UNIVERSIDAD NACIONAL DE INGENIERÍA

FACULTAD DE CIENCIAS

ESCUELA PROFESIONAL DE CIENCIA DE LA COMPUTACIÓN

2da parte del informe
TEMA:

Implementacion de Inteligencia Artificial en el reconomiento de aves migratorias de los Pantanos de Villa

CESAR MARTIN CRUZ SALAZAR

ALUMNOS:	**CÓDIGO:**
Jose Manuel Ravichagua Marin:	**20210086C**
Milagros Cristina Ruiz Arica:	**20210356K**
Linda Nicole Flores Salazar:	**20190587B**

Fecha de entrega : 30-12-2022

Lima - Perú
2022

Índice

1. Presentacion del Código

1.1. Importar librerias

[1]:
```
import sys
```

[2]:
```
import os
```

[3]:
```
from tensorflow import keras
```

[4]:
```
from tensorflow.keras import layers
```

[5]:
```
from tensorflow.keras.preprocessing.image import ImageDataGenerator
```

[6]:
```
from tensorflow.python.keras import optimizers
```

[7]:
```
from tensorflow.python.keras.models import Sequential
```

[8]:
```
from tensorflow.python.keras.layers import Dropout, Flatten, Dense,␣
↪Activation
```

[9]:
```
from tensorflow.python.keras.layers import Convolution2D, MaxPooling2D
```

[10]:
```
from tensorflow.python.keras import backend as K
```

[11]:
```
K.clear_session() #borrar lo anteiro desde cero
data_entrenamiento = './data/train'
data_validacion = './data/valid'
```

1.2. parametros

[12]:
```
epocas =  20 #numero de veces de iteracion
altura,longitud = 50,50 #tamaño de pixeles por imagen
batch_size = 32 #numero de imagnes a procesar
nb_train_samples= 4167 #una epoca tendra 4167 pasos
pasos_validacion = 72
filtrosConv1 =32 # profundidad
filtrosConv2 =64
tamano_filtro1 = (3,3) #alutra , long
tamano_filtro2 = (2,2)
tamano_pool = (2,2) #tamaño de filtro en maxpooling
clases = 12 #numero de especies de aves
learning_rate = 0.0004 #ajustes de la redd neural numero pequeño learnig␣
↪rate
```

1.3. preprocesamiento de imagenes

[13]:
```
entrenamiento_datagen = ImageDataGenerator(
    rescale =1./255, #mayor efificencia del entrenamiento
    shear_range = 0.2, #inclina las imagnes
    zoom_range = 0.3,#algunas hara zoom
    horizontal_flip = True #invertir imagenes
)
validacion_datagen = ImageDataGenerator(
    rescale=1./255
)
imagen_entrenamiento = entrenamiento_datagen.flow_from_directory(
    data_entrenamiento,
    target_size=(altura, longitud),
    batch_size=batch_size,
    class_mode='categorical')
```

```
Found 4167 images belonging to 12 classes.
```

[14]:
```
imagen_validacion = validacion_datagen.flow_from_directory(
    data_validacion,
    target_size=(altura, longitud),
    batch_size=batch_size,
    class_mode='categorical')
```

```
Found 71 images belonging to 12 classes.
```

1.4. crear la cnn la red neuronal convolucional

[15]:
```
cnn = Sequential() #cavrias capas apiladas entre ellas
#primera capa
cnn.add(Convolution2D(filtrosConv1,
                      tamano_filtro1,
                      padding = 'same',
                      input_shape=(altura,longitud,3),
                      activation='relu'))
cnn.add(MaxPooling2D(pool_size=tamano_pool))
```

[16]:
```
cnn.add(Convolution2D(filtrosConv2, tamano_filtro2, padding␣
 ↪="same",activation='relu'))
cnn.add(MaxPooling2D(pool_size=tamano_pool))
cnn.add(Flatten())
cnn.add(Dense(256, activation='relu'))
cnn.add(Dropout(0.5))
cnn.add(Dense(clases, activation='softmax'))
```

[17]:
```
import tensorflow as tf
```

[18]:
```
cnn.compile(loss='categorical_crossentropy',
            optimizer= 'adam',
            metrics=['accuracy'])
```

[19]:
```
cnn.summary() #resumen de la red neuronal
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 50, 50, 32)        896
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 25, 25, 32)        0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 25, 25, 64)        8256
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 12, 12, 64)        0
_________________________________________________________________
flatten (Flatten)            (None, 9216)              0
_________________________________________________________________
dense (Dense)                (None, 256)               2359552
_________________________________________________________________
dropout (Dropout)            (None, 256)               0
_________________________________________________________________
dense_1 (Dense)              (None, 12)                3084
=================================================================
Total params: 2,371,788
Trainable params: 2,371,788
Non-trainable params: 0
_________________________________________________________________
```

