UNIVERSIDAD DE BURGOS
ESCUELA POLITÉCNICA SUPERIOR
Grado en Ingeniería en Informática

TFG del Grado en Ingeniería Informática

Detección de bajas de aves

Presentado por Saúl Vetia de Juana
en Universidad de Burgos — 28 de enero de 2017
Tutor: Carlos Cambra Baseca
Roberto Carlos Casado Vara

UNIVERSIDAD DE BURGOS
ESCUELA POLITÉCNICA SUPERIOR
Grado en Ingeniería en Informática

D. Carlos Cambra Báseca, profesor del departamento de Ingeniería informática, área de ciencias de la computación e inteligencia artificial, y D. Roberto Casado Vara, profesor del departamento de Matemáticas y computación, área de Matemática aplicada.

Exponen:

Que el alumno D. Saúl Vetia de Juana, con DNI 71299742H, ha realizado el Trabajofinal de Grado en Ingeniería Informática titulado "Proyecto de detección de bajas de ves".

Y que dicho trabajo ha sido realizado por el alumno bajo la dirección del que suscribe, en virtud de lo cual se autoriza su presentación y defensa.

En Burgos, 28 de enero de 2017

Vº. Bº. del Tutor:

CAMBRA BASECA CARLOS - 18053084Q

Firmado digitalmente por CAMBRA BASECA CARLOS - 18053084Q
Fecha: 2022.06.10 15:57:36 +02'00'

D. Carlos Cambra Báseca

Vº. Bº. del co-tutor:

CASADO VARA ROBERTO CARLOS - 45686237A

Firmado digitalmente por CASADO VARA ROBERTO CARLOS - 45686237A
Fecha: 2022.06.10 14:03:36 +02'00'

D. Roberto Casado Vara

Resumen

El objetivo de este trabajo es desarrollar un programa que, dadas unas imágenes de gallinas clasificadas (vivas o muertas) se entrene una red neuronal con el objetivo de detectar bajas de aves en jaulas.

Para ello se investigará qué son las redes neuronales convolucionales, su funcionamiento y las técnicas de aprendizaje y reducción de sobreajuste para poder implementarlas en el programa.

Descriptores

Redes neuronales, redes neuronales convolucionales, CNN, Python, Tensorflow, Keras.

Índice general

Índice de figuras

Introducción

El objetivo de este trabajo es entrenar una red neuronal capaz de clasificar imágenes de aves según si están vivas o muertas.

Para ello se ha tomado como punto de partida un notebook público de objetivo similar llamado “Cat vs. Dog Image Classification”, el cual ha servido como base para proporcionar la estructura de la red neuronal. A partir de esta base se han investigado distintas técnicas que permiten mejorar la eficacia de la red, y se han realizado diversas pruebas para probarlas individualmente.

Finalmente, a partir de las pruebas que han ofrecido mejores resultados, se ha elaborado una versión final de la red.

Para poder explicar correctamente el proceso de decisión que ha llevado a la elaboración de la versión final del programa, se dividirá el trabajo en ocho apartados, tantos como pruebas se han realizado, en los cuales se explicarán las técnicas utilizadas y se observarán los resultados.

Objetivos del proyecto

El principal objetivo es optimizar una red neuronal convolucional capaz de clasificar imágenes pertenecientes a dos grupos distintos.

Para ello se pretende diseñar un algoritmo que, mediante diversas técnicas de redes neuronales, maximice la eficacia de la red y facilite la clasificación a partir de un conjunto de imágenes de entrenamiento reducido.

Para cumplir este objetivo analizaré los resultados obtenidos con cada versión para decidir qué técnicas puedo implementar en la versión final del código y cuales puedo descartar.

Conceptos teóricos

Redes neuronales: Introducción

Una red neuronal consiste en un conjunto de unidades, llamadas neuronas artificiales, conectadas entre sí para transmitirse señales y procesarlas, produciendo unos valores de salida determinados de acuerdo al objetivo que se proponga alcanzar.

Las unidades de procesamiento se organizan en capas. El modelo de una red neuronal se puede dividir en tres: una capa de entrada, con neuronas que reciben los datos de entrada, una o varias capas intermedias (también llamadas ocultas), y una capa de salida, que transmite la información ya procesada. Los datos de entrada se presentan en la primera capa y los valores se propagan desde cada neurona hasta la capa siguiente hasta que finalmente, se envía un resultado desde la capa de salida.

La red aprende examinando los registros individuales, generando una predicción para cada registro y realizando ajustes a las ponderaciones cuando realiza una predicción incorrecta. Este proceso se repite muchas veces y la red sigue mejorando sus predicciones hasta haber alcanzado uno o varios criterios de parada.

El proceso por el cual las redes se preparan para el procesado de información se denomina entrenamiento. Consiste en presentar a la red ejemplos para los que se conoce el resultado, y los resultados calculados se comparan con los resultados conocidos. La información procedente de esta comparación se pasa hacia atrás (backpropagation) para refinar los resultados. A medida que progresa el entrenamiento, la red se va haciendo cada vez más precisa en la replicación de resultados conocidos, de modo que pueda clasificar conjuntos de datos ajenos a los conjuntos de datos de entrenamiento (aprendizaje inductivo).

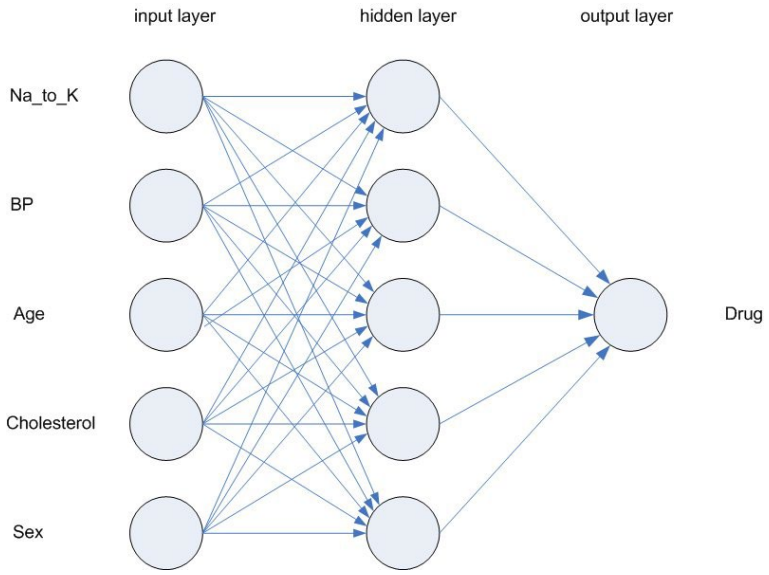

Ilustración 1: Esquema de una red neuronal

Conceptos relativos a las CNN

Las redes neuronales convolucionales (Convolutional Neural Networks) son un tipo de red neuronal especialmente utilizada en el campo de la visión artificial que consiste en una variación del perceptrón multicapa.

Las redes neuronales convolucionales constan de múltiples capas de convolución, tras las cuales se añade una función para realizar un mapeo causal no-lineal.

Su funcionamiento se puede dividir en dos fases: primero una fase de extracción de características durante la cual se utilizan neuronas convolucionales y neuronas de reducción de muestreo, y una fase de clasificación final sobre las características extraídas, donde se utilizan neuronas de tipo perceptrón simple.

Algunos conceptos necesarios para la comprensión del proceso de convolución:

Neuronas de reducción de muestreo

Son procesadores en matriz que realizan una operación sobre los datos de imagen 2D que pasan por ellas, en lugar de un único valor numérico.

El operador de convolución tiene el efecto de filtrar la imagen de entrada con un núcleo previamente entrenado. Así los datos se transforman de manera tal que, según la forma en la que el núcleo se haya entrenado, algunas características de la imagen reciben una asignación de un valor numérico mayor.

Los núcleos pueden procesar las imágenes de forma específica para realizar una tarea determinada con las imágenes, como por ejemplo detectar bordes.

Los núcleos utilizados por las redes neuronales convolucionales pueden extraer otras características más complejas. Estos núcleos se pueden importar igual que cualquier otra librería, como por ejemplo, el núcleo Tensorflow, el cual utilizaré durante el desarrollo de este trabajo.

Max-pooling

Es un proceso por el cual se hace un resumen de características sobre una región. Se busca el valor máximo dentro de esa área y se pasa dicho valor como resumen de las características de esa área.

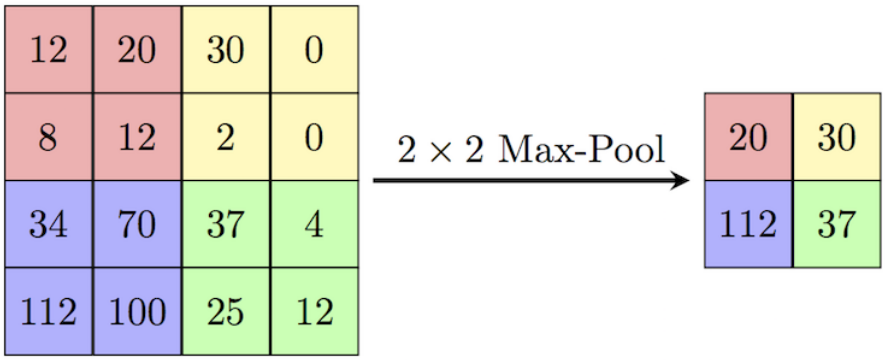

Ilustración 2: Max-pooling

Clasificación

Tras pasar por una o varias fases de extracción de características, los datos están listos para ser clasificados. Según las características extraídas, las imágenes se clasifican de acuerdo a una etiqueta u otra.

