Momento de Retroalimentación Individual: Implementación de un modelo de Deep Learning.

En este documento se utiliza un framework (Tensorflow/Keras) para entrenar un modelo de aprendizaje profundo destinado a clasificar imágenes de caterpillars (orugas), fish (peces) y forgs (ranas). El desempeño de este modelo se evalúa en su aproximación inicial y se realizan ajustes para mejorarlo.

El dataset utilizado es **Animals Detection Images Dataset**, conjunto de datos de detección de animales extraidos con Google Open Images V6+. Cuenta con conjuntos de entrenamiento y prueba e imágenes de ochenta animales diferentes. No obstante, para propósitos de esta entrega se seleccionaron sólo tres animales; Caterpillar, Fish y Frog; y se unificaron los conjuntos de entrenamiento y prueba para realizar la división de entrenamiento, prueba y validación en una porporción de 70%, 15% y 15% respectivamente. Para conocer más de este dataset favor de consultar la siguiente liga: https://www.kaggle.com/datasets/antoreepjana/animals-detection-images-dataset/?select=train.

Este trabajo es propiedad de **Arturo Garza Campuzano**, con matrícula **A00828096**.

1. Selección y unificación del dataset

Para seleccionar y unificar los datos originales de los animales Caterpillar, Fish y Frog se siguió el siguiente proceso:

1. Descarga y carga de las imágenes del dataset **Animals Detection Images Dataset**.
2. Selección de imágenes de Caterpillar, Fish y Frog.
3. Unificación de las imágenes de los conjuntos de entrenamiento y prueba en un solo conjunto. Para esto se hizo uso del siguiente código.
4. Separación del conjunto unificado en entrenamiento, prueba y validación.