[20]:
```
#entrenar nuestra rede neuronal
modelo = cnn.fit(
    imagen_entrenamiento,
    epochs=epocas,
    validation_data=imagen_validacion,
    steps_per_epoch=nb_train_samples/batch_size,
    validation_steps=pasos_validacion/batch_size)
```

```
Epoch 1/20
117/130 [=========================>...] - ETA: 5s - loss: 1.9469 - accuracy:
0.3627

C:\Users\ANGELLO\anaconda3\lib\site-packages\PIL\Image.py:945: UserWarning:
Palette images with Transparency expressed in bytes should be converted to␣
 ↪RGBA
```

```
images
  warnings.warn(
130/130 [==============================] - 53s 402ms/step - loss: 1.9171 -
accuracy: 0.3765 - val_loss: 3.1228 - val_accuracy: 0.1268
Epoch 2/20
130/130 [==============================] - 49s 380ms/step - loss: 1.6258 -
accuracy: 0.4780 - val_loss: 2.4760 - val_accuracy: 0.1127
Epoch 3/20
130/130 [==============================] - 43s 334ms/step - loss: 1.4332 -
accuracy: 0.5349 - val_loss: 2.2080 - val_accuracy: 0.2958
Epoch 4/20
130/130 [==============================] - 38s 296ms/step - loss: 1.3130 -
accuracy: 0.5635 - val_loss: 1.9760 - val_accuracy: 0.3239
Epoch 5/20
130/130 [==============================] - 39s 304ms/step - loss: 1.2261 -
accuracy: 0.5865 - val_loss: 1.8242 - val_accuracy: 0.3944
Epoch 6/20
130/130 [==============================] - 40s 307ms/step - loss: 1.1577 -
accuracy: 0.6180 - val_loss: 1.5575 - val_accuracy: 0.4507
Epoch 7/20
130/130 [==============================] - 39s 300ms/step - loss: 1.0887 -
accuracy: 0.6338 - val_loss: 1.5089 - val_accuracy: 0.5070
Epoch 8/20
130/130 [==============================] - 41s 320ms/step - loss: 1.0293 -
accuracy: 0.6566 - val_loss: 1.4601 - val_accuracy: 0.4507
Epoch 9/20
130/130 [==============================] - 49s 379ms/step - loss: 0.9544 -
accuracy: 0.6842 - val_loss: 1.2949 - val_accuracy: 0.5352
Epoch 10/20
130/130 [==============================] - 50s 388ms/step - loss: 0.9315 -
accuracy: 0.6734 - val_loss: 1.1735 - val_accuracy: 0.5775
Epoch 11/20
130/130 [==============================] - 51s 398ms/step - loss: 0.8747 -
accuracy: 0.7048 - val_loss: 1.0830 - val_accuracy: 0.5775
Epoch 12/20
130/130 [==============================] - 66s 513ms/step - loss: 0.8371 -
accuracy: 0.7149 - val_loss: 0.9437 - val_accuracy: 0.6620
Epoch 13/20
130/130 [==============================] - 60s 462ms/step - loss: 0.7907 -
accuracy: 0.7391 - val_loss: 0.9243 - val_accuracy: 0.6620
Epoch 14/20
130/130 [==============================] - 48s 372ms/step - loss: 0.7414 -
accuracy: 0.7492 - val_loss: 0.7617 - val_accuracy: 0.7183
Epoch 15/20
130/130 [==============================] - 47s 362ms/step - loss: 0.7419 -
accuracy: 0.7478 - val_loss: 0.7963 - val_accuracy: 0.6901
```

```
Epoch 16/20
130/130 [==============================] - 48s 370ms/step - loss: 0.6767 -
accuracy: 0.7723 - val_loss: 0.7055 - val_accuracy: 0.7042
Epoch 17/20
130/130 [==============================] - 47s 366ms/step - loss: 0.6569 -
accuracy: 0.7785 - val_loss: 0.7910 - val_accuracy: 0.7042
Epoch 18/20
130/130 [==============================] - 50s 389ms/step - loss: 0.6237 -
accuracy: 0.7855 - val_loss: 0.7244 - val_accuracy: 0.7042
Epoch 19/20
130/130 [==============================] - 49s 381ms/step - loss: 0.6148 -
accuracy: 0.7898 - val_loss: 0.5307 - val_accuracy: 0.8310
Epoch 20/20
130/130 [==============================] - 46s 357ms/step - loss: 0.5634 -
accuracy: 0.8071 - val_loss: 0.4456 - val_accuracy: 0.8451
```