Proceso de aprendizaje de las CNN

Se realiza a lo largo de varias etapas

Pre-procesamiento

Durante este paso se suelen normalizar los valores de los pixeles de la imagen, dado que estas redes suelen operar con valores comprendidos entre 0 y 1. Para ello se divide el valor asociado a cada pixel entre 255.

Proceso de convolución

Consiste en tomar grupos de pixeles cercanos de la imagen de entrada y operarlos matemáticamente a través de producto escalar con el núcleo. Estos núcleos son matrices de tamaño “n x n” que representa al conjunto de neuronas de entrada, y permiten extraer determinadas características de las imágenes de entrada.

Durante un paso de convolución puede haber muchos núcleos. Al conjunto de estos se le denomina “filtro”. Cada vez que se procesa un conjunto de imágenes mediante un filtro, se genera una matriz tridimensional de salida denominada “capa de neuronas ocultas”, que se volverán a procesar.

Función de activación

Es la función objetivo con la que se trata de ajustar el conjunto de datos. La más utilizada es ReLU (Rectified Linear Unit).

Esta función se define como “la parte positiva de sus argumentos” y toma la siguiente forma:

$$f(x) = x^+ = \max(0, x)$$

Ilustración 3: Función de activación

Muestreo

Paso en el que se toma una muestra de las neuronas más representativas para continuar el proceso de convolución.

Si no se muestrease y se realizara la convolución a partir de la capa obtenida, se consumirían muchos recursos computacionales, de modo que al reducir el número de muestras es posible conservar las mejores neuronas (con las características detectadas más importantes) y facilitar el proceso de convolución para la siguiente capa.

La técnica de muestreo más utilizada es la técnica previamente comentada de “max-pooling”.

Durante este paso se recorre cada una de las imágenes obtenidas, de izquierda a derecha y de arriba a abajo, pero en lugar de leer un solo pixel cada vez, se lee una cantidad “n x n”, y se guarda el valor más alto de entre los pixeles leídos. De esta forma, el tamaño de las imágenes resultantes se reduce exponencialmente, y se facilita el siguiente paso.

Conexión con capa tradicional

Para finalizar el proceso, se toma la última capa generada sobre la cual se aplicara muestreo y se transforma en una capa de neuronas tradicional. Esto significa que de tener una capa tridimensional pasamos a tener una capa de neuronas bidimensionales.

A esta nueva capa tradicional se le aplica una función llamada Softmax, que convierte en probabilidad el valor de las neuronas de salida.

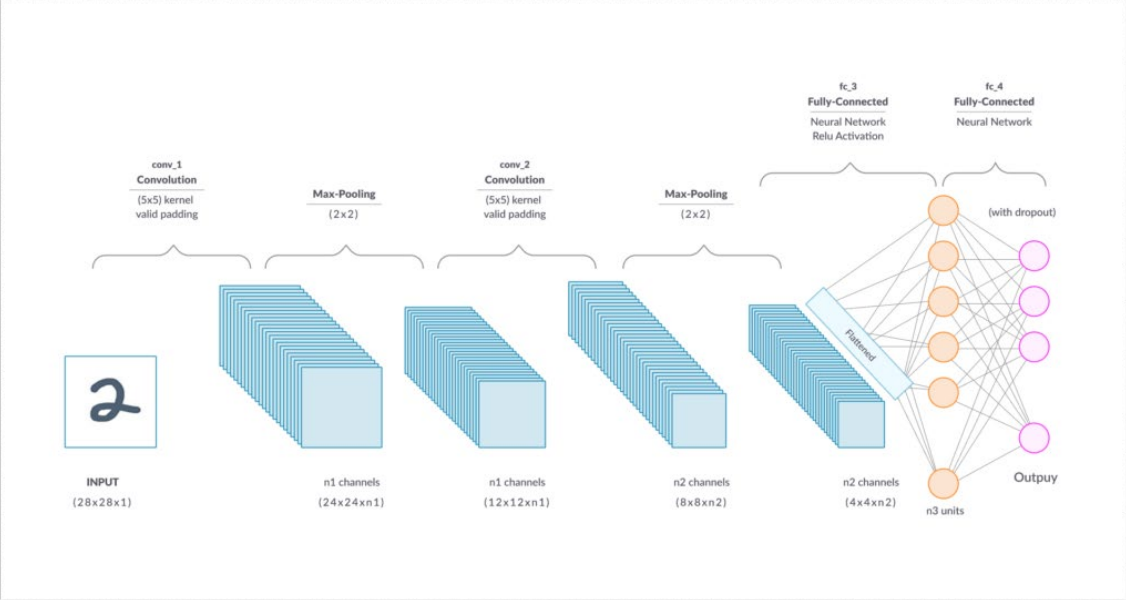

Ilustración 4: Esquema de una CNN

Técnicas y herramientas

Herramientas

Jupyter Notebook: Entorno de desarrollo web que permite la ejecución de código Python en el navegador.

Anaconda navigator: Interfaz de usuario que permite la ejecución de aplicaciones de desarrollo de Python, así como gestionar paquetes y versiones de Conda.

Keras: Biblioteca de redes neuronales de código abierto capaz de ejecutarse sobre Tensorflow y que permite implementar funciones para redes Deep Learning. Proporciona un conjunto de herramientas especializadas en redes neuronales convolucionales.

Tensorflow: Biblioteca de código abierto que implementa funciones de aprendizaje automático para construir y entrenar redes neuronales.

Técnicas

Manipulación de los bloques de convolución: El objetivo de las pruebas realizadas en este bloque es determinar hasta qué punto la modificación de los elementos que constituyen los bloques de convolución afecta al resultado.

Definimos un bloque de convolución como la unidad básica de procesamiento en las redes neuronales convolucionales. Constan de una capa de convolución y una capa de max-pooling.

Las capas de convolución sirven para realizar el proceso de convolución, que como se ha explicado anteriormente, es un proceso consistente en el producto escalar entre los valores de entrada y un conjunto bidimensional de pesos. Este conjunto de pesos es lo que se conoce como filtro.

Partiendo de esta premisa, he realizado tres pruebas consistentes en:

- Aumento del número de bloques de convolución
- Aumento del número de capas de convolución
- Incremento del tamaño del filtro

Manipulación de las capas fully connected: También conocidas como “Capas ocultas”, se trata de la penúltima capa de una red neuronal convolucional, ubicada antes de la capa de salida. Cada neurona de la capa anterior está conectada con todas las de la ésta capa.

En esta capa se toma el resultado obtenido en la capa de aplanado (Flatten) y se procesa mediante dos funciones: Una función afín (Affine function) y una función no lineal.

-Función afín: Se define como la combinación de una transformación lineal y una traslación.

$$\mathbf{x} \mapsto \mathbf{A}\mathbf{x} + \mathbf{b}$$

Ilustración 5: Affine function

-Función no lineal: Se define una función no lineal como aquella que no se representa mediante una línea recta. En el contexto de redes neuronales, hace referencia a la función de activación, es decir, la función objetivo con la que se trata de ajustar el conjunto de datos.

En este bloque se han realizado dos pruebas, con el objetivo de ver el grado en que la modificación de la capa oculta afecta al resultado:

- Aumento del nº de neuronas de la capa fully connected.
- Aumento del nº de capas fully connected.

Data augmentation: Data augmentation consiste en un conjunto de técnicas que alteran las imágenes de entrada con el objetivo de generar artificialmente datos.
De esta forma es posible entrenar una red neuronal con un conjunto de imágenes reducido, siempre y cuando sea lo suficientemente variado, y se evita el problema que supone tener que recoger grandes cantidades de datos de entrenamiento.

Estas técnicas de alteración consisten en realizar pequeñas modificaciones a las imágenes. Algunos ejemplos son:

- Rotación aleatoria
- Re-escalado
- Volteo vertical/horizontal
- Recortar la imagen
- Ampliar la imagen
- Cambiar a escala de grises
- Añadir ruido (blur)

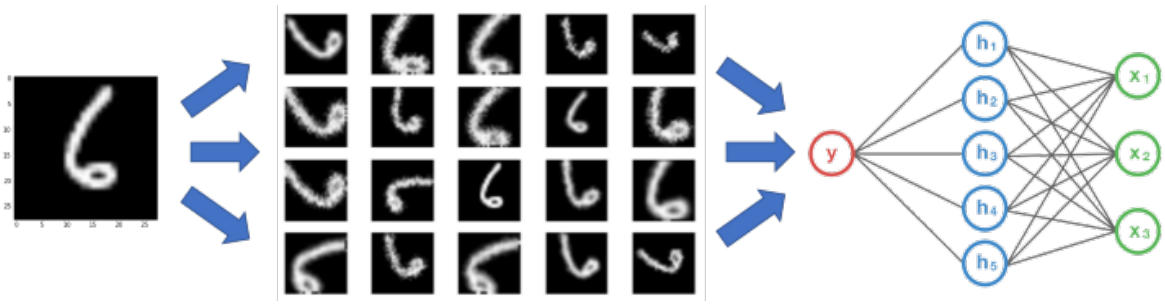

Ilustración 6: Data augmentation

Dropout: Es una técnica de regularización que tiene como objetivo eliminar el sobreajuste. Omite aleatoriamente neuronas de acuerdo a la cantidad indicada (0.25 significaría que se omite el 25% de las neuronas).

Aspectos relevantes del proyecto

En este apartado se expondrán los aspectos más relevantes de cada prueba, como por ejemplo el código empleado o los resultados obtenidos.

Cabe destacar que se ha hecho una división del 70% del total de imágenes para el conjunto de entrenamiento y el 30% restante para el conjunto de validación.

Para determinar la eficacia del modelo se utilizan las métricas de precisión (accuracy) y pérdida (loss). La precisión es una medida porcentual de las veces que se predice de forma correcta. La precisión de un modelo se mide mediante la diferencia entre la precisión de entrenamiento y la precisión de validación, tal que cuanto más similares sean, mejor es el modelo. La pérdida es un sumatorio de los errores cometidos para cada muestra. Yo me centraré más en la precisión.