2. Cargar librerias requeridas

```
!pip install tensorflow

import matplotlib.pyplot as plt
import numpy as np
import os

from tensorflow.keras import models
from tensorflow.keras import layers
from tensorflow.keras.applications import VGG16
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras import optimizers
```

```python
from tensorflow.keras import regularizers
from tensorflow.keras.layers import Dropout, Flatten, Dense,
GlobalAveragePooling2D,BatchNormalization, l2
from keras.metrics import top_k_categorical_accuracy
```

```
Requirement already satisfied: tensorflow in
/usr/local/lib/python3.10/dist-packages (2.14.0)
Requirement already satisfied: absl-py>=1.0.0 in
/usr/local/lib/python3.10/dist-packages (from tensorflow) (1.4.0)
Requirement already satisfied: astunparse>=1.6.0 in
/usr/local/lib/python3.10/dist-packages (from tensorflow) (1.6.3)
Requirement already satisfied: flatbuffers>=23.5.26 in
/usr/local/lib/python3.10/dist-packages (from tensorflow) (23.5.26)
Requirement already satisfied: gast!=0.5.0,!=0.5.1,!=0.5.2,>=0.2.1
in /usr/local/lib/python3.10/dist-packages (from tensorflow) (0.5.4)
Requirement already satisfied: google-pasta>=0.1.1 in
/usr/local/lib/python3.10/dist-packages (from tensorflow) (0.2.0)
Requirement already satisfied: h5py>=2.9.0 in
/usr/local/lib/python3.10/dist-packages (from tensorflow) (3.9.0)
Requirement already satisfied: libclang>=13.0.0 in
/usr/local/lib/python3.10/dist-packages (from tensorflow) (16.0.6)
Requirement already satisfied: ml-dtypes==0.2.0 in
/usr/local/lib/python3.10/dist-packages (from tensorflow) (0.2.0)
Requirement already satisfied: numpy>=1.23.5 in
/usr/local/lib/python3.10/dist-packages (from tensorflow) (1.23.5)
Requirement already satisfied: opt-einsum>=2.3.2 in
/usr/local/lib/python3.10/dist-packages (from tensorflow) (3.3.0)
Requirement already satisfied: packaging in
/usr/local/lib/python3.10/dist-packages (from tensorflow) (23.2)
Requirement already satisfied: protobuf!=4.21.0,!=4.21.1,!=4.21.2,!
=4.21.3,!=4.21.4,!=4.21.5,<5.0.0dev,>=3.20.3 in
/usr/local/lib/python3.10/dist-packages (from tensorflow) (3.20.3)
Requirement already satisfied: setuptools in
/usr/local/lib/python3.10/dist-packages (from tensorflow) (67.7.2)
Requirement already satisfied: six>=1.12.0 in
/usr/local/lib/python3.10/dist-packages (from tensorflow) (1.16.0)
Requirement already satisfied: termcolor>=1.1.0 in
/usr/local/lib/python3.10/dist-packages (from tensorflow) (2.3.0)
Requirement already satisfied: typing-extensions>=3.6.6 in
/usr/local/lib/python3.10/dist-packages (from tensorflow) (4.5.0)
Requirement already satisfied: wrapt<1.15,>=1.11.0 in
/usr/local/lib/python3.10/dist-packages (from tensorflow) (1.14.1)
Requirement already satisfied: tensorflow-io-gcs-filesystem>=0.23.1 in
/usr/local/lib/python3.10/dist-packages (from tensorflow) (0.34.0)
Requirement already satisfied: grpcio<2.0,>=1.24.3 in
/usr/local/lib/python3.10/dist-packages (from tensorflow) (1.59.0)
Requirement already satisfied: tensorboard<2.15,>=2.14 in
/usr/local/lib/python3.10/dist-packages (from tensorflow) (2.14.1)
Requirement already satisfied: tensorflow-estimator<2.15,>=2.14.0
in /usr/local/lib/python3.10/dist-packages (from tensorflow) (2.14.0)
```

```
Requirement already satisfied: keras<2.15,>=2.14.0 in
/usr/local/lib/python3.10/dist-packages (from tensorflow) (2.14.0)
Requirement already satisfied: wheel<1.0,>=0.23.0 in
/usr/local/lib/python3.10/dist-packages (from astunparse>=1.6.0-
>tensorflow) (0.41.2)
Requirement already satisfied: google-auth<3,>=1.6.3 in
/usr/local/lib/python3.10/dist-packages (from tensorboard<2.15,>=2.14-
>tensorflow) (2.17.3)
Requirement already satisfied: google-auth-oauthlib<1.1,>=0.5 in
/usr/local/lib/python3.10/dist-packages (from tensorboard<2.15,>=2.14-
>tensorflow) (1.0.0)
Requirement already satisfied: markdown>=2.6.8 in
/usr/local/lib/python3.10/dist-packages (from tensorboard<2.15,>=2.14-
>tensorflow) (3.5)
Requirement already satisfied: requests<3,>=2.21.0 in
/usr/local/lib/python3.10/dist-packages (from tensorboard<2.15,>=2.14-
>tensorflow) (2.31.0)
Requirement already satisfied: tensorboard-data-server<0.8.0,>=0.7.0
in /usr/local/lib/python3.10/dist-packages (from
tensorboard<2.15,>=2.14->tensorflow) (0.7.2)
Requirement already satisfied: werkzeug>=1.0.1 in
/usr/local/lib/python3.10/dist-packages (from tensorboard<2.15,>=2.14-
>tensorflow) (3.0.1)
Requirement already satisfied: cachetools<6.0,>=2.0.0 in
/usr/local/lib/python3.10/dist-packages (from google-auth<3,>=1.6.3-
>tensorboard<2.15,>=2.14->tensorflow) (5.3.2)
Requirement already satisfied: pyasn1-modules>=0.2.1 in
/usr/local/lib/python3.10/dist-packages (from google-auth<3,>=1.6.3-
>tensorboard<2.15,>=2.14->tensorflow) (0.3.0)
Requirement already satisfied: rsa<5,>=3.1.4 in
/usr/local/lib/python3.10/dist-packages (from google-auth<3,>=1.6.3-
>tensorboard<2.15,>=2.14->tensorflow) (4.9)
Requirement already satisfied: requests-oauthlib>=0.7.0 in
/usr/local/lib/python3.10/dist-packages (from google-auth-
oauthlib<1.1,>=0.5->tensorboard<2.15,>=2.14->tensorflow) (1.3.1)
Requirement already satisfied: charset-normalizer<4,>=2 in
/usr/local/lib/python3.10/dist-packages (from requests<3,>=2.21.0-
>tensorboard<2.15,>=2.14->tensorflow) (3.3.1)
Requirement already satisfied: idna<4,>=2.5 in
/usr/local/lib/python3.10/dist-packages (from requests<3,>=2.21.0-
>tensorboard<2.15,>=2.14->tensorflow) (3.4)
Requirement already satisfied: urllib3<3,>=1.21.1 in
/usr/local/lib/python3.10/dist-packages (from requests<3,>=2.21.0-
>tensorboard<2.15,>=2.14->tensorflow) (2.0.7)
Requirement already satisfied: certifi>=2017.4.17 in
/usr/local/lib/python3.10/dist-packages (from requests<3,>=2.21.0-
>tensorboard<2.15,>=2.14->tensorflow) (2023.7.22)
Requirement already satisfied: MarkupSafe>=2.1.1 in
/usr/local/lib/python3.10/dist-packages (from werkzeug>=1.0.1-
```

```
>tensorboard<2.15,>=2.14->tensorflow) (2.1.3)
Requirement already satisfied: pyasn1<0.6.0,>=0.4.6 in
/usr/local/lib/python3.10/dist-packages (from pyasn1-modules>=0.2.1-
>google-auth<3,>=1.6.3->tensorboard<2.15,>=2.14->tensorflow) (0.5.0)
Requirement already satisfied: oauthlib>=3.0.0 in
/usr/local/lib/python3.10/dist-packages (from requests-
oauthlib>=0.7.0->google-auth-oauthlib<1.1,>=0.5-
>tensorboard<2.15,>=2.14->tensorflow) (3.2.2)
```

