PRACTICA 2 DE FUNDAMENTOS DE LOS SISTEMAS INTELIGENTES: REDES NEURONALES

Nicolás Almeida Ramírez

Tinizara María Rodríguez Delgado

El objetivo de esta práctica es el entrenamiento de una red neuronal, para poder diferenciar entre los diferentes tipos de animales que conforman el dataset, en este caso son: elefante, mariposa, vaca, oveja y ardilla.

Comenzamos importando todas las librerías que usaremos, tanto para realizar el entreno de la red, como para los diferentes *plots* y cálculo de la matriz de confusión.

IMPORTS

In [2]:
```
from keras.preprocessing.image import ImageDataGenerator
from keras.models import Sequential
from keras.layers import Dense, Dropout
from keras.optimizers import RMSprop
from keras.layers import Dense, Dropout, Flatten
from keras.layers import Conv2D, MaxPooling2D
from keras.callbacks import EarlyStopping
from keras import backend as K
import keras
from time import time

import numpy as np
from sklearn.metrics import confusion_matrix
from matplotlib import pyplot as plt

print("Imports realizados")
```

```
Imports realizados
```

DATA AUGMENTATION

Con el *data augmentation* hacemos referencia a las diferentes variaciones que se le realizará a las imágenes que conforman el *dataset* con el fin de obtener más imágenes para poder entrenar la red.

En este caso, a las imágenes le aplicamos rotación, zoom o un factor de reescalada.

In [8]:
```
batch_size = 15

train_data_dir = './animals/dataset/train'
validation_data_dir = './animals/dataset/val'

train_datagen = ImageDataGenerator(
        rescale=1./255,
        rotation_range=15,
        zoom_range=0.1)

test_datagen = ImageDataGenerator(rescale=1./255)

train_generator = train_datagen.flow_from_directory(
        train_data_dir,
        target_size=(150, 150),
        batch_size=batch_size,
        class_mode='categorical')

validation_generator = test_datagen.flow_from_directory(
        validation_data_dir,
        target_size=(150, 150),
        batch_size=batch_size,
        class_mode='categorical')

print("Data obtenida")
```

```
Found 5736 images belonging to 5 classes.
Found 2460 images belonging to 5 classes.
Data obtenida
```

En el apartado del modelo, hacemos uso del secuencial, que es más simple, ya que se conforma de una pila lineal de capas.

Añadimos diferentes capas convolutivas, usando como función de activación RELU, debido a que probamos usando la función de activación ELU, y obteníamos unos valores muy bajos en cuanto a la *accuracy*, en comparación con los que obteníamos usando RELU.

In [41]:
```
model = Sequential()
model.add(Conv2D(32, kernel_size=(3, 3),
                 activation='relu',
                 input_shape=(150, 150, 3)))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Conv2D(128, (3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Flatten())
model.add(Dense(128, activation='relu'))

model.add(Dropout(0.2))
model.add(Dense(5, activation='softmax'))

model.compile(loss='mse',
              optimizer=keras.optimizers.SGD(learning_rate=1),
              metrics=['accuracy'])


print("Resumen del modelo")
model.summary()
```

En cuanto al entrenamiento, con 20 épocas, y un *bacth size* de 15, obtuvimos los mejores resultados, llegando a un 74% de *val_accuracy.*

In [42]:
```
epochs = 20

es = EarlyStopping(monitor='val_accuracy', mode='max', verbose=1, patience=3, restore_best_weights=True)

history = model.fit(
        train_generator,
        epochs=epochs,
        validation_data=validation_generator,
        callbacks = [es]
)
```

