0.5603 - val_loss: 0.6784 - val_accuracy: 0.5758
Epoch 2/40
616/616 [=====] - 78s 126ms/step - loss: 0.6583 - accuracy: 0.6110 - val_loss: 0.6997 - val_accuracy: 0.5924
Epoch 3/40
616/616 [=====] - 78s 126ms/step - loss: 0.6465 - accuracy: 0.6260 - val_loss: 0.6184 - val_accuracy: 0.6623
Epoch 4/40
616/616 [=====] - 78s 127ms/step - loss: 0.6315 - accuracy: 0.6419 - val_loss: 0.6370 - val_accuracy: 0.6364
Epoch 5/40
616/616 [=====] - 78s 127ms/step - loss: 0.6244 - accuracy: 0.6534 - val_loss: 0.5794 - val_accuracy: 0.6909
Epoch 6/40
616/616 [=====] - 76s 124ms/step - loss: 0.6073 - accuracy: 0.6655 - val_loss: 0.5889 - val_accuracy: 0.6746
Epoch 7/40
616/616 [=====] - 73s 118ms/step - loss: 0.5869 - accuracy: 0.6863 - val_loss: 0.5307 - val_accuracy: 0.7353
Epoch 8/40
616/616 [=====] - 74s 120ms/step - loss: 0.5792 - accuracy: 0.6993 - val_loss: 0.6081 - val_accuracy: 0.6496
Epoch 9/40
616/616 [=====] - 73s 118ms/step - loss: 0.5690 - accuracy: 0.7035 - val_loss: 0.5169 - val_accuracy: 0.7428
Epoch 10/40
616/616 [=====] - 72s 117ms/step - loss: 0.5589 - accuracy: 0.7135 - val_loss: 0.5013 - val_accuracy: 0.7580
Epoch 11/40
616/616 [=====] - 72s 118ms/step - loss: 0.5474 - accuracy: 0.7231 - val_loss: 0.5357 - val_accuracy: 0.7291
Epoch 12/40
616/616 [=====] - 72s 118ms/step - loss: 0.5398 - accuracy: 0.7263 - val_loss: 0.4919 - val_accuracy: 0.7597
```

```
callbacks=[tensorboardCNN2_AD]
```

```
)
Epoch 1/40
616/616 [=====] - 82s 132ms/step - loss: 0.6854 - accuracy: 0.5512 - val_loss: 0.6727 - val_accuracy: 0.6157
Epoch 2/40
616/616 [=====] - 81s 131ms/step - loss: 0.6711 - accuracy: 0.5904 - val_loss: 0.6419 - val_accuracy: 0.6255
Epoch 3/40
616/616 [=====] - 81s 132ms/step - loss: 0.6574 - accuracy: 0.6110 - val_loss: 0.6531 - val_accuracy: 0.6154
Epoch 4/40
616/616 [=====] - 81s 132ms/step - loss: 0.6398 - accuracy: 0.6312 - val_loss: 0.6183 - val_accuracy: 0.6037
Epoch 5/40
616/616 [=====] - 81s 132ms/step - loss: 0.6301 - accuracy: 0.6465 - val_loss: 0.5815 - val_accuracy: 0.7024
Epoch 6/40
616/616 [=====] - 81s 132ms/step - loss: 0.6193 - accuracy: 0.6588 - val_loss: 0.6123 - val_accuracy: 0.6463
Epoch 7/40
616/616 [=====] - 81s 132ms/step - loss: 0.6052 - accuracy: 0.6732 - val_loss: 0.5612 - val_accuracy: 0.7019
Epoch 8/40
616/616 [=====] - 81s 131ms/step - loss: 0.6031 - accuracy: 0.6717 - val_loss: 0.5563 - val_accuracy: 0.7193
Epoch 9/40
616/616 [=====] - 81s 131ms/step - loss: 0.5916 - accuracy: 0.6840 - val_loss: 0.5854 - val_accuracy: 0.6856
Epoch 10/40
616/616 [=====] - 81s 132ms/step - loss: 0.5857 - accuracy: 0.6885 - val_loss: 0.5471 - val_accuracy: 0.7285
Epoch 11/40
616/616 [=====] - 81s 132ms/step - loss: 0.5776 - accuracy: 0.6960 - val_loss: 0.5504 - val_accuracy: 0.7142
Epoch 12/40
616/616 [=====] - 81s 132ms/step - loss: 0.5726 - accuracy: 0.7028 - val_loss: 0.5475 - val_accuracy: 0.7190
Epoch 13/40
616/616 [=====] - 81s 131ms/step - loss: 0.5683 - accuracy: 0.7050 - val_loss: 0.5238 - val_accuracy: 0.7403
Epoch 14/40
616/616 [=====] - 81s 132ms/step - loss: 0.5660 - accuracy: 0.7098 - val_loss: 0.5136 - val_accuracy: 0.7546
Epoch 15/40
616/616 [=====] - 81s 132ms/step - loss: 0.5558 -
```

Figure 2. Código del entrenamiento

3.3. Predicción

Para la predicción de las imágenes, previamente al entrenamiento se guardó el mismo en un archivo .h5 para qué pueda ser leído en esta sección, posteriormente se prepara la imagen a ser reconocida y se incorpora como argumento a la función predict de TF. Por último, se puede visualizar la salida de la última neurona, representado si es un perro o un gato.