2. Definición de los directorios para utilizar dataset

Para acceder al conjunto de datos referido anteriormente se asignan los directorios correspondientes de Google Drive.

```
# Define image dataset directories
from google.colab import drive
drive.mount('/content/drive')

directory =
"/content/drive/MyDrive/TC3007C.501_Equipo7/Reto/Modelos/animals/"

base_dir = directory + 'dataset_split'
train_dir = os.path.join(base_dir,'train')
validation_dir = os.path.join(base_dir, 'validation')
test_dir = os.path.join(base_dir, 'test')

Mounted at /content/drive
```

3. Preprocesamiento y Augmentación de Datos

A continuación, se realiza la configuración del **preprocesamiento** y **augmentación** de datos de entrenamiento y validación para:

- **Normalizar datos**: permite que la red converja más rápido durante el entrenamiento y sea menos sensible a diferencias en la escala de los valores de píxeles entre las imágenes.
- **Aumentar datos**: técnica en la cual se aplican transformaciones aleatorias a las imágenes de entrenamiento para crear nuevas instancias de datos.
- **Reducir sobreajuste (overfitting)**: el sobreajuste ocurre cuando un modelo se adapta en exceso a los datos de entrenamiento y tiene dificultades para generalizar con nuevos datos. El aumento de datos contribuye a prevenir este sobreajuste.

Con el fin de generar lotes de datos de imágenes con ciertas transformaciones y aumentos se utiliza ImageDataGenerator de Keras:

- Transformaciones del conjunto de entrenamiento
  - Escalado de los valores de pixeles al rango de 0-1
  - Rotación aleatoria de la imagen de 40 grados

    - Desplazamientos horizontales y verticales aleatorios
    - Rangos aleatorios de deformación
    - Volteo horizontal aleatorio
- El conjunto de validación sólo se escala

Después se genera un batch de imágenes preprocesadas durante el entrenamiento.

```
train_datagen = ImageDataGenerator(
                                      rescale = 1./255,
                                      rotation_range = 40,
                                      width_shift_range = 0.2,
                                      height_shift_range = 0.2,
                                      shear_range = 0.2,
                                      zoom_range = 0.2,
                                      horizontal_flip = True,)

val_datagen = ImageDataGenerator(1./255)


train_generator = train_datagen.flow_from_directory(
                                      train_dir,
                                      target_size=(224, 224),
                                      batch_size=32,
                                      class_mode='categorical')


val_generator = val_datagen.flow_from_directory(
                                      validation_dir,
                                      target_size=(224, 224),
                                      batch_size=32,
                                      class_mode='categorical')

Found 1875 images belonging to 3 classes.
Found 400 images belonging to 3 classes.
```

4. Modelo 1

Se construye un modelo de *transfer learning* utilizando la arquitectura VGG16 como base y se agregan capas *fully connected*. Para esto se carga el modelo VGG16 pre-entrenado, excluyendo las capas *fully connected* al final (feature extraction) y se crea un modelo secuencial para agregar nuevas capas:

- Modelo VGG16 como capas de extracción de características
- Capa de normalización por batch (convergencia)
- Aplanado de las *features* para preparar las capas densas
- Dos capas densas de 256 unidades
- Capa de salida para una clasificación de tres clases

Finalmente se congelan las capas del modelo base VGG16 para no entrenar esos pesos y se entrenan las capas densas agregadas.

```python
# Load VGG16 model, excluding final fully connected layers
conv_base= VGG16(weights='imagenet',
                                            include_top = False,
                                            input_shape = (224,224,3))

# Create new sequential model for transfer learning
model = models.Sequential()

# Add VGG16 model as feature extractor
model.add(conv_base)
# Add batch normalization layer
model.add(layers.BatchNormalization())
# Flatten features for dense layers
model.add(layers.Flatten())
# Add dense classifier layers
model.add(layers.Dense(256,activation='relu'))
model.add(layers.Dense(256,activation='relu'))
# Output layer for binary classification
model.add(layers.Dense(3,activation='sigmoid'))

#Freeze base model weights
conv_base.trainable = False

#Print model summary
model.summary()
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-
applications/vgg16/vgg16_weights_tf_dim_ordering_tf_kernels_notop.h5
58889256/58889256 [==============================] - 0s 0us/step
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 vgg16 (Functional)          (None, 7, 7, 512)         14714688

 batch_normalization (Batch  (None, 7, 7, 512)         2048
 Normalization)

 flatten (Flatten)           (None, 25088)             0

 dense (Dense)               (None, 256)               6422784

 dense_1 (Dense)             (None, 256)               65792

 dense_2 (Dense)             (None, 3)                 771

=================================================================
Total params: 21206083 (80.89 MB)
Trainable params: 6490371 (24.76 MB)
```

```
Non-trainable params: 14715712 (56.14 MB)
_________________________________________________________________
```