Versión original del código

En esta prueba se muestra el código en su versión básica, sin modificar, adaptada para que lea las imágenes proporcionadas para este trabajo.

Librerías y ubicación de imágenes

Este apartado será común a todas las pruebas y a la versión final del código

Librerías

In [1]:
```
import urllib.request
import os
import zipfile

import matplotlib.image as mpimg
import matplotlib.pyplot as plt


import tensorflow as tf

from tensorflow.keras.optimizers import RMSprop
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.preprocessing import image


import numpy as np
```

Ilustración 7: Librerías utilizadas

Ubicación de imágenes de modelo y de test.

```
#Directorio base
base_dir = 'C:/Users/Saul/Desktop/SAUL/INFORMATICA/UBU 4º/2º semestre/TFG/Fotos_Proyecto_Detección_de_Bajas'

train_dir = os.path.join(base_dir, 'Entrenamiento')
validation_dir = os.path.join(base_dir, 'Validacion')

# Directorio con las fotos de gallinas clasificadas
train_alive_dir = os.path.join(train_dir, 'Gallinas_vivas')
train_dead_dir = os.path.join(train_dir, 'Gallinas_muertas')

# Directorio con las imágenes de modelo
validation_alive_dir = os.path.join(validation_dir, 'Gallinas_vivas')
validation_dead_dir = os.path.join(validation_dir, 'Gallinas_muertas')
```

Ilustración 8: Dirección de las imágenes

Definición del modelo

Versión original

```
model = tf.keras.models.Sequential([
    # tf.keras.layers.Conv2D(filtros, tamaño_kernel, activacion, input_shape)
    # input_size es la forma deseada de la imagen (150x150) con 3 bytes de color
    # Bloque de convolución 1
    tf.keras.layers.Conv2D(16, (3,3), activation='relu', input_shape=(150, 150, 3)),
    tf.keras.layers.MaxPooling2D(2,2),

    # Bloque de convolución 2
    tf.keras.layers.Conv2D(32, (3,3), activation='relu'),
    tf.keras.layers.MaxPooling2D(2,2),

    # Bloque de convolución 3
    tf.keras.layers.Conv2D(64, (3,3), activation='relu'),
    tf.keras.layers.MaxPooling2D(2,2),

    # "Aplana" los resultados para pasárselos a una DNN (Deep neural network)
    tf.keras.layers.Flatten(),

    # Neuronas de capa oculta: 512
    tf.keras.layers.Dense(512, activation='relu')
    ,
    # Neurona de salida que devuelve dos valores: 0 para gallinas muertas y 1 para gallinas vivas.
    tf.keras.layers.Dense(1, activation='sigmoid')
])
```

Ilustración 9: Definición del modelo original

```
from tensorflow.keras.optimizers import RMSprop

model.compile(optimizer=RMSprop(learning_rate=0.001),
              loss='binary_crossentropy',
              metrics = ['accuracy'])
```

```
# Reescalado de imágenes a 1./255.
train_datagen = ImageDataGenerator( rescale = 1.0/255. )
test_datagen  = ImageDataGenerator( rescale = 1.0/255. )

# --------------------------------------------------------------------
# Entrenamiento de las imágenes en lotes de 5 utilizando "train_datagen"
# --------------------------------------------------------------------
train_generator = train_datagen.flow_from_directory(train_dir,
                                                    batch_size=5,
                                                    class_mode='binary',
                                                    target_size=(150, 150))

# --------------------------------------------------------------------
# Validación de las imágenes en lotes de 5 utilizando "test_datagen"
# --------------------------------------------------------------------
validation_generator =  test_datagen.flow_from_directory(validation_dir,
                                                         batch_size=5,
                                                         class_mode  = 'binary',
                                                         target_size = (150, 150))
```

Ilustración 10: Generadores entrenamiento-validación

En este modelo solamente disponemos de tres bloques de convolución, consistentes en una capa de convolución y una capa de max-pooling. Cada capa de convolución tiene el doble de filtros que la anterior.

También consta de una capa de aplanado (Flatten), una capa oculta (Fully connected) de 512 neuronas y la capa de salida.

Entrenamiento

```
history = model.fit(train_generator,
                                validation_data=validation_generator,
                                #steps_per_epoch=25,
                                epochs=25,
                                #validation_steps=25,
                                verbose=2)
```

Ilustración 11: Entrenamiento del modelo

El único cambio realizado en esta parte es la eliminación de los parámetros “steps_per_epoch” y “validation_steps”. Esto se debe a que según el número de imágenes de entrada es posible que la red se quede sin datos, es decir, que el nº de pasos por ciclo*nº imágenes/ciclo debe de ser igual al nº de imágenes total.

Resultados obtenidos

```
Epoch 25/25
33/33 - 53s - loss: 2.2080e-09 - accuracy: 1.0000 - val_loss: 2.8492 - val_accuracy: 0.7286 - 53s/epoch - 2s/step
```

Como se puede apreciar, la diferencia entre las funciones de entrenamiento (loss, accuracy) y validación (val_loss, val_accuracy) son demasiado grandes, lo cual es indicativo de que se está dando sobreajuste.

```
resultado = model.evaluate(validation_generator, verbose=0)
print(f'Pérdida: {resultado[0]} / Precisión: {resultado[1]}')
```

```
Pérdida: 2.849167823791504 / Precisión: 0.7285714149475098
```

Ilustración 12: Valores pérdida/precisión del modelo original

Representación gráfica

```python
#------------------------------------------------------------
# Recupero los valores para dibujar las gráficas
#------------------------------------------------------------
acc = history.history['accuracy']
val_acc = history.history['val_accuracy']
loss = history.history['loss']
val_loss = history.history['val_loss']

epochs = range(len(acc))

#------------------------------------------------
# Accuracy vs val_accuracy
#------------------------------------------------
plt.plot(epochs, acc, 'bo', label='Training accuracy')
plt.plot(epochs, val_acc, 'b', label='Validation accuracy')
plt.title('accuracy vs val_accuracy')
plt.legend()

plt.figure()

#------------------------------------------------
# Loss vs val_loss
#------------------------------------------------
plt.plot(epochs, loss, 'bo', label='Training Loss')
plt.plot(epochs, val_loss, 'b', label='Validation Loss')
plt.title('loss vs val_loss')
plt.legend()

plt.show()
```

Ilustración 13: Recuperación de valores

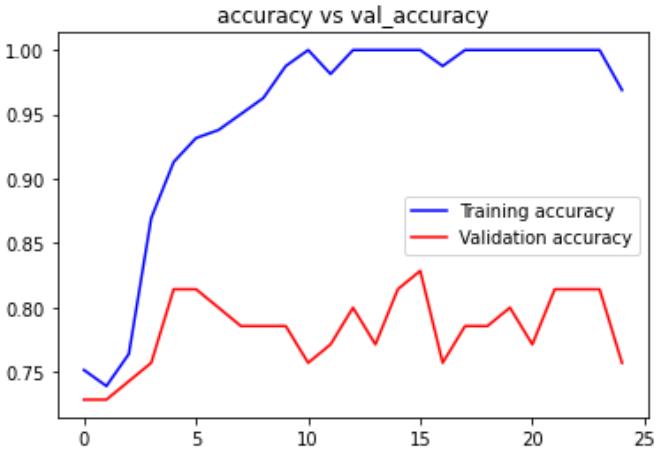

Ilustración 14: Comparativa de precisión

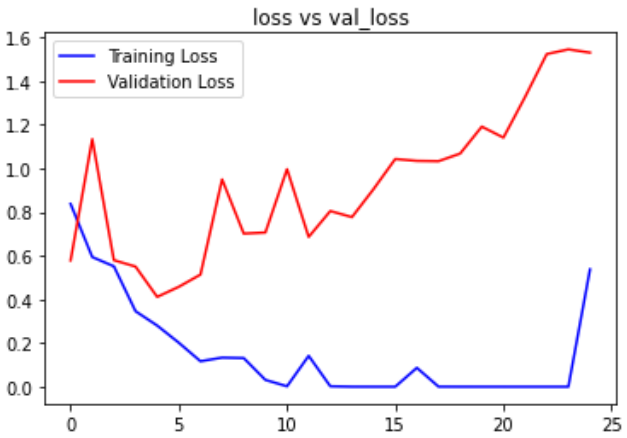

Ilustración 15: Comparativa de pérdida

Prueba nº 1: Aumento de bloques de convolución

Definición del modelo

```
model = tf.keras.models.Sequential([
    # tf.keras.layers.Conv2D(filtros, tamaño_kernel, activacion, input_shape)
    # input_size es la forma deseada de la imagen (150x150) con 3 bytes de color
    # Bloque de convolución 1
    tf.keras.layers.Conv2D(100, (3,3), activation='relu', input_shape=(150, 150, 3)),
    tf.keras.layers.MaxPooling2D(2,2),

    # Bloque de convolución 2
    tf.keras.layers.Conv2D(100, (3,3), activation='relu'),
    tf.keras.layers.MaxPooling2D(2,2),

    # Bloque de convolución 3
    tf.keras.layers.Conv2D(100, (3,3), activation='relu'),
    tf.keras.layers.MaxPooling2D(2,2),

    # Bloque de convolución 4
    tf.keras.layers.Conv2D(100, (3,3), activation='relu'),
    tf.keras.layers.MaxPooling2D(2,2),

    # Bloque de convolución 5
    tf.keras.layers.Conv2D(100, (3,3), activation='relu'),
    tf.keras.layers.MaxPooling2D(2,2),

    # "Aplana" los resultados para pasárselos a una DNN (Deep neural network)
    tf.keras.layers.Flatten(),

    # Neuronas de capa oculta: 512
    tf.keras.layers.Dense(512, activation='relu')
    ,
    # Neurona de salida que devuelve dos valores: 0 para gallinas muertas y 1 para gallinas vivas.
    tf.keras.layers.Dense(1, activation='sigmoid')
])
```

Ilustración 16: Modelo nº1

Para realizar esta prueba he añadido 2 bloques de convolución, consistentes en una capa de convolución y una max-pooling cada uno. He puesto el número de neuronas de cada capa a 100, por ser un número ni demasiado grande ni demasiado pequeño. El resto de parámetros no se ha modificado.

