Universidad Internacional de Valencia

Proyecto de Programación Grupal

Dataset de 525 especies de pájaros

AUTORES

GABRIEL DÍAZ IRELAND, ALBERTO ENRIQUE GARCÍA DE CASTRO CRESPO, UNAI ARAMBARRI YEREGUI

ASIGNATURA

REDES NEURONALES Y DEEP LEARNING

PROFESORA

ROCÍO DEL AMOR DEL AMOR

07MIAR08_EntregaFinal

November 25, 2023

0.1 INTRODUCCIÓN

En esta actividad, el alumno debe evaluar y comparar dos estrategias para la clasificación de imágenes empleando el dataset asignado. El/La alumnx deberá resolver el reto proponiendo una solución válida basada en aprendizaje profundo, más concretamente en redes neuronales convolucionales (CNNs). Será indispensable que la solución propuesta siga el pipeline visto en clase para resolver este tipo de tareas de inteligencia artificial: 1. Carga del conjunto de datos 2. Inspección del conjunto de datos 3. Acondicionamiento del conjunto de datos 4. Desarrollo de la arquitectura de red neuronal y entrenamiento de la solución 5. Monitorización del proceso de entrenamiento para la toma de decisiones 6. Evaluación del modelo predictivo y planteamiento de la siguiente prueba experimental

Para resolver la actividad propuesta, organizammos los siguientes bloques.

1. Exploratory Data Analysis (EDA)
2. CNN "From Scratch"
3. Modelos por Transfer Learning: VGG16 | INCEPTIONV3
4. Conclusiones y Comentarios Finales.
5. Anexo de Experimentos.

0.2 EXPLORATOY DATA ANALYSIS

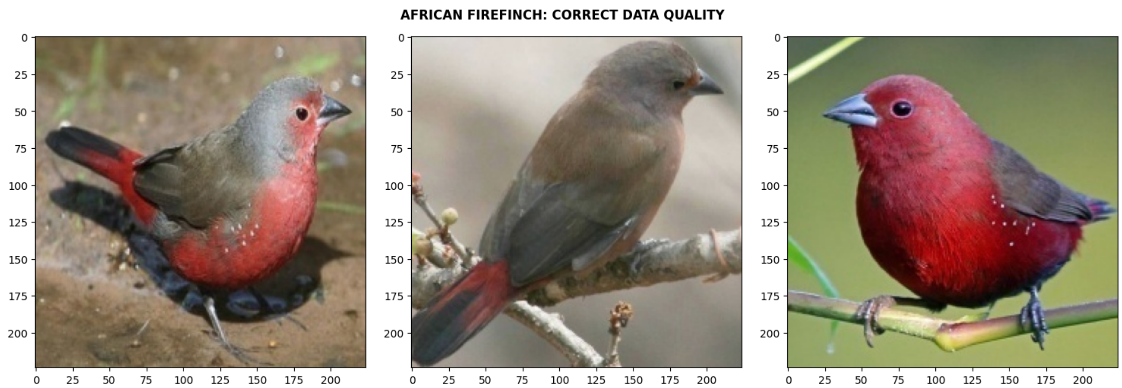
Antes de empezar con la modelización de redes CNN, utilizamos un apartado de Exploración de datos, con el objetivo de entender más nuestro dataset. Primero, Importamos las librerías finales necesarias para el proyecto y preparamos el enviroment de trabajo. También cargamos y revisamos los datos en una pequeña exploración visual, de tamaño, y de forma.

[29]:
```
# Importamos librerías necesarias
import numpy as np
import os
import random
import matplotlib.pyplot as plt
import cv2

import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
from tensorflow.keras import preprocessing
from tensorflow.keras.preprocessing import image_dataset_from_directory
from keras.preprocessing.image import ImageDataGenerator
```

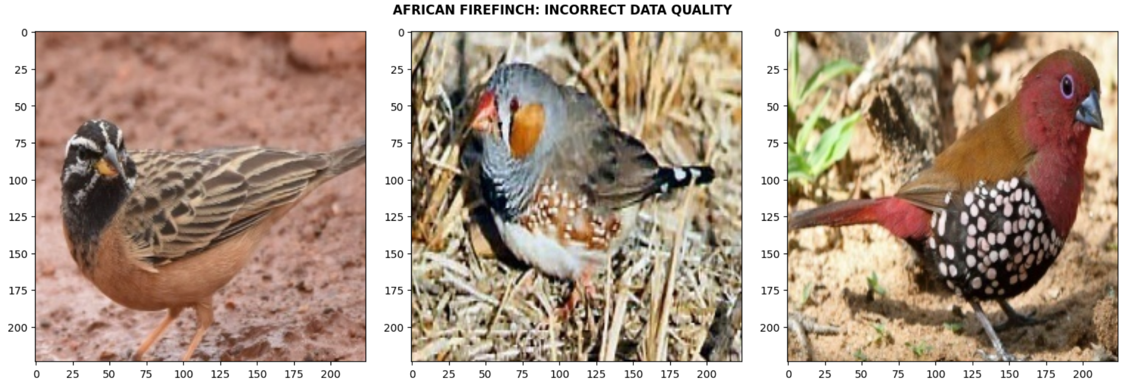
[30]:
```
# API keys de Kaggle para la descarga del dataset
os.environ['KAGGLE_USERNAME'] = 'alumno'
os.environ['KAGGLE_KEY'] = 'Key'
```

[31]:
```
# Descargamos el dataset y unzip
import kaggle

kaggle.api.dataset_download_files('gpiosenka/100-bird-species', unzip=True)
```

[19]:
```
# De manera visual y descargando algunas clases, exploramos las imágenes␣
 ↪disponibles, intentando sacar una conclusión estimada de la calidad del␣
 ↪dataset.
import matplotlib.pyplot as plt
import numpy as np
import cv2

idx = "017"
img1 = cv2.imread('kaggle.birds/train/AFRICAN FIREFINCH/' + str(idx) + '.jpg',␣
 ↪cv2.COLOR_BGR2RGB)
img1 = cv2.cvtColor(img1, cv2.COLOR_BGR2RGB)

idx = "051"
img2 = cv2.imread('kaggle.birds/train/AFRICAN FIREFINCH/' + str(idx) + '.jpg',␣
 ↪cv2.COLOR_BGR2RGB)
img2 = cv2.cvtColor(img2, cv2.COLOR_BGR2RGB)

idx = "006"
img3 = cv2.imread('kaggle.birds/train/AFRICAN FIREFINCH/' + str(idx) + '.jpg',␣
 ↪cv2.COLOR_BGR2RGB)
img3 = cv2.cvtColor(img3, cv2.COLOR_BGR2RGB)




fig, axes = plt.subplots(1, 3, figsize=(15, 5))


axes[0].imshow(img1)
axes[1].imshow(img2)
axes[2].imshow(img3)

fig.suptitle('AFRICAN FIREFINCH: CORRECT DATA QUALITY', fontweight='bold',␣
 ↪color='black')
fig.tight_layout()
plt.show()
```



[20]:
```
idx = 100
img1 = cv2.imread('kaggle.birds/train/AFRICAN FIREFINCH/'  + str(idx) + '.jpg',␣
 ↪cv2.COLOR_BGR2RGB)
img1 = cv2.cvtColor(img1, cv2.COLOR_BGR2RGB)

idx = 115
img2 = cv2.imread('kaggle.birds/train/AFRICAN FIREFINCH/'  + str(idx) + '.jpg',␣
 ↪cv2.COLOR_BGR2RGB)
img2 = cv2.cvtColor(img2, cv2.COLOR_BGR2RGB)

idx = 128
img3 = cv2.imread('kaggle.birds/train/AFRICAN FIREFINCH/'  + str(idx) + '.jpg',␣
 ↪cv2.COLOR_BGR2RGB)
img3 = cv2.cvtColor(img3, cv2.COLOR_BGR2RGB)

fig, axes = plt.subplots(1, 3, figsize=(15, 5))

axes[0].imshow(img1)
axes[1].imshow(img2)
axes[2].imshow(img3)

fig.suptitle('AFRICAN FIREFINCH: INCORRECT DATA QUALITY', fontweight='bold',␣
 ↪color='black')
fig.tight_layout()
plt.show()
```



Tras revisar las colecciones de imágenes, encontramos pájaros mal clasificados, no correspondientes a la especie etiquetada. Se observa esto en más de una clase (No solo en "AFRICAN FIREFINCH" conocido en españo como jilguero africano. En el ejemplo, podemos ver que están etiquetados como AFRICAN FIREFINCH un escribano pechirrojo canela (izquierda), un jilguero australiano (centro) y una estrilda de gorja rosada (derecha).

Esto, justifica el uso de técnicas más adelante como son la clusterización en grupos de imágenes y el data augmentation.

[32]:
```
# Definición de directorios
train_directory = 'train'
test_directory = 'test'
val_directory = 'valid'
```

[ ]:
```
# Númeramos categorías
categories = os.listdir(train_directory)
print(str(len(categories)),'CATEGORIES are ', categories)

category_count = len(categories)
```

```
525 CATEGORIES are  ['CARMINE BEE-EATER', 'IVORY BILLED ARACARI', 'FRIGATE',
'COPPERSMITH BARBET', 'GOLDEN CHEEKED WARBLER', 'GREAT TINAMOU', 'NORTHERN
SHOVELER', 'ASIAN OPENBILL STORK', 'EMERALD TANAGER', 'AMERICAN GOLDFINCH',
'BARROWS GOLDENEYE', 'ELEGANT TROGON', 'DUSKY LORY', 'CHUKAR PARTRIDGE', 'COMMON
POORWILL', 'BLONDE CRESTED WOODPECKER', 'AZARAS SPINETAIL', 'OSTRICH', 'MALAGASY
WHITE EYE', 'BORNEAN PHEASANT', 'FAIRY BLUEBIRD', 'COLLARED ARACARI', 'GREATER
PRAIRIE CHICKEN', 'GILDED FLICKER', 'DOWNY WOODPECKER', 'ALEXANDRINE PARAKEET',
'NORTHERN BEARDLESS TYRANNULET', 'AMETHYST WOODSTAR', 'LARK BUNTING', 'CUBAN
TODY', 'DEMOISELLE CRANE', 'GOULDIAN FINCH', 'WATTLED CURASSOW', 'ANNAS
HUMMINGBIRD', 'PURPLE SWAMPHEN', 'HELMET VANGA', 'COCKATOO', 'COCK OF THE
ROCK', 'GOLDEN PARAKEET', 'IVORY GULL', 'ROCK DOVE', 'VENEZUELIAN TROUPIAL',
'HARLEQUIN QUAIL', 'NORTHERN FULMAR', 'SCARLET FACED LIOCICHLA', 'RED TAILED
THRUSH', 'STRIATED CARACARA', 'ORANGE BREASTED TROGON', 'CAPE MAY WARBLER',
'PEACOCK', 'GREY CUCKOOSHRIKE', 'CRANE HAWK', 'BLACK HEADED CAIQUE', 'FLAME
```

BOWERBIRD', 'BURCHELLS COURSER', 'DUSKY ROBIN', 'NICOBAR PIGEON', 'DOUBLE BARRED FINCH', 'EASTERN BLUEBIRD', 'VULTURINE GUINEAFOWL', 'CHESTNET BELLIED EUPHONIA', 'CINNAMON TEAL', 'PHILIPPINE EAGLE', 'EASTERN YELLOW ROBIN', 'BEARDED BELLBIRD', 'RED FACED CORMORANT', 'BARN OWL', 'PINK ROBIN', 'WHITE EARED HUMMINGBIRD', 'CHINESE BAMBOO PARTRIDGE', 'LAUGHING GULL', 'NORTHERN CARDINAL', 'FLAME TANAGER', 'WATTLED LAPWING', 'ASIAN GREEN BEE EATER', 'COMMON HOUSE MARTIN', 'BLACK THROATED BUSHTIT', 'BLACK VENTED SHEARWATER', 'HOUSE SPARROW', 'OCELLATED TURKEY', 'BLUE THROATED PIPING GUAN', 'AMERICAN PIPIT', 'SPOTTED CATBIRD', 'CAATINGA CACHOLOTE', 'MAGPIE GOOSE', 'BLUE GROUSE', 'LILAC ROLLER', 'GREEN WINGED DOVE', 'AVADAVAT', 'STRIPPED SWALLOW', 'ASHY STORM PETREL', 'GILA WOODPECKER', 'GREEN MAGPIE', 'AZURE TANAGER', 'BEARDED REEDLING', 'JANDAYA PARAKEET', 'RED CROSSBILL', 'SMITHS LONGSPUR', 'JABIRU', 'CRESTED CARACARA', 'WOODLAND KINGFISHER', 'DARK EYED JUNCO', 'COPPERY TAILED COUCAL', 'AMERICAN COOT', 'BALD EAGLE', 'BROWN NOODY', 'CRIMSON CHAT', 'RED BELLIED PITTA', 'WALL CREAPER', 'BLACK-CAPPED CHICKADEE', 'PARAKETT AUKLET', 'EURASIAN MAGPIE', 'D-ARNAUDS BARBET', 'TURQUOISE MOTMOT', 'FASCIATED WREN', 'OKINAWA RAIL', 'RED TAILED HAWK', 'CLARKS GREBE', 'EASTERN BLUEBONNET', 'HIMALAYAN MONAL', 'ABYSSINIAN GROUND HORNBILL', 'SCARLET IBIS', 'GREEN BROADBILL', 'ALBATROSS', 'GREAT JACAMAR', 'SUNBITTERN', 'ABBOTTS BOOBY', 'AUCKLAND SHAQ', 'FIERY MINIVET', 'BLUE HERON', 'CHESTNUT WINGED CUCKOO', 'RED LEGGED HONEYCREEPER', 'AMERICAN KESTREL', 'DUNLIN', 'RUDY KINGFISHER', 'HARLEQUIN DUCK', 'WHITE THROATED BEE EATER', 'GLOSSY IBIS', 'CALIFORNIA QUAIL', 'RED BROWED FINCH', 'BLACK COCKATO', 'AFRICAN OYSTER CATCHER', 'FOREST WAGTAIL', 'PHAINOPEPLA', 'GREAT POTOO', 'INLAND DOTTEREL', 'EASTERN GOLDEN WEAVER', 'FAIRY PENGUIN', 'PURPLE FINCH', 'GAMBELS QUAIL', 'GOLDEN EAGLE', 'GO AWAY BIRD', 'DARJEELING WOODPECKER', 'GREATER PEWEE', 'FAIRY TERN', 'CRESTED NUTHATCH', 'RAZORBILL', 'AFRICAN EMERALD CUCKOO', 'GRANDALA', 'GREATOR SAGE GROUSE', 'BUSH TURKEY', 'CRAB PLOVER', 'ANDEAN GOOSE', 'CRESTED OROPENDOLA', 'HIMALAYAN BLUETAIL', 'CLARKS NUTCRACKER', 'BLACK SKIMMER', 'NORTHERN GOSHAWK', 'RED WISKERED BULBUL', 'AMERICAN FLAMINGO', 'VICTORIA CROWNED PIGEON', 'RED FACED WARBLER', 'RAINBOW LORIKEET', 'FIRE TAILLED MYZORNIS', 'BROWN CREPPER', 'TOUCHAN', 'RUFOUS TREPE', 'RED WINGED BLACKBIRD', 'DAURIAN REDSTART', 'CRESTED AUKLET', 'POMARINE JAEGER', 'FRILL BACK PIGEON', 'RED BEARDED BEE EATER', 'RED KNOT', 'COMMON IORA', 'CRIMSON SUNBIRD', 'MARABOU STORK', 'DOUBLE EYED FIG PARROT', 'PAINTED BUNTING', 'GRAY KINGBIRD', 'BAIKAL TEAL', 'LAZULI BUNTING', 'PLUSH CRESTED JAY', 'MALLARD DUCK', 'BUFFLEHEAD', 'BLUE GROSBEAK', 'BORNEAN LEAFBIRD', 'AMERICAN WIGEON', 'SHOEBILL', 'GREAT XENOPS', 'BLUE COAU', 'BANDED STILT', 'TEAL DUCK', 'HOODED MERGANSER', 'PYRRHULOXIA', 'JAVA SPARROW', 'CANVASBACK', 'MALABAR HORNBILL', 'CHINESE POND HERON', 'AZURE BREASTED PITTA', 'AUSTRAL CANASTERO', 'GREAT GRAY OWL', 'GREY HEADED FISH EAGLE', 'BIRD OF PARADISE', 'CHATTERING LORY', 'ANDEAN LAPWING', 'PUNA TEAL', 'ORNATE HAWK EAGLE', 'AMERICAN ROBIN', 'BAY-BREASTED WARBLER', 'CASPIAN TERN', 'RED FODY', 'YELLOW BELLIED FLOWERPECKER', 'SORA', 'BALTIMORE ORIOLE', 'TRICOLORED BLACKBIRD', 'GOLDEN CHLOROPHONIA', 'HEPATIC TANAGER', 'KIWI', 'NORTHERN JACANA', 'KING EIDER', 'COMMON LOON', 'BALD IBIS', 'SWINHOES PHEASANT', 'VIOLET CUCKOO', 'HOUSE FINCH', 'RED BILLED TROPICBIRD', 'BULWERS PHEASANT', 'SATYR TRAGOPAN', 'SHORT BILLED DOWITCHER', 'MASKED LAPWING', 'VIOLET GREEN SWALLOW', 'WILLOW PTARMIGAN', 'GOLDEN PHEASANT', 'RUBY THROATED HUMMINGBIRD', 'MILITARY MACAW', 'NOISY FRIARBIRD', 'CAMPO FLICKER',

'CHUCAO TAPACULO', 'PYGMY KINGFISHER', 'INDIAN BUSTARD', 'HARPY EAGLE', 'TAIWAN MAGPIE', 'EASTERN ROSELLA', 'CAPE LONGCLAW', 'RED SHOULDERED HAWK', 'OYSTER CATCHER', 'SNOW PARTRIDGE', 'WOOD THRUSH', 'WHITE NECKED RAVEN', 'ROUGH LEG BUZZARD', 'WHITE CRESTED HORNBILL', 'HOATZIN', 'NORTHERN GANNET', 'SNOWY OWL', 'CINNAMON FLYCATCHER', 'TASMANIAN HEN', 'AZURE JAY', 'SCARLET MACAW', 'DALMATIAN PELICAN', 'EASTERN WIP POOR WILL', 'ANTBIRD', 'BLACK VULTURE', 'WHITE TAILED TROPIC', 'COMMON FIRECREST', 'RED NAPED TROGON', 'VIOLET TURACO', 'GREY HEADED CHACHALACA', 'ZEBRA DOVE', 'CRESTED WOOD PARTRIDGE', 'IBISBILL', 'MERLIN', 'WILD TURKEY', 'GREAT KISKADEE', 'BLUE MALKOHA', 'AMERICAN AVOCET', 'GRAY CATBIRD', 'BAND TAILED GUAN', 'BLACK NECKED STILT', 'BOBOLINK', 'JAPANESE ROBIN', 'STRIPPED MANAKIN', 'ALPINE CHOUGH', 'ANTILLEAN EUPHONIA', 'WILSONS BIRD OF PARADISE', 'BLUE GRAY GNATCATCHER', 'SNOW GOOSE', 'BLACK BAZA', 'CRESTED COUA', 'BEARDED BARBET', 'CURL CRESTED ARACURI', 'SQUACCO HERON', 'ASIAN DOLLARD BIRD', 'FAN TAILED WIDOW', 'ECUADORIAN HILLSTAR', 'CANARY', 'INDIAN PITTA', 'ROSEATE SPOONBILL', 'RING-NECKED PHEASANT', 'EASTERN TOWEE', 'OILBIRD', 'BORNEAN BRISTLEHEAD', 'NORTHERN MOCKINGBIRD', 'HORNED GUAN', 'SCARLET CROWNED FRUIT DOVE', 'PATAGONIAN SIERRA FINCH', 'TURKEY VULTURE', 'CRESTED KINGFISHER', 'LOONEY BIRDS', 'AFRICAN FIREFINCH', 'BLOOD PHEASANT', 'INDIAN VULTURE', 'GURNEYS PITTA', 'PARADISE TANAGER', 'WHITE BREASTED WATERHEN', 'CERULEAN WARBLER', 'AMERICAN DIPPER', 'VERMILION FLYCATHER', 'LITTLE AUK', 'OVENBIRD', 'VERDIN', 'IMPERIAL SHAQ', 'EURASIAN BULLFINCH', 'CEDAR WAXWING', 'MALEO', 'QUETZAL', 'BLACK-THROATED SPARROW', 'HAWAIIAN GOOSE', 'GUINEA TURACO', 'HORNED SUNGEM', 'SANDHILL CRANE', 'CALIFORNIA CONDOR', 'CHARA DE COLLAR', 'AZURE TIT', 'JOCOTOCO ANTPITTA', 'KOOKABURRA', 'AMERICAN REDSTART', 'INCA TERN', 'LIMPKIN', 'BLACK AND YELLOW BROADBILL', 'RUBY CROWNED KINGLET', 'JACK SNIPE', 'ANHINGA', 'HAWFINCH', 'YELLOW BREASTED CHAT', 'STORK BILLED KINGFISHER', 'BELTED KINGFISHER', 'ASIAN CRESTED IBIS', 'ENGGANO MYNA', 'GOLD WING WARBLER', 'COLLARED CRESCENTCHEST', 'ARARIPE MANAKIN', 'HOOPOES', 'BLACK-NECKED GREBE', 'BANDED PITA', 'MANGROVE CUCKOO', 'VARIED THRUSH', 'MYNA', 'NORTHERN RED BISHOP', 'HAMERKOP', 'SNOWY SHEATHBILL', 'CALIFORNIA GULL', 'GREAT ARGUS', 'EASTERN MEADOWLARK', 'VEERY', 'CRESTED SHRIKETIT', 'CAPUCHINBIRD', 'MASKED BOOBY', 'ROADRUNNER', 'GOLDEN BOWER BIRD', 'KING VULTURE', 'RUFUOS MOTMOT', 'BLACKBURNIAM WARBLER', 'MCKAYS BUNTING', 'AFRICAN CROWNED CRANE', 'BLACK FACED SPOONBILL', 'LESSER ADJUTANT', 'CRESTED FIREBACK', 'BREWERS BLACKBIRD', 'IBERIAN MAGPIE', 'APAPANE', 'TIT MOUSE', 'HYACINTH MACAW', 'WHITE CHEEKED TURACO', 'BAR-TAILED GODWIT', 'APOSTLEBIRD', 'CRESTED SERPENT EAGLE', 'INDIGO FLYCATCHER', 'BLACK TAIL CRAKE', 'TOWNSENDS WARBLER', 'TAILORBIRD', 'ABBOTTS BABBLER', 'ROSY FACED LOVEBIRD', 'NORTHERN FLICKER', 'GRAY PARTRIDGE', 'ROYAL FLYCATCHER', 'SNOWY PLOVER', 'AFRICAN PIED HORNBILL', 'SPANGLED COTINGA', 'BARRED PUFFBIRD', 'ALTAMIRA YELLOWTHROAT', 'BROWN HEADED COWBIRD', 'EUROPEAN TURTLE DOVE', 'RED HEADED DUCK', 'KAGU', 'SAYS PHOEBE', 'GYRFALCON', 'GUINEAFOWL', 'VISAYAN HORNBILL', 'PEREGRINE FALCON', 'BARN SWALLOW', 'BLUE THROATED TOUCANET', 'GREY PLOVER', 'SPLENDID WREN', 'ELLIOTS PHEASANT', 'CHIPPING SPARROW', 'GOLDEN PIPIT', 'CREAM COLORED WOODPECKER', 'BLUE DACNIS', 'RUDDY SHELDUCK', 'BRANDT CORMARANT', 'ORIENTAL BAY OWL', 'SCARLET TANAGER', 'KILLDEAR', 'PALM NUT VULTURE', 'SNOWY EGRET', 'CAPPED HERON', 'COMMON GRACKLE', 'SPOTTED WHISTLING DUCK', 'SURF SCOTER', 'MOURNING DOVE', 'PUFFIN', 'CASSOWARY', 'LONG-EARED OWL', 'CROW', 'ROSE BREASTED COCKATOO', 'WOOD DUCK', 'EARED PITA', 'FIORDLAND

```
PENGUIN', 'COMMON STARLING', 'AMERICAN BITTERN', 'NORTHERN PARULA', 'EVENING
GROSBEAK', 'CABOTS TRAGOPAN', 'BLACK THROATED HUET', 'INDIGO BUNTING', 'ASHY
THRUSHBIRD', 'VIOLET BACKED STARLING', 'YELLOW HEADED BLACKBIRD', 'HORNED LARK',
'MIKADO  PHEASANT', 'MASKED BOBWHITE', 'TRUMPTER SWAN', 'BALI STARLING',
'WHIMBREL', 'BLACK SWAN', 'SPOON BILED SANDPIPER', 'ALBERTS TOWHEE', 'PALILA',
'OSPREY', 'TAWNY FROGMOUTH', 'BLACK THROATED WARBLER', 'DOUBLE BRESTED
CORMARANT', 'BLACK BREASTED PUFFBIRD', 'MALACHITE KINGFISHER', 'SAMATRAN
THRUSH', 'RUFOUS KINGFISHER', 'REGENT BOWERBIRD', 'CUBAN TROGON', 'ANDEAN
SISKIN', 'CAPE GLOSSY STARLING', 'CAPE ROCK THRUSH', 'STRIPED OWL', 'CINNAMON
ATTILA', 'JACOBIN PIGEON', 'BROWN THRASHER', 'EURASIAN GOLDEN ORIOLE', 'GREEN
JAY', 'TAKAHE', 'EMPEROR PENGUIN', 'INDIAN ROLLER', 'BANANAQUIT', 'WRENTIT',
'YELLOW CACIQUE', 'WHITE BROWED CRAKE', 'CACTUS WREN', 'PURPLE MARTIN', 'LUCIFER
HUMMINGBIRD', 'BANDED BROADBILL', 'ROSE BREASTED GROSBEAK', 'KNOB BILLED DUCK',
'GANG GANG COCKATOO', 'UMBRELLA BIRD', 'MANDRIN DUCK', 'PARUS MAJOR', 'SUPERB
STARLING', 'GROVED BILLED ANI', 'AUSTRALASIAN FIGBIRD', 'PURPLE GALLINULE',
'TREE SWALLOW', 'EGYPTIAN GOOSE', 'STEAMER DUCK', 'LOGGERHEAD SHRIKE', 'EMU',
'SAND MARTIN', 'AFRICAN PYGMY GOOSE', 'KAKAPO', 'TROPICAL KINGBIRD', 'ANIANIAU',
'EUROPEAN GOLDFINCH', 'SRI LANKA BLUE MAGPIE', 'IWI', 'RED HEADED WOODPECKER',
'BLACK FRANCOLIN', 'ORANGE BRESTED BUNTING']
```