5. Entrenamiento del Modelo 1

Después de haber aplicado *transfer learning* se compila y entrena el modelo:

- Configuración del modelo especificando la función de pérdida, el optimizador y las métricas a monotorizar
- Entrenamiento del modelo con los datos de entrenamiento y validación
- Guardar el modelo entrenado

```
model.compile(loss='categorical_crossentropy',

     optimizer=optimizers.RMSprop(learning_rate=2e-5),
                              metrics=['accuracy'])

history = model.fit(train_generator,
                    steps_per_epoch=len(train_generator),
                    epochs =5,
                    validation_data=val_generator,
                    validation_steps=len(val_generator))

model.save('vgg_caterpillar_fish_frog.h5')
```

```
Epoch 1/5
59/59 [==============================] - ETA: 0s - loss: 0.8650 - 
accuracy: 0.6117

/usr/local/lib/python3.10/dist-packages/keras/src/preprocessing/
image.py:1862: UserWarning: This ImageDataGenerator specifies 
`featurewise_center`, but it hasn't been fit on any training data. Fit 
it first by calling `.fit(numpy_data)`.
  warnings.warn(

59/59 [==============================] - 1505s 25s/step - loss: 0.8650 
- accuracy: 0.6117 - val_loss: 7.5154 - val_accuracy: 0.5350
Epoch 2/5
59/59 [==============================] - 1489s 25s/step - loss: 0.6773 
- accuracy: 0.7163 - val_loss: 5.8016 - val_accuracy: 0.6825
Epoch 3/5
59/59 [==============================] - 1439s 24s/step - loss: 0.5918 
- accuracy: 0.7616 - val_loss: 5.2963 - val_accuracy: 0.7600
Epoch 4/5
59/59 [==============================] - 1439s 24s/step - loss: 0.5272 
- accuracy: 0.7888 - val_loss: 4.8078 - val_accuracy: 0.8100
Epoch 5/5
59/59 [==============================] - 1445s 25s/step - loss: 0.4775 
- accuracy: 0.8176 - val_loss: 5.7429 - val_accuracy: 0.8125
```

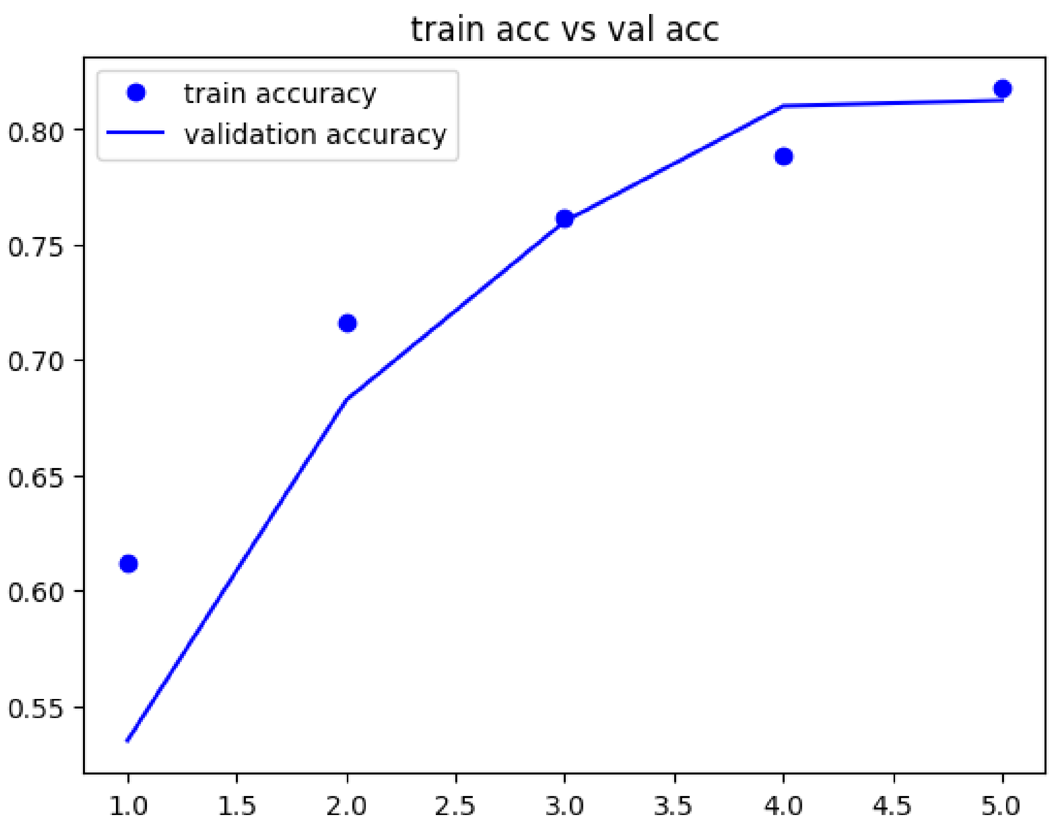
```
/usr/local/lib/python3.10/dist-packages/keras/src/engine/
training.py:3079: UserWarning: You are saving your model as an HDF5
file via `model.save()`. This file format is considered legacy. We
recommend using instead the native Keras format, e.g.
`model.save('my_model.keras')`.
  saving_api.save_model(

----------------------------------------------------------------------
-----
KeyError                                  Traceback (most recent call
last)
<ipython-input-8-3861820c6b4c> in <cell line: 13>()
     11 model.save('vgg_caterpillar_fish_frog.h5')
     12
---> 13 acc = history.history['acc']
     14 val_acc = history.history['val_acc']
     15 loss = history.history['loss']

KeyError: 'acc'
```