[23]:
```
target_dir = './modelo/'
if not os.path.exists(target_dir):
  os.mkdir(target_dir)
cnn.save('./modelo/modelo.h5')
cnn.save_weights('./modelo/pesos.h5')
```

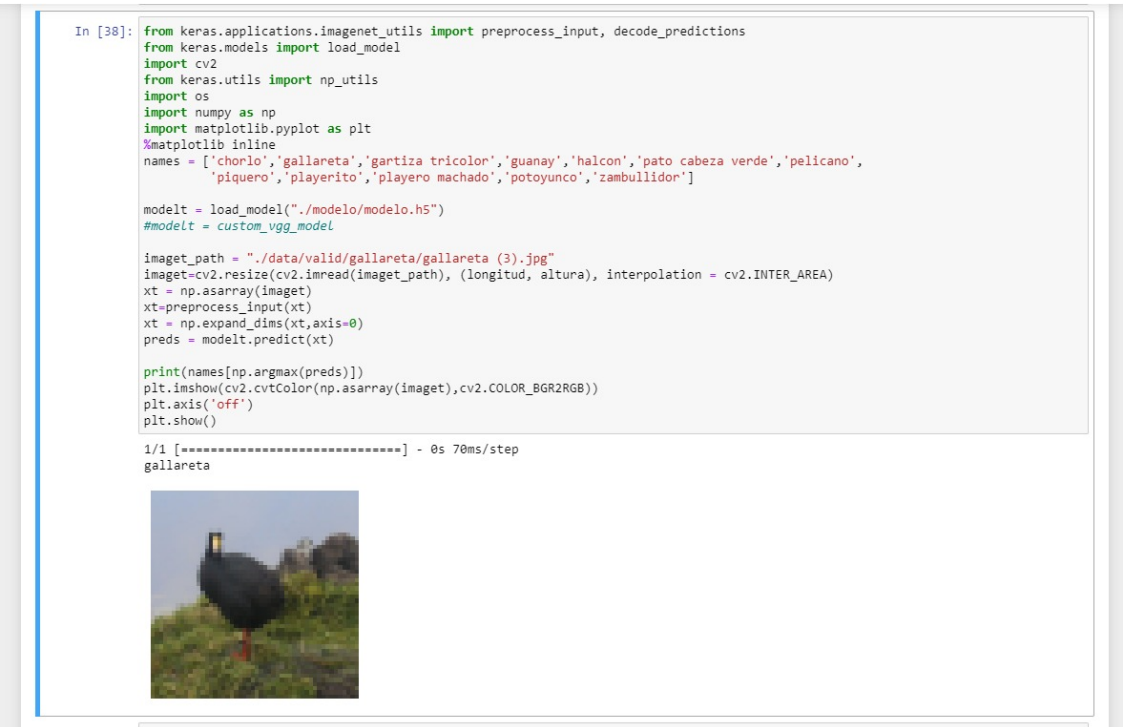
[67]:
```
from keras.applications.imagenet_utils import preprocess_input,␣
 ↪decode_predictions
from keras.models import load_model
import cv2
from keras.utils import np_utils
import os
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
names = ['chorlo','gallareta','gartiza tricolor','guanay','halcon','pato␣
 ↪cabeza verde','pelicano',
         'piquero','playerito','playero␣
 ↪machado','potoyunco','zambullidor']

modelt = load_model("./modelo/modelo.h5")
#modelt = custom_vgg_model

imaget_path = "./data/valid/gallareta/gallareta (4).jpg"
imaget=cv2.resize(cv2.imread(imaget_path), (longitud, altura),␣
 ↪interpolation = cv2.INTER_AREA)
xt = np.asarray(imaget)
xt=preprocess_input(xt)
xt = np.expand_dims(xt,axis=0)
preds = modelt.predict(xt)
```

```
print(names[np.argmax(preds)])
plt.imshow(cv2.cvtColor(np.asarray(imaget),cv2.COLOR_BGR2RGB))
plt.axis('off')
plt.show()
```

```
1/1 [==============================] - 0s 60ms/step
pato cabeza verde
```

2. Resultados

Primer resultado (válido):



Segundo resultado (válido) :