```

<Python.core.display.HTML object>
[129]: <ipython>
import tensorflow as tf
import tensorflow_datasets as tfds
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from sklearn.metrics import confusion_matrix, f1_score, roc_curve,precision_score, recall_score, accuracy_score, roc_auc_score
from sklearn.metrics import classification_report
from tensorflow.keras import Model
from tensorflow.keras import layers
from tensorflow.keras import Model
from tensorflow.keras import Sequential
from tensorflow.keras import Input
from tensorflow.keras import Model
ModelP = load_model('perros-gatos-cnn_V1.h5')
ModelP2 = load_model('perros-gatos-dense-ad_V2.h5')

[130]: <ipython>
import numpy as np
#from google.colab import files
from keras.preprocessing import image
import keras
import tensorflow as tf

#uploaded = files.upload()
import matplotlib.pyplot as plt
import cv2
ModelP.load_weights('perros-gatos-cnn_V1.h5')
ModelP2.load_weights('perros-gatos-dense-ad_V2.h5')

[131]: <ipython>
import numpy as np
#from google.colab import files
from keras.preprocessing import image
import keras
import tensorflow as tf

#uploaded = files.upload()
import matplotlib.pyplot as plt
import cv2

plt.figure(figsize=(10,10))
Imagen = cv2.imread('ImgPer/Per8.jpg')
plt.subplot(1, 2, 1)
plt.xticks([])
plt.yticks([])
plt.imshow(Imagen, cmap='gray')
plt.title('Original')

TAMANO_IMG=100

Imagen = cv2.resize(Imagen, (TAMANO_IMG, TAMANO_IMG))
Imagen = cv2.cvtColor(Imagen, cv2.COLOR_BGR2GRAY)
plt.subplot(1, 2, 2)
plt.xticks([])
plt.yticks([])
plt.imshow(Imagen, cmap='gray')
plt.title('Para el modelo')

23

# predicting images
img=Imagen
#img = keras.utils.load_img(path, target_size=(100, 100))
x = keras.utils.img_to_array(img)
x = np.expand_dims(x, axis=0)

images = np.reshape(x)
classes = ModelP.predict(images, batch_size=10)
print("retorno de la prediccion = ",classes[0])
if classes[0]>0.5:
    print(" ** ES PERRO **")
else:
    print(" ** ES GATO **")