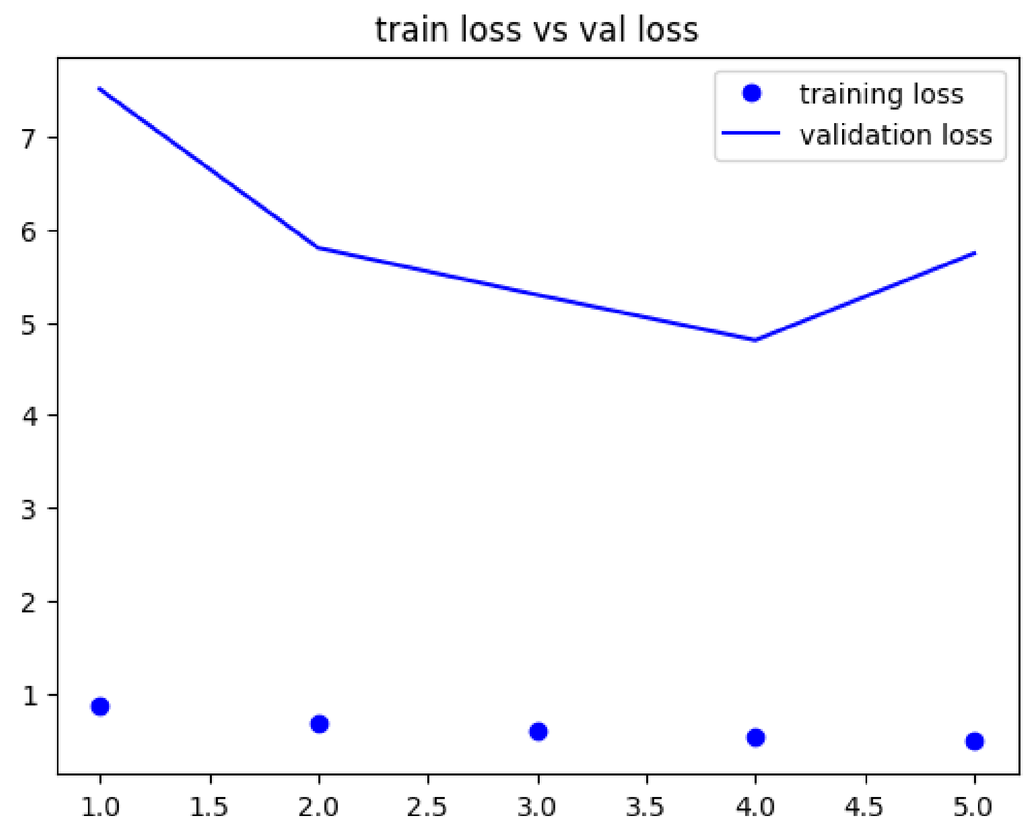
Finalmente se extraen las métricas de dicho entrenamiento, graficándolas, y se evalua el modelo sobre el conjunto de prueba. Este proceso consta de los siguientes pasos:

- Extracción de accuracy y loss del entrenamiento y validación
- Gráfica accuracy de entrenamiento vs validación
- Gráfica loss de entrenamiento vs validación
- Generador de datos de prueba con augmentación

```
acc = history.history['accuracy']
val_acc = history.history['val_accuracy']
loss = history.history['loss']
val_loss = history.history['val_loss']

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

plt.plot(epochs,acc,'bo',label='train accuracy')
plt.plot(epochs,val_acc, 'b', label='validation accuracy')
plt.title('train acc vs val acc')
plt.legend()

plt.figure()

plt.plot(epochs,loss, 'bo', label ='training loss')
plt.plot(epochs,val_loss, 'b', label = 'validation loss')
plt.title('train loss vs val loss')
plt.legend()

plt.show()

test_datagen = ImageDataGenerator(1./255)
```

```
test_generator = test_datagen.flow_from_directory(
                            test_dir,
                            target_size = (224, 224),
                            batch_size = 32,
                            class_mode= 'categorical')

test_loss, test_acc = model.evaluate_generator(test_generator, steps =
len(test_generator))
print('\ntest acc :\n', test_acc)
```






```
Found 406 images belonging to 3 classes.

<ipython-input-9-8ba4bf3ad9e8>:29: UserWarning:
`Model.evaluate_generator` is deprecated and will be removed in a
future version. Please use `Model.evaluate`, which supports
generators.
  test_loss, test_acc = model.evaluate_generator(test_generator, steps
= len(test_generator))


test acc :
 0.8103448152542114
```

