Final Transfer Learning

Héctor Manuel Cárdenas Yáñez | A01634615

Siddhartha López Valenzuela | A00227694

Álvaro Morán Errejón | A01638034

Isaí Ambrocio | A01625101

Librerias utilizadas

```
import matplotlib.pyplot as plt
import numpy as np
import os
import random
import tensorflow as tf
from tensorflow.keras import layers
```

Descomprimimos las carpetas, convertimos las imágenes a JPG, asignamos las etiquetas, creamos nuestro lote `BATCH_SIZE` y cambiamos el tamaño de las imágenes `IMG_SIZE`

```
import zipfile
from PIL import Image

def convert_images_to_jpeg(directory_path):
    # Iterate over all files in the directory
    for filename in os.listdir(directory_path):
        file_path = os.path.join(directory_path, filename)

        # Check if the file is an image (based on file extension)
        if filename.lower().endswith(('.png', '.jpg', '.jpeg',
'.tiff', '.bmp', '.gif')):
            # Open the image using PIL
            with Image.open(file_path) as img:
                # Convert and save the image as JPEG
                new_filename = os.path.splitext(filename)[0] + '.jpg'
# Change file extension to .jpg
                img.convert('RGB').save(os.path.join(directory_path,
new_filename), 'JPEG')

                # Delete the original image if it's not a JPEG
                if not filename.lower().endswith('.jpg'):
                    os.remove(file_path)

def unzip_dataset(zip_path, extract_to):
    with zipfile.ZipFile(zip_path, 'r') as zip_ref:
```

```
        zip_ref.extractall(extract_to)

zip_mouses_path = '/content/mouse-20231017T003417Z-001.zip'
zip_keyboards_path = '/content/teclado-20231017T003419Z-001.zip'
zip_monitors_path = '/content/monitor-20231017T003415Z-001.zip'

extract_mouses_dir = os.path.join('extracted_path_mouses')
extract_keyboards_dir = os.path.join('extracted_path_keyboards')
extract_monitors_dir = os.path.join('extracted_path_monitors')


def assign_labels(dataset, label):
    def _assign_labels(images, _):
        return images, tf.ones_like(_, dtype=tf.int32) * label
    return dataset.map(_assign_labels)

unzip_dataset(zip_mouses_path, extract_mouses_dir)
unzip_dataset(zip_keyboards_path, extract_keyboards_dir)
unzip_dataset(zip_monitors_path, extract_monitors_dir)

convert_images_to_jpeg(extract_mouses_dir)
convert_images_to_jpeg(extract_keyboards_dir)
convert_images_to_jpeg(extract_monitors_dir)

BATCH_SIZE = 32
IMG_SIZE = (160,160)
train_mouses_dataset = tf.keras.utils.image_dataset_from_directory(
    extract_mouses_dir,
    shuffle=True,
    batch_size=BATCH_SIZE,
    image_size=IMG_SIZE,
    label_mode='int'
)

train_keyboards_dataset = tf.keras.utils.image_dataset_from_directory(
    extract_keyboards_dir,
    shuffle=True,
    batch_size=BATCH_SIZE,
    image_size=IMG_SIZE,
    label_mode='int'
)

train_monitors_dataset = tf.keras.utils.image_dataset_from_directory(
    extract_monitors_dir,
    shuffle=True,
    batch_size=BATCH_SIZE,
    image_size=IMG_SIZE,
    label_mode='int'
)
```

```
train_mouses_dataset = assign_labels(train_mouses_dataset, 0)
train_keyboards_dataset = assign_labels(train_keyboards_dataset, 1)
train_monitors_dataset = assign_labels(train_monitors_dataset, 2)

train_dataset =
train_mouses_dataset.concatenate(train_keyboards_dataset).concatenate(
train_monitors_dataset)
```

```
Found 186 files belonging to 1 classes.
Found 253 files belonging to 1 classes.
Found 200 files belonging to 1 classes.
```