Resultados obtenidos

```
Epoch 25/25
33/33 - 56s - loss: 0.1619 - accuracy: 0.9627 - val_loss: 0.9503 - val_accuracy: 0.7286 - 56s/epoch - 2s/step
```

Ilustración 17: Resultados prueba 1

```
Pérdida: 0.9502633213996887 / Precisión: 0.7285714149475098
```

Ilustración 18: Valores de pérdida y precisión 1

Como podemos observar, la precisión al clasificar los datos del modelo es muy alta, mientras que la precisión al clasificar los datos de validación es de casi igual que en la versión original.

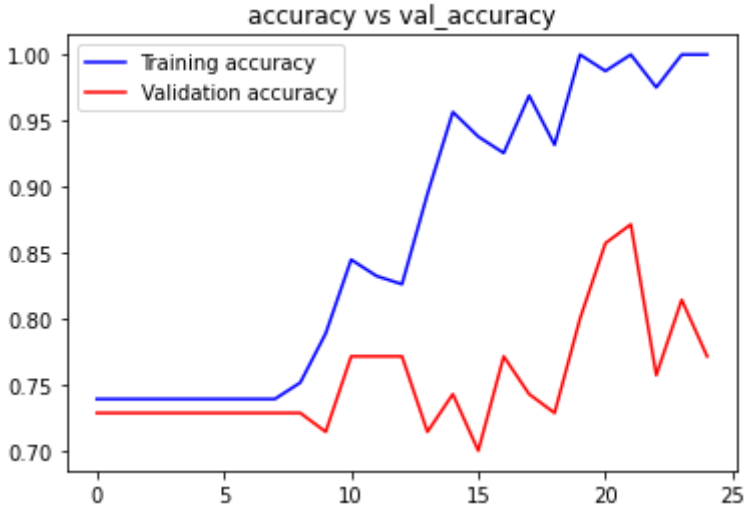

Ilustración 19: Precisión prueba nº1

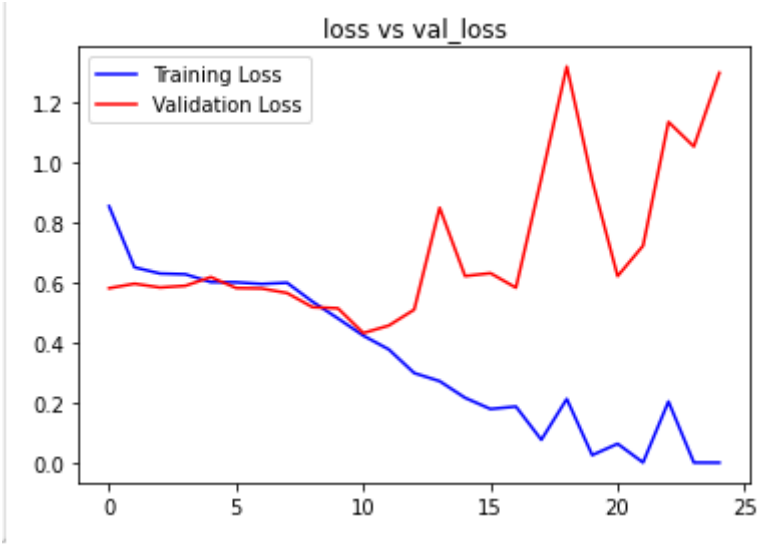

Ilustración 20: Pérdida prueba nº1

Prueba nº 2: Aumento del número de capas de convolución

Definición del modelo

```
model = tf.keras.models.Sequential([
    # tf.keras.layers.Conv2D(filtros, tamaño_kernel, activacion, input_shape)
    # input_size es la forma deseada de la imagen (150x150) con 3 bytes de color
    # Bloque de convolución 1
    tf.keras.layers.Conv2D(100, (3,3), activation='relu', input_shape=(150, 150, 3)),
    tf.keras.layers.Conv2D(100, (3,3), activation='relu', input_shape=(150, 150, 3)),
    tf.keras.layers.MaxPooling2D(2,2),

    # Bloque de convolución 2
    tf.keras.layers.Conv2D(100, (3,3), activation='relu'),
    tf.keras.layers.Conv2D(100, (3,3), activation='relu'),
    tf.keras.layers.MaxPooling2D(2,2),

    # Bloque de convolución 3
    tf.keras.layers.Conv2D(100, (3,3), activation='relu'),
    tf.keras.layers.Conv2D(100, (3,3), activation='relu'),
    tf.keras.layers.MaxPooling2D(2,2),

    # "Aplana" los resultados para pasárselos a una DNN (Deep neural network)
    tf.keras.layers.Flatten(),

    # Neuronas de capa oculta: 512
    tf.keras.layers.Dense(512, activation='relu')
    ,
    # Neurona de salida que devuelve dos valores: 0 para gallinas muertas y 1 para gallinas vivas.
    tf.keras.layers.Dense(1, activation='sigmoid')
])
```

Ilustración 21: Modelo nº2

Éste modelo consta de tres bloques, cada uno con dos capas de convolución y una max-pooling. El número de neuronas de cada capa sigue siendo 100.

Resultados obtenidos

```
Epoch 24/25
33/33 - 84s - loss: 8.0262e-07 - accuracy: 1.0000 - val_loss: 6.1463 - val_accuracy: 0.6714 - 84s/epoch - 3s/step
Epoch 25/25
33/33 - 84s - loss: 2.9753e-07 - accuracy: 1.0000 - val_loss: 6.8979 - val_accuracy: 0.6714 - 84s/epoch - 3s/step
```

```
resultado = model.evaluate(validation_generator, verbose=0)
print(f'Pérdida: {resultado[0]} / Precisión: {resultado[1]}')
```

```
Pérdida: 6.8979105949401855 / Precisión: 0.6714285612106323
```

Ilustración 22: Resultados prueba 2

Como podemos observar, no sólo hay sobreajuste, sino que la precisión de validación es muy pobre.

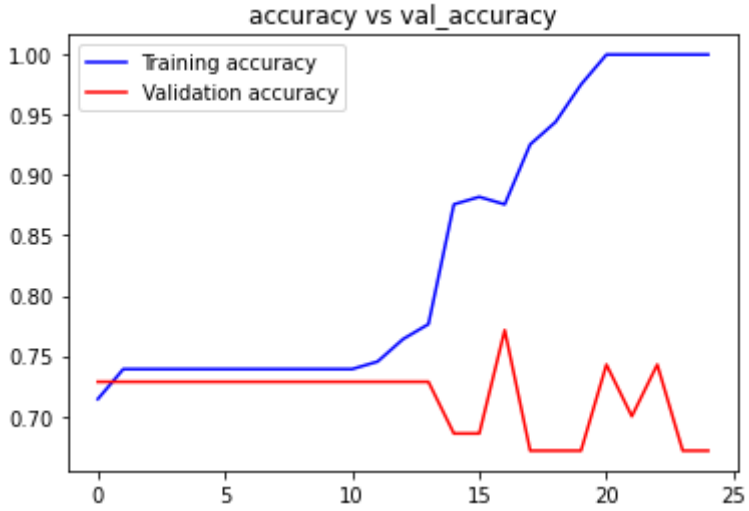

Ilustración 23: Precisión prueba nº 2

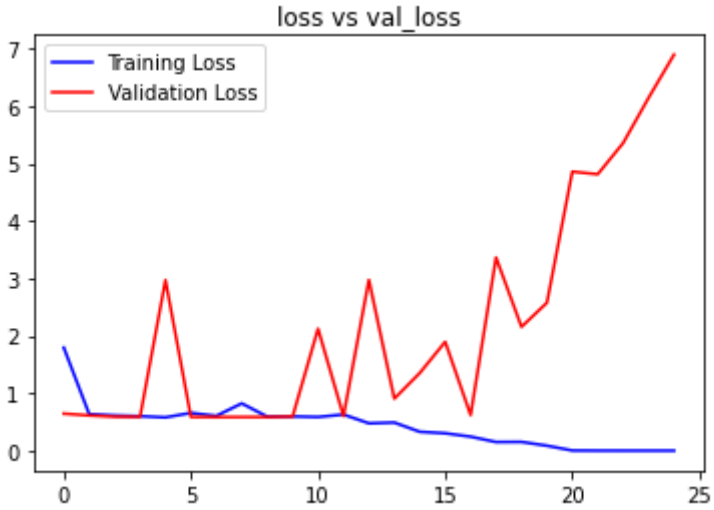

Ilustración 24: Pérdida prueba nº2

Prueba nº 3: Aumento del número de filtros

Definición del modelo

```
# Bloque de convolución 1
tf.keras.layers.Conv2D(64, (3,3), activation='relu', input_shape=(150, 150, 3)),
tf.keras.layers.MaxPooling2D(2,2),

# Bloque de convolución 2
tf.keras.layers.Conv2D(128, (3,3), activation='relu'),
tf.keras.layers.MaxPooling2D(2,2),

# Bloque de convolución 3
tf.keras.layers.Conv2D(256, (3,3), activation='relu'),
tf.keras.layers.MaxPooling2D(2,2),
```

Ilustración 25: Modelo nº3

Al igual que el anterior, este modelo tiene tres bloques. Cada bloque tiene ahora una sola capa de convolución y una max-pooling, pero el número de filtros se incrementa en cada capa, de tal manera que, igual que en la versión original, cada capa tiene el doble de filtros que la anterior, con la diferencia de que se ha multiplicado el nº de filtros de cada capa por cuatro con respecto de la versión original.