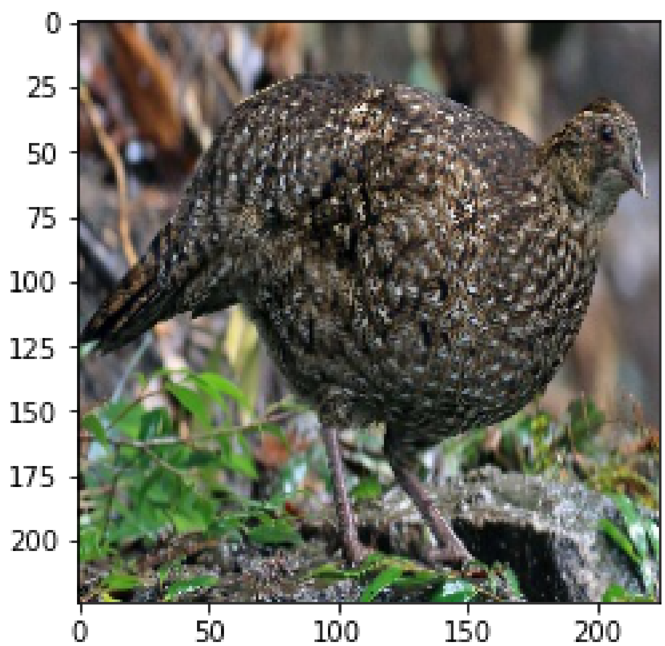
[34]:

```python
# Continuamos con la creación de generadores de datos de imágenes para
 ↪entrenamiento, validación y prueba y aplicación de la ampliación de datos
 ↪con transformaciones específicas solo al conjunto de entrenamiento
augmented_gen = ImageDataGenerator(
    rescale=1./255,
    rotation_range=40,
    width_shift_range=0.2,
    height_shift_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True,
    fill_mode='nearest')

general_datagen = ImageDataGenerator(rescale = 1./255) # for training,
 ↪validation and testing data

train_generator = general_datagen.flow_from_directory(
    train_directory,
    target_size = (224, 224),
    batch_size = 32
)
valid_generator = general_datagen.flow_from_directory(
    val_directory,
    target_size = (224, 224),
    batch_size = 32
)
test_generator = general_datagen.flow_from_directory(
    test_directory,
```

```
    target_size = (224, 224),
    batch_size = 32
)
```

```
Found 84635 images belonging to 525 classes.
Found 2625 images belonging to 525 classes.
Found 2625 images belonging to 525 classes.
```

[35]:
```
# Método para visualizar una imagen aleatoria
def plot_image(generator):
    images_in_batch = next(generator) # images_in_batch retornará (batch_size,␣
 ↪height, width, n_channels)
    img = images_in_batch[0][0] # img retornará (height, width, n_chennels)

    plt.imshow(img)

plot_image(train_generator)
```



[22]:
```
#Revisamos características generales de la imagen

print(len(img)) ## Cada imagen contiene 244 columnas.
print(len(img[1])) # ECada imagen contiene 244 filas.
print(len(img[1][1])) # Cada imagen contiene 3 valores, uno para cada banda.
print(np.max(img))
print(type(img[1][1][1]))
```

```
224
224
3
255
<class 'numpy.uint8'>
```

[36]:
```
# Mostramos el número de grupos (o lotes) en los generadores de entrenamiento y␣
↪validación, proporcionando una visión de cuántos pasos por época
train_groups = len(train_generator)
valid_groups = len(valid_generator) # validation_step

print(f"Train groups: {train_groups}")
print(f"Validation groups: {valid_groups}")
```

```
Train groups: 2645
Validation groups: 83
```

0.3 CNN FROM SCRATCH

[38]:
```
# Al principio el modelo solo tenía dos bloques que gradualmente pasaban de 64␣
↪a 128 filtros y daban una precisión baja del 55%. Con dropouts, la␣
↪normalización por lotes, configurando los filtros a 64, y añadiendo otro␣
↪bloque y cambios en la capa densa antes de la salida notamos cambios␣
↪significativos
# Disponemos de estos experimentos en el apartado 5.Anexo. para consulta. Aquí,␣
↪vemos el la arquitectura y resultado final.
keras.backend.clear_session()

inputs = keras.Input(shape = (224, 224, 3))

x = layers.Conv2D(filters = 64, kernel_size = 3, padding = "same", use_bias =␣
↪False)(inputs)
x = layers.BatchNormalization()(x)
x = layers.Activation("relu")(x)

x = layers.Conv2D(filters = 64, kernel_size = 3, padding = "valid", use_bias =␣
↪False)(x)
x = layers.BatchNormalization()(x)
x = layers.Activation("relu")(x)

x = layers.MaxPooling2D(pool_size = 2)(x)
x = layers.BatchNormalization()(x)
x = layers.Dropout(0.5)(x)


x = layers.Conv2D(filters = 64, kernel_size = 3, padding = "same", use_bias =␣
↪False)(x)
x = layers.BatchNormalization()(x)
```

```
x = layers.Activation("relu")(x)

x = layers.Conv2D(filters = 64, kernel_size = 3, padding = "same", use_bias =␣
 ↪False)(x)
x = layers.BatchNormalization()(x)
x = layers.Activation("relu")(x)

x = layers.MaxPooling2D(pool_size = 2)(x)
x = layers.BatchNormalization()(x)
x = layers.Dropout(0.5)(x)


x = layers.Conv2D(filters = 64, kernel_size = 3, padding = "same", use_bias =␣
 ↪False)(x)
x = layers.BatchNormalization()(x)
x = layers.Activation("relu")(x)

x = layers.Conv2D(filters = 64, kernel_size = 3, padding = "same", use_bias =␣
 ↪False)(x)
x = layers.BatchNormalization()(x)
x = layers.Activation("relu")(x)

x = layers.MaxPooling2D(pool_size = 2)(x)
x = layers.BatchNormalization()(x)
x = layers.Dropout(0.5)(x)


x = layers.Conv2D(filters = 64, kernel_size = 3, padding = "same", use_bias =␣
 ↪False)(x)
x = layers.BatchNormalization()(x)
x = layers.Activation("relu")(x)


x = layers.Flatten()(x)


x = layers.Dense(512)(x)
x = layers.BatchNormalization()(x)
x = layers.Activation("relu")(x)
x = layers.Dropout(0.5)(x)

outputs = layers.Dense(category_count, activation = "softmax")(x)

base_model = keras.Model(inputs, outputs)

base_model.summary()
```

```
Model: "model"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         [(None, 224, 224, 3)]     0
_________________________________________________________________
conv2d (Conv2D)              (None, 224, 224, 64)      1728
_________________________________________________________________
batch_normalization (BatchNo (None, 224, 224, 64)      256
_________________________________________________________________
activation (Activation)      (None, 224, 224, 64)      0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 222, 222, 64)      36864
_________________________________________________________________
batch_normalization_1 (Batch (None, 222, 222, 64)      256
_________________________________________________________________
activation_1 (Activation)    (None, 222, 222, 64)      0
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 111, 111, 64)      0
_________________________________________________________________
batch_normalization_2 (Batch (None, 111, 111, 64)      256
_________________________________________________________________
dropout (Dropout)            (None, 111, 111, 64)      0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 111, 111, 64)      36864
_________________________________________________________________
batch_normalization_3 (Batch (None, 111, 111, 64)      256
_________________________________________________________________
activation_2 (Activation)    (None, 111, 111, 64)      0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 111, 111, 64)      36864
_________________________________________________________________
batch_normalization_4 (Batch (None, 111, 111, 64)      256
_________________________________________________________________
activation_3 (Activation)    (None, 111, 111, 64)      0
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 55, 55, 64)        0
_________________________________________________________________
batch_normalization_5 (Batch (None, 55, 55, 64)        256
_________________________________________________________________
dropout_1 (Dropout)          (None, 55, 55, 64)        0
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 55, 55, 64)        36864
_________________________________________________________________
batch_normalization_6 (Batch (None, 55, 55, 64)        256
_________________________________________________________________
activation_4 (Activation)    (None, 55, 55, 64)        0
_________________________________________________________________
```

```
conv2d_5 (Conv2D)            (None, 55, 55, 64)        36864
_________________________________________________________________
batch_normalization_7 (Batch (None, 55, 55, 64)        256
_________________________________________________________________
activation_5 (Activation)    (None, 55, 55, 64)        0
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 27, 27, 64)        0
_________________________________________________________________
batch_normalization_8 (Batch (None, 27, 27, 64)        256
_________________________________________________________________
dropout_2 (Dropout)          (None, 27, 27, 64)        0
_________________________________________________________________
conv2d_6 (Conv2D)            (None, 27, 27, 64)        36864
_________________________________________________________________
batch_normalization_9 (Batch (None, 27, 27, 64)        256
_________________________________________________________________
activation_6 (Activation)    (None, 27, 27, 64)        0
_________________________________________________________________
flatten (Flatten)            (None, 46656)             0
_________________________________________________________________
dense (Dense)                (None, 512)               23888384
_________________________________________________________________
batch_normalization_10 (Batc (None, 512)               2048
_________________________________________________________________
activation_7 (Activation)    (None, 512)               0
_________________________________________________________________
dropout_3 (Dropout)          (None, 512)               0
_________________________________________________________________
dense_1 (Dense)              (None, 525)               269325
=================================================================
Total params: 24,385,229
Trainable params: 24,382,925
Non-trainable params: 2,304
_________________________________________________________________
```

[39]:

```
base_model.compile(optimizer =keras.optimizers.Adam(lr = 0.001),
               loss = 'categorical_crossentropy',
               metrics = ['accuracy'])

# Establecemos dos callbacks: EarlyStopping y ReduceLearningRate
history = base_model.fit(
    train_generator,
    steps_per_epoch = train_groups,
    epochs = 20, # añadir más épocas aumenta el accuracy un 1% o 2%
    validation_data = valid_generator,
    validation_steps = valid_groups,
    verbose = 1,
```

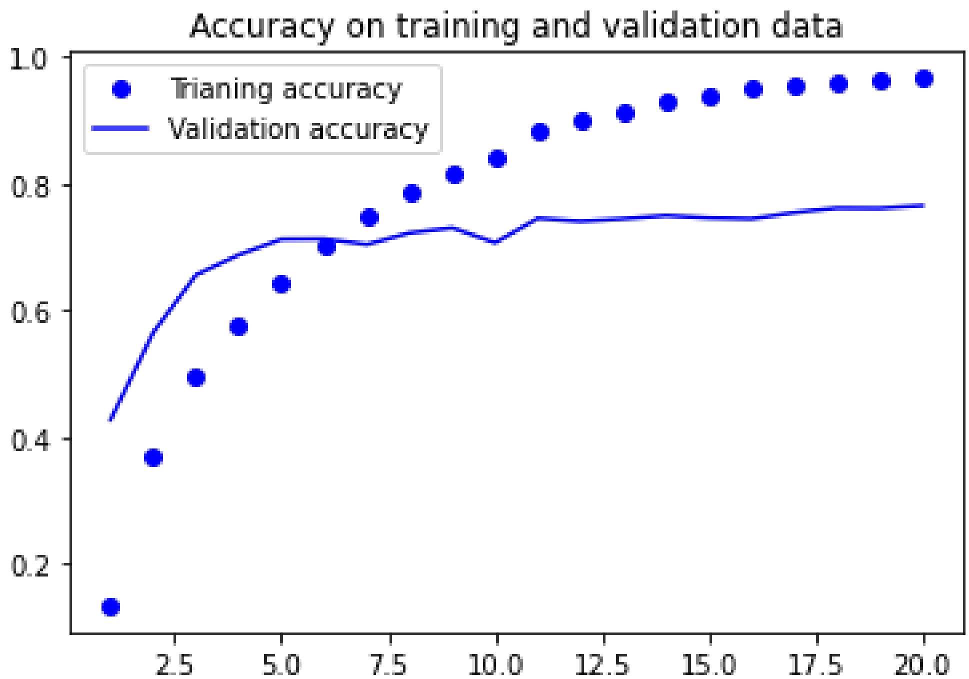
```
    callbacks=[keras.callbacks.EarlyStopping(monitor='val_accuracy', patience =␣
↪5, restore_best_weights = True),
               keras.callbacks.ReduceLROnPlateau(monitor = 'val_loss', factor =␣
↪0.7, patience = 2, verbose = 1),
    keras.callbacks.ModelCheckpoint(
            filepath = "intial_model.keras",
            save_best_only = True,
            monitor = "val_loss")
    ])
```

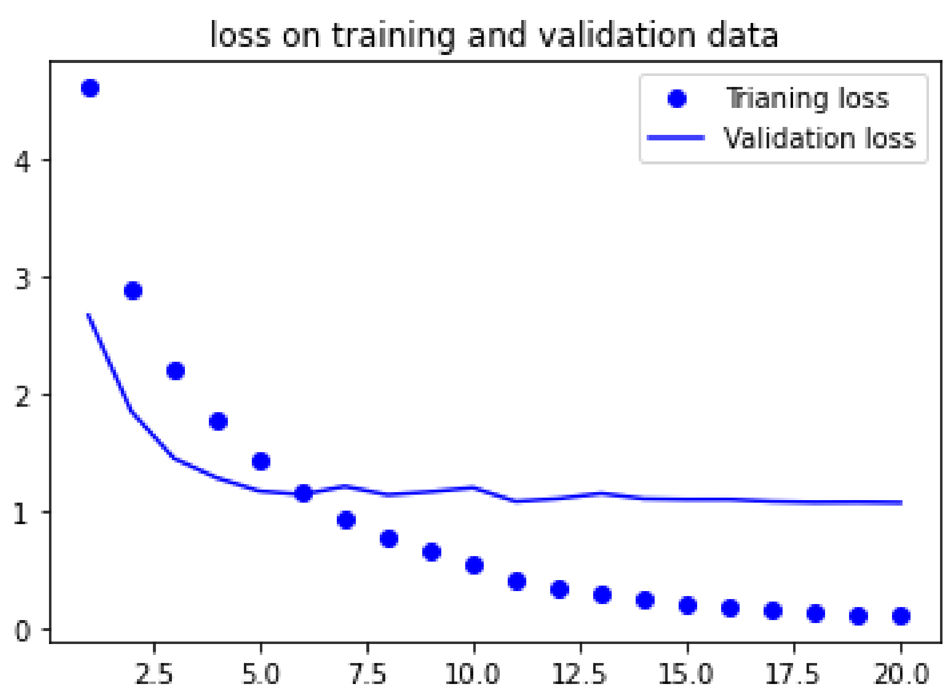
```
Epoch 1/20
2645/2645 [==============================] - 200s 75ms/step - loss: 5.4692 -
accuracy: 0.0638 - val_loss: 2.6702 - val_accuracy: 0.4278
Epoch 2/20
2645/2645 [==============================] - 199s 75ms/step - loss: 3.0352 -
accuracy: 0.3432 - val_loss: 1.8511 - val_accuracy: 0.5653
Epoch 3/20
2645/2645 [==============================] - 437s 165ms/step - loss: 2.2215 -
accuracy: 0.4922 - val_loss: 1.4484 - val_accuracy: 0.6560
Epoch 4/20
2645/2645 [==============================] - 569s 215ms/step - loss: 1.7327 -
accuracy: 0.5836 - val_loss: 1.2815 - val_accuracy: 0.6876
Epoch 5/20
2645/2645 [==============================] - 767s 290ms/step - loss: 1.3764 -
accuracy: 0.6549 - val_loss: 1.1683 - val_accuracy: 0.7124
Epoch 6/20
2645/2645 [==============================] - 716s 271ms/step - loss: 1.1034 -
accuracy: 0.7134 - val_loss: 1.1383 - val_accuracy: 0.7128
Epoch 7/20
2645/2645 [==============================] - 702s 265ms/step - loss: 0.8845 -
accuracy: 0.7633 - val_loss: 1.2086 - val_accuracy: 0.7040
Epoch 8/20
2645/2645 [==============================] - 536s 203ms/step - loss: 0.7121 -
accuracy: 0.8025 - val_loss: 1.1358 - val_accuracy: 0.7223
Epoch 9/20
2645/2645 [==============================] - 498s 188ms/step - loss: 0.6111 -
accuracy: 0.8258 - val_loss: 1.1622 - val_accuracy: 0.7303
Epoch 10/20
2645/2645 [==============================] - 436s 165ms/step - loss: 0.5136 -
accuracy: 0.8512 - val_loss: 1.1972 - val_accuracy: 0.7067

Epoch 00010: ReduceLROnPlateau reducing learning rate to 0.0007000000332482159.
Epoch 11/20
2645/2645 [==============================] - 391s 148ms/step - loss: 0.3953 -
accuracy: 0.8839 - val_loss: 1.0791 - val_accuracy: 0.7451
Epoch 12/20
2645/2645 [==============================] - 391s 148ms/step - loss: 0.3238 -
```

```
accuracy: 0.9051 - val_loss: 1.1037 - val_accuracy: 0.7406
Epoch 13/20
2645/2645 [==============================] - 399s 151ms/step - loss: 0.2820 -
accuracy: 0.9150 - val_loss: 1.1479 - val_accuracy: 0.7444

Epoch 00013: ReduceLROnPlateau reducing learning rate to 0.0004900000232737511.
Epoch 14/20
2645/2645 [==============================] - 387s 146ms/step - loss: 0.2345 -
accuracy: 0.9303 - val_loss: 1.1021 - val_accuracy: 0.7497
Epoch 15/20
2645/2645 [==============================] - 399s 151ms/step - loss: 0.2012 -
accuracy: 0.9394 - val_loss: 1.0964 - val_accuracy: 0.7459

Epoch 00015: ReduceLROnPlateau reducing learning rate to 0.00034300000406801696.
Epoch 16/20
2645/2645 [==============================] - 387s 146ms/step - loss: 0.1687 -
accuracy: 0.9494 - val_loss: 1.0968 - val_accuracy: 0.7444
Epoch 17/20
2645/2645 [==============================] - 398s 151ms/step - loss: 0.1459 -
accuracy: 0.9561 - val_loss: 1.0791 - val_accuracy: 0.7543

Epoch 00017: ReduceLROnPlateau reducing learning rate to 0.00024009999469853935.
Epoch 18/20
2645/2645 [==============================] - 393s 149ms/step - loss: 0.1340 -
accuracy: 0.9586 - val_loss: 1.0718 - val_accuracy: 0.7615
Epoch 19/20
2645/2645 [==============================] - 388s 147ms/step - loss: 0.1195 -
accuracy: 0.9624 - val_loss: 1.0737 - val_accuracy: 0.7611
Epoch 20/20
2645/2645 [==============================] - 389s 147ms/step - loss: 0.1105 -
accuracy: 0.9675 - val_loss: 1.0693 - val_accuracy: 0.7653
```