Realizamos el filtrado de imágenes y aquellas que no sean validas son removidas.

```
def filter_and_remove_invalid_images(directory_path):
    """Check each image in the directory, and if
    TensorFlow cannot read it, remove it."""

    removed_files = []
    for filename in os.listdir(directory_path):
        file_path = os.path.join(directory_path, filename)

        try:
            _ = tf.io.read_file(file_path)
            _ = tf.image.decode_image(_, channels=3)
        except Exception as e:
            os.remove(file_path)
            removed_files.append(filename)
    return removed_files
mouses_image_dir = os.path.join(extract_mouses_dir, "mouse")
keyboards_image_dir = os.path.join(extract_keyboards_dir, "teclado")
monitors_image_dir = os.path.join(extract_monitors_dir, "monitor")

removed_mouses = filter_and_remove_invalid_images(mouses_image_dir)
removed_keyboards =
filter_and_remove_invalid_images(keyboards_image_dir)
removed_monitors =
filter_and_remove_invalid_images(monitors_image_dir)

removed_mouses, removed_keyboards, removed_monitors
```

Configuración de los conjuntos de datos de imágenes para entrenar el modelo.

```
BATCH_SIZE = 32
IMG_SIZE = (160,160)
train_mouses_dataset = tf.keras.utils.image_dataset_from_directory(
    extract_mouses_dir,
    shuffle=True,
    batch_size=BATCH_SIZE,
    image_size=IMG_SIZE,
```

```
    label_mode='int'
)

train_keyboards_dataset = tf.keras.utils.image_dataset_from_directory(
    extract_keyboards_dir,
    shuffle=True,
    batch_size=BATCH_SIZE,
    image_size=IMG_SIZE,
    label_mode='int'
)

train_monitors_dataset = tf.keras.utils.image_dataset_from_directory(
    extract_monitors_dir,
    shuffle=True,
    batch_size=BATCH_SIZE,
    image_size=IMG_SIZE,
    label_mode='int'
)

train_mouses_dataset = assign_labels(train_mouses_dataset, 0)
train_keyboards_dataset = assign_labels(train_keyboards_dataset, 1)
train_monitors_dataset = assign_labels(train_monitors_dataset, 2)

train_dataset =
train_mouses_dataset.concatenate(train_keyboards_dataset).concatenate(
train_monitors_dataset)
```

```
Found 183 files belonging to 1 classes.
Found 245 files belonging to 1 classes.
Found 199 files belonging to 1 classes.
```

Creamos un nuevo dataset para la validación y prueba basándonos en un método similar al visto en clase.

```
train_dataset = train_dataset.shuffle(buffer_size=10000)

total_batches = tf.data.experimental.cardinality(train_dataset)

test_dataset = train_dataset.take(total_batches // 5)
temp_dataset = train_dataset.skip(total_batches // 5)

val_batches = tf.data.experimental.cardinality(temp_dataset)
validation_dataset = temp_dataset.take(val_batches // 5)
train_dataset = temp_dataset.skip(val_batches // 5)
```

Mejoramos la eficiencia en el acceso a los datos durante el entrenamiento.

```
AUTOTUNE = tf.data.AUTOTUNE

train_dataset = train_dataset.prefetch(buffer_size=AUTOTUNE)
```

```
validation_dataset = validation_dataset.prefetch(buffer_size=AUTOTUNE)
test_dataset = test_dataset.prefetch(buffer_size=AUTOTUNE)

data_augmentation = tf.keras.Sequential([
    layers.RandomFlip("horizontal_and_vertical"),
    layers.RandomRotation(0.2),
])

rescale =tf.keras.layers.Rescaling(1./127.5, offset =-1)
preprocess_input = tf.keras.applications.mobilenet_v2.preprocess_input

IMG_SIZE = (160,160)

IMG_SHAPE = IMG_SIZE + (3,)
print(IMG_SHAPE)

base_model = tf.keras.applications.ResNet50(input_shape=IMG_SHAPE,
include_top=False, weights='imagenet')
```

```
(160, 160, 3)
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/resnet/
resnet50_weights_tf_dim_ordering_tf_kernels_notop.h5
94765736/94765736 [==============================] - 5s 0us/step
```