Resultados obtenidos

```
Epoch 25/25
33/33 - 64s - loss: 2.0253e-06 - accuracy: 1.0000 - val_loss: 1.9740 - val_accuracy: 0.7429 - 64s/epoch - 2s/step
```

```
1 resultado = model.evaluate(validation_generator, verbose=0)
2 print(f'Pérdida: {resultado[0]} / Precisión: {resultado[1]}')
```

```
Pérdida: 1.9740493297576904 / Precisión: 0.7428571581840515
```

Ilustración 26: Resultados prueba 3

Si bien los resultados son algo mejores que en la prueba anterior, sigue mostrando sobreajuste.

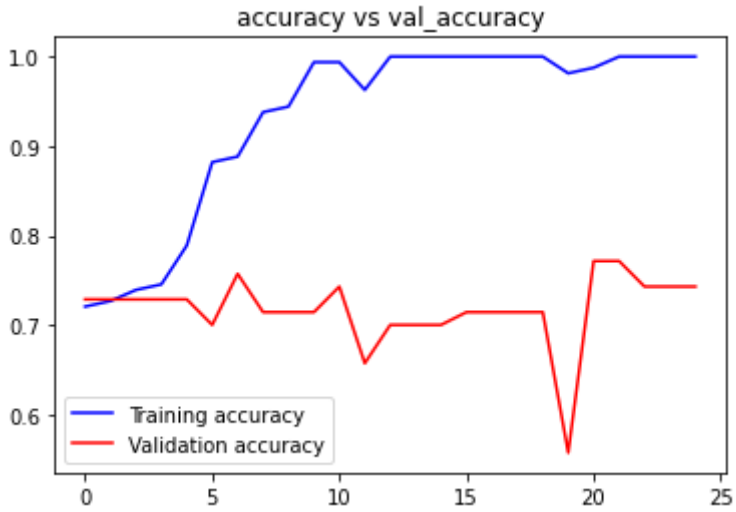

Ilustración 27: Precisión prueba nº3

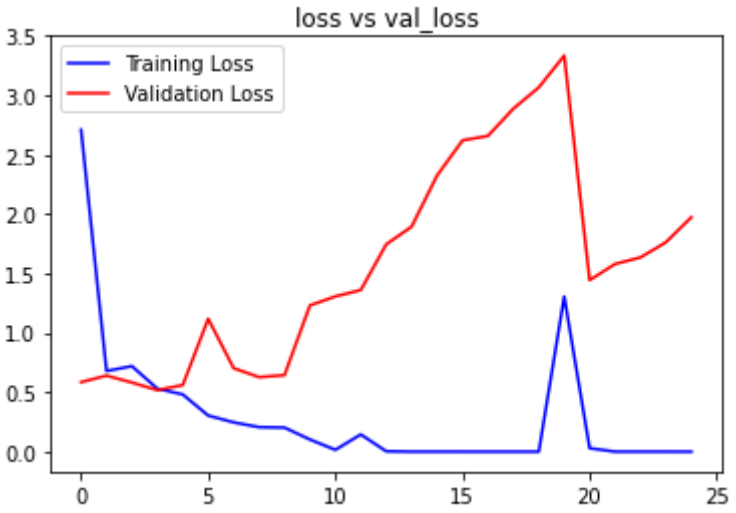

Ilustración 28: Pérdida prueba nº3

Prueba nº4: Combinación de las pruebas anteriores

Definición del modelo

```
# Bloque de convolución 1
tf.keras.layers.Conv2D(32, (3,3), activation='relu', input_shape=(150, 150, 3)),
tf.keras.layers.Conv2D(32, (3,3), activation='relu', input_shape=(150, 150, 3)),
tf.keras.layers.MaxPooling2D(2,2),

# Bloque de convolución 2
tf.keras.layers.Conv2D(64, (3,3), activation='relu'),
tf.keras.layers.Conv2D(64, (3,3), activation='relu'),
tf.keras.layers.MaxPooling2D(2,2),

# Bloque de convolución 3
tf.keras.layers.Conv2D(128, (3,3), activation='relu'),
tf.keras.layers.Conv2D(128, (3,3), activation='relu'),
tf.keras.layers.MaxPooling2D(2,2),

# Bloque de convolución 4
tf.keras.layers.Conv2D(256, (3,3), activation='relu'),
tf.keras.layers.Conv2D(256, (3,3), activation='relu'),
tf.keras.layers.MaxPooling2D(2,2),
```

Ilustración 29: Modelo nº4

Esta prueba combina características de todas las anteriores. El modelo consta de cuatro bloques de convolución de tres capas cada uno: dos capas de convolución y una de max-pooling. La distribución de filtros sigue el mismo patrón que en la prueba 3 (cada bloque tiene el doble de filtros que el anterior, y en este caso se multiplica el nº de filtros por bloque original por dos).

Resultados obtenidos

```
Epoch 100/100
33/33 - 56s - loss: 8.7921e-06 - accuracy: 1.0000 - val_loss: 2.6835 - val_accuracy: 0.8000 - 56s/epoch - 2s/step
```

```
1 resultado = model.evaluate(validation_generator, verbose=0)
2 print(f'Pérdida: {resultado[0]} / Precisión: {resultado[1]}')
```

```
Pérdida: 2.6835219860076904 / Precisión: 0.800000011920929
```

Ilustración 30: Resultados prueba 4

En este caso se ha entrenado la red durante 100 ciclos, para poder observar mejor su comportamiento. Como los modelos anteriores presentaban sobreajuste en ciclos muy tempranos se realizó el entrenamiento en 25 ciclos para ahorrar tiempo. Aquí sin embargo no se empiezan a mostrar signos de sobreajuste hasta el ciclo nº 50, de modo que se ha dejado dicho número de ciclos a 100 para garantizar la corrección de las conclusiones.

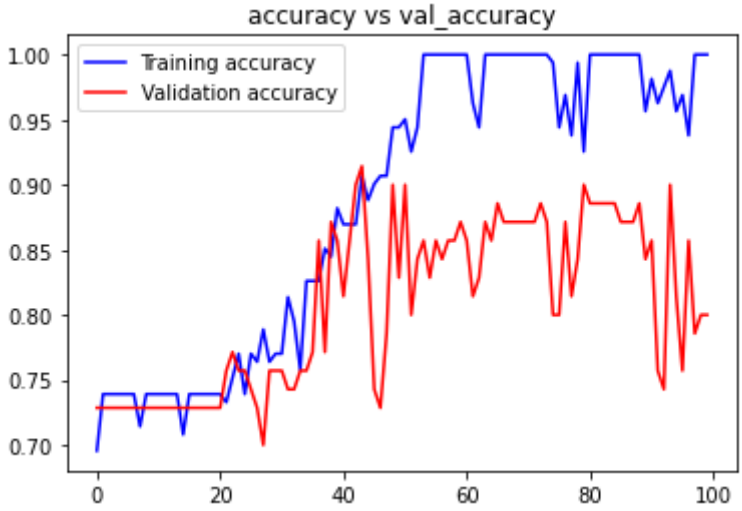

Ilustración 31: Precisión prueba nº4

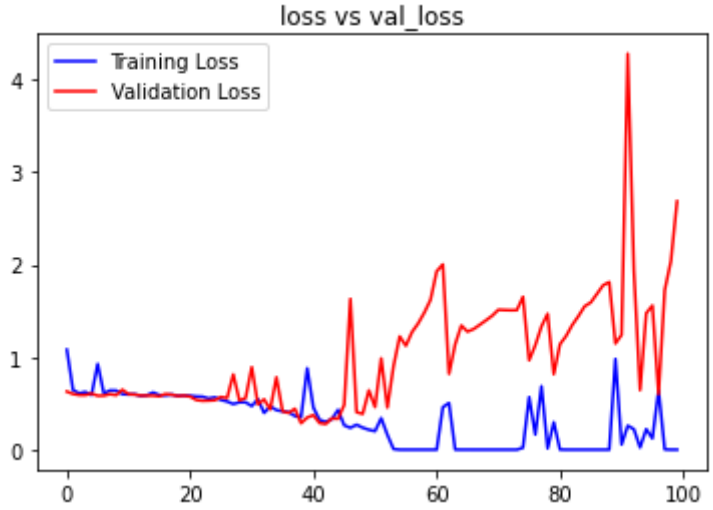

Ilustración 32: Pérdida prueba nº4

Prueba nº5: Dropout

Definición del modelo

```
# Bloque de convolución 1
tf.keras.layers.Conv2D(100, (3,3), activation='relu', input_shape=(150, 150, 3)),
tf.keras.layers.MaxPooling2D(2,2),
tf.keras.layers.Dropout(0.25),

# Bloque de convolución 2
tf.keras.layers.Conv2D(100, (3,3), activation='relu'),
tf.keras.layers.MaxPooling2D(2,2),
tf.keras.layers.Dropout(0.25),

# Bloque de convolución 3
tf.keras.layers.Conv2D(100, (3,3), activation='relu'),
tf.keras.layers.MaxPooling2D(2,2),
tf.keras.layers.Dropout(0.25),
```

Ilustración 33: Modelo nº5

Para la realización de este modelo se ha introducido una capa de Dropout en cada bloque de convolución, en la cual se omiten el 25% de las neuronas durante el proceso de entrenamiento. El resto de parámetros es igual que en la versión original (tres bloques, una capa de convolución por cada uno) pero con 100 neuronas.