6. Modelo 2

Con el fin de mejorar los resultados obtenidos del modelo anterior se reconstruye el modelo añadiendo/reemplazando el modelo con las siguientes capas:

- Capa densa con 512 unidades, aplicando técnicas de regularización (a cambio de la primera capa de 256 unidades)
- Capa de dropout, para objetivos de regularización
- Añadidura de técnicas de regularización de la última capa densa de 256 unidades

```python
# Load VGG16 model, excluding final fully connected layers
conv_base = VGG16(weights='imagenet', include_top=False,
input_shape=(224, 224, 3))

# Create a new sequential model for transfer learning
model_2 = models.Sequential()

# Add VGG16 model as a feature extractor
model_2.add(conv_base)
# Add batch normalization layer
model_2.add(layers.BatchNormalization())
# Flatten features for dense layers
model_2.add(layers.Flatten())

# Add dense classifier layers with regularization
model_2.add(layers.Dense(512, activation='relu',
kernel_regularizer=regularizers.l2(0.01)))
model_2.add(layers.Dropout(0.5))  # Add dropout for regularization
model_2.add(layers.Dense(256, activation='relu',
kernel_regularizer=regularizers.l2(0.01)))

# Output layer for multiclass classification (3 classes)
model_2.add(layers.Dense(3, activation='softmax'))

# Freeze base model weights
conv_base.trainable = False

# Print model summary
model_2.summary()
```

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

 batch_normalization_2 (Bat  (None, 7, 7, 512)         2048
 chNormalization)

 flatten_2 (Flatten)         (None, 25088)             0

 dense_3 (Dense)             (None, 512)               12845568

 dropout (Dropout)           (None, 512)               0

 dense_4 (Dense)             (None, 256)               131328

 dense_5 (Dense)             (None, 3)                 771

=================================================================
```

```
Total params: 27694403 (105.65 MB)
Trainable params: 12978691 (49.51 MB)
Non-trainable params: 14715712 (56.14 MB)
_________________________________________________________________
```

6. Entrenamiento del Modelo 2

Se entrena este nuevo modelo como se entrenó el anterior.

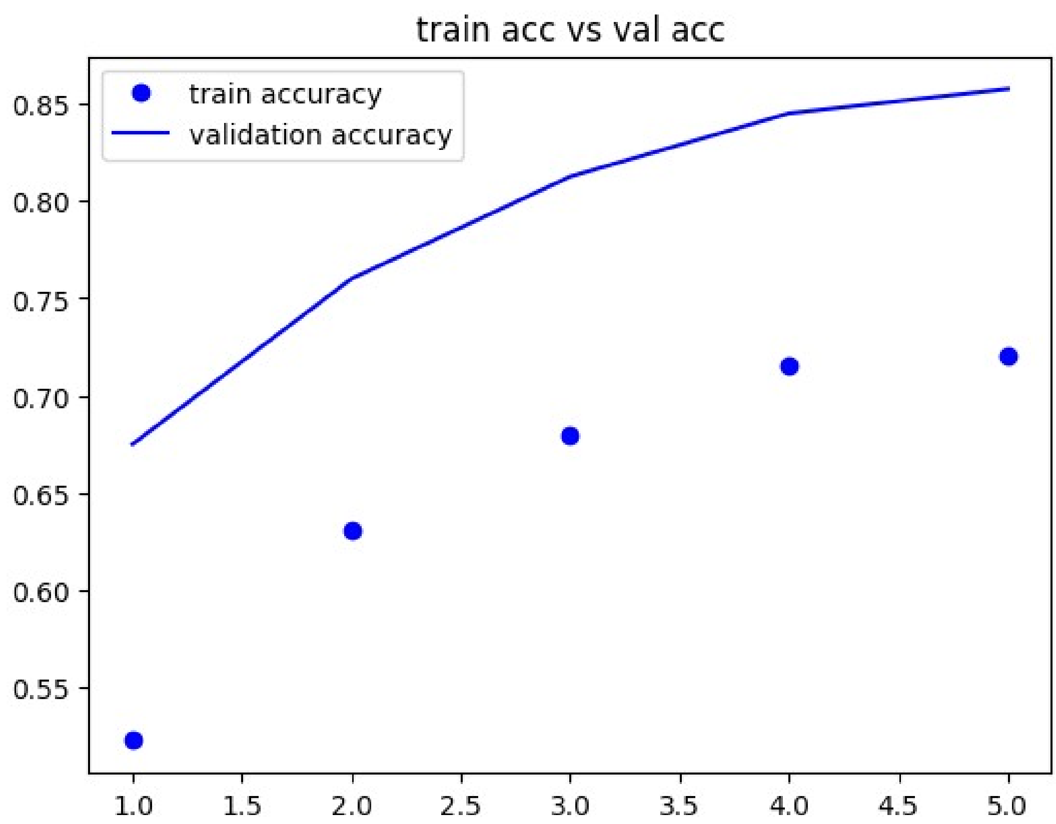
```
model_2.compile(loss='categorical_crossentropy',