[40]:
```python
accuracy = history.history["accuracy"]
val_accuracy = history.history["val_accuracy"]

loss = history.history["loss"]
val_loss = history.history["val_loss"]

epochs = range(1, len(accuracy) + 1)

plt.plot(epochs, accuracy, "bo", label = "Train accuracy")
plt.plot(epochs, val_accuracy, "b-", label = "Val accuracy")
plt.title("Acc on train and val data")
plt.legend()
plt.figure()

plt.plot(epochs, loss, "bo", label = "Trianing loss")
```

```
plt.plot(epochs, val_loss, "b-", label = "Validation loss")
plt.title("Loss on train and val data")
plt.legend()
plt.show()
```





[41]:
```python
# Comprobando el modelo en el dataset de test
test_model_base = keras.models.load_model("intial_model.keras")
_, test_acc = test_model_base.evaluate(test_generator)
print(f"The accuracy of the intial model on the test set is : {test_acc:.3f}")
```

```
83/83 [==============================] - 17s 207ms/step - loss: 0.8728 -
accuracy: 0.7981
The accuracy of the intial model on the test set is : 0.798
```

The accuracy of the intial model on the test set is : 0.798

0.4 TRANSFER LEARNING: VGG16 | INCEPTIONV3

RNDL_VGG16_INCEPTIONV3

November 25, 2023

[1]:
```
import opendatasets as od
dataset_url='https://www.kaggle.com/datasets/gpiosenka/100-bird-species'
od.download(dataset_url)
```

```
Skipping, found downloaded files in ".\100-bird-species" (use force=True to
force download)
```

[2]:
```
import tensorflow as tf
gpus = tf.config.experimental.list_physical_devices('GPU')
tf.config.experimental.set_memory_growth(gpus[0], True)
```

[3]:
```
import matplotlib.pyplot as plt
import os
from tensorflow.keras.preprocessing.image import img_to_array, load_img,␣
↪ImageDataGenerator
from tensorflow.keras.applications.vgg16 import preprocess_input
```

[4]:
```
def plot_curves(model_history,filepath_image,count):
    accuracy = model_history.history['accuracy']
    val_accuracy = model_history.history['val_accuracy']

    loss = model_history.history['loss']
    val_loss = model_history.history['val_loss']

    epochs = range(len(accuracy))
    plt.plot(epochs, accuracy, 'b', label='Training accuracy')
    plt.plot(epochs, val_accuracy, 'r', label='Validation accuracy')

    plt.title('Training and validation accuracy')
    plt.legend()
    plt.figure()
    plt.plot(epochs, loss, 'b', label='Training loss')
    plt.plot(epochs, val_loss, 'r', label='Validation loss')

    plt.title('Training and validation')
    plt.legend()
    plt.show()
    plt.savefig(filepath_image, dpi=100)
```

```
    count+=1
```

[5]:
```
#Para crear directorios que guarden los modelos
def crea_directorio(model_name):
    folder_path="Models/"+model_name
    os.makedirs(folder_path, exist_ok=True)
    model_path=folder_path+"/"+model_name+"-{val_accuracy:.2f}.hdf5"
    graph_path=folder_path+"/learning_curves_"+model_name+".png"
    return model_path,graph_path
```

[6]:
```
def evalua_modelo(model,test_data,model_name,dic_resultados):
    results = model.evaluate(test_data, verbose=0)
    dic_resultados[model_name]={"test_loss":results[0],"test_accuracy":
 ↪results[1]}
    return f"{model_name}- test_loss: {results[0]} - test_accuracy {results[1]}"
resultados={}
```

[7]:
```
train_data = ImageDataGenerator(preprocessing_function =␣
 ↪preprocess_input,rescale=1./255)
train_generator = train_data.flow_from_directory('100-bird-species/train/',
                                                batch_size=32,
                                                target_size=(224,224),
                                                 shuffle=True,
                                                class_mode='categorical')
```

```
Found 84635 images belonging to 525 classes.
```

[8]:
```
val_data = ImageDataGenerator(preprocessing_function =␣
 ↪preprocess_input,rescale=1./255)
val_generator = val_data.flow_from_directory('100-bird-species/valid/',
                                            batch_size=32,
                                            target_size=(224,224),
                                            shuffle=True,
                                            class_mode='categorical')
```

```
Found 2625 images belonging to 525 classes.
```

[9]:
```
test_data = ImageDataGenerator(preprocessing_function =␣
 ↪preprocess_input,rescale=1./255)
test_generator = test_data.flow_from_directory('100-bird-species/test/',
                                              batch_size=32,
                                              target_size=(224,224),
                                              shuffle=True,
                                              class_mode='categorical')
```

```
Found 2625 images belonging to 525 classes.
```

[10]:
```
#importamos las librerias
from tensorflow.keras.applications import VGG16
from tensorflow.keras import layers, models
from tensorflow.keras.callbacks import EarlyStopping, ModelCheckpoint,␣
↪ReduceLROnPlateau
from tensorflow.keras.optimizers import Adam
```

0.0.1 Red Pre-entrenada

A continuación se realizarán tareas de ***transfer learning*** y ***fine-tuning*** comparando la arquitectura VGG16 y la arquitectura InceptionV3.

0.0.2 VGG16 y VGG19

Se tratan de redes convolucionales desarrolladas por K. Simonyan y A. Zisserman, de la Universidad Oxford, que ganó el Desafío de Reconocimiento Visual a Gran Escala de ImageNet (ILSVRC) en 2014. Se trata de una arquitectura bastante simple, compuesta por un número progresivo de bloques convolucionales con filtros de tamaño 3x3. Entre cada bloque convolucional una capa maxpooling se encarga de reducir a la mitad las dimensiones de los mapas de activación. Los números 16 y 19 hacen referencia a las capas de profundidad de cada algoritmo.

0.0.3 Modelo BASE

[11]:
```
#Cargamos el base_model con los pesos de ImageNet y los congelamos
base_model = VGG16(weights='imagenet', include_top=False,␣
↪input_shape=(224,224,3))
base_model.trainable=False
```

[12]:
```
#Creamos el top model (Clasificador)
flatten_layer = layers.Flatten()
dense_layer_1 = layers.Dense(2048, activation='relu')
dense_layer_2 = layers.Dense(1024,activation='relu')
prediction_layer = layers.Dense(525, activation='softmax')

#Unimos los modelos con la api secuencialc
model = models.Sequential([
    base_model,
    flatten_layer,
    dense_layer_1,
    dense_layer_2,
    prediction_layer
])

#Compilamos el modelo
model.compile(
    optimizer=Adam(learning_rate=0.0003),
```

```
    loss='categorical_crossentropy',
    metrics=['accuracy'],
)

model.summary()
```

```
Model: "sequential"

_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 vgg16 (Functional)          (None, 7, 7, 512)         14714688

 flatten (Flatten)           (None, 25088)             0

 dense (Dense)               (None, 2048)              51382272

 dense_1 (Dense)             (None, 1024)              2098176

 dense_2 (Dense)             (None, 525)               538125

=================================================================
Total params: 68,733,261
Trainable params: 54,018,573
Non-trainable params: 14,714,688
_________________________________________________________________
```

[13]:
```
#Creamos directorio para guardar el modelo
count=1
model_name=base_model.name+"_entrega_"+str(count)
model_path,graph_path=crea_directorio(model_name)
```

[14]:
```
tf.keras.utils.plot_model(model, to_file=graph_path.
 ↪replace("learning_curves","graph"), show_shapes=True)
```

```
You must install pydot (`pip install pydot`) and install graphviz (see
instructions at https://graphviz.gitlab.io/download/) for
plot_model/model_to_dot to work.
```

[15]:
```
#Definimos los callbacks
lr =ReduceLROnPlateau(monitor="val_loss",
                      factor=0.5,
                      patience=2)

es = EarlyStopping(monitor='val_accuracy',
                   mode='max', patience=5,
                   restore_best_weights=True)

mcp = ModelCheckpoint(filepath=model_path,
```

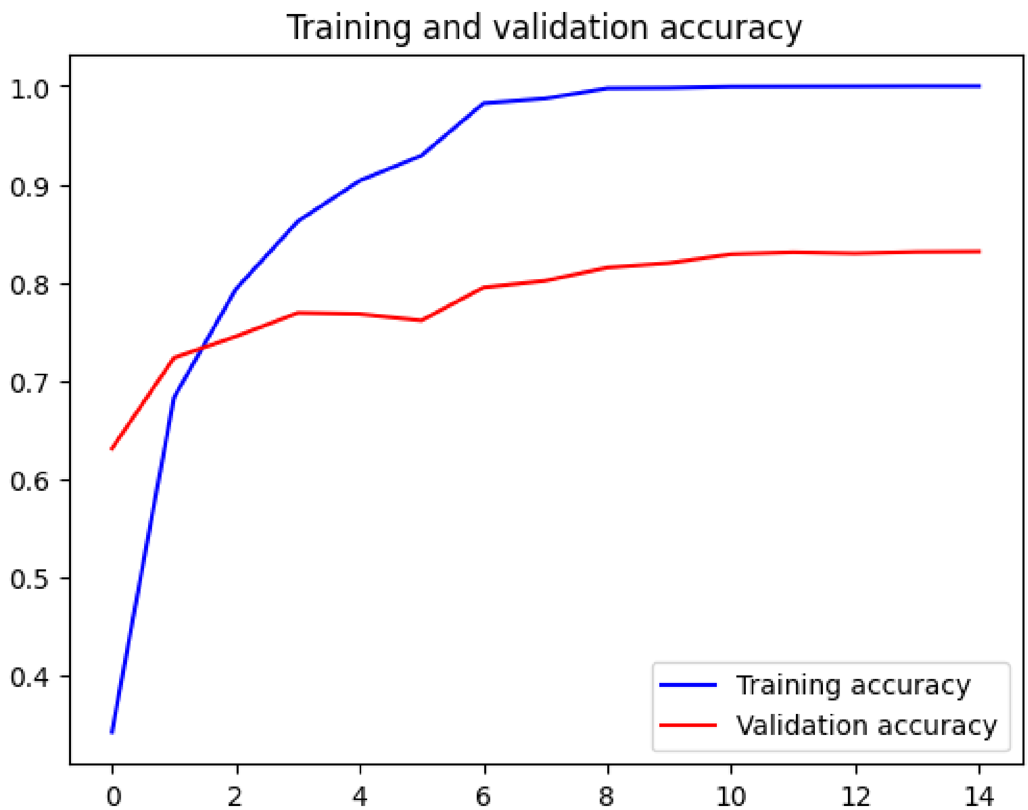
```
                           save_best_only=True,
                           monitor='val_accuracy',
                           mode='max')
```

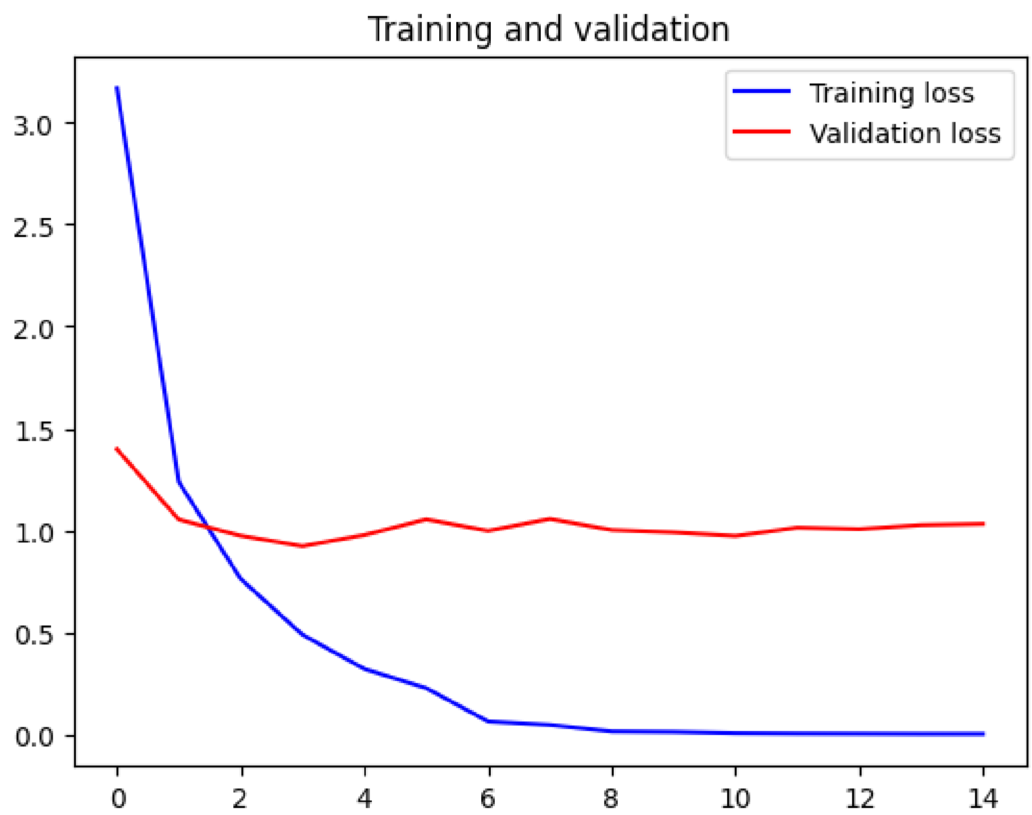
[16]:
```
#entrenamos el modelo
history=model.fit(train_generator, epochs=15, validation_data = val_generator,␣
 ↪callbacks=[es,lr,mcp])
```

```
Epoch 1/15
2645/2645 [==============================] - 367s 137ms/step - loss: 3.1696 -
accuracy: 0.3427 - val_loss: 1.3978 - val_accuracy: 0.6312 - lr: 3.0000e-04
Epoch 2/15
2645/2645 [==============================] - 358s 135ms/step - loss: 1.2372 -
accuracy: 0.6829 - val_loss: 1.0523 - val_accuracy: 0.7234 - lr: 3.0000e-04
Epoch 3/15
2645/2645 [==============================] - 363s 137ms/step - loss: 0.7615 -
accuracy: 0.7934 - val_loss: 0.9729 - val_accuracy: 0.7451 - lr: 3.0000e-04
Epoch 4/15
2645/2645 [==============================] - 358s 135ms/step - loss: 0.4873 -
accuracy: 0.8623 - val_loss: 0.9220 - val_accuracy: 0.7691 - lr: 3.0000e-04
Epoch 5/15
2645/2645 [==============================] - 355s 134ms/step - loss: 0.3196 -
accuracy: 0.9037 - val_loss: 0.9764 - val_accuracy: 0.7680 - lr: 3.0000e-04
Epoch 6/15
2645/2645 [==============================] - 358s 135ms/step - loss: 0.2254 -
accuracy: 0.9294 - val_loss: 1.0530 - val_accuracy: 0.7615 - lr: 3.0000e-04
Epoch 7/15
2645/2645 [==============================] - 363s 137ms/step - loss: 0.0617 -
accuracy: 0.9825 - val_loss: 0.9972 - val_accuracy: 0.7950 - lr: 1.5000e-04
Epoch 8/15
2645/2645 [==============================] - 369s 139ms/step - loss: 0.0446 -
accuracy: 0.9873 - val_loss: 1.0551 - val_accuracy: 0.8019 - lr: 1.5000e-04
Epoch 9/15
2645/2645 [==============================] - 358s 135ms/step - loss: 0.0126 -
accuracy: 0.9977 - val_loss: 1.0012 - val_accuracy: 0.8152 - lr: 7.5000e-05
Epoch 10/15
2645/2645 [==============================] - 359s 136ms/step - loss: 0.0103 -
accuracy: 0.9981 - val_loss: 0.9899 - val_accuracy: 0.8198 - lr: 7.5000e-05
Epoch 11/15
2645/2645 [==============================] - 375s 142ms/step - loss: 0.0041 -
accuracy: 0.9994 - val_loss: 0.9722 - val_accuracy: 0.8290 - lr: 3.7500e-05
Epoch 12/15
2645/2645 [==============================] - 356s 135ms/step - loss: 0.0025 -
accuracy: 0.9996 - val_loss: 1.0127 - val_accuracy: 0.8309 - lr: 3.7500e-05
Epoch 13/15
2645/2645 [==============================] - 368s 139ms/step - loss: 0.0018 -
accuracy: 0.9997 - val_loss: 1.0054 - val_accuracy: 0.8297 - lr: 1.8750e-05
Epoch 14/15
```

```
2645/2645 [==============================] - 436s 165ms/step - loss: 8.2685e-04
- accuracy: 0.9999 - val_loss: 1.0254 - val_accuracy: 0.8312 - lr: 1.8750e-05
Epoch 15/15
2645/2645 [==============================] - 437s 165ms/step - loss: 6.0145e-04
- accuracy: 0.9999 - val_loss: 1.0311 - val_accuracy: 0.8316 - lr: 9.3750e-06
```

[17]:
```
plot_curves(history,graph_path,count)
```






```
<Figure size 640x480 with 0 Axes>
```

[18]:
```
evalua_modelo(model,test_generator,model_name,resultados)
```

[18]:
```
'vgg16_entrega_1- test_loss: 0.7535420656204224 - test_accuracy
0.8655238151550293'
```

0.0.4 Modelo Base 2

[ ]:
```

```

[19]:
```
#Creamos el top model (Clasificador)
flatten_layer = layers.Flatten()
dense_layer_1 = layers.Dense(2048, activation='relu')
dense_layer_2 = layers.Dense(2048,activation='relu')
dense_layer_3 = layers.Dense(1024,activation='relu')
prediction_layer = layers.Dense(525, activation='softmax')

#Unimos los modelos con la api secuencialc
model = models.Sequential([
    base_model,
```

```
    flatten_layer,
    dense_layer_1,
    dense_layer_2,
    dense_layer_3,
    prediction_layer
])

#Compilamos el modelo
model.compile(
    optimizer=Adam(learning_rate=0.0003),
    loss='categorical_crossentropy',
    metrics=['accuracy'],
)

model.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 vgg16 (Functional)          (None, 7, 7, 512)         14714688

 flatten_1 (Flatten)         (None, 25088)             0

 dense_3 (Dense)             (None, 2048)              51382272

 dense_4 (Dense)             (None, 2048)              4196352

 dense_5 (Dense)             (None, 1024)              2098176

 dense_6 (Dense)             (None, 525)               538125

=================================================================
Total params: 72,929,613
Trainable params: 58,214,925
Non-trainable params: 14,714,688
_________________________________________________________________
```

[20]:
```
#Creamos directorio para guardar el modelo
count=2
model_name=base_model.name+"_entrega_"+str(count)
model_path,graph_path=crea_directorio(model_name)
```

[21]:
```
tf.keras.utils.plot_model(model, to_file=graph_path.
 ↪replace("learning_curves","graph"), show_shapes=True)
```

```
You must install pydot (`pip install pydot`) and install graphviz (see
instructions at https://graphviz.gitlab.io/download/) for
```

plot_model/model_to_dot to work.

[22]:
```
#Definimos los callbacks
lr =ReduceLROnPlateau(monitor="val_loss",
                      factor=0.5,
                      patience=2)

es = EarlyStopping(monitor='val_accuracy',
                   mode='max', patience=5,
                   restore_best_weights=True)

mcp = ModelCheckpoint(filepath=model_path,
                      save_best_only=True,
                      monitor='val_accuracy',
                      mode='max')
```

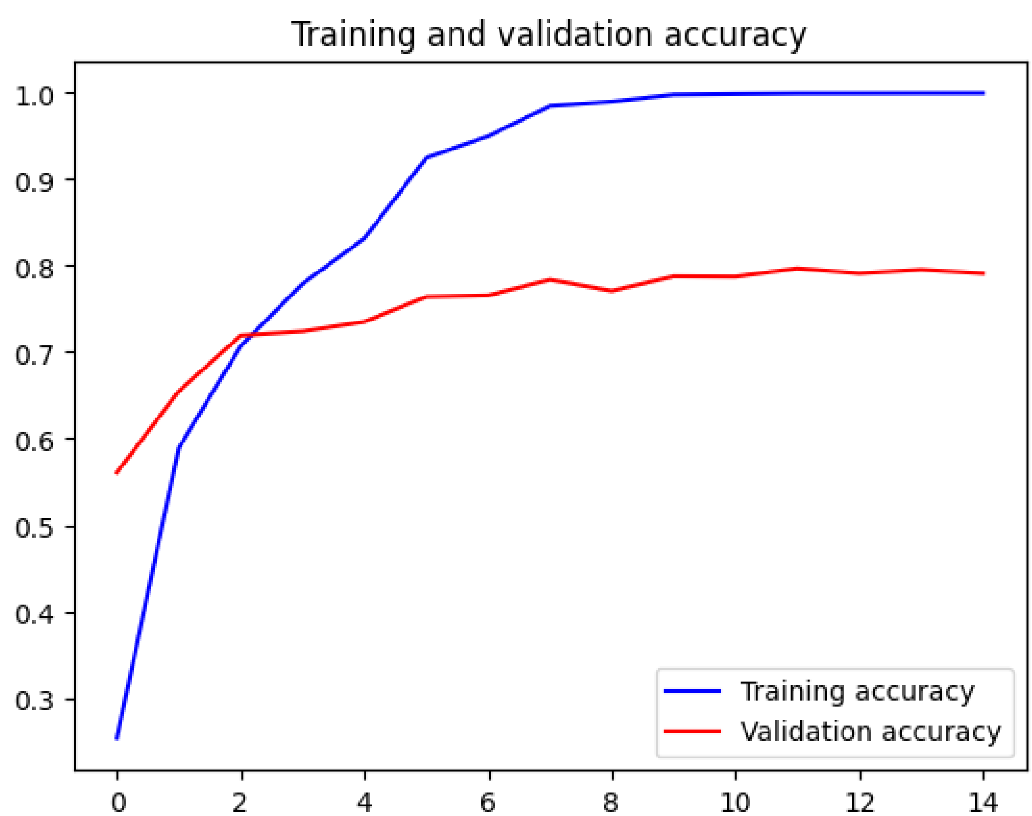
[23]:
```
#entrenamos el modelo
history=model.fit(train_generator, epochs=15, validation_data = val_generator,␣
 ↪callbacks=[es,lr,mcp])
```