Resultados obtenidos

```
Epoch 100/100
33/33 - 60s - loss: 2.3627e-10 - accuracy: 1.0000 - val_loss: 3.5175 - val_accuracy: 0.7429 - 60s/epoch - 2s/step
```

```
1 resultado = model.evaluate(validation_generator, verbose=0)
2 print(f'Pérdida: {resultado[0]} / Precisión: {resultado[1]}')
```

```
Pérdida: 3.5175092220306396 / Precisión: 0.7428571581840515
```

Ilustración 34: Resultados prueba 5

Los resultados no son buenos, de hecho son muy similares a los primeros resultados obtenidos.

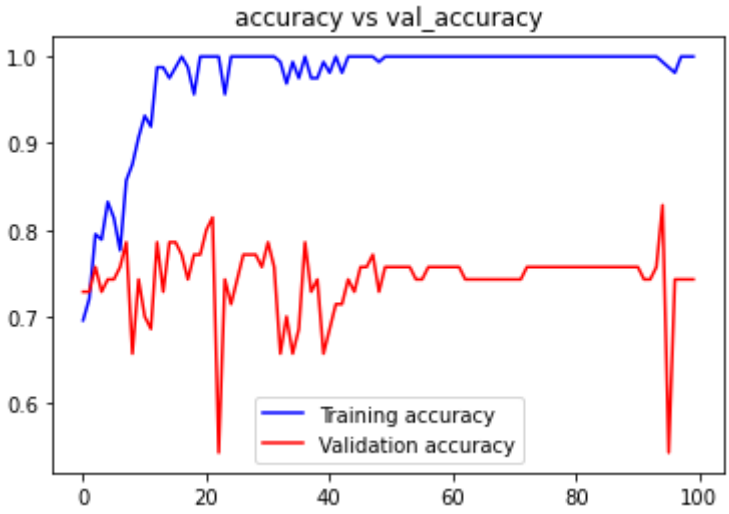

Ilustración 35: Precisión prueba nº5

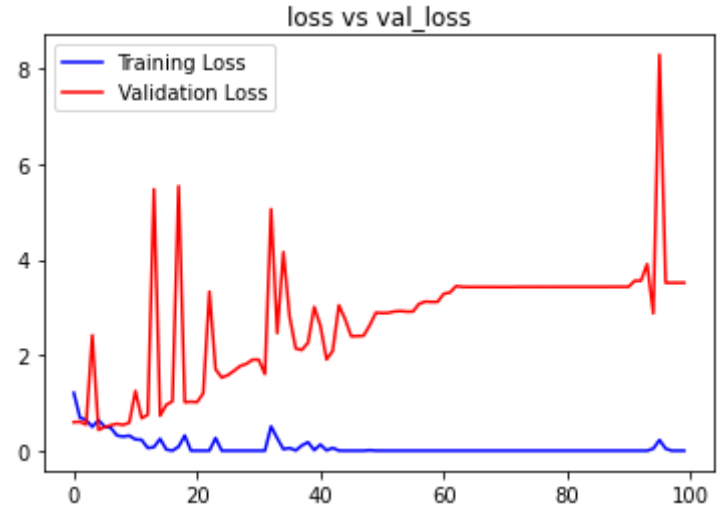

Ilustración 36: Pérdida prueba nº5

Prueba nº6: Aumento de neuronas de la capa fully connected

Definición del modelo

```
    # "Aplana" los resultados para pasárselos a una DNN (Deep neural network)
    tf.keras.layers.Flatten(),

    # Neuronas de capa oculta: 512
    #Original: 512
    #Nuevo: 1024
    tf.keras.layers.Dense(1024, activation='relu')
    ,
    # Neurona de salida que devuelve dos valores: 0 para gallinas muertas y 1 para gallinas vivas.
    tf.keras.layers.Dense(1, activation='sigmoid')
])
```

Ilustración 37: Modelo nº6

En este modelo se ha duplicado el nº de neuronas de la capa fully connected con respecto de la versión original.

Resultados obtenidos

```
Epoch 25/25
33/33 - 66s - loss: 1.1137e-09 - accuracy: 1.0000 - val_loss: 3.0041 - val_accuracy: 0.7000 - 66s/epoch - 2s/step
```

```
resultado = model.evaluate(validation_generator, verbose=0)
print(f'Pérdida: {resultado[0]} / Precisión: {resultado[1]}')
```

```
Pérdida: 3.004074811935425 / Precisión: 0.699999988079071
```

Ilustración 38: Resultados prueba 6

Como se puede observar, la red presenta sobreajuste, y resultados muy pobres con el conjunto de validación.

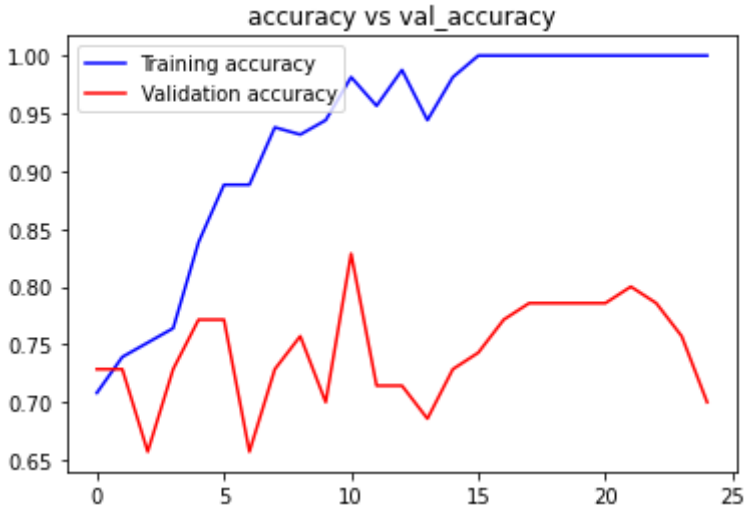

Ilustración 39: Precisión prueba nº6

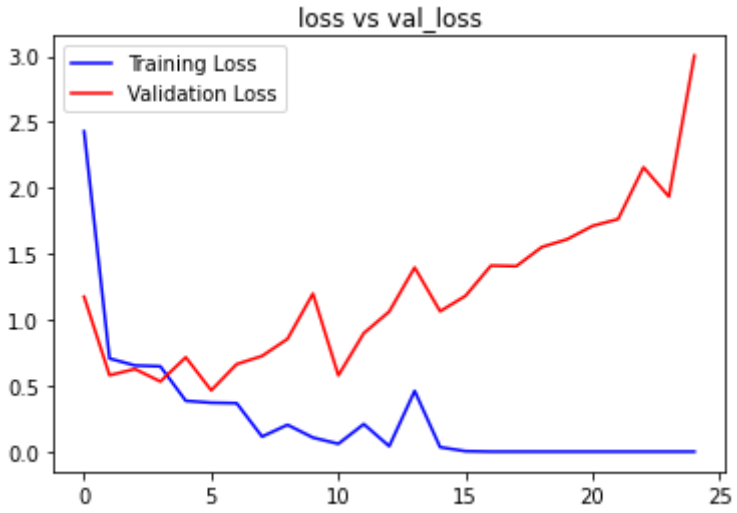

Ilustración 40: Pérdida prueba nº6

Prueba nº7: Incremento de capas fully connected

Definición del modelo

```
    # Capas Fully connected
    tf.keras.layers.Dense(500, activation='relu'),
    tf.keras.layers.Dense(550, activation='relu'),
    tf.keras.layers.Dense(600, activation='relu'),
    tf.keras.layers.Dense(128, activation='relu'),

    # Neurona de salida que devuelve dos valores: 0 para gallinas muertas y 1 para gallinas vivas.
    tf.keras.layers.Dense(1, activation='sigmoid')
])
```

Ilustración 41: Modelo nº7

En este modelo se han implementado tres capas ocultas extra.

Resultados obtenidos

```
Epoch 25/25
33/33 - 58s - loss: 3.6129e-10 - accuracy: 1.0000 - val_loss: 4.9994 - val_accuracy: 0.7429 - 58s/epoch - 2s/step
```

```
1 resultado = model.evaluate(validation_generator, verbose=0)
2 print(f'Pérdida: {resultado[0]} / Precisión: {resultado[1]}')
```

```
Pérdida: 4.999392986297607 / Precisión: 0.7428571581840515
```

Ilustración 42: Resultados prueba 7

Como se puede observar, los resultados son muy pobres, equiparables a los obtenidos en las primeras pruebas.