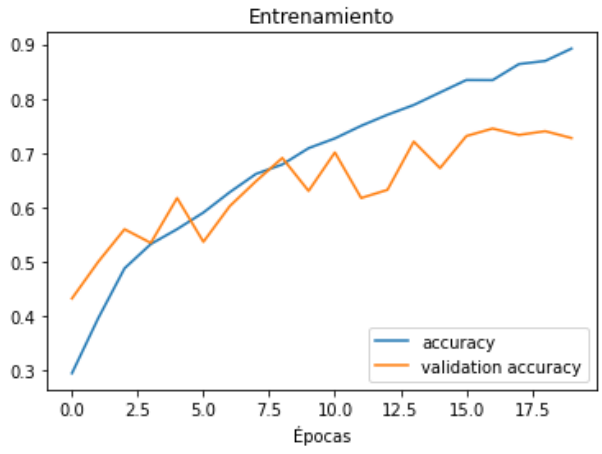
Ilustramos los resultados con las siguientes gráficas:

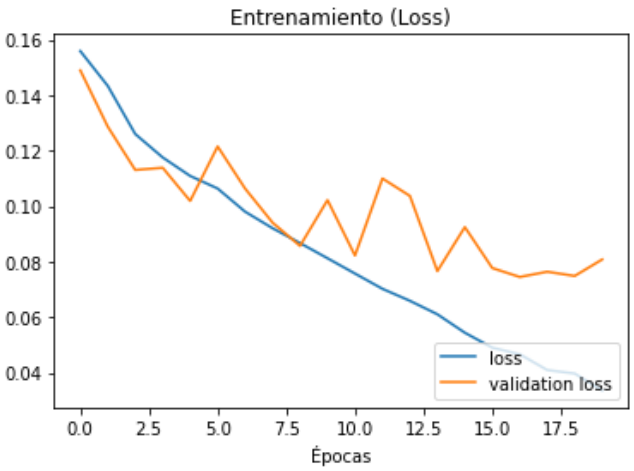
GRAFICA PERDIDA

In [44]:
```
plt.plot(history.history['loss'], label='loss')
plt.plot(history.history['val_loss'], label='validation loss')

plt.title('Entrenamiento perdida')
plt.xlabel('Épocas')
plt.legend(loc="lower right")
plt.show()
```

GRAFICA PRECISION

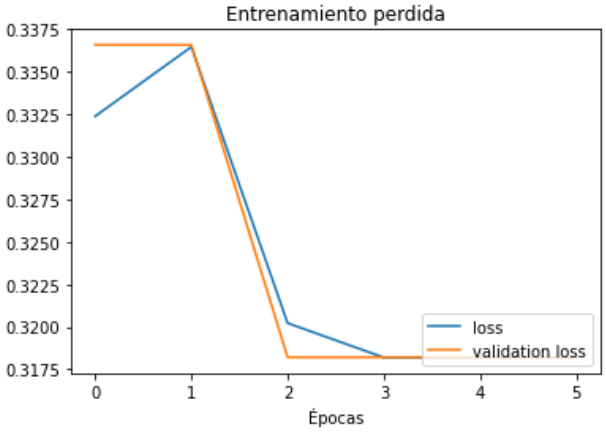
In [9]:
```
plt.plot(history.history['accuracy'], label='accuracy')
plt.plot(history.history['val_accuracy'], label='validation accuracy')

plt.title('Entrenamiento precision')
plt.xlabel('Épocas')
plt.legend(loc="lower right")

plt.show()
```