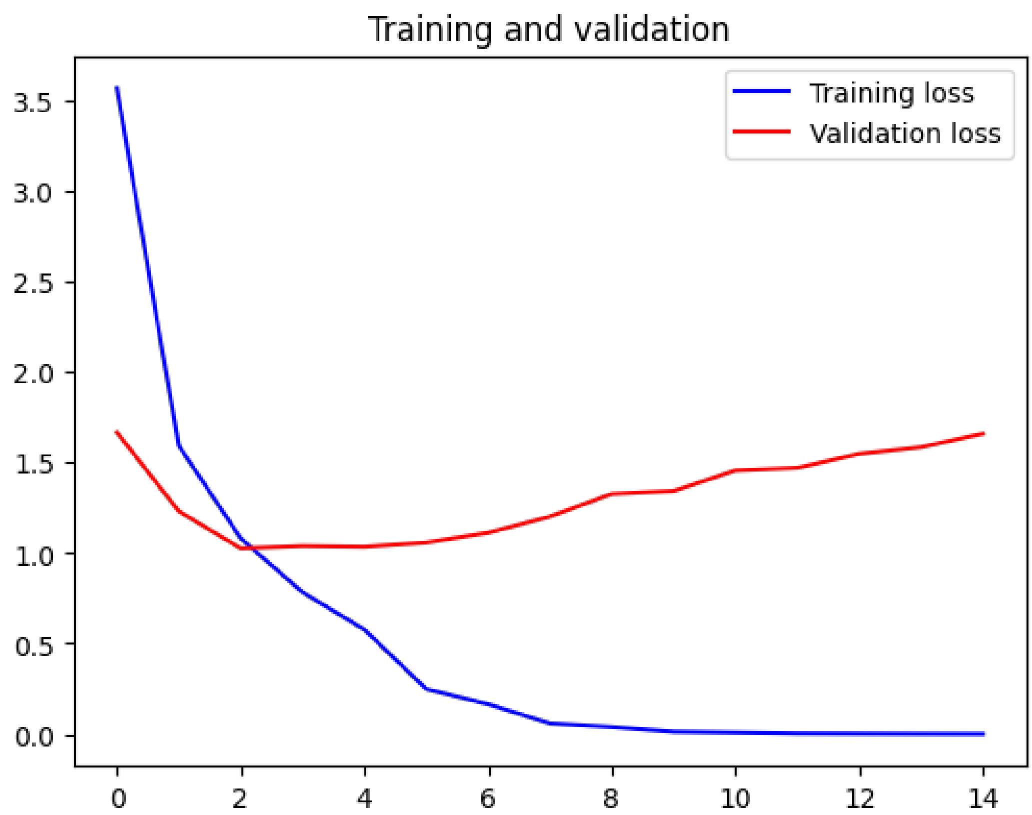
```
Epoch 1/15
2645/2645 [==============================] - 432s 163ms/step - loss: 3.5672 -
accuracy: 0.2535 - val_loss: 1.6661 - val_accuracy: 0.5608 - lr: 3.0000e-04
Epoch 2/15
2645/2645 [==============================] - 408s 154ms/step - loss: 1.5930 -
accuracy: 0.5892 - val_loss: 1.2308 - val_accuracy: 0.6549 - lr: 3.0000e-04
Epoch 3/15
2645/2645 [==============================] - 358s 135ms/step - loss: 1.0814 -
accuracy: 0.7070 - val_loss: 1.0259 - val_accuracy: 0.7192 - lr: 3.0000e-04
Epoch 4/15
2645/2645 [==============================] - 392s 148ms/step - loss: 0.7833 -
accuracy: 0.7792 - val_loss: 1.0383 - val_accuracy: 0.7242 - lr: 3.0000e-04
Epoch 5/15
2645/2645 [==============================] - 362s 137ms/step - loss: 0.5774 -
accuracy: 0.8317 - val_loss: 1.0343 - val_accuracy: 0.7352 - lr: 3.0000e-04
Epoch 6/15
2645/2645 [==============================] - 374s 141ms/step - loss: 0.2489 -
accuracy: 0.9249 - val_loss: 1.0577 - val_accuracy: 0.7642 - lr: 1.5000e-04
Epoch 7/15
2645/2645 [==============================] - 366s 138ms/step - loss: 0.1652 -
accuracy: 0.9500 - val_loss: 1.1117 - val_accuracy: 0.7657 - lr: 1.5000e-04
Epoch 8/15
2645/2645 [==============================] - 372s 141ms/step - loss: 0.0590 -
accuracy: 0.9849 - val_loss: 1.2010 - val_accuracy: 0.7840 - lr: 7.5000e-05
Epoch 9/15
2645/2645 [==============================] - 356s 134ms/step - loss: 0.0392 -
accuracy: 0.9899 - val_loss: 1.3264 - val_accuracy: 0.7710 - lr: 7.5000e-05
Epoch 10/15
```

```
2645/2645 [==============================] - 362s 137ms/step - loss: 0.0130 -
accuracy: 0.9981 - val_loss: 1.3425 - val_accuracy: 0.7878 - lr: 3.7500e-05
Epoch 11/15
2645/2645 [==============================] - 362s 137ms/step - loss: 0.0084 -
accuracy: 0.9990 - val_loss: 1.4559 - val_accuracy: 0.7874 - lr: 3.7500e-05
Epoch 12/15
2645/2645 [==============================] - 356s 135ms/step - loss: 0.0040 -
accuracy: 0.9996 - val_loss: 1.4700 - val_accuracy: 0.7970 - lr: 1.8750e-05
Epoch 13/15
2645/2645 [==============================] - 366s 138ms/step - loss: 0.0027 -
accuracy: 0.9997 - val_loss: 1.5476 - val_accuracy: 0.7912 - lr: 1.8750e-05
Epoch 14/15
2645/2645 [==============================] - 360s 136ms/step - loss: 0.0017 -
accuracy: 0.9999 - val_loss: 1.5846 - val_accuracy: 0.7954 - lr: 9.3750e-06
Epoch 15/15
2645/2645 [==============================] - 366s 138ms/step - loss: 9.9707e-04
- accuracy: 1.0000 - val_loss: 1.6585 - val_accuracy: 0.7912 - lr: 9.3750e-06
```

[24]:

```
plot_curves(history,graph_path,count)
```






```
<Figure size 640x480 with 0 Axes>
```

[25]:
```
evalua_modelo(model,test_generator,model_name,resultados)
```

[25]:
```
'vgg16_entrega_2- test_loss: 1.3398350477218628 - test_accuracy
0.8110476136207581'
```

0.0.5 Fine tunning 1

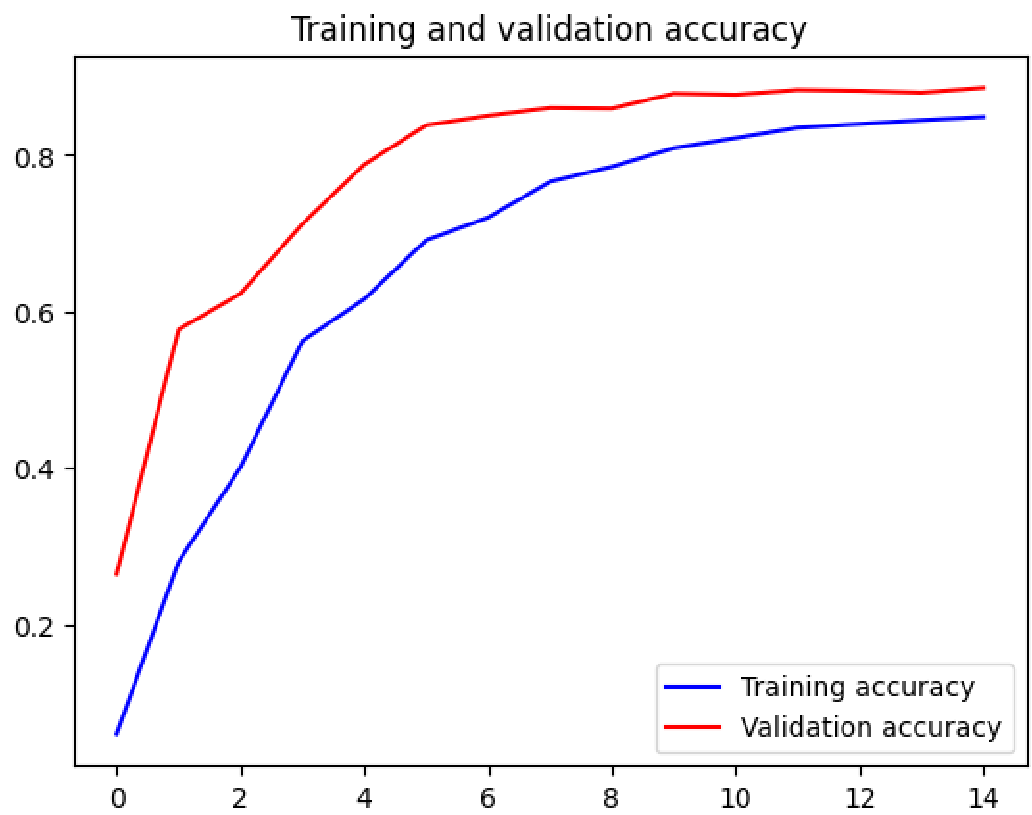
[26]:
```
#Cargamos el base_model con los pesos de ImageNet y los congelamos
base_model = VGG16(weights='imagenet', include_top=False,␣
 ↪input_shape=(224,224,3))
base_model.trainable=True
for layer in base_model.layers[:-3]:
    layer.trainable=False

# Creamos el top model (Clasificador)
flatten_layer = layers.Flatten()
dense_layer_1 = layers.Dense(1024, activation='relu')
batch_norm_layer_1 = layers.BatchNormalization()
dropout_layer_1 = layers.Dropout(0.5)
```

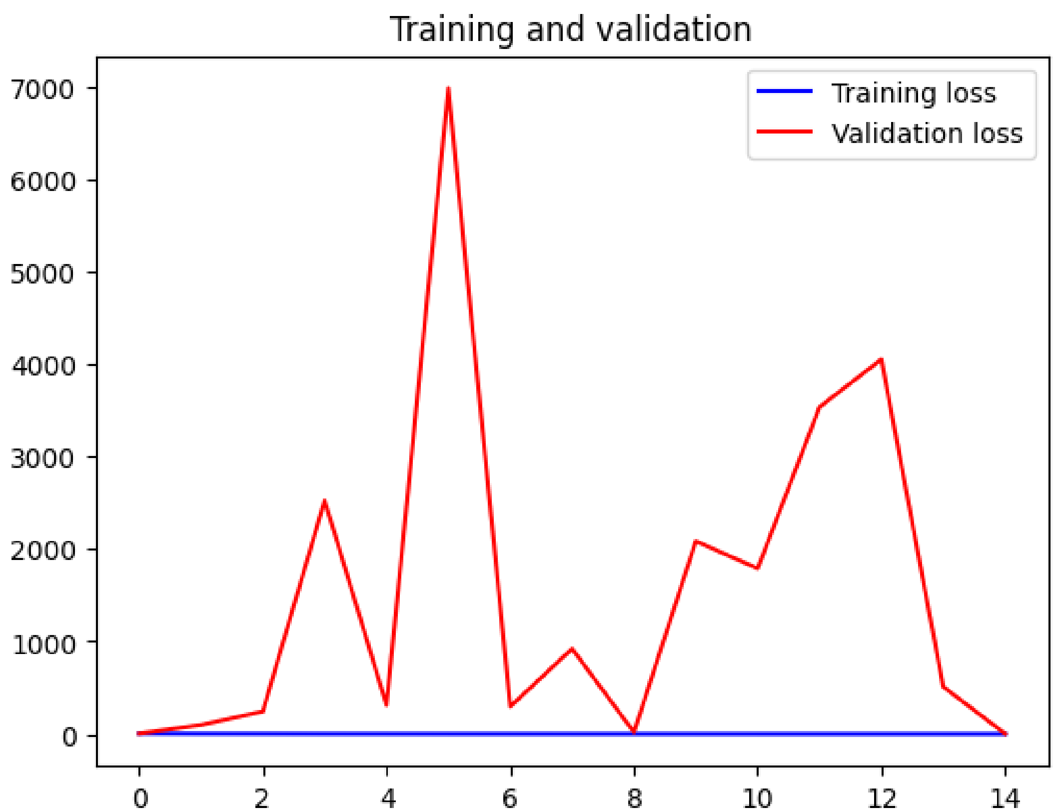
```
dense_layer_2 = layers.Dense(512, activation='relu')
batch_norm_layer_2 = layers.BatchNormalization()
dropout_layer_2 = layers.Dropout(0.5)
prediction_layer = layers.Dense(525, activation='softmax')

# Unimos los modelos con la API secuencial
model = models.Sequential([
    base_model,
    flatten_layer,
    dense_layer_1,
    batch_norm_layer_1,
    dropout_layer_1,
    dense_layer_2,
    batch_norm_layer_2,
    dropout_layer_2,
    prediction_layer
])

#Compilamos el modelo
model.compile(
    optimizer=Adam(learning_rate=0.01),
    loss='categorical_crossentropy',
    metrics=['accuracy'],
)

model.summary()


#Creamos directorio para guardar el modelo
count=1 #contador de experimentos iniciado, se actualiza al graficar
model_name=base_model.name+"_tunned_entrega_"+str(count)
model_path,graph_path=crea_directorio(model_name)

tf.keras.utils.plot_model(model, to_file=graph_path.
 ↪replace("learning_curves","graph"), show_shapes=True)

#Definimos los callbacks
lr =ReduceLROnPlateau(monitor="val_loss",
                      factor=0.5,
                      patience=2)

es = EarlyStopping(monitor='val_accuracy',
                   mode='max', patience=5,
                   restore_best_weights=True)

mcp = ModelCheckpoint(filepath=model_path,
                      save_best_only=True,
```

```
                    monitor='val_accuracy',
                    mode='max')

#entrenamos el modelo
history=model.fit(train_generator, epochs=15, validation_data = val_generator,␣
 ↪callbacks=[es,lr,mcp])

plot_curves(model.history,graph_path,count)

evalua_modelo(model,test_generator,model_name,resultados)
```

```
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 vgg16 (Functional)          (None, 7, 7, 512)         14714688

 flatten_2 (Flatten)         (None, 25088)             0

 dense_7 (Dense)             (None, 1024)              25691136

 batch_normalization (BatchN  (None, 1024)             4096
 ormalization)

 dropout (Dropout)           (None, 1024)              0

 dense_8 (Dense)             (None, 512)               524800

 batch_normalization_1 (Batc  (None, 512)              2048
 hNormalization)

 dropout_1 (Dropout)         (None, 512)               0

 dense_9 (Dense)             (None, 525)               269325

=================================================================
Total params: 41,206,093
Trainable params: 31,207,949
Non-trainable params: 9,998,144
_________________________________________________________________
You must install pydot (`pip install pydot`) and install graphviz (see
instructions at https://graphviz.gitlab.io/download/) for
plot_model/model_to_dot to work.
Epoch 1/15
2645/2645 [==============================] - 375s 141ms/step - loss: 5.1009 -
accuracy: 0.0608 - val_loss: 4.6265 - val_accuracy: 0.2648 - lr: 0.0100
Epoch 2/15
2645/2645 [==============================] - 380s 144ms/step - loss: 3.1285 -
```

```
accuracy: 0.2804 - val_loss: 97.2868 - val_accuracy: 0.5771 - lr: 0.0100
Epoch 3/15
2645/2645 [==============================] - 375s 142ms/step - loss: 2.5310 -
accuracy: 0.4010 - val_loss: 242.9237 - val_accuracy: 0.6229 - lr: 0.0100
Epoch 4/15
2645/2645 [==============================] - 377s 143ms/step - loss: 1.7376 -
accuracy: 0.5626 - val_loss: 2525.1626 - val_accuracy: 0.7120 - lr: 0.0050
Epoch 5/15
2645/2645 [==============================] - 379s 143ms/step - loss: 1.4931 -
accuracy: 0.6159 - val_loss: 310.1700 - val_accuracy: 0.7878 - lr: 0.0050
Epoch 6/15
2645/2645 [==============================] - 377s 143ms/step - loss: 1.1597 -
accuracy: 0.6912 - val_loss: 6984.4053 - val_accuracy: 0.8381 - lr: 0.0025
Epoch 7/15
2645/2645 [==============================] - 384s 145ms/step - loss: 1.0318 -
accuracy: 0.7200 - val_loss: 295.1431 - val_accuracy: 0.8503 - lr: 0.0025
Epoch 8/15
2645/2645 [==============================] - 375s 142ms/step - loss: 0.8476 -
accuracy: 0.7658 - val_loss: 922.5612 - val_accuracy: 0.8602 - lr: 0.0012
Epoch 9/15
2645/2645 [==============================] - 384s 145ms/step - loss: 0.7761 -
accuracy: 0.7850 - val_loss: 17.3073 - val_accuracy: 0.8594 - lr: 0.0012
Epoch 10/15
2645/2645 [==============================] - 382s 144ms/step - loss: 0.6699 -
accuracy: 0.8087 - val_loss: 2084.7622 - val_accuracy: 0.8785 - lr: 6.2500e-04
Epoch 11/15
2645/2645 [==============================] - 375s 142ms/step - loss: 0.6212 -
accuracy: 0.8216 - val_loss: 1789.6329 - val_accuracy: 0.8770 - lr: 6.2500e-04
Epoch 12/15
2645/2645 [==============================] - 383s 145ms/step - loss: 0.5754 -
accuracy: 0.8348 - val_loss: 3535.7356 - val_accuracy: 0.8830 - lr: 3.1250e-04
Epoch 13/15
2645/2645 [==============================] - 375s 142ms/step - loss: 0.5549 -
accuracy: 0.8396 - val_loss: 4051.4465 - val_accuracy: 0.8819 - lr: 3.1250e-04
Epoch 14/15
2645/2645 [==============================] - 377s 142ms/step - loss: 0.5298 -
accuracy: 0.8445 - val_loss: 510.4235 - val_accuracy: 0.8796 - lr: 1.5625e-04
Epoch 15/15
2645/2645 [==============================] - 379s 143ms/step - loss: 0.5157 -
accuracy: 0.8486 - val_loss: 2.6249 - val_accuracy: 0.8857 - lr: 1.5625e-04
```


Training and validation accuracy
0.2
0.4
0.6
0.8
0
2
4
6
8
10
12
14
Training accuracy
Validation accuracy



[26]: 'vgg16_tunned_entrega_1- test_loss: 641.4629516601562 - test_accuracy
0.9062857031822205'

<Figure size 640x480 with 0 Axes>

0.0.6 Fine tunning 2

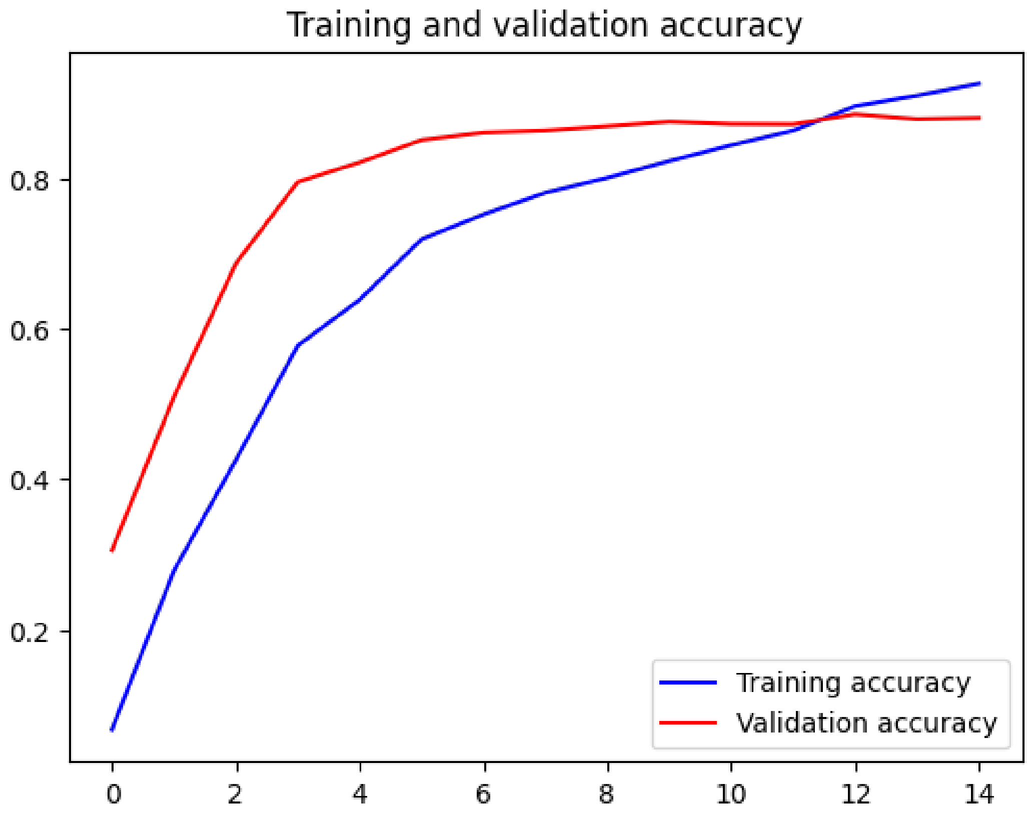
[27]:
```
#Cargamos el base_model con los pesos de ImageNet y los congelamos
base_model = VGG16(weights='imagenet', include_top=False,␣
↪input_shape=(224,224,3))
base_model.trainable=True
for layer in base_model.layers[:-3]:
    layer.trainable=False

# Creamos el top model (Clasificador)
flatten_layer = layers.Flatten()
dense_layer_1 = layers.Dense(512, activation='relu')
batch_norm_layer_1 = layers.BatchNormalization()
dropout_layer_1 = layers.Dropout(0.5)
dense_layer_2 = layers.Dense(256, activation='relu')
batch_norm_layer_2 = layers.BatchNormalization()
```