     optimizer=optimizers.RMSprop(learning_rate=2e-5),
                                metrics=['accuracy'])

history = model_2.fit(train_generator,
                     steps_per_epoch=len(train_generator),
                     epochs =5,
                     validation_data=val_generator,
                     validation_steps=len(val_generator))

model_2.save('vgg_caterpillar_fish_frog_2.h5')
```

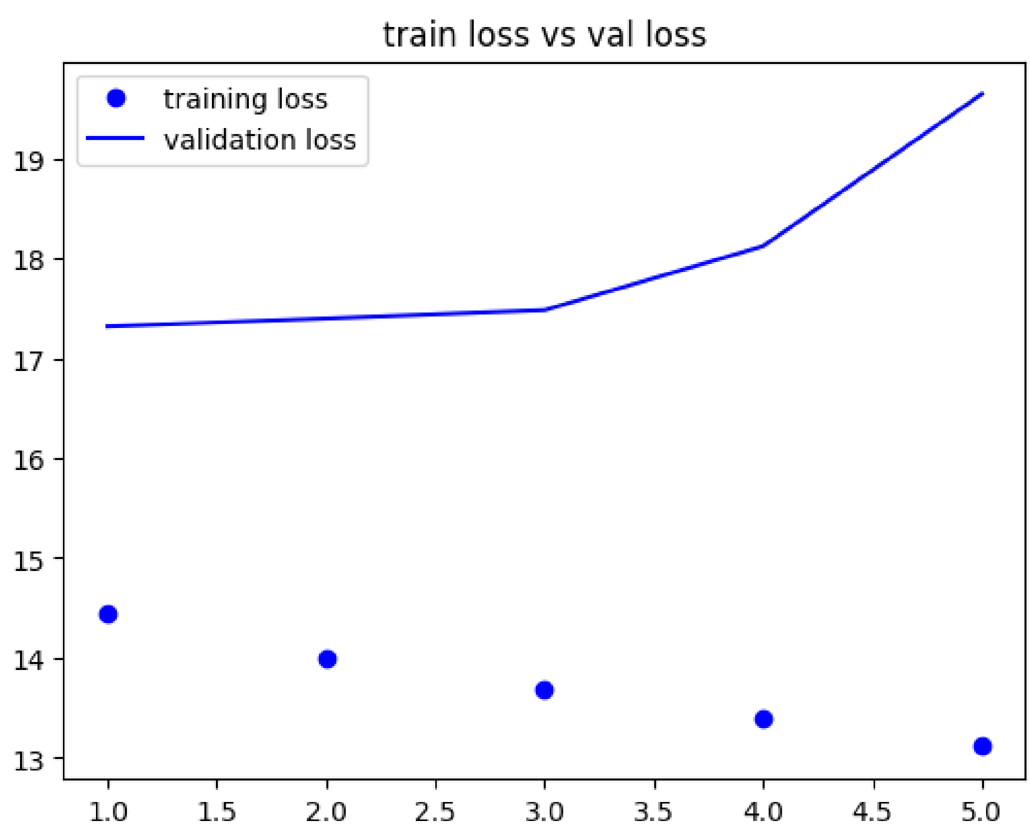
```
Epoch 1/5
59/59 [==============================] - 1580s 27s/step - loss:
14.4371 - accuracy: 0.5227 - val_loss: 17.3180 - val_accuracy: 0.6750
Epoch 2/5
59/59 [==============================] - 1577s 27s/step - loss:
14.0059 - accuracy: 0.6309 - val_loss: 17.3979 - val_accuracy: 0.7600
Epoch 3/5
59/59 [==============================] - 1485s 25s/step - loss:
13.6791 - accuracy: 0.6795 - val_loss: 17.4811 - val_accuracy: 0.8125
Epoch 4/5
59/59 [==============================] - 1431s 24s/step - loss:
13.3912 - accuracy: 0.7157 - val_loss: 18.1211 - val_accuracy: 0.8450
Epoch 5/5
59/59 [==============================] - 1440s 24s/step - loss:
13.1157 - accuracy: 0.7205 - val_loss: 19.6427 - val_accuracy: 0.8575

---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call
last)
<ipython-input-13-1277ae8c0edd> in <cell line: 13>()
     11 model_2.save('vgg_caterpillar_fish_frog.h5')
     12
---> 13 acc = history.history['acc']
     14 val_acc = history.history['val_acc']
     15 loss = history.history['loss']
```

```
KeyError: 'acc'
```

Se obtienen las mismas métricas, gráficas y resultados del conjunto de prueba.