1/1 [=====] - 0s 20ms/step
retorno de la prediccion = [1.]
** ES PERRO **




```

Figure 3. Código para la Predicción

4. Conclusiones Generales

Luego de la elaboración del código, implementación de la base de datos, entrenamiento del modelo y de la predicción de imágenes diferentes a las de la base de datos, se llegaron a las siguientes conclusiones. En el proyecto se realizaron tres modelos: uno con capas densas, el segundo con CNN (convolutional neural network), y el tercero con CNN con DropOut=0.5 para los datos de inicio.

- Para el primer modelo con 30 épocas se obtuvo un accuracy de 0.6316.
- Para el segundo modelo con 30 épocas se obtuvo un accuracy de 0.9957.
- Para el tercer modelo con 30 épocas se obtuvo un accuracy de 0.9851.

Teniendo el mejor valor de accuracy para el modelo 2 “CNN”, que es una CNN sin dropout.

Se volvió a realizar el entrenamiento con nuevos datos, los cuales fueron generados de las imágenes del principio, pero con modificaciones visuales (rotación, desplazamiento, zoom).

- Para el primer modelo ”Denso-ad” con 25 épocas se obtuvo un accuracy de 0.5184.
- Para el segundo modelo ”CNN-ad” con 40 épocas se obtuvo un accuracy de 0.8085.
- Para el tercer modelo ”CNN2-ad” con 40 épocas se obtuvo un accuracy de 0.7684.

El mejor modelo obtenido tomando en cuenta el accuracy fue el **CNN-ad**, el cual tiene una precisión del 0.8085.

Se ejecutó la prueba con los dos modelos con más accuracy y mediante una prueba con un conjunto nuevo de datos, se obtuvo una precisión escasa, llegando a la conclusión que existe overfitting debido a una saturación en las épocas.

Se pudo determinar que existe sobreentrenamiento (overfitting) para el segundo modelo sin adicionamiento de datos ”CNN” figura 4, ya que desde la época 15 empieza a llegar a un ajuste constante. Por otra parte , se realizaron pruebas con un conjunto de imágenes nuevas, teniendo un resultado erróneo, debido a un modelo memorón.

Para este modelo, se pudo identificar que existe una buena respuesta entre el número de épocas y el accuracy. Por otra parte, las perdidas son graduales. En conclusión, se puede decir que el modelo es bueno tanto para el mismo conjunto de datos de entrenamiento como para datos nuevos, ya que, según pruebas con nuevas imágenes, se tuvo un resultado satisfactorio.

Se determinó utilizar el modelo CNN sin adicionamiento de datos con menos épocas para eliminar el sobreentrenamiento, obteniendo las siguientes características: Para el segundo modelo ”CNN” con 20 épocas se obtuvo un accuracy de 0.9887.

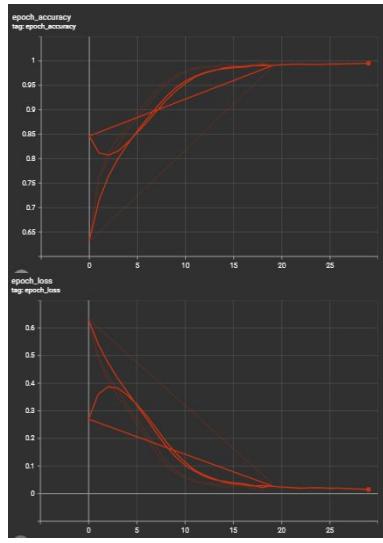


Figure 4. Accuracy del modelo CNN

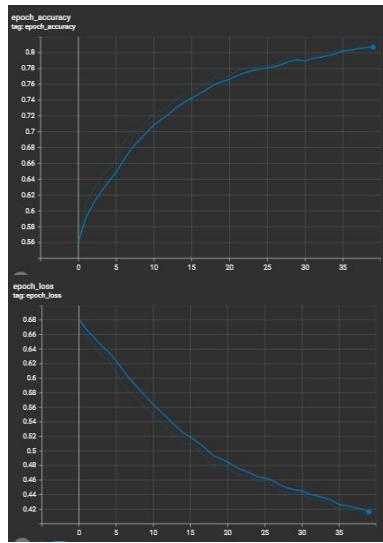


Figure 5. Accuracy del modelo CNN-ad

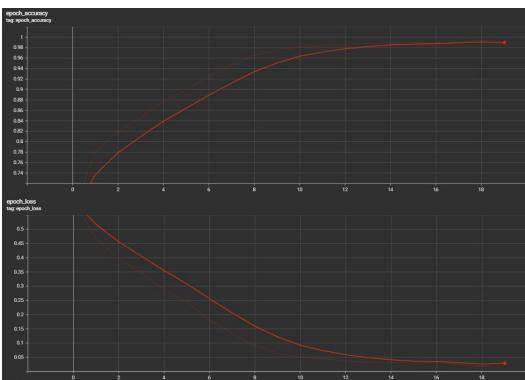


Figure 6. Accuracy del modelo CNN con 20 épocas

Se llamó al entrenamiento, previo: ModelF=load_model('perros-gatos-cnn_V1.h5').

A continuación se probó un set de datos externos para determinar la predicción de datos que no son del dataset.