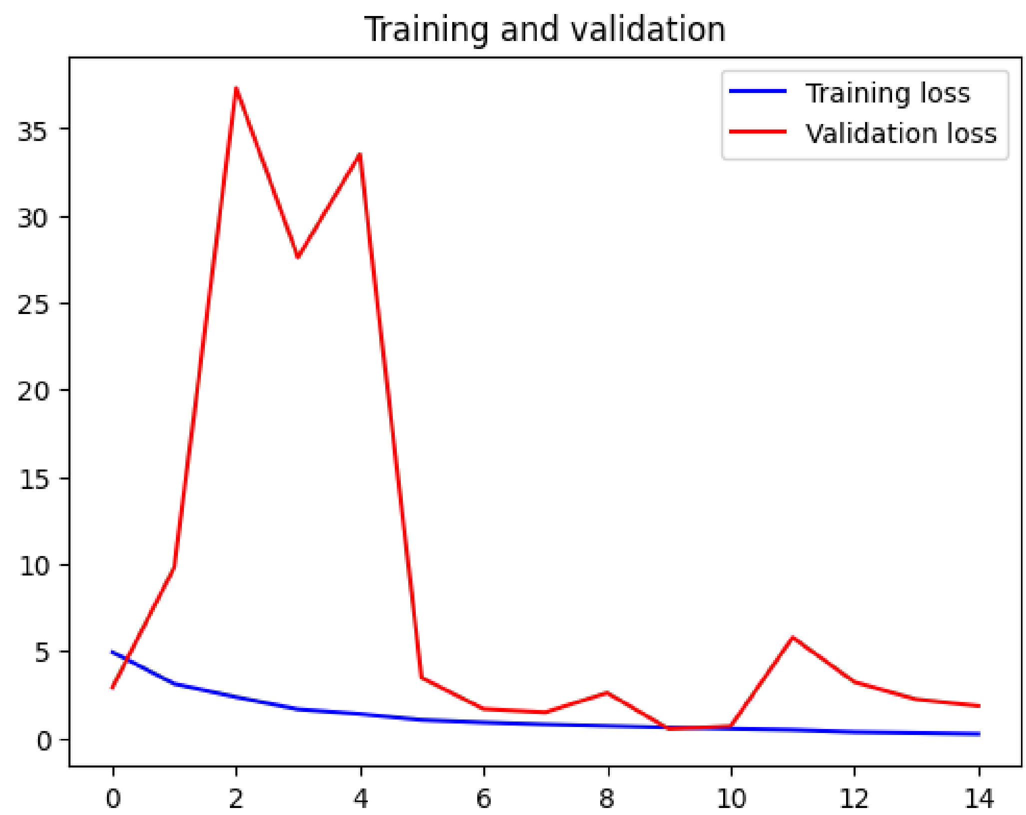
```
dropout_layer_2 = layers.Dropout(0.5)
prediction_layer = layers.Dense(525, activation='softmax')

# Unimos los modelos con la API secuencial
model = models.Sequential([
    base_model,
    flatten_layer,
    dense_layer_1,
    batch_norm_layer_1,
    dropout_layer_1,
    dense_layer_2,
    batch_norm_layer_2,
    dropout_layer_2,
    prediction_layer
])

#Compilamos el modelo
model.compile(
    optimizer=Adam(learning_rate=0.01),
    loss='categorical_crossentropy',
    metrics=['accuracy'],
)

model.summary()


#Creamos directorio para guardar el modelo
count=2 #contador de experimentos iniciado, se actualiza al graficar
model_name=base_model.name+"_tunned_entrega_"+str(count)
model_path,graph_path=crea_directorio(model_name)

tf.keras.utils.plot_model(model, to_file=graph_path.
 ↪replace("learning_curves","graph"), show_shapes=True)

#Definimos los callbacks
lr =ReduceLROnPlateau(monitor="val_loss",
                      factor=0.5,
                      patience=2)

es = EarlyStopping(monitor='val_accuracy',
                   mode='max', patience=5,
                   restore_best_weights=True)

mcp = ModelCheckpoint(filepath=model_path,
                      save_best_only=True,
                      monitor='val_accuracy',
                      mode='max')
```

```
#entrenamos el modelo
history=model.fit(train_generator, epochs=15, validation_data = val_generator,␣
↪callbacks=[es,lr,mcp])

plot_curves(model.history,graph_path,count)

evalua_modelo(model,test_generator,model_name,resultados)
```

```
Model: "sequential_3"

_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 vgg16 (Functional)          (None, 7, 7, 512)         14714688

 flatten_3 (Flatten)         (None, 25088)             0

 dense_10 (Dense)            (None, 512)               12845568

 batch_normalization_2 (Batc  (None, 512)              2048
 hNormalization)

 dropout_2 (Dropout)         (None, 512)               0

 dense_11 (Dense)            (None, 256)               131328

 batch_normalization_3 (Batc  (None, 256)              1024
 hNormalization)

 dropout_3 (Dropout)         (None, 256)               0

 dense_12 (Dense)            (None, 525)               134925

=================================================================
Total params: 27,829,581
Trainable params: 17,832,973
Non-trainable params: 9,996,608
_________________________________________________________________
You must install pydot (`pip install pydot`) and install graphviz (see
instructions at https://graphviz.gitlab.io/download/) for
plot_model/model_to_dot to work.
Epoch 1/15
2645/2645 [==============================] - 367s 138ms/step - loss: 4.9257 -
accuracy: 0.0679 - val_loss: 2.9032 - val_accuracy: 0.3059 - lr: 0.0100
Epoch 2/15
2645/2645 [==============================] - 374s 141ms/step - loss: 3.1066 -
accuracy: 0.2786 - val_loss: 9.8014 - val_accuracy: 0.5093 - lr: 0.0100
Epoch 3/15
```

```
2645/2645 [==============================] - 372s 141ms/step - loss: 2.3447 -
accuracy: 0.4256 - val_loss: 37.3082 - val_accuracy: 0.6865 - lr: 0.0100
Epoch 4/15
2645/2645 [==============================] - 369s 140ms/step - loss: 1.6441 -
accuracy: 0.5774 - val_loss: 27.5941 - val_accuracy: 0.7947 - lr: 0.0050
Epoch 5/15
2645/2645 [==============================] - 370s 140ms/step - loss: 1.3749 -
accuracy: 0.6383 - val_loss: 33.5124 - val_accuracy: 0.8206 - lr: 0.0050
Epoch 6/15
2645/2645 [==============================] - 371s 140ms/step - loss: 1.0351 -
accuracy: 0.7186 - val_loss: 3.4567 - val_accuracy: 0.8503 - lr: 0.0025
Epoch 7/15
2645/2645 [==============================] - 371s 140ms/step - loss: 0.8918 -
accuracy: 0.7515 - val_loss: 1.6634 - val_accuracy: 0.8602 - lr: 0.0025
Epoch 8/15
2645/2645 [==============================] - 371s 140ms/step - loss: 0.7771 -
accuracy: 0.7804 - val_loss: 1.4707 - val_accuracy: 0.8629 - lr: 0.0025
Epoch 9/15
2645/2645 [==============================] - 373s 141ms/step - loss: 0.6875 -
accuracy: 0.8001 - val_loss: 2.5875 - val_accuracy: 0.8686 - lr: 0.0025
Epoch 10/15
2645/2645 [==============================] - 379s 143ms/step - loss: 0.6026 -
accuracy: 0.8224 - val_loss: 0.5113 - val_accuracy: 0.8747 - lr: 0.0025
Epoch 11/15
2645/2645 [==============================] - 370s 140ms/step - loss: 0.5222 -
accuracy: 0.8434 - val_loss: 0.6812 - val_accuracy: 0.8716 - lr: 0.0025
Epoch 12/15
2645/2645 [==============================] - 366s 138ms/step - loss: 0.4555 -
accuracy: 0.8627 - val_loss: 5.7728 - val_accuracy: 0.8712 - lr: 0.0025
Epoch 13/15
2645/2645 [==============================] - 370s 140ms/step - loss: 0.3345 -
accuracy: 0.8952 - val_loss: 3.2083 - val_accuracy: 0.8846 - lr: 0.0012
Epoch 14/15
2645/2645 [==============================] - 374s 141ms/step - loss: 0.2847 -
accuracy: 0.9092 - val_loss: 2.2157 - val_accuracy: 0.8781 - lr: 0.0012
Epoch 15/15
2645/2645 [==============================] - 365s 138ms/step - loss: 0.2310 -
accuracy: 0.9251 - val_loss: 1.8362 - val_accuracy: 0.8796 - lr: 6.2500e-04
```


Training and validation accuracy
0.8
0.6
0.4
0.2
0
2
4
6
8
10
12
14
Training accuracy
Validation accuracy



[27]: 'vgg16_tunned_entrega_2- test_loss: 4.894068241119385 - test_accuracy 0.9028571248054504'

<Figure size 640x480 with 0 Axes>

0.1 Inceptionv3

Esta arqiutectura convolucional también se introdujo en 2014, en este caso por Christian Szegedy, de Google. Se trata de una red convolucional de 48 capas de profundidad que utiliza modulos "inception", bloques convolucionales con filtros de diferentes tamaños (1x1, 3x3 y 5x5) que posteriormente concatena. Aunque parece una arquitectura más compleja resulta más eficiente computacionalmente que VGG16.

[28]:
```
from tensorflow.keras.applications.inception_v3 import preprocess_input
```

[29]:
```
train_data = ImageDataGenerator(preprocessing_function =␣
 ↪preprocess_input,rescale=1./255)
train_generator = train_data.flow_from_directory('100-bird-species/train/',
                                                 batch_size=32,
                                                 target_size=(224,224),
                                                  shuffle=True,
                                                 class_mode='categorical')
```

```
Found 84635 images belonging to 525 classes.
```

[30]:
```python
val_data = ImageDataGenerator(preprocessing_function =␣
 ↪preprocess_input,rescale=1./255)
val_generator = val_data.flow_from_directory('100-bird-species/valid/',
                                             batch_size=32,
                                             target_size=(224,224),
                                             shuffle=True,
                                             class_mode='categorical')
```

```
Found 2625 images belonging to 525 classes.
```

[31]:
```python
test_data = ImageDataGenerator(preprocessing_function =␣
 ↪preprocess_input,rescale=1./255)
test_generator = test_data.flow_from_directory('100-bird-species/test/',
                                               batch_size=32,
                                               target_size=(224,224),
                                               shuffle=True,
                                               class_mode='categorical')
```

```
Found 2625 images belonging to 525 classes.
```

[32]:
```python
#importamos las librerias
from tensorflow.keras.applications import InceptionV3
from tensorflow.keras import layers, models
from tensorflow.keras.callbacks import EarlyStopping, ModelCheckpoint,␣
 ↪ReduceLROnPlateau
from tensorflow.keras.optimizers import Adam
```

[33]:
```python
#Cargamos el base_model con los pesos de ImageNet y los congelamos
base_model = InceptionV3(weights='imagenet', include_top=False,␣
 ↪input_shape=(224,224,3))
base_model.trainable=False
```

0.1.1 Modelo base (Average Pooling)

[34]:
```python
#Creamos el top model (Clasificador)
Average_layer = layers.GlobalAveragePooling2D()
prediction_layer = layers.Dense(525, activation='softmax')

#Unimos los modelos con la api secuencialc
model = models.Sequential([
    base_model,
    Average_layer,
    prediction_layer
])

#Compilamos el modelo
```

```
model.compile(
    optimizer=Adam(learning_rate=0.03),
    loss='categorical_crossentropy',
    metrics=['accuracy'],
)

model.summary()
```

```
Model: "sequential_4"

_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 inception_v3 (Functional)   (None, 5, 5, 2048)        21802784

 global_average_pooling2d (G  (None, 2048)             0
 lobalAveragePooling2D)

 dense_13 (Dense)            (None, 525)               1075725

=================================================================
Total params: 22,878,509
Trainable params: 1,075,725
Non-trainable params: 21,802,784
_________________________________________________________________
```

[35]:
```
#Creamos directorio para guardar el modelo
count=0 #contador de experimentos iniciado, se actualiza al graficar
model_name=base_model.name+"_entrega_"+str(count)
model_path,graph_path=crea_directorio(model_name)
```

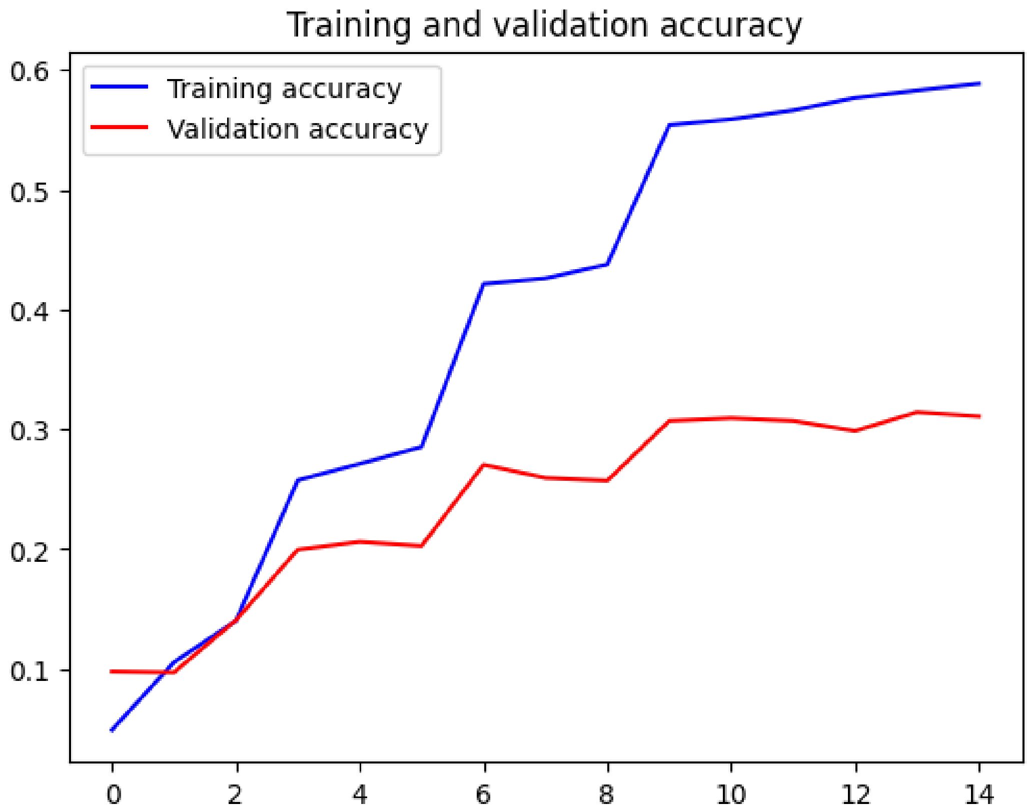
[36]:
```
#Definimos los callbacks
lr =ReduceLROnPlateau(monitor="val_loss",
                      factor=0.5,
                      patience=2)

es = EarlyStopping(monitor='val_accuracy',
                   mode='max', patience=5,
                   restore_best_weights=True)

mcp = ModelCheckpoint(filepath=model_path,
                      save_best_only=True,
                      monitor='val_accuracy',
                      mode='max')
```

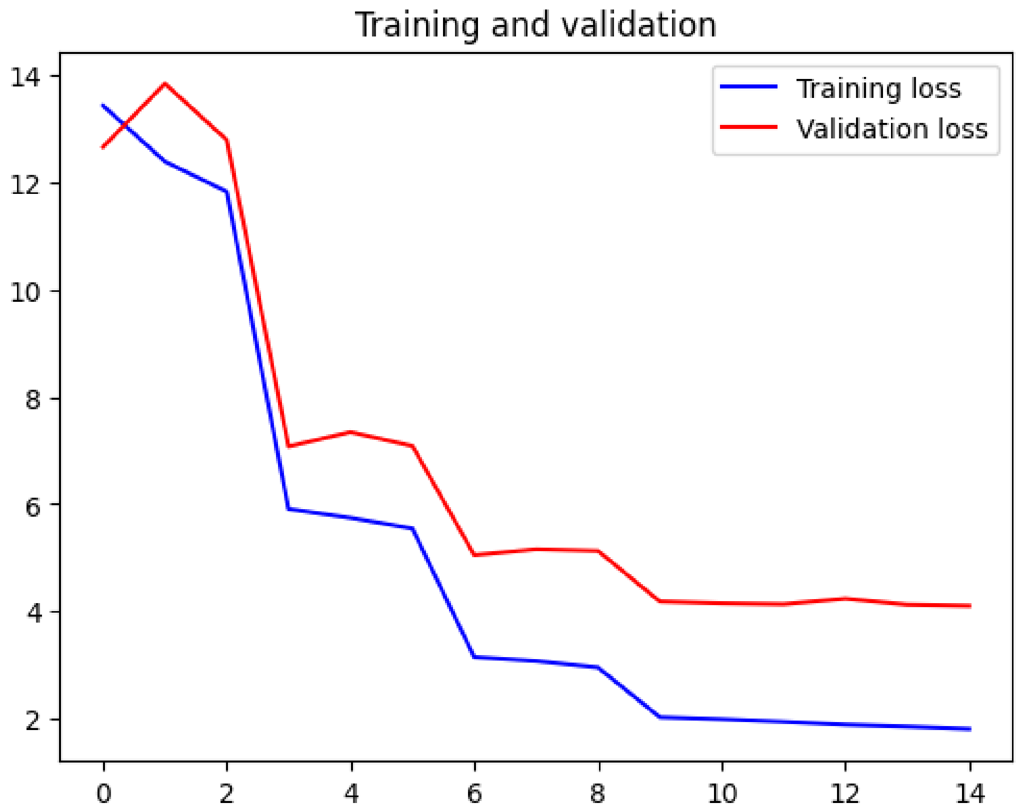
[37]:
```
#entrenamos el modelo
history=model.fit(train_generator, epochs=15, validation_data = val_generator,␣
 ↪callbacks=[es,lr,mcp])
```

```
Epoch 1/15
2645/2645 [==============================] - 185s 68ms/step - loss: 13.4386 -
accuracy: 0.0491 - val_loss: 12.6641 - val_accuracy: 0.0979 - lr: 0.0300
Epoch 2/15
2645/2645 [==============================] - 177s 67ms/step - loss: 12.3933 -
accuracy: 0.1056 - val_loss: 13.8489 - val_accuracy: 0.0971 - lr: 0.0300
Epoch 3/15
2645/2645 [==============================] - 186s 70ms/step - loss: 11.8331 -
accuracy: 0.1401 - val_loss: 12.7971 - val_accuracy: 0.1406 - lr: 0.0300
Epoch 4/15
2645/2645 [==============================] - 176s 67ms/step - loss: 5.9058 -
accuracy: 0.2575 - val_loss: 7.0758 - val_accuracy: 0.1996 - lr: 0.0150
Epoch 5/15
2645/2645 [==============================] - 170s 64ms/step - loss: 5.7413 -
accuracy: 0.2712 - val_loss: 7.3414 - val_accuracy: 0.2061 - lr: 0.0150
Epoch 6/15
2645/2645 [==============================] - 171s 65ms/step - loss: 5.5433 -
accuracy: 0.2853 - val_loss: 7.0848 - val_accuracy: 0.2027 - lr: 0.0150
Epoch 7/15
2645/2645 [==============================] - 181s 69ms/step - loss: 3.1418 -
accuracy: 0.4214 - val_loss: 5.0456 - val_accuracy: 0.2705 - lr: 0.0075
Epoch 8/15
2645/2645 [==============================] - 194s 73ms/step - loss: 3.0687 -
accuracy: 0.4260 - val_loss: 5.1551 - val_accuracy: 0.2594 - lr: 0.0075
Epoch 9/15
2645/2645 [==============================] - 185s 70ms/step - loss: 2.9489 -
accuracy: 0.4376 - val_loss: 5.1257 - val_accuracy: 0.2571 - lr: 0.0075
Epoch 10/15
2645/2645 [==============================] - 177s 67ms/step - loss: 2.0195 -
accuracy: 0.5540 - val_loss: 4.1790 - val_accuracy: 0.3070 - lr: 0.0037
Epoch 11/15
2645/2645 [==============================] - 179s 68ms/step - loss: 1.9811 -
accuracy: 0.5589 - val_loss: 4.1454 - val_accuracy: 0.3093 - lr: 0.0037
Epoch 12/15
2645/2645 [==============================] - 183s 69ms/step - loss: 1.9342 -
accuracy: 0.5662 - val_loss: 4.1290 - val_accuracy: 0.3070 - lr: 0.0037
Epoch 13/15
2645/2645 [==============================] - 177s 67ms/step - loss: 1.8797 -
accuracy: 0.5767 - val_loss: 4.2328 - val_accuracy: 0.2987 - lr: 0.0037
Epoch 14/15
2645/2645 [==============================] - 184s 69ms/step - loss: 1.8436 -
accuracy: 0.5829 - val_loss: 4.1197 - val_accuracy: 0.3143 - lr: 0.0037
Epoch 15/15
2645/2645 [==============================] - 187s 71ms/step - loss: 1.8001 -
accuracy: 0.5885 - val_loss: 4.0989 - val_accuracy: 0.3109 - lr: 0.0037
```

[38]:

```
plot_curves(model.history,graph_path,count)
```


Training and validation accuracy
Training accuracy
Validation accuracy
0.6
0.5
0.4
0.3
0.2
0.1
0
2
4
6
8
10
12
14



<Figure size 640x480 with 0 Axes>

[39]:
```
evalua_modelo(model,test_generator,model_name,resultados)
```

[39]: 'inception_v3_entrega_0- test_loss: 4.006126880645752 - test_accuracy 0.3310476243495941'

0.1.2 Modelo Base (MaxPooling)

[40]:
```
#Creamos el top model (Clasificador)
Max_layer = layers.GlobalMaxPooling2D()
prediction_layer = layers.Dense(525, activation='softmax')

#Unimos los modelos con la api secuencialc
model = models.Sequential([
    base_model,
    Max_layer,
    prediction_layer
])

#Compilamos el modelo
```

```
model.compile(
    optimizer=Adam(learning_rate=0.03),
    loss='categorical_crossentropy',
    metrics=['accuracy'],
)

model.summary()
```

```
Model: "sequential_5"

_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 inception_v3 (Functional)   (None, 5, 5, 2048)        21802784

 global_max_pooling2d (Globa  (None, 2048)             0
 lMaxPooling2D)

 dense_14 (Dense)            (None, 525)               1075725

=================================================================
Total params: 22,878,509
Trainable params: 1,075,725
Non-trainable params: 21,802,784
_________________________________________________________________
```

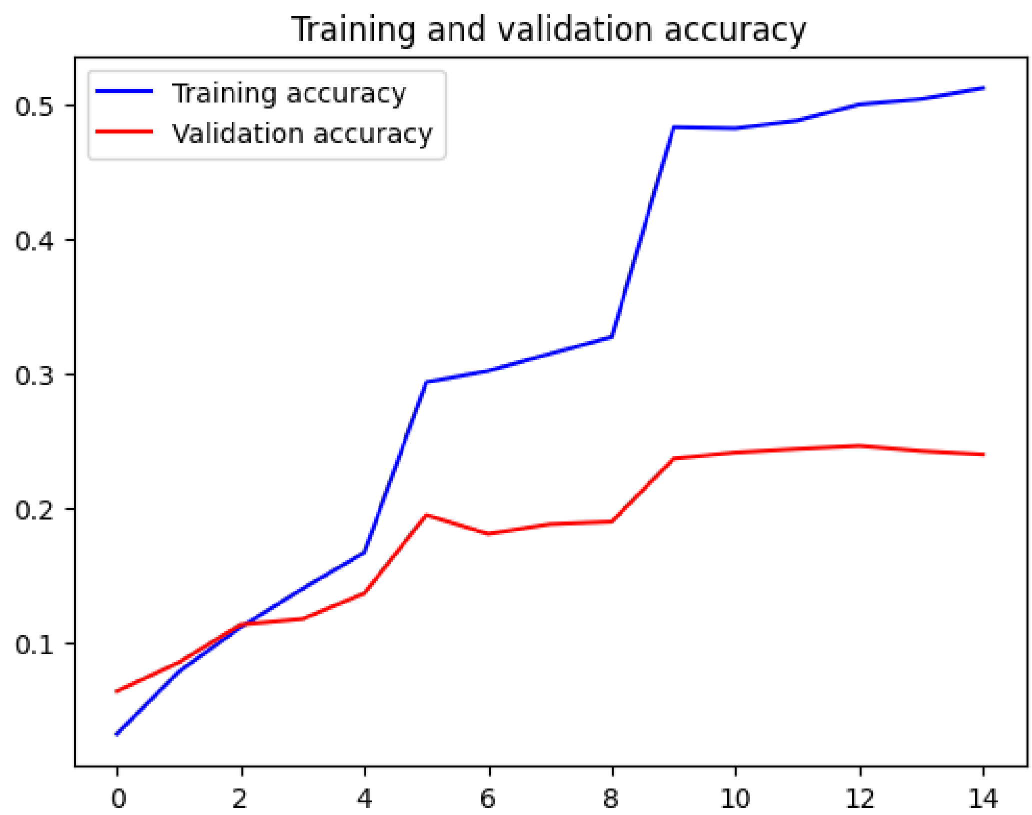
[41]:
```
#Creamos directorio para guardar el modelo
count=0 #contador de experimentos iniciado, se actualiza al graficar
model_name=base_model.name+"_max_entrega_"+str(count)
model_path,graph_path=crea_directorio(model_name)
```