Extraemos las características.

```
image_batch, label_batch = next(iter(train_dataset))
feature_batch = base_model(image_batch)
print(feature_batch.shape)
```

```
(32, 5, 5, 2048)
```

Congelado de las capas.

```
base_model.trainable=False
base_model.summary()

global_average_layer = tf.keras.layers.GlobalAveragePooling2D()
feature_batch_average = global_average_layer(feature_batch)
print(feature_batch_average.shape)
```

```
(32, 2048)
```

Capa densa para realizar predicciones sobre las características extraídas por el modelo base.

```
prediction_layer = tf.keras.layers.Dense(3, activation='softmax')
prediction_batch = prediction_layer(feature_batch_average)
print(prediction_batch.shape)
```

```
(32, 3)
```

Modelo completo

```
inputs=tf.keras.Input(shape=(160,160,3))
x = data_augmentation(inputs)
x = preprocess_input(x)
x = base_model(x, training=False)
x = global_average_layer(x)
x = tf.keras.layers.Dropout(0.2)(x)
outputs = prediction_layer(x)
model = tf.keras.Model(inputs,outputs)
```

Compilado del modelo.

```
base_learning_rate = 0.0001
model.compile(optimizer=tf.keras.optimizers.RMSprop(learning_rate=base
_learning_rate/10),

loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=False),
              metrics=['accuracy'])


model.summary()

initial_epochs = 10
#sin entrenar nada, el puro rendimiento del modelo que descargarmos
sin epochas
#ni entrenamiento
loss0, accuracy0 = model.evaluate(validation_dataset)

3/3 [==============================] - 5s 64ms/step - loss: 1.3103 -
accuracy: 0.0521
```

Historial a través de las épocas.

```
history = model.fit(
    train_dataset,
    epochs=initial_epochs,
    validation_data=(validation_dataset)
)

Epoch 1/10
14/14 [==============================] - 19s 599ms/step - loss: 1.0846
- accuracy: 0.4005 - val_loss: 1.1880 - val_accuracy: 0.2188
Epoch 2/10
14/14 [==============================] - 6s 284ms/step - loss: 1.0825
- accuracy: 0.4169 - val_loss: 1.2888 - val_accuracy: 0.1771
Epoch 3/10
14/14 [==============================] - 6s 307ms/step - loss: 1.1054
```

```
- accuracy: 0.3672 - val_loss: 1.3056 - val_accuracy: 0.1354
Epoch 4/10
14/14 [==============================] - 5s 244ms/step - loss: 1.1064
- accuracy: 0.3475 - val_loss: 1.2903 - val_accuracy: 0.1250
Epoch 5/10
14/14 [==============================] - 8s 454ms/step - loss: 1.0802
- accuracy: 0.4119 - val_loss: 1.2077 - val_accuracy: 0.1647
Epoch 6/10
14/14 [==============================] - 5s 232ms/step - loss: 1.1123
- accuracy: 0.3768 - val_loss: 1.0969 - val_accuracy: 0.2644
Epoch 7/10
14/14 [==============================] - 6s 306ms/step - loss: 1.0811
- accuracy: 0.4169 - val_loss: 1.1341 - val_accuracy: 0.2069
Epoch 8/10
14/14 [==============================] - 5s 233ms/step - loss: 1.0624
- accuracy: 0.4275 - val_loss: 1.1953 - val_accuracy: 0.1146
Epoch 9/10
14/14 [==============================] - 8s 458ms/step - loss: 1.0811
- accuracy: 0.4159 - val_loss: 1.2137 - val_accuracy: 0.1494
Epoch 10/10
14/14 [==============================] - 5s 234ms/step - loss: 1.1165
- accuracy: 0.3481 - val_loss: 1.1157 - val_accuracy: 0.3239
```

Ajuste fino.

```
base_model.trainable = True
fine_tune_at = 100
for layer in base_model.layers[:fine_tune_at]:
    layer.trainable = False
```