```
acc = history.history['accuracy']
val_acc = history.history['val_accuracy']
loss = history.history['loss']
val_loss = history.history['val_loss']

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

plt.plot(epochs,acc,'bo',label='train accuracy')
plt.plot(epochs,val_acc, 'b', label='validation accuracy')
plt.title('train acc vs val acc')
plt.legend()

plt.figure()

plt.plot(epochs,loss, 'bo', label ='training loss')
plt.plot(epochs,val_loss, 'b', label = 'validation loss')
plt.title('train loss vs val loss')
plt.legend()

plt.show()

test_datagen = ImageDataGenerator(1./255)
test_generator = test_datagen.flow_from_directory(
                            test_dir,
                            target_size = (224, 224),
                            batch_size = 32,
                            class_mode= 'categorical')

test_loss, test_acc = model_2.evaluate_generator(test_generator, steps
= len(test_generator))
print('\ntest acc :\n', test_acc)
```


train acc vs val acc
train accuracy
validation accuracy
0.85
0.80
0.75
0.70
0.65
0.60
0.55
1.0
1.5
2.0
2.5
3.0
3.5
4.0
4.5
5.0


train loss vs val loss
training loss
validation loss
19
18
17
16
15
14
13
1.0
1.5
2.0
2.5
3.0
3.5
4.0
4.5
5.0

```
Found 406 images belonging to 3 classes.

<ipython-input-14-7e2ae5d96a8d>:29: UserWarning:
`Model.evaluate_generator` is deprecated and will be removed in a
future version. Please use `Model.evaluate`, which supports
generators.
  test_loss, test_acc = model_2.evaluate_generator(test_generator,
steps = len(test_generator))
/usr/local/lib/python3.10/dist-packages/keras/src/preprocessing/image.
py:1862: UserWarning: This ImageDataGenerator specifies
`featurewise_center`, but it hasn't been fit on any training data. Fit
it first by calling `.fit(numpy_data)`.
  warnings.warn(


test acc :
 0.8546798229217529
```

7. Conclusiones

```python
import pandas as pd

model1_results = {
    'Epoch': [1, 2, 3, 4, 5],
    'Train_Loss': [0.8650, 0.6773, 0.5918, 0.5272, 0.4775],
    'Train_Accuracy': [0.6117, 0.7163, 0.7616, 0.7888, 0.8176],
    'Validation_Loss': [7.5154, 5.8016, 5.2963, 4.8078, 5.7429],
    'Validation_Accuracy': [0.5350, 0.6825, 0.7600, 0.8100, 0.8125]
}

model2_results = {
    'Epoch': [1, 2, 3, 4, 5],
    'Train_Loss': [14.4371, 14.0059 , 13.6791, 13.3912, 13.1157],
    'Train_Accuracy': [0.5227, 0.6309, 0.6795, 0.7157, 0.7205],
    'Validation_Loss': [17.3180, 17.3979, 17.4811, 18.1211, 19.6427],
    'Validation_Accuracy': [0.6750, 0.7600, 0.8125, 0.8450, 0.8575]
}

test_results = {
    'Model': [1, 2],
    'Test_Accuracy': [0.8103, 0.8547]
}

model1_results_df = pd.DataFrame(model1_results)
model2_results_df = pd.DataFrame(model2_results)
test_results_df = pd.DataFrame(test_results)

model1_results_df
```

```
   Epoch  Train_Loss  Train_Accuracy  Validation_Loss
Validation_Accuracy
0      1      0.8650          0.6117           7.5154
0.5350
1      2      0.6773          0.7163           5.8016
0.6825
2      3      0.5918          0.7616           5.2963
0.7600
3      4      0.5272          0.7888           4.8078
0.8100
4      5      0.4775          0.8176           5.7429
0.8125
```

```
model2_results_df
```

```
   Epoch  Train_Loss  Train_Accuracy  Validation_Loss
Validation_Accuracy
0      1     14.4371          0.5227          17.3180
0.6750
1      2     14.0059          0.6309          17.3979
0.7600
2      3     13.6791          0.6795          17.4811
0.8125
3      4     13.3912          0.7157          18.1211
0.8450
4      5     13.1157          0.7205          19.6427
0.8575
```

```
test_results_df
```

```
   Model  Test_Accuracy
0      1         0.8103
1      2         0.8547
```

Considerando los resultados de ambos modelos se puede observar que el mejor modelo es el **Modelo 1**. Si bien, el resultado del accuracy sobre el conjunto de prueba es mayor en el **Modelo 2**, los resultados en los conjunto de entrenamiento y validación hay mayor estabilidad en el **Modelo 1**; donde las gráficas se presentan de modo rasonable.