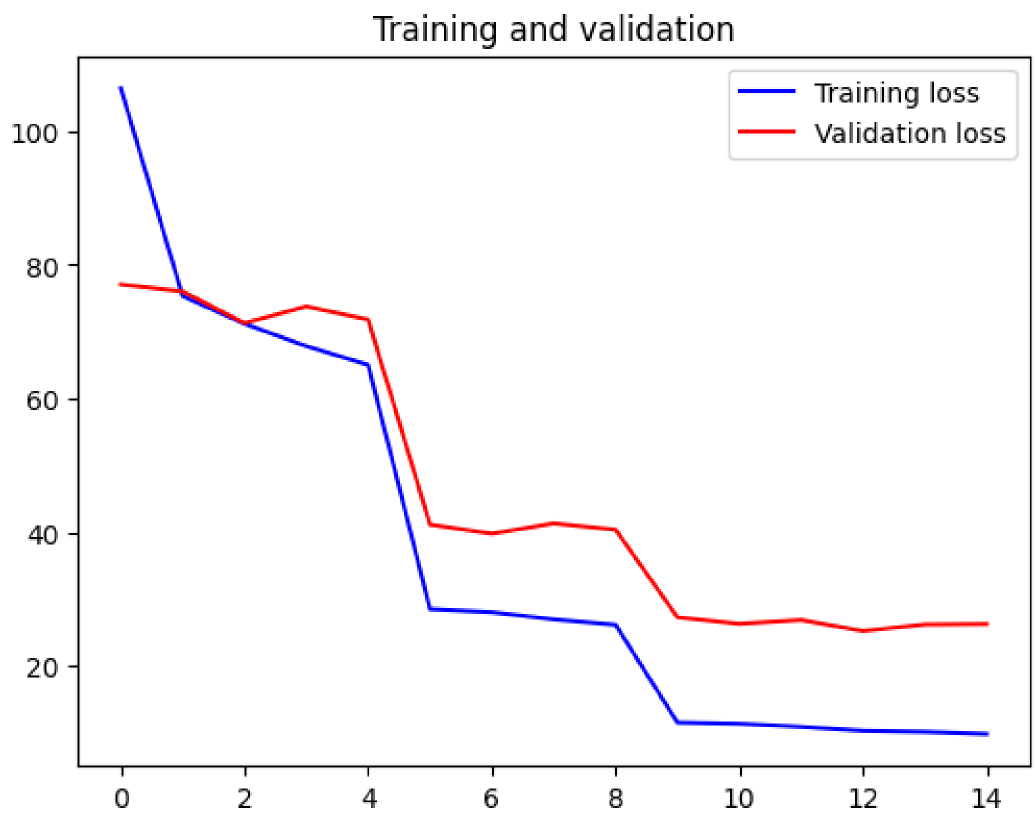
[42]:
```
#Definimos los callbacks
lr =ReduceLROnPlateau(monitor="val_loss",
                      factor=0.5,
                      patience=2)

es = EarlyStopping(monitor='val_accuracy',
                   mode='max', patience=5,
                   restore_best_weights=True)

mcp = ModelCheckpoint(filepath=model_path,
                      save_best_only=True,
                      monitor='val_accuracy',
                      mode='max')
```

[43]:
```
#entrenamos el modelo
history=model.fit(train_generator, epochs=15, validation_data = val_generator,
 ↪callbacks=[es,lr,mcp])
```

```
Epoch 1/15
2645/2645 [==============================] - 180s 67ms/step - loss: 106.4178 -
accuracy: 0.0322 - val_loss: 77.0727 - val_accuracy: 0.0640 - lr: 0.0300
Epoch 2/15
2645/2645 [==============================] - 185s 70ms/step - loss: 75.3160 -
accuracy: 0.0785 - val_loss: 76.0156 - val_accuracy: 0.0853 - lr: 0.0300
Epoch 3/15
2645/2645 [==============================] - 187s 71ms/step - loss: 71.1766 -
accuracy: 0.1115 - val_loss: 71.2899 - val_accuracy: 0.1135 - lr: 0.0300
Epoch 4/15
2645/2645 [==============================] - 182s 69ms/step - loss: 67.8334 -
accuracy: 0.1402 - val_loss: 73.7807 - val_accuracy: 0.1177 - lr: 0.0300
Epoch 5/15
2645/2645 [==============================] - 189s 72ms/step - loss: 65.0344 -
accuracy: 0.1672 - val_loss: 71.8182 - val_accuracy: 0.1368 - lr: 0.0300
Epoch 6/15
2645/2645 [==============================] - 182s 69ms/step - loss: 28.5123 -
accuracy: 0.2938 - val_loss: 41.1550 - val_accuracy: 0.1950 - lr: 0.0150
Epoch 7/15
2645/2645 [==============================] - 181s 68ms/step - loss: 28.0524 -
accuracy: 0.3023 - val_loss: 39.7950 - val_accuracy: 0.1810 - lr: 0.0150
Epoch 8/15
2645/2645 [==============================] - 191s 72ms/step - loss: 26.9821 -
accuracy: 0.3150 - val_loss: 41.3317 - val_accuracy: 0.1882 - lr: 0.0150
Epoch 9/15
2645/2645 [==============================] - 177s 67ms/step - loss: 26.1769 -
accuracy: 0.3275 - val_loss: 40.3937 - val_accuracy: 0.1901 - lr: 0.0150
Epoch 10/15
2645/2645 [==============================] - 183s 69ms/step - loss: 11.5538 -
accuracy: 0.4836 - val_loss: 27.2937 - val_accuracy: 0.2370 - lr: 0.0075
Epoch 11/15
2645/2645 [==============================] - 176s 66ms/step - loss: 11.3758 -
accuracy: 0.4827 - val_loss: 26.3195 - val_accuracy: 0.2415 - lr: 0.0075
Epoch 12/15
2645/2645 [==============================] - 186s 70ms/step - loss: 10.9162 -
accuracy: 0.4886 - val_loss: 26.8867 - val_accuracy: 0.2442 - lr: 0.0075
Epoch 13/15
2645/2645 [==============================] - 190s 72ms/step - loss: 10.3186 -
accuracy: 0.5005 - val_loss: 25.2436 - val_accuracy: 0.2465 - lr: 0.0075
Epoch 14/15
2645/2645 [==============================] - 186s 70ms/step - loss: 10.1598 -
accuracy: 0.5045 - val_loss: 26.2185 - val_accuracy: 0.2427 - lr: 0.0075
Epoch 15/15
2645/2645 [==============================] - 189s 72ms/step - loss: 9.8549 -
accuracy: 0.5126 - val_loss: 26.2917 - val_accuracy: 0.2400 - lr: 0.0075
```

[44]:
```
plot_curves(model.history,graph_path,count)
```


Training and validation accuracy
Training accuracy
Validation accuracy
0.5
0.4
0.3
0.2
0.1
0
2
4
6
8
10
12
14




```
<Figure size 640x480 with 0 Axes>
```

[45]:
```
evalua_modelo(model,test_generator,model_name,resultados)
```

[45]:
```
'inception_v3_max_entrega_0- test_loss: 25.72496795654297 - test_accuracy
0.2579047679901123'
```

0.1.3 Modelo Data Augmentation

[46]:
```
data_augmentation = tf.keras.Sequential([
    tf.keras.layers.RandomFlip(mode="horizontal", seed=42),
    tf.keras.layers.RandomRotation(factor=0.05, seed=42),
    tf.keras.layers.RandomZoom(0.05, seed=42),
])
```

[47]:
```
base_model = tf.keras.applications.InceptionV3(include_top=␣
  ↪False,weights='imagenet')

# 2. Freeze the base model
base_model.trainable = False
model=tf.keras.Sequential([
```

```
    tf.keras.layers.Input(shape =(224,224,3), name = "input-layer"),
    data_augmentation,
    base_model,
    tf.keras.layers.GlobalAveragePooling2D(name =␣
 ↪"global_average_pooling_layer"),
    tf.keras.layers.BatchNormalization(),
    tf.keras.layers.Dropout(0.3),
    tf.keras.layers.Dense(256,activation="relu"),
    tf.keras.layers.BatchNormalization(),
    tf.keras.layers.Dropout(0.3),
    tf.keras.layers.Dense(525, activation = "softmax", name = "output-layer")
])


# 9. Compile the model
model.compile(loss = "categorical_crossentropy",
                optimizer = tf.keras.optimizers.Adam(learning_rate = 0.01),
                metrics = ["accuracy"])
model.summary()
```

```
Model: "sequential_7"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 sequential_6 (Sequential)   (None, 224, 224, 3)       0

 inception_v3 (Functional)   (None, None, None, 2048)  21802784

 global_average_pooling_laye  (None, 2048)             0
 r (GlobalAveragePooling2D)

 batch_normalization_192 (Ba  (None, 2048)            8192
 tchNormalization)

 dropout_4 (Dropout)         (None, 2048)              0

 dense_15 (Dense)            (None, 256)               524544

 batch_normalization_193 (Ba  (None, 256)             1024
 tchNormalization)

 dropout_5 (Dropout)         (None, 256)               0

 output-layer (Dense)        (None, 525)               134925

=================================================================
Total params: 22,471,469
Trainable params: 664,077
```

```
Non-trainable params: 21,807,392
_________________________________________________________________
```

[48]:
```
#Creamos directorio para guardar el modelo
count=1
model_name=base_model.name+"_Data_Aug_Layers_Entrega"+str(count)
model_path,graph_path=crea_directorio(model_name)
```

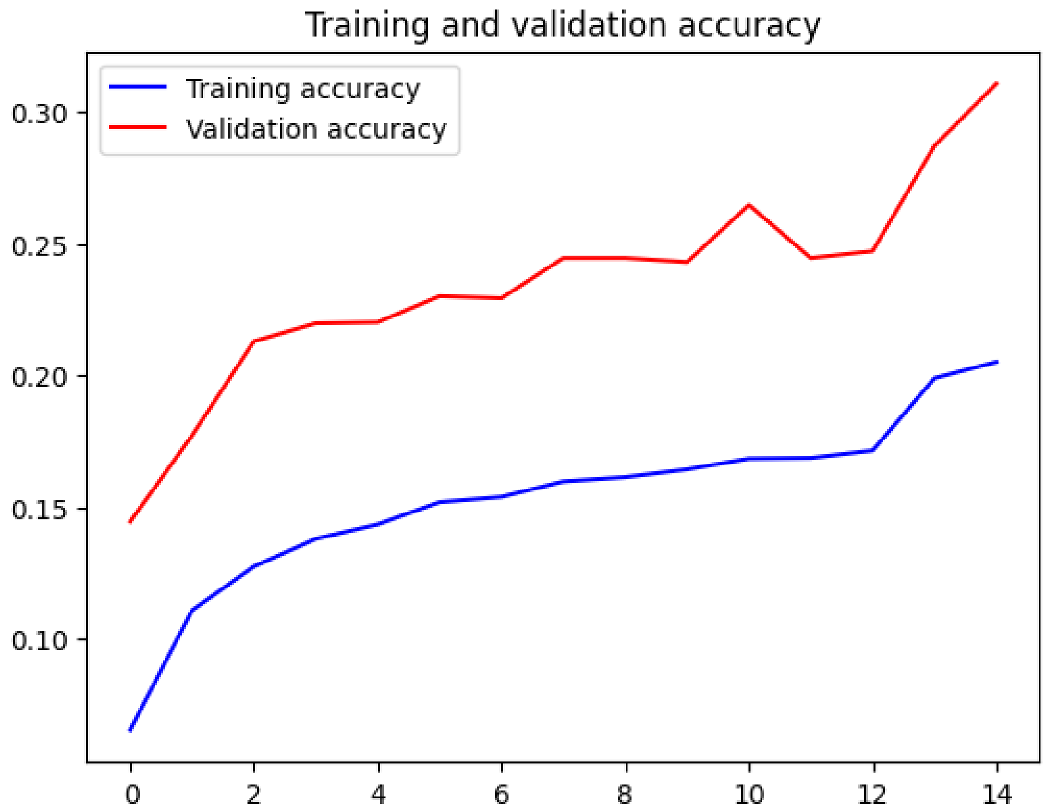
[49]:
```
#Definimos los callbacks
lr =ReduceLROnPlateau(monitor="val_loss",
                      factor=0.5,
                      patience=2)

es = EarlyStopping(monitor='val_accuracy',
                   mode='max', patience=5,
                   restore_best_weights=True)

mcp = ModelCheckpoint(filepath=model_path,
                      save_best_only=True,
                      monitor='val_accuracy',
                      mode='max')
```

[50]:
```
#entrenamos el modelo
history=model.fit(train_generator, epochs=15, validation_data = val_generator,␣
 ↪callbacks=[es,lr,mcp])
```

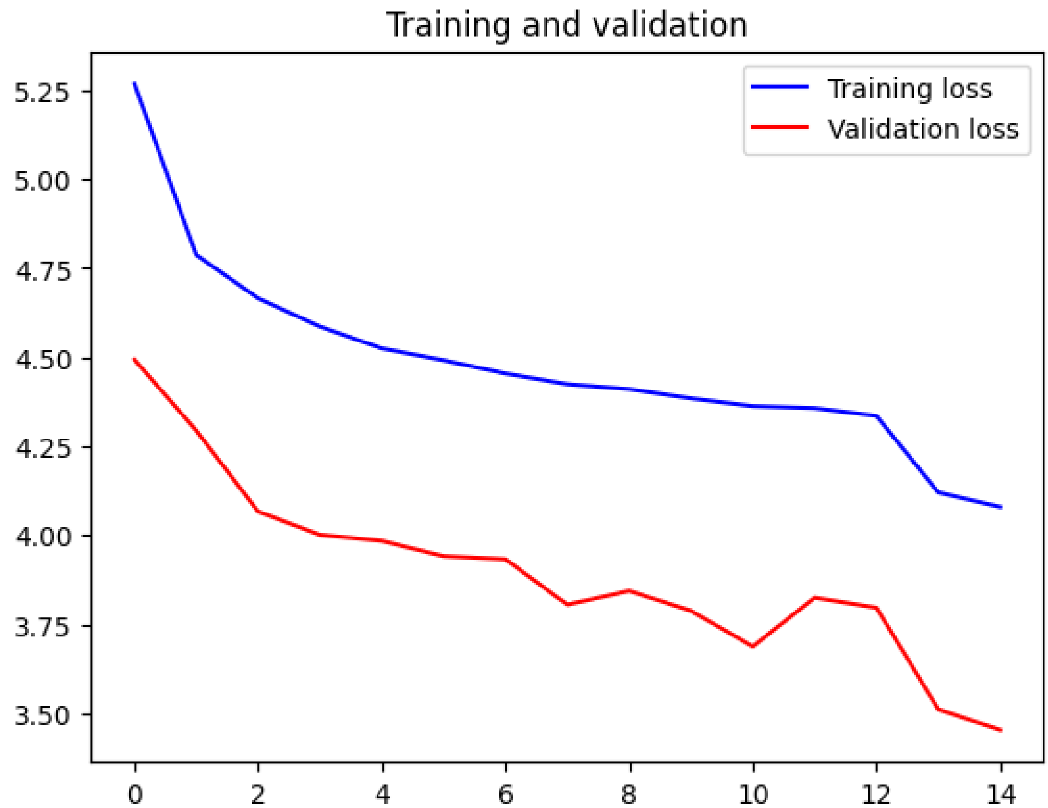
```
Epoch 1/15
2645/2645 [==============================] - 194s 72ms/step - loss: 5.2676 -
accuracy: 0.0652 - val_loss: 4.4933 - val_accuracy: 0.1444 - lr: 0.0100
Epoch 2/15
2645/2645 [==============================] - 199s 75ms/step - loss: 4.7868 -
accuracy: 0.1107 - val_loss: 4.2938 - val_accuracy: 0.1771 - lr: 0.0100
Epoch 3/15
2645/2645 [==============================] - 201s 76ms/step - loss: 4.6652 -
accuracy: 0.1273 - val_loss: 4.0664 - val_accuracy: 0.2130 - lr: 0.0100
Epoch 4/15
2645/2645 [==============================] - 185s 70ms/step - loss: 4.5856 -
accuracy: 0.1379 - val_loss: 4.0010 - val_accuracy: 0.2198 - lr: 0.0100
Epoch 5/15
2645/2645 [==============================] - 207s 78ms/step - loss: 4.5246 -
accuracy: 0.1433 - val_loss: 3.9841 - val_accuracy: 0.2202 - lr: 0.0100
Epoch 6/15
2645/2645 [==============================] - 186s 70ms/step - loss: 4.4912 -
accuracy: 0.1518 - val_loss: 3.9410 - val_accuracy: 0.2301 - lr: 0.0100
Epoch 7/15
2645/2645 [==============================] - 191s 72ms/step - loss: 4.4540 -
accuracy: 0.1538 - val_loss: 3.9326 - val_accuracy: 0.2293 - lr: 0.0100
Epoch 8/15
```

```
2645/2645 [==============================] - 198s 75ms/step - loss: 4.4241 -
accuracy: 0.1597 - val_loss: 3.8050 - val_accuracy: 0.2446 - lr: 0.0100
Epoch 9/15
2645/2645 [==============================] - 199s 75ms/step - loss: 4.4100 -
accuracy: 0.1613 - val_loss: 3.8438 - val_accuracy: 0.2446 - lr: 0.0100
Epoch 10/15
2645/2645 [==============================] - 191s 72ms/step - loss: 4.3841 -
accuracy: 0.1642 - val_loss: 3.7879 - val_accuracy: 0.2430 - lr: 0.0100
Epoch 11/15
2645/2645 [==============================] - 210s 79ms/step - loss: 4.3624 -
accuracy: 0.1683 - val_loss: 3.6876 - val_accuracy: 0.2648 - lr: 0.0100
Epoch 12/15
2645/2645 [==============================] - 190s 72ms/step - loss: 4.3568 -
accuracy: 0.1686 - val_loss: 3.8247 - val_accuracy: 0.2446 - lr: 0.0100
Epoch 13/15
2645/2645 [==============================] - 190s 72ms/step - loss: 4.3351 -
accuracy: 0.1714 - val_loss: 3.7964 - val_accuracy: 0.2472 - lr: 0.0100
Epoch 14/15
2645/2645 [==============================] - 193s 73ms/step - loss: 4.1201 -
accuracy: 0.1989 - val_loss: 3.5111 - val_accuracy: 0.2872 - lr: 0.0050
Epoch 15/15
2645/2645 [==============================] - 213s 81ms/step - loss: 4.0795 -
accuracy: 0.2051 - val_loss: 3.4536 - val_accuracy: 0.3109 - lr: 0.0050
```

[51]:
```
plot_curves(history,graph_path,count)
```



Training and validation accuracy
Training accuracy
Validation accuracy
0.30
0.25
0.20
0.15
0.10
0
2
4
6
8
10
12
14



<Figure size 640x480 with 0 Axes>

[52]:
```
evalua_modelo(model,test_generator,model_name,resultados)
```

[52]:
```
'inception_v3_Data_Aug_Layers_Entrega1- test_loss: 3.38978910446167 -
test_accuracy 0.31771427392959595'
```

0.1.4 Fine Tune 1

[53]:
```
data_augmentation = tf.keras.Sequential([
    tf.keras.layers.RandomFlip(mode="horizontal", seed=42),
    tf.keras.layers.RandomRotation(factor=0.05, seed=42),
    tf.keras.layers.RandomZoom(0.05, seed=42),
])
```

[54]:
```
base_model = tf.keras.applications.InceptionV3(include_top=␣
 ↪False,weights='imagenet')

# 2. Freeze the base model
base_model.trainable = True
for layer in base_model.layers[:-22]:
```

```
  layer.trainable = False

model=tf.keras.Sequential([
    tf.keras.layers.Input(shape =(224,224,3), name = "input-layer"),
    data_augmentation,
    base_model,
    tf.keras.layers.GlobalAveragePooling2D(name =␣
 ↪"global_average_pooling_layer"),
    tf.keras.layers.Dense(525, activation = "softmax", name = "output-layer")
])


# 9. Compile the model
model.compile(loss = "categorical_crossentropy",
                optimizer = tf.keras.optimizers.Adam(learning_rate = 0.01),
                metrics = ["accuracy"])
model.summary()
```

```
Model: "sequential_9"

_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 sequential_8 (Sequential)   (None, 224, 224, 3)       0

 inception_v3 (Functional)   (None, None, None, 2048)  21802784

 global_average_pooling_laye  (None, 2048)             0
 r (GlobalAveragePooling2D)

 output-layer (Dense)        (None, 525)               1075725

=================================================================
Total params: 22,878,509
Trainable params: 3,895,821
Non-trainable params: 18,982,688

_________________________________________________________________
```

[55]:
```
#Creamos directorio para guardar el modelo
count=1
model_name=base_model.name+"_FT_entrega"+str(count)
model_path,graph_path=crea_directorio(model_name)
```

[56]:
```
#Definimos los callbacks
lr =ReduceLROnPlateau(monitor="val_loss",
                      factor=0.5,
                      patience=2)
```