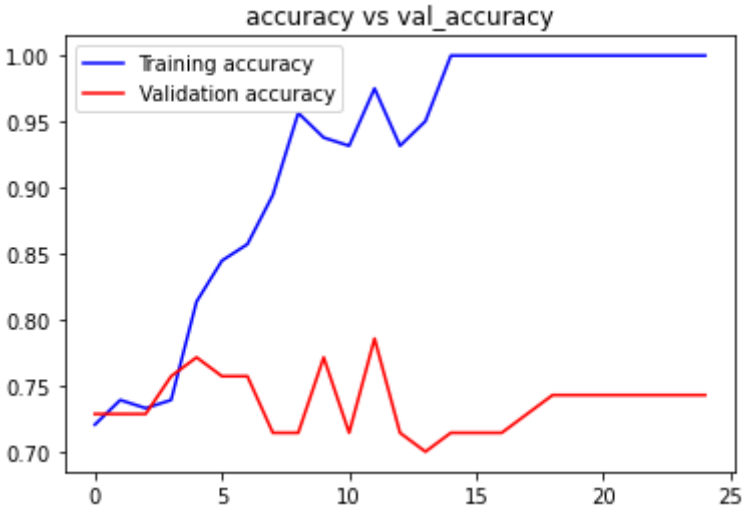

Ilustración 43: Precisión prueba nº7

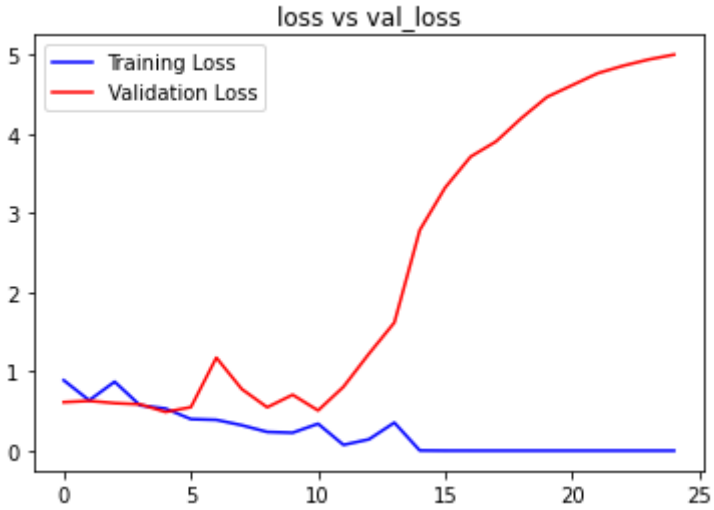

Ilustración 44: Pérdida prueba nº7

Prueba nº8: Data augmentation

Definición del modelo

```
model = tf.keras.models.Sequential([
    # tf.keras.layers.Conv2D(filtros, tamaño_kernel, activacion, input_shape)
    # input_size es la forma deseada de la imagen (150x150) con 3 bytes de color
    # Bloque de convolución 1
    tf.keras.layers.Conv2D(100, (3,3), activation='relu', input_shape=(150, 150, 3)),
    tf.keras.layers.MaxPooling2D(2,2),

    # Bloque de convolución 2
    tf.keras.layers.Conv2D(100, (3,3), activation='relu'),
    tf.keras.layers.MaxPooling2D(2,2),

    # Bloque de convolución 3
    tf.keras.layers.Conv2D(100, (3,3), activation='relu'),
    tf.keras.layers.MaxPooling2D(2,2),

    # "Aplana" los resultados para pasárselos a una DNN (Deep neural network)
    tf.keras.layers.Flatten(),

    # Neuronas de capa oculta: 512
    tf.keras.layers.Dense(512, activation='relu')
    ,
    # Neurona de salida que devuelve dos valores: 0 para gallinas muertas y 1 para gallinas vivas.
    tf.keras.layers.Dense(1, activation='sigmoid')
])
```

Ilustración 45: Modelo nº8

En este caso el modelo se ha dejado tal y como en la versión original, simplemente se ha dejado el número de filtros a 100.

Las técnicas de data augmentation se han implementado antes del entrenamiento. Previamente sólo se re-escalaban las imágenes para normalizarlas, ahora se ha implementado además, rotación de 90º, alteración del brillo entre 0.2 y 0.8, volteo vertical y volteo horizontal.

```
# DATA AUGMENTATION: SE HA APLICADO ROTACIÓN, BRILLO Y VOLTEO (VERTICAL Y HORIZONTAL)
train_datagen = ImageDataGenerator(rotation_range=90,
                                   brightness_range=(0.2, 0.8),
                                   horizontal_flip=True,
                                   vertical_flip=True,
                                   rescale = 1.0/255.)

test_datagen  = ImageDataGenerator( rescale = 1.0/255. )

# --------------------------------------------------------------------
# Entrenamiento de las imágenes en lotes de 5 utilizando "train_datagen"
# --------------------------------------------------------------------
train_generator = train_datagen.flow_from_directory(train_dir,
                                                    batch_size=5,
                                                    class_mode='binary',
                                                    target_size=(150, 150))

# --------------------------------------------------------------------
# Validación de las imágenes en lotes de 5 utilizando "test_datagen"
# --------------------------------------------------------------------
validation_generator =  test_datagen.flow_from_directory(validation_dir,
                                                         batch_size=5,
                                                         class_mode  = 'binary',
                                                         target_size = (150, 150))
```

Ilustración 46: Data augmentation

Resultados obtenidos

```
Epoch 100/100
33/33 - 61s - loss: 0.4249 - accuracy: 0.7950 - val_loss: 1.1536 - val_accuracy: 0.7429 - 61s/epoch - 2s/step
```

```
1 resultado = model.evaluate(validation_generator, verbose=0)
2 print(f'Pérdida: {resultado[0]} / Precisión: {resultado[1]}')
```

```
Pérdida: 1.1536223888397217 / Precisión: 0.7428571581840515
```

Ilustración 47: Resultados prueba 8

Como podemos observar, los resultados son mucho mejores. Apenas hay diferencia entre la precisión de entrenamiento y la de validación, lo que significa que la red es capaz de generalizar con éxito.

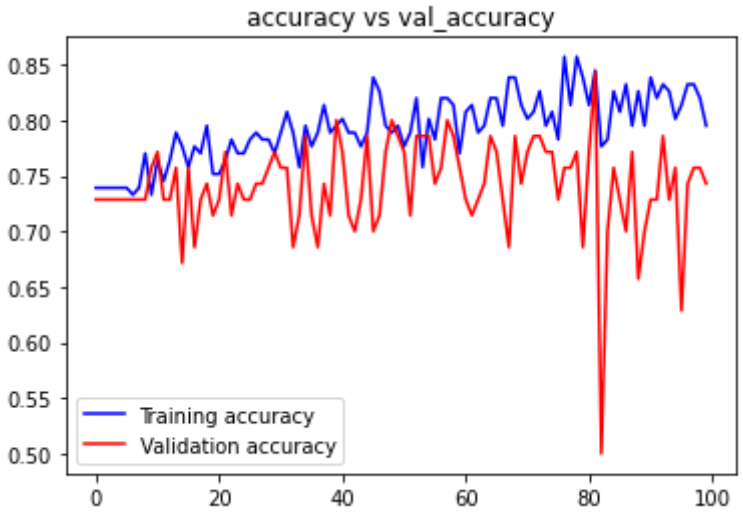

Ilustración 48: Precisión prueba nº8

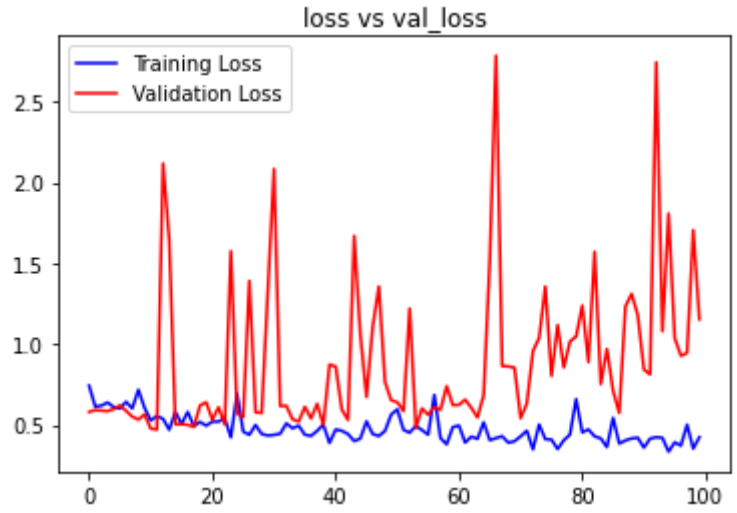

Ilustración 49: Pérdida prueba nº8

Versión final

Definición del modelo

```
model = tf.keras.models.Sequential([
    # tf.keras.layers.Conv2D(filtros, tamaño_kernel, activacion, input_shape)
    # input_size es la forma deseada de la imagen (150x150) con 3 bytes de color
    # Bloque de convolución 1
    tf.keras.layers.Conv2D(80, (3,3), activation='relu', input_shape=(150, 150, 3)),
    tf.keras.layers.MaxPooling2D(2,2),

    # Bloque de convolución 2
    tf.keras.layers.Conv2D(100, (3,3), activation='relu'),
    tf.keras.layers.MaxPooling2D(2,2),

    # Bloque de convolución 3
    tf.keras.layers.Conv2D(100, (3,3), activation='relu'),
    tf.keras.layers.Conv2D(100, (3,3), activation='relu'),
    tf.keras.layers.MaxPooling2D(2,2),

    #Dropout
    tf.keras.layers.Dropout(0.25),

    # "Aplana" los resultados para pasárselos a una DNN (Deep neural network)
    tf.keras.layers.Flatten(),

    # Neuronas de capa oculta: 512
    tf.keras.layers.Dense(256, activation='relu'),
    tf.keras.layers.Dense(32, activation='relu'),

    # Neurona de salida que devuelve dos valores: 0 para gallinas muertas y 1 para gallinas vivas.
    tf.keras.layers.Dense(1, activation='sigmoid')
])
```

Ilustración 50: Modelo final

Para esta versión se han combinado los mejores aspectos de las demás pruebas.
Este modelo consta de tres bloques de convolución, dos de los cuales solo tienen una capa de convolución, mientras que el último tiene 2. Asimismo, el nº de filtros de los dos últimos bloques es mayor que el primero. Se ha implementado una capa Dropout que omite el 25% de las neuronas, y una capa oculta extra. Además, en las capas ocultas se ha reducido el nº de neuronas, en comparación con versiones anteriores.

```
# DATA AUGMENTATION: ROTACIÓN, BRILLO, VOLTEO, REESCALADO
train_datagen = ImageDataGenerator(rotation_range=90,
                                   brightness_range=(0.2, 0.8),
                                   horizontal_flip=True,
                                   vertical_flip=True,
                                   rescale = 1.0/255.)

test_datagen  = ImageDataGenerator( rescale = 1.0/255. )
```

Ilustración 51: Data augmentation V.final

Las técnicas de Data augmentation se han implementado igual que en la prueba 8.