Compilado del modelo y su respectivo resumen.

```
model.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=base_le
arning_rate/10),

loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=False),
              metrics=['accuracy'])

model.summary()

Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_2 (InputLayer)        [(None, 160, 160, 3)]     0

 sequential (Sequential)     (None, 160, 160, 3)       0

 tf.math.truediv (TFOpLambd  (None, 160, 160, 3)       0
 a)
```

```
 tf.math.subtract (TFOpLamb  (None, 160, 160, 3)       0
 da)

 resnet50 (Functional)       (None, 5, 5, 2048)        23587712

 global_average_pooling2d (  (None, 2048)              0
 GlobalAveragePooling2D)

 dropout (Dropout)           (None, 2048)              0

 dense (Dense)               (None, 3)                 6147

=================================================================
Total params: 23593859 (90.00 MB)
Trainable params: 19459075 (74.23 MB)
Non-trainable params: 4134784 (15.77 MB)
_________________________________________________________________
```

Ajuste para mejorar el modelo.

```
fine_tune_epochs=10
total_epochs = initial_epochs + fine_tune_epochs

history = model.fit(
    train_dataset,
    epochs=total_epochs,
    initial_epoch = history.epoch[-1],
    validation_data=(validation_dataset)
)

Epoch 10/20
14/14 [==============================] - 28s 432ms/step - loss: 1.1479
- accuracy: 0.3301 - val_loss: 1.0230 - val_accuracy: 0.4941
Epoch 11/20
14/14 [==============================] - 7s 357ms/step - loss: 0.8827
- accuracy: 0.5888 - val_loss: 1.0107 - val_accuracy: 0.6588
Epoch 12/20
14/14 [==============================] - 6s 278ms/step - loss: 0.7676
- accuracy: 0.6359 - val_loss: 1.1863 - val_accuracy: 0.2000
Epoch 13/20
14/14 [==============================] - 6s 322ms/step - loss: 0.5807
- accuracy: 0.7568 - val_loss: 0.8054 - val_accuracy: 0.5938
Epoch 14/20
14/14 [==============================] - 6s 281ms/step - loss: 0.5566
- accuracy: 0.7565 - val_loss: 0.3219 - val_accuracy: 0.7931
Epoch 15/20
14/14 [==============================] - 7s 366ms/step - loss: 0.6843
- accuracy: 0.7047 - val_loss: 0.5834 - val_accuracy: 0.7188
Epoch 16/20
```

```
14/14 [==============================] - 6s 280ms/step - loss: 0.4699
- accuracy: 0.8301 - val_loss: 0.3627 - val_accuracy: 0.8542
Epoch 17/20
14/14 [==============================] - 7s 356ms/step - loss: 0.4135
- accuracy: 0.8599 - val_loss: 0.4109 - val_accuracy: 0.8229
Epoch 18/20
14/14 [==============================] - 6s 289ms/step - loss: 0.4234
- accuracy: 0.8164 - val_loss: 0.4516 - val_accuracy: 0.8592
Epoch 19/20
14/14 [==============================] - 10s 510ms/step - loss: 0.3786
- accuracy: 0.8594 - val_loss: 0.5226 - val_accuracy: 0.8873
Epoch 20/20
14/14 [==============================] - 6s 320ms/step - loss: 0.3714
- accuracy: 0.8592 - val_loss: 0.2478 - val_accuracy: 0.9155
```

Hacemos las gráficas para comparar el cambio de `Training Accuracy` y `Validation Accuracy` a traves de las épocas.