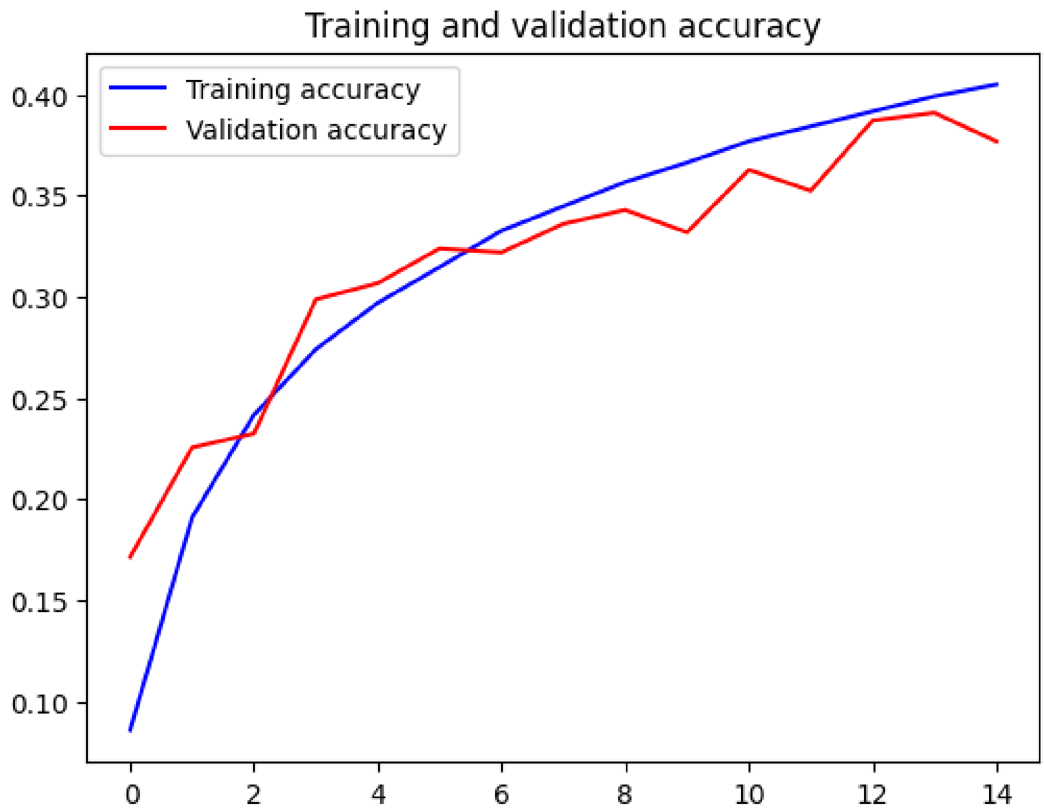
```
es = EarlyStopping(monitor='val_accuracy',
                   mode='max', patience=5,
                   restore_best_weights=True)

mcp = ModelCheckpoint(filepath=model_path,
                      save_best_only=True,
                      monitor='val_accuracy',
                      mode='max')
```

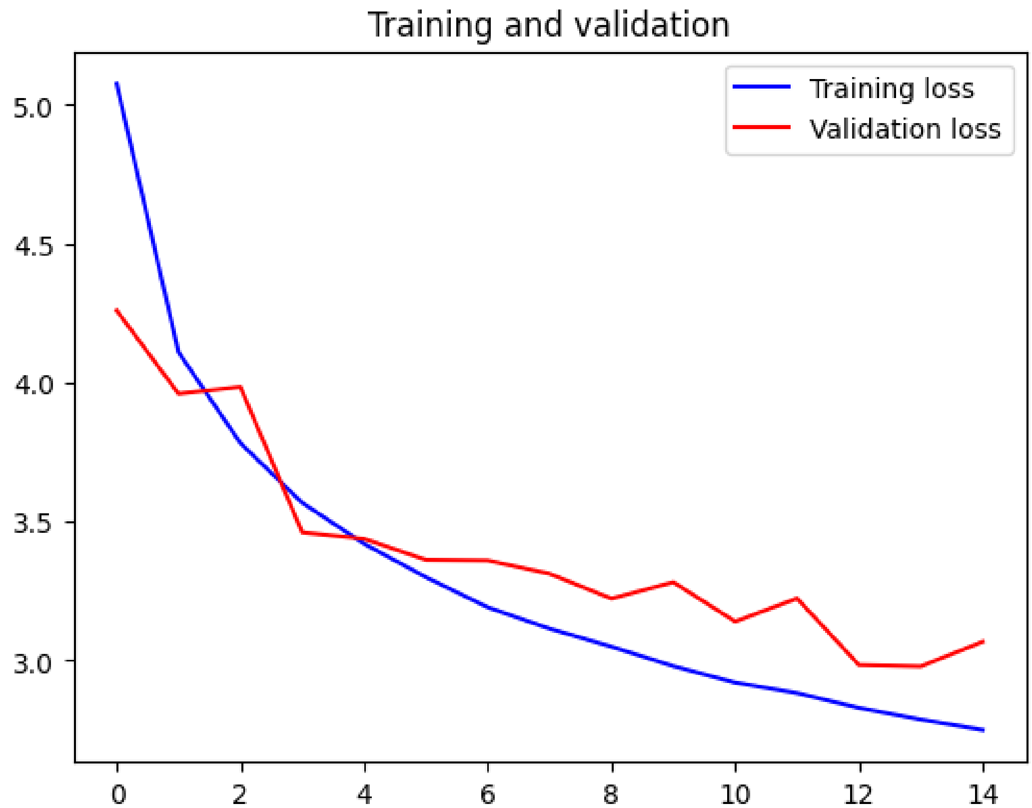
[57]:
```
#entrenamos el modelo
history=model.fit(train_generator, epochs=15, validation_data = val_generator,␣
 ↪callbacks=[es,lr,mcp])
```

```
Epoch 1/15
2645/2645 [==============================] - 214s 79ms/step - loss: 5.0753 -
accuracy: 0.0859 - val_loss: 4.2593 - val_accuracy: 0.1714 - lr: 0.0100
Epoch 2/15
2645/2645 [==============================] - 203s 77ms/step - loss: 4.1093 -
accuracy: 0.1909 - val_loss: 3.9597 - val_accuracy: 0.2255 - lr: 0.0100
Epoch 3/15
2645/2645 [==============================] - 206s 78ms/step - loss: 3.7820 -
accuracy: 0.2416 - val_loss: 3.9841 - val_accuracy: 0.2324 - lr: 0.0100
Epoch 4/15
2645/2645 [==============================] - 203s 77ms/step - loss: 3.5671 -
accuracy: 0.2740 - val_loss: 3.4607 - val_accuracy: 0.2987 - lr: 0.0100
Epoch 5/15
2645/2645 [==============================] - 213s 81ms/step - loss: 3.4199 -
accuracy: 0.2969 - val_loss: 3.4376 - val_accuracy: 0.3067 - lr: 0.0100
Epoch 6/15
2645/2645 [==============================] - 209s 79ms/step - loss: 3.2993 -
accuracy: 0.3146 - val_loss: 3.3620 - val_accuracy: 0.3238 - lr: 0.0100
Epoch 7/15
2645/2645 [==============================] - 207s 78ms/step - loss: 3.1911 -
accuracy: 0.3326 - val_loss: 3.3597 - val_accuracy: 0.3219 - lr: 0.0100
Epoch 8/15
2645/2645 [==============================] - 196s 74ms/step - loss: 3.1141 -
accuracy: 0.3447 - val_loss: 3.3116 - val_accuracy: 0.3360 - lr: 0.0100
Epoch 9/15
2645/2645 [==============================] - 203s 77ms/step - loss: 3.0494 -
accuracy: 0.3566 - val_loss: 3.2223 - val_accuracy: 0.3429 - lr: 0.0100
Epoch 10/15
2645/2645 [==============================] - 201s 76ms/step - loss: 2.9793 -
accuracy: 0.3663 - val_loss: 3.2813 - val_accuracy: 0.3318 - lr: 0.0100
Epoch 11/15
2645/2645 [==============================] - 204s 77ms/step - loss: 2.9204 -
accuracy: 0.3768 - val_loss: 3.1400 - val_accuracy: 0.3627 - lr: 0.0100
Epoch 12/15
2645/2645 [==============================] - 202s 76ms/step - loss: 2.8824 -
```

```
accuracy: 0.3842 - val_loss: 3.2231 - val_accuracy: 0.3524 - lr: 0.0100
Epoch 13/15
2645/2645 [==============================] - 206s 78ms/step - loss: 2.8293 -
accuracy: 0.3916 - val_loss: 2.9837 - val_accuracy: 0.3870 - lr: 0.0100
Epoch 14/15
2645/2645 [==============================] - 196s 74ms/step - loss: 2.7874 -
accuracy: 0.3990 - val_loss: 2.9786 - val_accuracy: 0.3909 - lr: 0.0100
Epoch 15/15
2645/2645 [==============================] - 199s 75ms/step - loss: 2.7501 -
accuracy: 0.4048 - val_loss: 3.0670 - val_accuracy: 0.3768 - lr: 0.0100
```

[58]:
```
plot_curves(history,graph_path,count)
```






```
<Figure size 640x480 with 0 Axes>
```

[59]:
```
evalua_modelo(model,test_generator,model_name,resultados)
```

[59]:
```
'inception_v3_FT_entrega1- test_loss: 3.0062930583953857 - test_accuracy
0.38514286279678345'
```

0.1.5 Fine Tunne 2

[60]:
```
data_augmentation = tf.keras.Sequential([
    tf.keras.layers.RandomFlip(mode="horizontal", seed=42),
    tf.keras.layers.RandomRotation(factor=0.05, seed=42),
    tf.keras.layers.RandomZoom(0.05, seed=42),
])
```

[61]:
```
base_model = tf.keras.applications.InceptionV3(include_top=␣
↪False,weights='imagenet')

# 2. Freeze the base model
base_model.trainable = True
for layer in base_model.layers[:-22]:
```

```
  layer.trainable = False

model=tf.keras.Sequential([
    tf.keras.layers.Input(shape =(224,224,3), name = "input-layer"),
    data_augmentation,
    base_model,
    tf.keras.layers.GlobalAveragePooling2D(name =␣
 ↪"global_average_pooling_layer"),
    tf.keras.layers.Activation("relu"),
    tf.keras.layers.BatchNormalization(),
    tf.keras.layers.Dropout(0.3),
    tf.keras.layers.Dense(2048, activation="relu"),
    tf.keras.layers.BatchNormalization(),
    tf.keras.layers.Dropout(0.3),
    tf.keras.layers.Dense(525, activation = "softmax", name = "output-layer")
])


# 9. Compile the model
model.compile(loss = "categorical_crossentropy",
                optimizer = tf.keras.optimizers.Adam(learning_rate = 0.0003),
                metrics = ["accuracy"])
model.summary()
```

```
Model: "sequential_11"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 sequential_10 (Sequential)  (None, 224, 224, 3)       0

 inception_v3 (Functional)   (None, None, None, 2048)  21802784

 global_average_pooling_laye  (None, 2048)             0
 r (GlobalAveragePooling2D)

 activation_376 (Activation)  (None, 2048)             0

 batch_normalization_382 (Ba  (None, 2048)             8192
 tchNormalization)

 dropout_6 (Dropout)         (None, 2048)              0

 dense_16 (Dense)            (None, 2048)              4196352

 batch_normalization_383 (Ba  (None, 2048)             8192
 tchNormalization)

 dropout_7 (Dropout)         (None, 2048)              0
```

```
 output-layer (Dense)        (None, 525)              1075725

=================================================================
Total params: 27,091,245
Trainable params: 8,100,365
Non-trainable params: 18,990,880
_________________________________________________________________
```

[62]:

```
#Creamos directorio para guardar el modelo
count=2
model_name=base_model.name+"_FT_entrega_"+str(count)
model_path,graph_path=crea_directorio(model_name)
```

[63]:

```
#Definimos los callbacks
lr =ReduceLROnPlateau(monitor="val_loss",
                      factor=0.5,
                      patience=2)

es = EarlyStopping(monitor='val_accuracy',
                   mode='max', patience=5,
                   restore_best_weights=True)

mcp = ModelCheckpoint(filepath=model_path,
                      save_best_only=True,
                      monitor='val_accuracy',
                      mode='max')
```

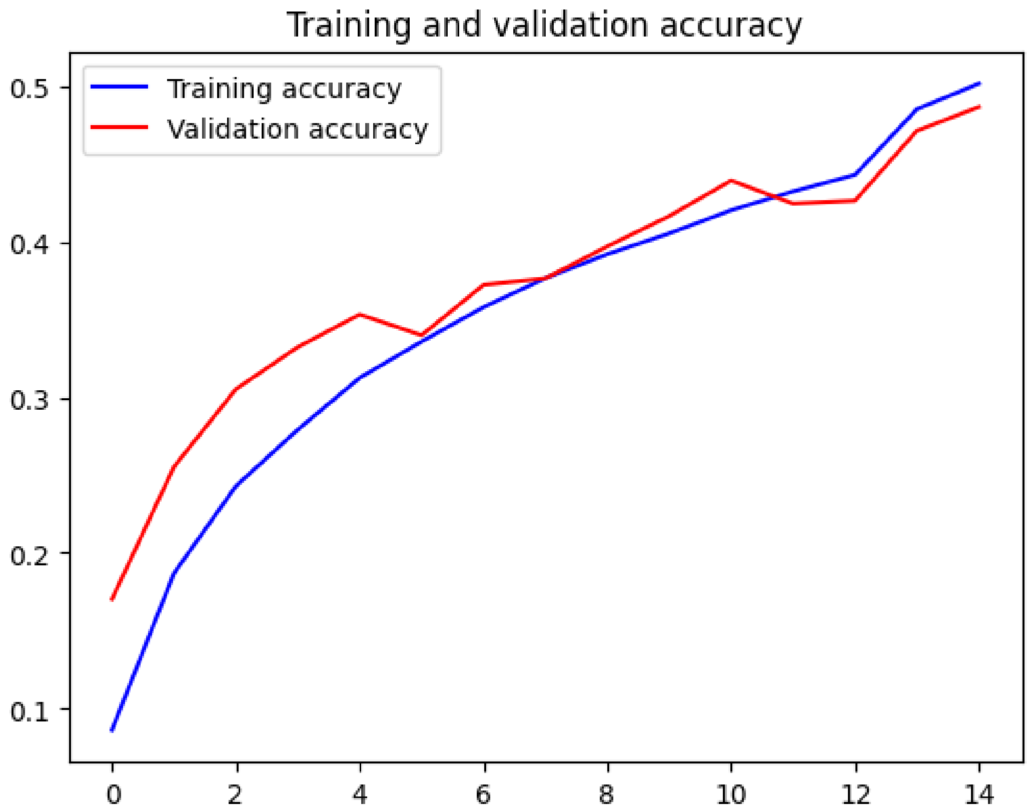
[64]:

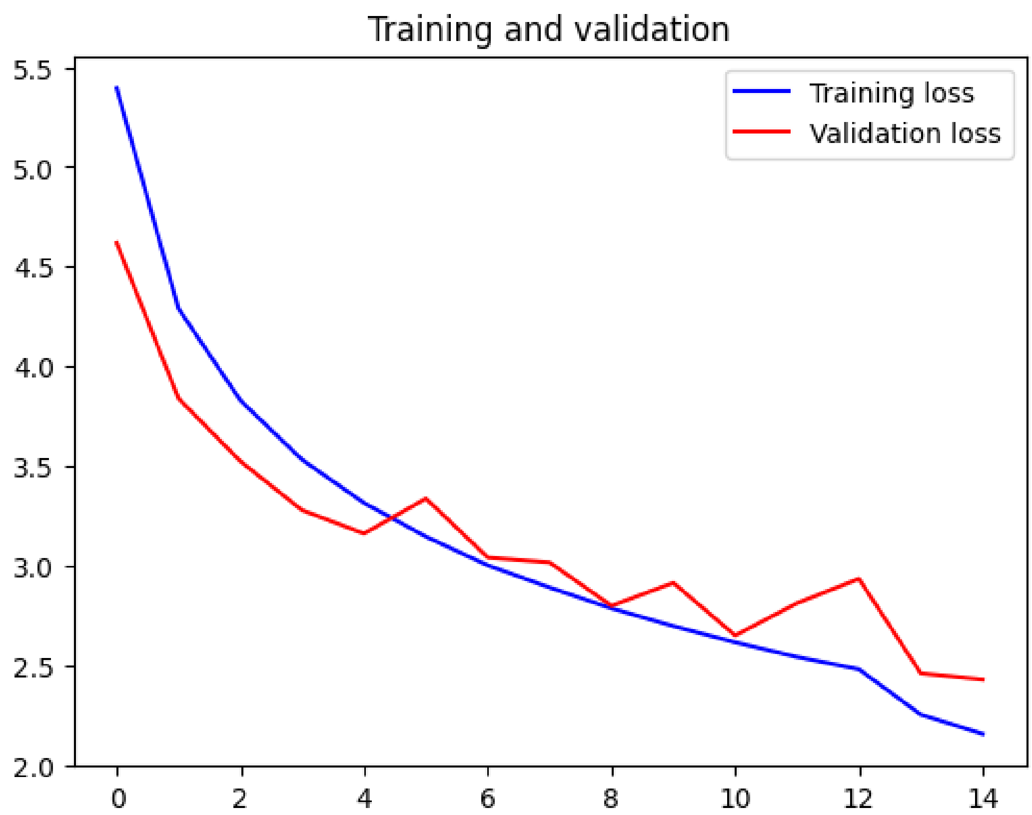
```
#entrenamos el modelo
history=model.fit(train_generator, epochs=15, validation_data = val_generator,␣
 ↪callbacks=[es,lr,mcp])
```

```
Epoch 1/15
2645/2645 [==============================] - 212s 78ms/step - loss: 5.3945 -
accuracy: 0.0863 - val_loss: 4.6188 - val_accuracy: 0.1703 - lr: 3.0000e-04
Epoch 2/15
2645/2645 [==============================] - 225s 85ms/step - loss: 4.2896 -
accuracy: 0.1868 - val_loss: 3.8394 - val_accuracy: 0.2552 - lr: 3.0000e-04
Epoch 3/15
2645/2645 [==============================] - 205s 78ms/step - loss: 3.8296 -
accuracy: 0.2429 - val_loss: 3.5244 - val_accuracy: 0.3051 - lr: 3.0000e-04
Epoch 4/15
2645/2645 [==============================] - 208s 79ms/step - loss: 3.5318 -
accuracy: 0.2791 - val_loss: 3.2788 - val_accuracy: 0.3322 - lr: 3.0000e-04
Epoch 5/15
2645/2645 [==============================] - 225s 85ms/step - loss: 3.3154 -
accuracy: 0.3124 - val_loss: 3.1621 - val_accuracy: 0.3531 - lr: 3.0000e-04
Epoch 6/15
```

```
2645/2645 [==============================] - 212s 80ms/step - loss: 3.1467 -
accuracy: 0.3358 - val_loss: 3.3376 - val_accuracy: 0.3398 - lr: 3.0000e-04
Epoch 7/15
2645/2645 [==============================] - 211s 80ms/step - loss: 3.0027 -
accuracy: 0.3579 - val_loss: 3.0422 - val_accuracy: 0.3722 - lr: 3.0000e-04
Epoch 8/15
2645/2645 [==============================] - 213s 81ms/step - loss: 2.8918 -
accuracy: 0.3765 - val_loss: 3.0173 - val_accuracy: 0.3764 - lr: 3.0000e-04
Epoch 9/15
2645/2645 [==============================] - 213s 80ms/step - loss: 2.7874 -
accuracy: 0.3919 - val_loss: 2.8004 - val_accuracy: 0.3970 - lr: 3.0000e-04
Epoch 10/15
2645/2645 [==============================] - 218s 82ms/step - loss: 2.6984 -
accuracy: 0.4053 - val_loss: 2.9153 - val_accuracy: 0.4164 - lr: 3.0000e-04
Epoch 11/15
2645/2645 [==============================] - 214s 81ms/step - loss: 2.6168 -
accuracy: 0.4201 - val_loss: 2.6516 - val_accuracy: 0.4392 - lr: 3.0000e-04
Epoch 12/15
2645/2645 [==============================] - 210s 79ms/step - loss: 2.5438 -
accuracy: 0.4322 - val_loss: 2.8134 - val_accuracy: 0.4244 - lr: 3.0000e-04
Epoch 13/15
2645/2645 [==============================] - 204s 77ms/step - loss: 2.4821 -
accuracy: 0.4429 - val_loss: 2.9355 - val_accuracy: 0.4263 - lr: 3.0000e-04
Epoch 14/15
2645/2645 [==============================] - 203s 77ms/step - loss: 2.2549 -
accuracy: 0.4851 - val_loss: 2.4617 - val_accuracy: 0.4712 - lr: 1.5000e-04
Epoch 15/15
2645/2645 [==============================] - 209s 79ms/step - loss: 2.1581 -
accuracy: 0.5014 - val_loss: 2.4309 - val_accuracy: 0.4865 - lr: 1.5000e-04
```

[65]:
```
plot_curves(history,graph_path,count)
```


Training and validation accuracy
Training accuracy
Validation accuracy
0.5
0.4
0.3
0.2
0.1
0
2
4
6
8
10
12
14




```
<Figure size 640x480 with 0 Axes>
```

[66]:
```
evalua_modelo(model,test_generator,model_name,resultados)
```

[66]:
```
'inception_v3_FT_entrega_2- test_loss: 2.3123550415039062 - test_accuracy
0.5047619342803955'
```

0.1.6 Fine Tunne 3

[67]:
```
data_augmentation = tf.keras.Sequential([
    tf.keras.layers.RandomFlip(mode="horizontal", seed=42),
    tf.keras.layers.RandomRotation(factor=0.05, seed=42),
    tf.keras.layers.RandomZoom(0.05, seed=42),
])
```

[68]:
```
base_model = tf.keras.applications.InceptionV3(include_top=
 ↪False,weights='imagenet')

# 2. Freeze the base model
base_model.trainable = True
for layer in base_model.layers[:-22]:
```

```
  layer.trainable = False

model=tf.keras.Sequential([
    tf.keras.layers.Input(shape =(224,224,3), name = "input-layer"),
    data_augmentation,
    base_model,
    tf.keras.layers.Flatten(),
    tf.keras.layers.Activation("relu"),
    tf.keras.layers.BatchNormalization(),
    tf.keras.layers.Dropout(0.5),
    tf.keras.layers.Dense(2048, activation="relu"),
    tf.keras.layers.BatchNormalization(),
    tf.keras.layers.Dropout(0.5),
    tf.keras.layers.Dense(1024, activation="relu"),
    tf.keras.layers.BatchNormalization(),
    tf.keras.layers.Dropout(0.5),
    tf.keras.layers.Dense(526, activation="relu"),
    tf.keras.layers.BatchNormalization(),
    tf.keras.layers.Dropout(0.5),
    tf.keras.layers.Dense(525, activation = "softmax", name = "output-layer")
])


# 9. Compile the model
model.compile(loss = "categorical_crossentropy",
                optimizer = tf.keras.optimizers.Adam(learning_rate = 0.0003),
                metrics = ["accuracy"])
model.summary()
```

```
Model: "sequential_13"

_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 sequential_12 (Sequential)  (None, 224, 224, 3)       0

 inception_v3 (Functional)   (None, None, None, 2048)  21802784

 flatten_4 (Flatten)         (None, 51200)             0

 activation_471 (Activation)  (None, 51200)            0

 batch_normalization_478 (Ba  (None, 51200)            204800
 tchNormalization)

 dropout_8 (Dropout)         (None, 51200)             0

 dense_17 (Dense)            (None, 2048)              104859648
```

```
 batch_normalization_479 (Ba  (None, 2048)             8192
 tchNormalization)

 dropout_9 (Dropout)         (None, 2048)              0

 dense_18 (Dense)            (None, 1024)              2098176

 batch_normalization_480 (Ba  (None, 1024)             4096
 tchNormalization)

 dropout_10 (Dropout)        (None, 1024)              0

 dense_19 (Dense)            (None, 526)               539150

 batch_normalization_481 (Ba  (None, 526)              2104
 tchNormalization)

 dropout_11 (Dropout)        (None, 526)               0

 output-layer (Dense)        (None, 525)               276675

=================================================================
Total params: 129,795,625
Trainable params: 110,703,341
Non-trainable params: 19,092,284
_________________________________________________________________
```