Figure 7. Predicción con imágenes externas al dataset mediante el segundo modelo CNN

Luego del entrenamiento de los diferentes modelos de predicción, hallando sus accuracy respectivos y ejecutando las pruebas con imágenes externas al Dataset; Se llegó a la conclusión de que el modelo más apropiado para la predicción de perros y gatos de imágenes externas al Dataset original es el modelo **CNN con 20 épocas** perros-gatos-cnn_V1.h5, siendo un modelo óptimo tanto en el tiempo de entrenamiento como la predicción. Este modelo presentó un accuracy de 0.9887 figura 6 y no mostró indicios de tener Overfitting.

Vale la pena mencionar que el modelo CNN_ad (CNN con datos añadidos) presentó también un accuracy bueno (0.8085) sin indicios de Overfitting pero una confusión entre perros y gatos del mismo dataset o imágenes externas. Este modelo puede ser mejorado mediante la adición de más épocas; sin embargo, este proceso implica mayor tiempo de procesamiento y entrenamiento (5H Aprox. para 60 épocas).

References

- [1] I. Tiñini Alvares [@IsRaTiAlv], Workshop CVwithML, 10 de junio de 2020. Accedido el 2 de diciembre de 2022. [En línea]. Disponible: https://github.com/IsRaTiAlv/Workshop_CVwithML
- [2] J. Elson, J. Douceur, J. Howell y S. Jared. "Dataset of TensorFlow - cats vs dogs". TensorFlow. https://www.tensorflow.org/datasets/catalog/cats_vs_dogs(accedido el 12 de diciembre de 2022).
- [3] F. Moutarde. "Deep-Learning: Introduction to Convolutional Neural Networks". Minesparis. https://people.minesparis.psl.eu/fabien.moutarde/ES_MachineLearning/Practical_deepLearning-convNets/convnet-notebook.html (accedido el 12 de diciembre de 2022)
- [4] L. Yuan. "Convolutional neural Network-XGBoost for accuracy enhancement of breast cancer detection". Journal of Physics: Conference Series. <https://iopscience.iop.org/article/10.1088/1742-6596/1918/4/042016/pdf>(accedido el 12 de diciembre de 2022).
- [5] A. Prangishvili, O. Namicheishvili, and M. Ramazashvili, "Convolutional Neural Networks," Works of Georgian Technical University, no. 3(517), pp. 33–56, Sep. 2020. Accessed: Dec. 12, 2022. [Online]. Available: <https://doi.org/10.36073/1512-0996-2020-3-33-56>
- [6] M. Mirkhan and M. R. Meybodi, "Restricted Convolutional Neural Networks," Neural Processing Letters, vol. 50, no. 2, pp. 1705–1733, Nov. 2018. Accessed: Dec. 12, 2022. [Online]. Available: <https://doi.org/10.1007/s11063-018-9954-x>
- [7] Ning Jin and Derong Liu, "Wavelet basis function neural networks for sequential learning," IEEE Transactions on Neural Networks, vol. 19, no. 3, pp. 523–528, Mar. 2008. Accessed: May 21, 2022. [Online]. Available: <https://doi.org/10.1109/tnn.2007.911749>
- [8] V. P. Balam, V. U. Sameer, and S. Chinara, "Automated classification system for drowsiness detection using convolutional neural network and electroencephalogram," IET Intelligent Transport Systems, vol. 15, no. 4, pp. 514–524, Feb. 2021. Accessed: May 21, 2022. [Online]. Available: <https://doi.org/10.1049/itr2.12041>