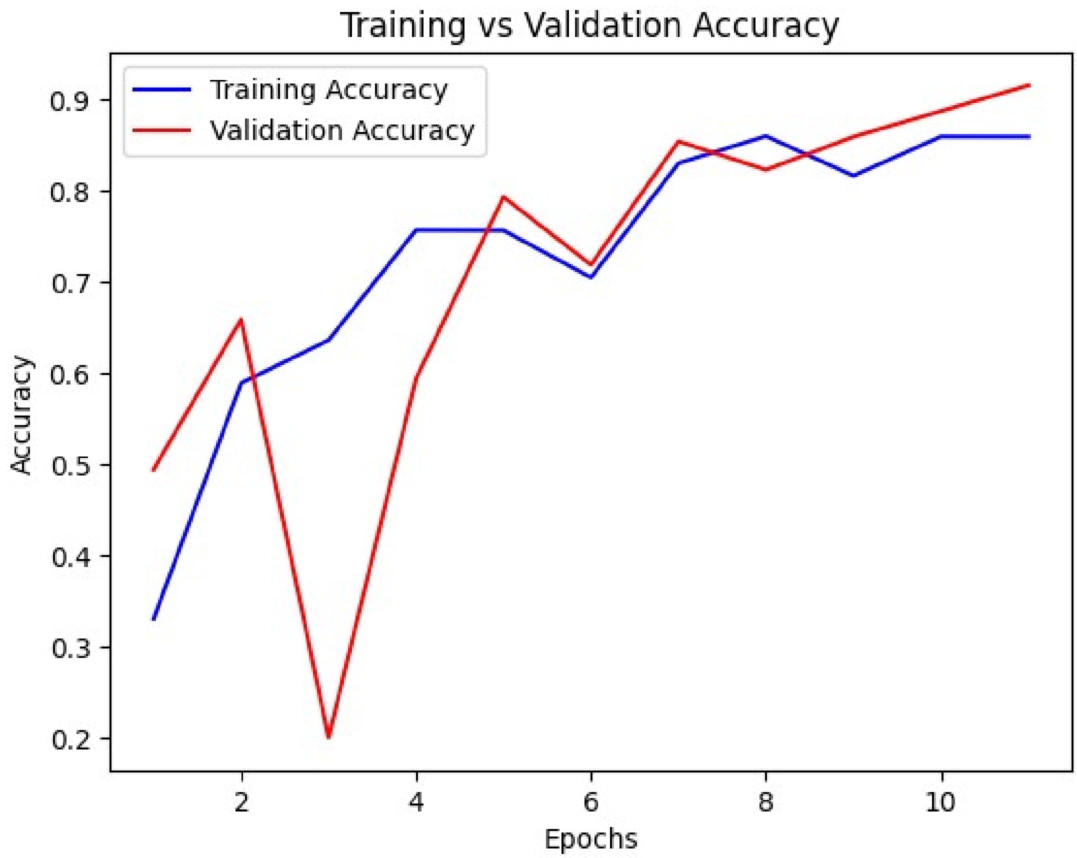
```python
import matplotlib.pyplot as plt


train_acc = history.history['accuracy']
val_acc = history.history['val_accuracy']

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

plt.plot(epochs, train_acc, 'b', label='Training Accuracy')
plt.plot(epochs, val_acc, 'r', label='Validation Accuracy')
plt.title('Training vs Validation Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()

plt.show()
```




```
print(image_batch.shape)
```

```
(1, 21, 160, 160, 3)
```

Redimensionamos las imágenes del conjunto de prueba, realizamos las predicciones con el modelo previamente entrenado y visualizamos una imagen del conjunto de prueba junto con la etiqueta verdadera y la predicha.

```
def resize_image(image, label):
    return tf.image.resize(image, [160, 160]), label

test_dataset = test_dataset.map(resize_image)

image_batch, label_batch =
next(iter(test_dataset.shuffle(1000).batch(1)))

if len(image_batch.shape) == 5:
    image_batch = image_batch[0]

image = image_batch[0]
true_label = label_batch[0]

def extract_scalar(value):
```

```
    if isinstance(value, (np.ndarray, tf.Tensor)):
        return int(value.flatten()[0])
    return int(value)

predictions = model.predict(image_batch)
predicted_label = np.argmax(predictions[0])

class_names = ["Mouse", "Keyboard", "Monitor"]

true_label_value = extract_scalar(true_label.numpy())

plt.imshow(image.numpy().astype("uint8"))
plt.title(f"True label: {class_names[true_label_value]} \n Predicted:
{class_names[predicted_label]}")
plt.axis("off")
plt.show()

1/1 [==============================] - 0s 26ms/step
```

True label: Keyboard
Predicted: Keyboard