[69]:

```
#Creamos directorio para guardar el modelo
count=3
model_name=base_model.name+"_FT_entrega_"+str(count)
model_path,graph_path=crea_directorio(model_name)
```

[70]:

```
#Definimos los callbacks
lr =ReduceLROnPlateau(monitor="val_loss",
                      factor=0.5,
                      patience=2)

es = EarlyStopping(monitor='val_accuracy',
                   mode='max', patience=5,
                   restore_best_weights=True)

mcp = ModelCheckpoint(filepath=model_path,
                      save_best_only=True,
                      monitor='val_accuracy',
                      mode='max')
```

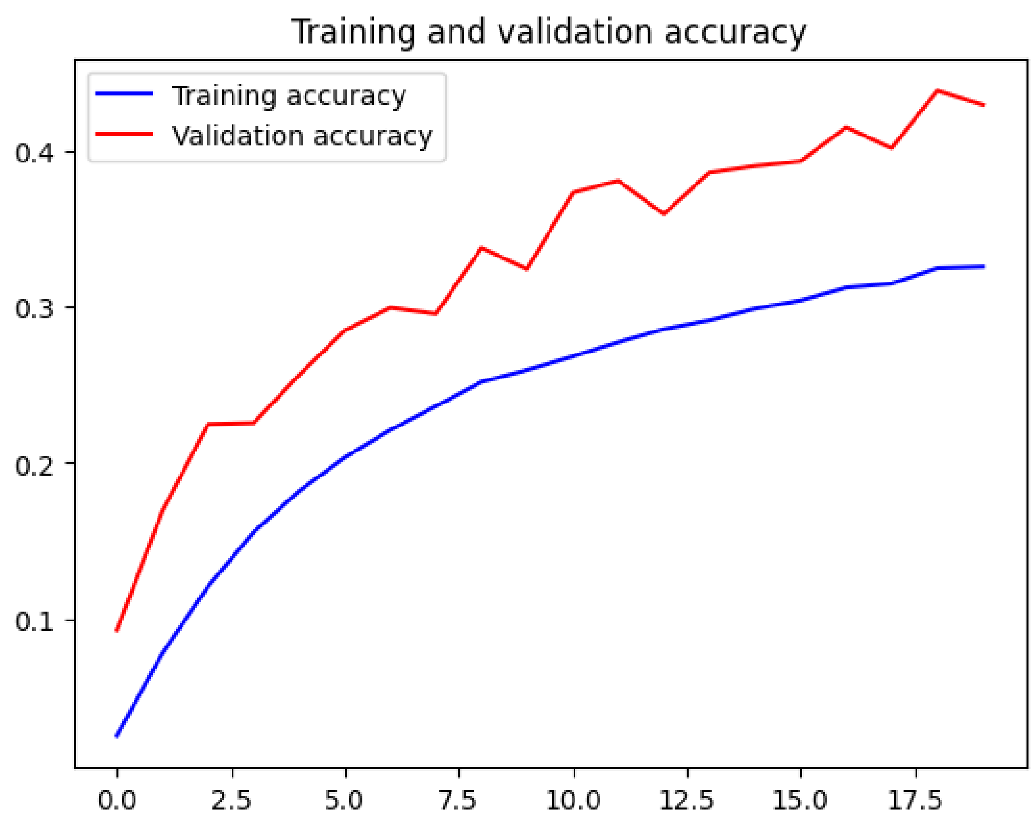
[71]:
```
#entrenamos el modelo
history=model.fit(train_generator, epochs=20, validation_data = val_generator,␣
↪callbacks=[es,lr,mcp])
```

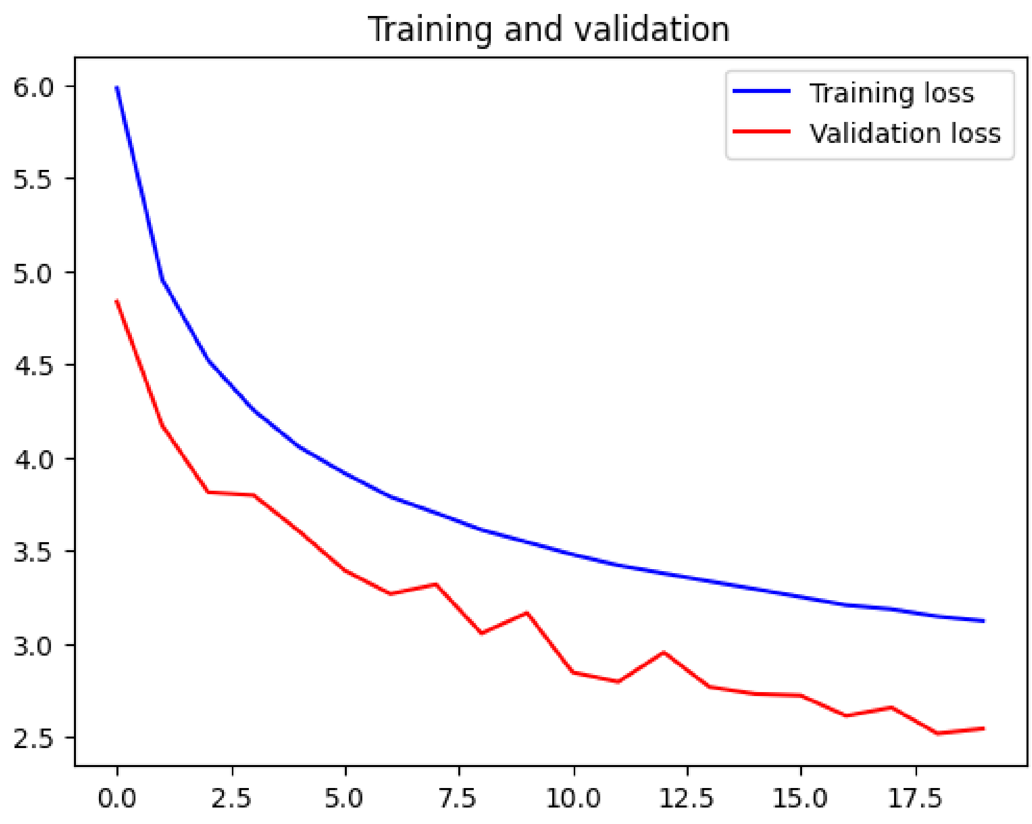
```
Epoch 1/20
2645/2645 [==============================] - 261s 97ms/step - loss: 5.9800 -
accuracy: 0.0250 - val_loss: 4.8334 - val_accuracy: 0.0926 - lr: 3.0000e-04
Epoch 2/20
2645/2645 [==============================] - 260s 98ms/step - loss: 4.9500 -
accuracy: 0.0778 - val_loss: 4.1670 - val_accuracy: 0.1691 - lr: 3.0000e-04
Epoch 3/20
2645/2645 [==============================] - 268s 101ms/step - loss: 4.5197 -
accuracy: 0.1207 - val_loss: 3.8108 - val_accuracy: 0.2248 - lr: 3.0000e-04
Epoch 4/20
2645/2645 [==============================] - 279s 105ms/step - loss: 4.2510 -
accuracy: 0.1555 - val_loss: 3.7950 - val_accuracy: 0.2255 - lr: 3.0000e-04
Epoch 5/20
2645/2645 [==============================] - 268s 101ms/step - loss: 4.0537 -
accuracy: 0.1819 - val_loss: 3.6012 - val_accuracy: 0.2564 - lr: 3.0000e-04
Epoch 6/20
2645/2645 [==============================] - 255s 96ms/step - loss: 3.9124 -
accuracy: 0.2034 - val_loss: 3.3907 - val_accuracy: 0.2850 - lr: 3.0000e-04
Epoch 7/20
2645/2645 [==============================] - 255s 96ms/step - loss: 3.7864 -
accuracy: 0.2210 - val_loss: 3.2650 - val_accuracy: 0.2994 - lr: 3.0000e-04
Epoch 8/20
2645/2645 [==============================] - 256s 97ms/step - loss: 3.6993 -
accuracy: 0.2363 - val_loss: 3.3160 - val_accuracy: 0.2956 - lr: 3.0000e-04
Epoch 9/20
2645/2645 [==============================] - 274s 103ms/step - loss: 3.6089 -
accuracy: 0.2518 - val_loss: 3.0532 - val_accuracy: 0.3379 - lr: 3.0000e-04
Epoch 10/20
2645/2645 [==============================] - 256s 97ms/step - loss: 3.5427 -
accuracy: 0.2595 - val_loss: 3.1635 - val_accuracy: 0.3242 - lr: 3.0000e-04
Epoch 11/20
2645/2645 [==============================] - 270s 102ms/step - loss: 3.4767 -
accuracy: 0.2682 - val_loss: 2.8441 - val_accuracy: 0.3733 - lr: 3.0000e-04
Epoch 12/20
2645/2645 [==============================] - 263s 99ms/step - loss: 3.4186 -
accuracy: 0.2774 - val_loss: 2.7932 - val_accuracy: 0.3810 - lr: 3.0000e-04
Epoch 13/20
2645/2645 [==============================] - 259s 98ms/step - loss: 3.3748 -
accuracy: 0.2857 - val_loss: 2.9508 - val_accuracy: 0.3596 - lr: 3.0000e-04
Epoch 14/20
2645/2645 [==============================] - 261s 99ms/step - loss: 3.3326 -
accuracy: 0.2914 - val_loss: 2.7659 - val_accuracy: 0.3863 - lr: 3.0000e-04
Epoch 15/20
2645/2645 [==============================] - 276s 104ms/step - loss: 3.2914 -
```

```
accuracy: 0.2988 - val_loss: 2.7278 - val_accuracy: 0.3905 - lr: 3.0000e-04
Epoch 16/20
2645/2645 [==============================] - 262s 99ms/step - loss: 3.2485 -
accuracy: 0.3041 - val_loss: 2.7198 - val_accuracy: 0.3935 - lr: 3.0000e-04
Epoch 17/20
2645/2645 [==============================] - 259s 98ms/step - loss: 3.2046 -
accuracy: 0.3124 - val_loss: 2.6107 - val_accuracy: 0.4152 - lr: 3.0000e-04
Epoch 18/20
2645/2645 [==============================] - 265s 100ms/step - loss: 3.1829 -
accuracy: 0.3150 - val_loss: 2.6553 - val_accuracy: 0.4019 - lr: 3.0000e-04
Epoch 19/20
2645/2645 [==============================] - 259s 98ms/step - loss: 3.1444 -
accuracy: 0.3249 - val_loss: 2.5160 - val_accuracy: 0.4389 - lr: 3.0000e-04
Epoch 20/20
2645/2645 [==============================] - 257s 97ms/step - loss: 3.1203 -
accuracy: 0.3258 - val_loss: 2.5415 - val_accuracy: 0.4297 - lr: 3.0000e-04
```

[72]:

```
plot_curves(history,graph_path,count)
```






```
<Figure size 640x480 with 0 Axes>
```

[73]:
```
evalua_modelo(model,test_generator,model_name,resultados)
```

[73]:
```
'inception_v3_FT_entrega_3- test_loss: 2.501204490661621 - test_accuracy
0.43504762649536133'
```

Conclusiones_Anexo

November 26, 2023

0.1 Conclusiones y Comentarios Finales

Con los experimentos, se consigue llegar al objetivo: evaluar y realizar clasificación de imágenes con distintas especies de pájaros mediante redes neuronales y CNN ("From Scratch" y Transfer Learning VGG16 e InceptionV3)

El proceso de entrenamiento sigue el pipeline visto en clase: carga del conjunto de datos, inspección del conjunto de datos, acondicionamiento del conjunto de datos, desarrollo de la arquitectura de red neuronal y entrenamiento de la solución. Monitorizando en todo momento el proceso de entrenamiento para la toma de decisiones, evaluación del modelo predictivo y planteamiento de la siguiente prueba experimental.

"Data pre_processing"

Gracias a las técnicas observadas en clase y al análisis visual, nos conseguimos dar cuenta de anomalías en el dataset y de imágenes de pájaros que NO se corresponden a la especie en la que están clasficadas, tomando acción a posteriori con data agumentation y la clusterizción para la detección de outliers.

"From Scratch VS Fine Tuning"

Las conclusiones obtenidas de la comparación entre la estrategia "From Scratch" y los modelos por Transfer Learning (VGG16 val: 0,85 , acc: 0,87)(InceptionV3 val: 0,45 , acc: 0,5) son que, de los modelos pre-entrenados obtienen resultados parecidos en términos de precisión y tiempo de entrenamiento que la red neuronal creada "From Scratch" (val: 0,76, acc: 0,96) para el caso de el modelo VGG16. Una explicación razonable para esto es que los alumnos han dedicado más tiempo de experimentación a crear la arquitectura de la red neuronal desde Scratch. Además, para entrenar la arquitectura de la red "From Scratch" se utiliza computación en la nube con recursos adicionales no proporcionados desde la Universidad, en cambio ambos mdoelos de Transfer Learning son entrenados en local/colab.

"VGG16 vs Inception V3"

También se observa que la arquitectura VGG16(VGG16 val: 0,85 , acc: 0,87) es más efectiva que InceptionV3 (InceptionV3 val: 0,45 , acc: 0,5) en la clasificación de imágenes. Una explicación posible de esto es el bajo número de épocas de entrenamiento. VGG16 es una arquitectura que se centra en el MaxPooling y reducción de la dimensionalidad. InceptionV3 tiene una arquitectura más compleja en el que se tienen en cuenta factores como el escalado. Quizá, cambiando el número de épocas, descongelando más capas y reducciendo el valor alfa de descenso del gradiente (paso) podríamos conseguir que InceptionV3 se acercara más a VGG16. Observamos en las curvas de

entrenamiento, que todavía no han llegado a estabilizarse en una parábola completa para InceptionV3, indicando que tenemos espacio de mejora con más entrenamiento pero no el hardware adecuado. Por otro lado, se observa que InceptionV3 es más eficiente en el entrenamiento que VGG16.

Como conclusión final, mencionar que el trabajo en equipo y la coordinación entre alumnos ha sido excelente, organizando las sesiones y los plazos de entrega desde el primero momento. Es una ventaja muy grande cuando los trabajos se hacen equipo y por eso queremos agradecer la organización de portafolios como este en el máster.

0.2 Anexo de Experimentos

0.3 First Model From Experiment (0,3 val acc)

[ ]:
```
from sklearn.preprocessing import LabelEncoder
from tensorflow import keras
from keras.models import Sequential
from keras import layers as L
from keras.utils import to_categorical

# Create a LabelEncoder object
label_encoder = LabelEncoder()

# Fit the LabelEncoder to the train_labels data
label_encoder.fit(train_labels)

# Create a dictionary that maps each unique label to an integer
label_map = {}
for i, label in enumerate(label_encoder.classes_):
 label_map[label] = i  # Escape the non-breaking space

# Convert the train_labels array to one-hot encoded vectors
train_labels_onehot , test_labels_onehot = to_categorical(label_encoder.
 ↪transform(train_labels)),to_categorical(label_encoder.transform(test_labels))

# Define the CNN model
model = Sequential()
model.add(L.Conv2D(32, 3, activation='relu', input_shape=(224, 224, 3)))
model.add(L.MaxPooling2D((2, 2)))
model.add(L.Conv2D(64, 3, activation='relu'))
model.add(L.MaxPooling2D((2, 2)))
model.add(L.Flatten())
model.add(L.Dense(128, activation='relu'))
model.add(L.Dense(len(label_map), activation='softmax'))

# Compile the model
model.compile(loss='categorical_crossentropy', optimizer='adam',␣
 ↪metrics=['accuracy'])
```

```
# Train the model
model.fit(train_images, train_labels_onehot, epochs=8, batch_size=128,␣
↪validation_data=(test_images, test_labels_onehot))

# Evaluate the model
test_loss, test_acc = model.evaluate(test_images, test_labels_onehot)
print('Test loss:', test_loss)
print('Test accuracy:', test_acc)

# To decode the predictions, you can use the inverse_transform method of the␣
↪LabelEncoder object. Here's an example of how to do this:
predicted_labels = model.predict(test_images)
predicted_labels_decoded = label_encoder.inverse_transform(predicted_labels.
↪argmax(axis=1))
print('Predicted labels:', predicted_labels_decoded)
```

0.4 Second Model From Experiment (0,7 val acc)

[ ]:
```
from sklearn.preprocessing import LabelEncoder
from tensorflow import keras
from keras.models import Sequential
from keras import layers as L
from keras.utils import to_categorical
from keras.callbacks import EarlyStopping

# Create a LabelEncoder object
label_encoder = LabelEncoder()

# Fit the LabelEncoder to the train_labels data
label_encoder.fit(train_labels)

# Create a dictionary that maps each unique label to an integer
label_map = {}
for i, label in enumerate(label_encoder.classes_):
 label_map[label] = i

# Convert the train_labels array to one-hot encoded vectors
train_labels_onehot, test_labels_onehot = to_categorical(label_encoder.
↪transform(train_labels)), to_categorical(label_encoder.
↪transform(test_labels))

model = Sequential()
# Capas convolucionales
model.add(L.Conv2D(64, (3, 3), padding='same', activation='relu',␣
↪input_shape=(224, 224, 3)))
```

```
model.add(L.MaxPooling2D((2, 2), strides=(2, 2)))
model.add(L.Conv2D(128, (3, 3), padding='same', activation='relu'))
model.add(L.MaxPooling2D((2, 2), strides=(2, 2)))
model.add(L.Conv2D(256, (3, 3), padding='same', activation='relu'))
model.add(L.MaxPooling2D((2, 2), strides=(2, 2)))
model.add(L.Conv2D(512, (3, 3), padding='same', activation='relu'))
model.add(L.MaxPooling2D((2, 2), strides=(2, 2)))
model.add(L.Conv2D(512, (3, 3), padding='same', activation='relu'))
model.add(L.MaxPooling2D((2, 2), strides=(2, 2)))

# Capas fully-connected
model.add(L.Flatten())
model.add(L.Dense(4096, activation='relu'))
model.add(L.Dropout(0.5))
model.add(L.Dense(4096, activation='relu'))
model.add(L.Dropout(0.5))
model.add(L.Dense(len(set(train_labels)), activation='softmax'))

# Compile the model
model.compile(loss='categorical_crossentropy', optimizer='adadelta',␣
 ↪metrics=['accuracy'])

# Define the early stopping callback
early_stopping = EarlyStopping(monitor='val_accuracy', patience=15) #,␣
 ↪min_delta=0.01

# Train the model
model.fit(train_images, train_labels_onehot, epochs=45, batch_size=128,␣
 ↪validation_data=(test_images, test_labels_onehot),␣
 ↪callbacks=[early_stopping])

# Evaluate the model
test_loss, test_acc = model.evaluate(test_images, test_labels_onehot)
print('Test loss:', test_loss)
print('Test accuracy:', test_acc)

# To decode the predictions, you can use the inverse_transform method of the␣
 ↪LabelEncoder object. Here's an example of how to do this:
predicted_labels = model.predict(test_images)
predicted_labels_decoded = label_encoder.inverse_transform(predicted_labels.
 ↪argmax(axis=1))
print('Predicted labels:', predicted_labels_decoded)
```

0.5 Third Model tipo SVG

[ ]:
```
from sklearn.preprocessing import LabelEncoder
from tensorflow import keras
from keras.models import Sequential
from keras import layers as L
from keras.utils import to_categorical
from keras.callbacks import EarlyStopping

# Create a LabelEncoder object
label_encoder = LabelEncoder()

# Fit the LabelEncoder to the train_labels data
label_encoder.fit(train_labels)

# Create a dictionary that maps each unique label to an integer
label_map = {}
for i, label in enumerate(label_encoder.classes_):
 label_map[label] = i

# Convert the train_labels array to one-hot encoded vectors
train_labels_onehot, test_labels_onehot = to_categorical(label_encoder.
 ↪transform(train_labels)), to_categorical(label_encoder.
 ↪transform(test_labels))

# Create the model
model = Sequential()

# Feature extractor network
model.add(L.ZeroPadding2D((32, 32)))
model.add(L.Conv2D(64, (3, 3), activation='relu'))
model.add(L.MaxPooling2D((2, 2)))

model.add(L.ZeroPadding2D((16, 16)))
model.add(L.Conv2D(128, (3, 3), activation='relu'))
model.add(L.MaxPooling2D((2, 2)))

model.add(L.ZeroPadding2D((8, 8)))
model.add(L.Conv2D(256, (3, 3), activation='relu'))
model.add(L.MaxPooling2D((2, 2)))

model.add(L.ZeroPadding2D((4, 4)))
model.add(L.Conv2D(512, (3, 3), activation='relu'))
model.add(L.MaxPooling2D((2, 2)))

# Add additional convolutional layers
model.add(L.Conv2D(512, (3, 3), activation='relu'))
```

```
model.add(L.Conv2D(1024, (3, 3), activation='relu'))
model.add(L.Conv2D(1024, (3, 3), activation='relu'))

# Prediction layers for class and location
model.add(L.Conv2D(21, (1, 1), activation='softmax'))
model.add(L.Conv2D(len(train_labels), (1, 1), activation='linear'))

# Compile the model
model.compile(loss='categorical_crossentropy', optimizer='adadelta',␣
 ↪metrics=['accuracy'])

# Define the early stopping callback
early_stopping = EarlyStopping(monitor='val_accuracy', patience=15) #,␣
 ↪min_delta=0.01

# Train the model
model.fit(train_images, train_labels_onehot, epochs=45, batch_size=128,␣
 ↪validation_data=(test_images, test_labels_onehot),␣
 ↪callbacks=[early_stopping])

# Evaluate the model
test_loss, test_acc = model.evaluate(test_images, test_labels_onehot)
print('Test loss:', test_loss)
print('Test accuracy:', test_acc)

# To decode the predictions, you can use the inverse_transform method of the␣
 ↪LabelEncoder object. Here's an example of how to do this:
predicted_labels = model.predict(test_images)
predicted_labels_decoded = label_encoder.inverse_transform(predicted_labels.
 ↪argmax(axis=1))
print('Predicted labels:', predicted_labels_decoded)
```