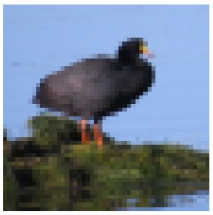
In [39]:
```
from keras.applications.imagenet_utils import preprocess_input, decode_predictions
from keras.models import load_model
import cv2
from keras.utils import np_utils
import os
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
names = ['chorlo','gallareta','gartiza tricolor','guanay','halcon','pato cabeza verde','pelicano',
         'piquero','playerito','playero machado','potoyunco','zambullidor']

modelt = load_model("./modelo/modelo.h5")
#modelt = custom_vgg_model

imaget_path = "./data/valid/gallareta/gallareta (4).jpg"
imaget=cv2.resize(cv2.imread(imaget_path), (longitud, altura), interpolation = cv2.INTER_AREA)
xt = np.asarray(imaget)
xt=preprocess_input(xt)
xt = np.expand_dims(xt,axis=0)
preds = modelt.predict(xt)

print(names[np.argmax(preds)])
plt.imshow(cv2.cvtColor(np.asarray(imaget),cv2.COLOR_BGR2RGB))
plt.axis('off')
plt.show()
```

```
1/1 [==============================] - 0s 78ms/step
gallareta
```

Tercer resultado (válido) :

In [40]:
```
from keras.applications.imagenet_utils import preprocess_input, decode_predictions
from keras.models import load_model
import cv2
from keras.utils import np_utils
import os
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
names = ['chorlo','gallareta','gartiza tricolor','guanay','halcon','pato cabeza verde','pelicano',
         'piquero','playerito','playero machado','potoyunco','zambullidor']

modelt = load_model("./modelo/modelo.h5")
#modelt = custom_vgg_model

imaget_path = "./data/valid/gallareta/gallareta (5).jpg"
imaget=cv2.resize(cv2.imread(imaget_path), (longitud, altura), interpolation = cv2.INTER_AREA)
xt = np.asarray(imaget)
xt=preprocess_input(xt)
xt = np.expand_dims(xt,axis=0)
preds = modelt.predict(xt)

print(names[np.argmax(preds)])
plt.imshow(cv2.cvtColor(np.asarray(imaget),cv2.COLOR_BGR2RGB))
plt.axis('off')
plt.show()
```

```
1/1 [==============================] - 0s 67ms/step
gallareta
```



Cuarto resultado (incorrecto):

```
In [36]: from keras.applications.imagenet_utils import preprocess_input, decode_predictions
         from keras.models import load_model
         import cv2
         from keras.utils import np_utils
         import os
         import numpy as np
         import matplotlib.pyplot as plt
         %matplotlib inline
         names = ['chorlo','gallareta','gartiza tricolor','guanay','halcon','pato cabeza verde','pelicano',
                  'piquero','playerito','playero machado','potoyunco','zambullidor']

         modelt = load_model("./modelo/modelo.h5")
         #modelt = custom_vgg_model

         imaget_path = "./data/valid/gallareta/gallareta (1).jpg"
         imaget=cv2.resize(cv2.imread(imaget_path), (longitud, altura), interpolation = cv2.INTER_AREA)
         xt = np.asarray(imaget)
         xt=preprocess_input(xt)
         xt = np.expand_dims(xt,axis=0)
         preds = modelt.predict(xt)

         print(names[np.argmax(preds)])
         plt.imshow(cv2.cvtColor(np.asarray(imaget),cv2.COLOR_BGR2RGB))
         plt.axis('off')
         plt.show()

         1/1 [==============================] - 0s 70ms/step
         pato cabeza verde
```

3. Observaciones

- A medida que aumenta el número de épocas aumenta el valor de precisión mientras que el valor de perdida disminuye.
- Un aumento en la cantidad de épocas generaría una mayor eficacia en nuestra red neuronal.
- El modelo usado para entrenar la red neuronal presenta un menor tiempo al entrenarla que otros tipos de modelos en los cuales toma horas para la misma cantidad de épocas.

4. Conclusiones

- Al hacer la recopilación de información de nuestro programa utilizado por las redes neuronales convolucionales se evidencia que muchas de ellas tienen el propósito dado de este proyecto(Modelo VGG16). Los motivos son por su baja complejidad computacional y su poca utilización de recursos de la imagen, lo hacen que tenga un tiempo de respuesta aceptable, y el reconocimiento sea de un 84.5 % de precisión.
- Con esto pretendemos a facilitar el reconocimiento de imágenes para especies de aves y el impacto de la contaminación que provoca que ciertas especies se encuentren en peligro de extinción por eso tratamos de informar a base de las inteligencias artificiales.
- Con lo construido y aplicado de nuestra red neuronal podemos ver alguna mejora por

la cantidad de información que podemos recibir cada año, asi la efectividad de aquel entrenamiento de nuestra aplicación pueda tener más impacto en nuestro objetivo, la cual es poder ser una herramienta util en la conservación de la vida silvestre y la proteccion de especies en peligro de extinción .