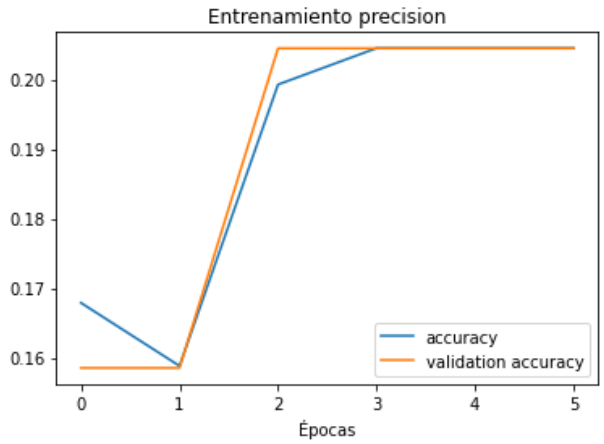




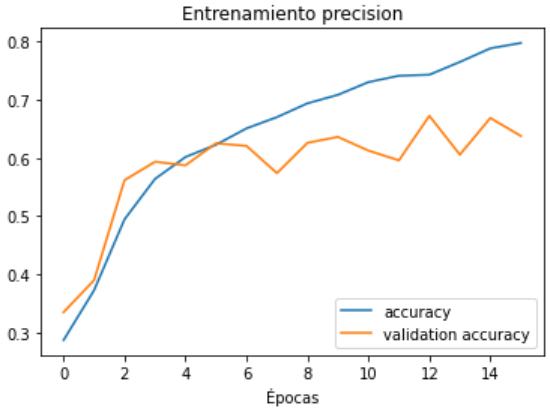
Después de haber realizado varias pruebas para alcanzar el accuracy citado anteriormente, comenzamos a variar los diferentes hiperparámetros, para ver si existía alguna relación entre ello, obteniendo los siguientes resultados:

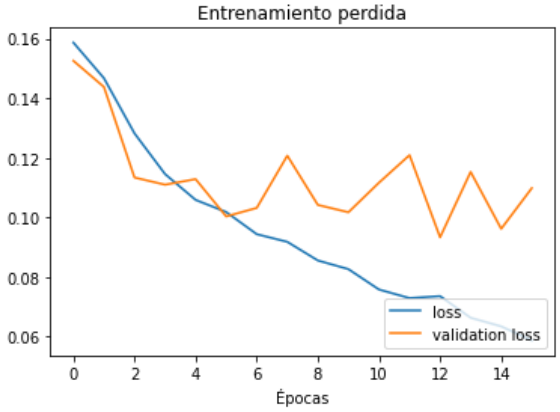
CASO 1: El cambio empleado fue usar la función de activación ELU



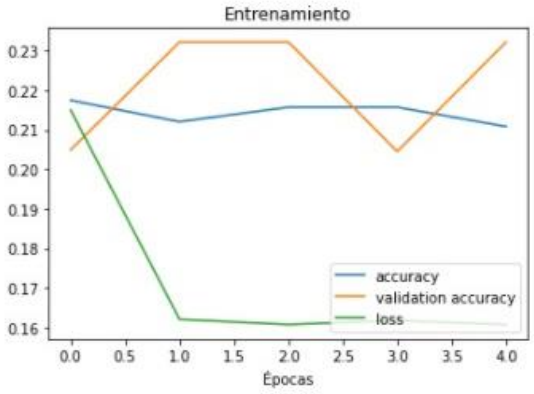


CASO 2: Se cambio el tamaño del *kernel* de 3x3 a 4x4, y entrenando en 100 épocas, aunque el entrenamiento se detuvo mucho antes de llegar a otros valores obtenidos previamente, debido al *EarlyStopping* puesto con una paciencia igual a 3.





CASO 3: El cambio empleado fue poner el batch size igual a 1, con 100 épocas



MATRIZ DE CONFUSION

```
In [11]: Y_pred = model.predict(validation_generator, 15)
         y_pred=np.argmax(Y_pred, axis=1)
         print('Matriz de confusion')
         print(confusion_matrix(validation_generator.classes, y_pred))

         Matriz de confusion
         [[ 52 112  80  66  80]
          [ 86 166 117  90 112]
          [ 67 151 100  77 109]
          [ 66 127  93  91 115]
          [ 68 140 113  79 103]]
```

Categorical cross entropy

La entropía cruzada describe la distancia entre la salida real y la esperada. Cuanto menor es el valor de la entropía, más cercanas serán las distribuciones de probabilidad.

Su funcionamiento se basa en la función de activación softmax, que es usada habitualmente como activación en la última capa de una red neuronal ya que su resultado se puede interpretar como una distribución de probabilidad.

$$\text{Loss} = \sum_{\substack{\text{output} \\ \text{size}}} - t_i \log(y_i)$$