Resultados obtenidos

```
Epoch 100/100
33/33 - 63s - loss: 0.4945 - accuracy: 0.7888 - val_loss: 1.2009 - val_accuracy: 0.7714 - 63s/epoch - 2s/step
```

```
1 resultado = model.evaluate(validation_generator, verbose=0)
2 print(f'Pérdida: {resultado[0]} / Precisión: {resultado[1]}')
```

```
Pérdida: 1.2008684873580933 / Precisión: 0.7714285850524902
```

Ilustración 52: Resultado final

Como se puede observar, los resultados que ofrece este modelo son muy superiores a los resultados obtenidos en cualquiera de las pruebas anteriores.

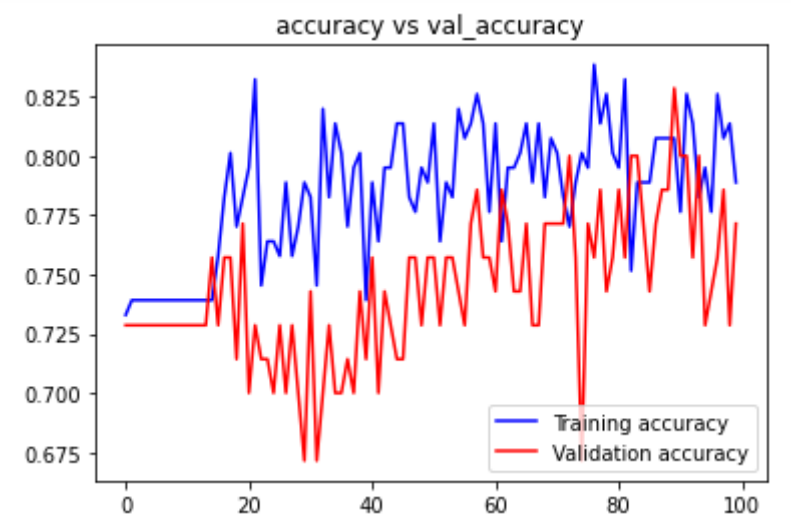

Ilustración 53: Precisión versión final

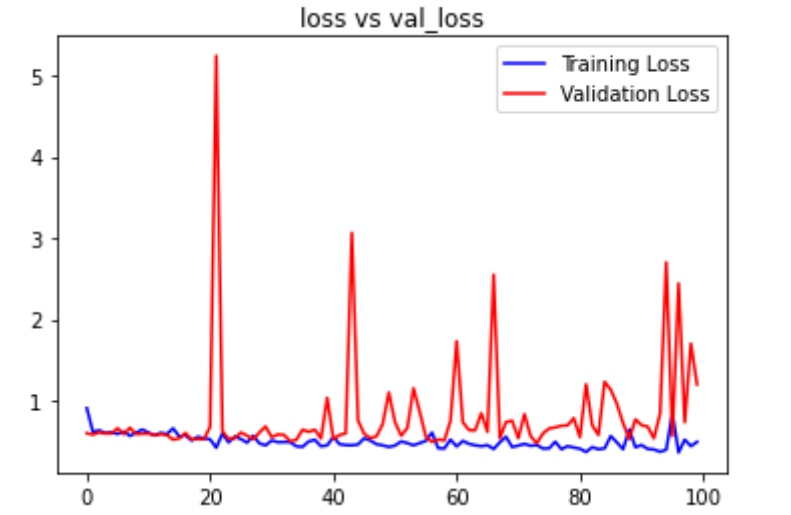

Ilustración 54: Pérdida versión final

Trabajos relacionados

Como se ha indicado al principio del documento, la idea de este trabajo es elaborar un programa que entrene una red neuronal convolucional, de manera que al sacar una foto de una jaula con una cámara, la red pueda reconocer si hay una gallina muerta en su interior.

Este trabajo se enmarca dentro del campo de la visión artificial, es decir la capacidad de adquirir, procesar y analizar imágenes del mundo real mediante un ordenador para producir algún resultado.

Tras buscar detenidamente trabajos similares, he encontrado varios que cumplen objetivos similares, pero ninguno igual que éste. A continuación se exponen algunos trabajos relacionados con la visión artificial publicados hasta la fecha.

• ***Desarrollo de un sistema de detección de personas en ambientes de interior mediante el uso de cámaras ojo de pez y algoritmos de Deep Learning. Autora: Clara Menduina Fernández.***

Este trabajo trata de detectar personas en espacios cerrados utilizando cámaras de ojo de pez y algoritmos de clasificación de Machine Learning tales como KNN, Random Forest o Regresión logística.

• ***Diseño de una aplicación de reconocimiento óptico de caracteres mediante Deep Learning. Autora: Paula Miles Uribe.***

Este trabajo trata sobre la implementación de un algoritmo que reconozca caracteres escritos y clasificarlos, para lo cual se implementa una red neuronal convolucional que recibe imágenes de dichos caracteres y extrae sus características para posterior clasificación.

Conclusiones y líneas de trabajo futuras

Versión	Accuracy	Val_accuracy
Versión original	1	0’7286
Prueba 1: Bloques de convolución	0’9627	0’7286
Prueba 2: Capas de convolución	1	0’6714
Prueba 3: Nº filtros	1	0’7429
Prueba 4: Combinación de pruebas 1, 2 y 3	1	0’8
Prueba 5: Dropout	1	0’7429
Prueba 6: Neuronas fully connected	1	0’7
Prueba 7: Capas fully connected	1	0’7429
Prueba 8: Data augmentation	0’795	0’7429
Versión final	0’7888	0’7714

Tras observar los resultados podemos concluir que la técnica que mejor ha funcionado es la de data augmentation. Si bien todas las técnicas han contribuido a mejorar la precisión al clasificar el conjunto de validación, la que más ha contribuido a eliminar el sobreajuste es el data augmentation.

De cara al futuro quedaría implementar este código en una cámara con Python integrado para darle el uso esperado, que es el de poder sacar fotos de las jaulas y que clasifique las imágenes de gallinas obtenidas.

Bibliografía

Como funcionan las convolutional neural networks vision por ordenador. (29 de 9 de 2018). Obtenido de https://www.aprendemachinelearning.com/como-funcionan-las-convolutional-neural-networks-vision-por-ordenador/

IBM corporation. (17 de 8 de 2021). Recuperado el 18 de 4 de 2022, de https://www.ibm.com/docs/es/spss-modeler/SaaS?topic=networks-neural-model

AI Wiki. (s.f.). Obtenido de Accuracy and Loss: https://machine-learning.paperspace.com/wiki/accuracy-and-loss#:~:text=Unlike%20accuracy%2C%20loss%20is%20not,is%20to%20minimize%20this%20value

Barrios, J. (s.f.). *juanbarrios.com*. Recuperado el 20 de 4 de 2022, de https://www.juanbarrios.com/redes-neurales-convolucionales/

Brownlee, J. (22 de 4 de 2019). *A gentle introduction to Pooling layers for convolutional neural networks*. Obtenido de https://machinelearningmastery.com/pooling-layers-for-convolutional-neural-networks/

Brownlee, J. (17 de 4 de 2019). *How do convolutional layers work in Deep Learning neural networks?* Obtenido de https://machinelearningmastery.com/convolutional-layers-for-deep-learning-neural-networks/

Cuemath. (s.f.). Obtenido de Non-linear function: https://www.cuemath.com/calculus/nonlinear-functions/

Fully connected layers in convolutional neural networks. (s.f.). Obtenido de https://indiantechwarrior.com/fully-connected-layers-in-convolutional-neural-networks/

Gandhi, A. (2021). *Data Augmentation | How to use Deep Learning when you have Limited Data — Part 2*. Obtenido de https://nanonets.com/blog/data-augmentation-how-to-use-deep-learning-when-you-have-limited-data-part-2/

Géron, A. (5 de 11 de 2018). *Hands-on Machine Learning with Scikit-Learn, Keras & Tensorflow*. Obtenido de https://www.knowledgeisle.com/wp-content/uploads/2019/12/2-Aurélien-Géron-Hands-On-Machine-Learning-with-Scikit-Learn-Keras-and-Tensorflow_-Concepts-Tools-and-Techniques-to-Build-Intelligent-Systems-O'Reilly-Media-2019.pdf

Google Colaboratory (Colab). (s.f.). *"Cat vs. Dog Image Classification"*. Obtenido de https://colab.research.google.com/github/google/eng-edu/blob/master/ml/pc/exercises/image_classification_part1.ipynb

Keras. (s.f.). Obtenido de https://es.wikipedia.org/wiki/Keras

Max pooling. (s.f.). Obtenido de https://paperswithcode.com/method/max-pooling

Sanagapati, P. (2019). *What is Dropout Regularization?* Obtenido de https://www.kaggle.com/code/pavansanagapati/what-is-dropout-regularization-find-

out/notebook

Sharma, P. (s.f.). *Convolution filters*. Obtenido de https://iq.opengenus.org/convolution-filters/#:~:text=Convolution%20filters%20are%20filters%20(multi,covered%20along%20with%20basic%20idea

Stateofheart AI. (s.f.). Obtenido de Affine layer: https://www.stateoftheart.ai/concepts/5b906a33-ce2a-4610-8667-2e826826aaa0

Takimoglu, A. (30 de 4 de 2021). *What is Data Augmentation? Techniques & Examples in 2022*. Obtenido de https://research.aimultiple.com/data-augmentation/

Tensorflow. (s.f.). Obtenido de https://es.wikipedia.org/wiki/TensorFlow#TensorFlow

Ye, A. (19 de 6 de 2020). *Finally, an intuitive explanation of why ReLU works*. Obtenido de https://towardsdatascience.com/if-rectified-linear-units-are-linear-how-do-they-add-nonlinearity-40247d3e4792