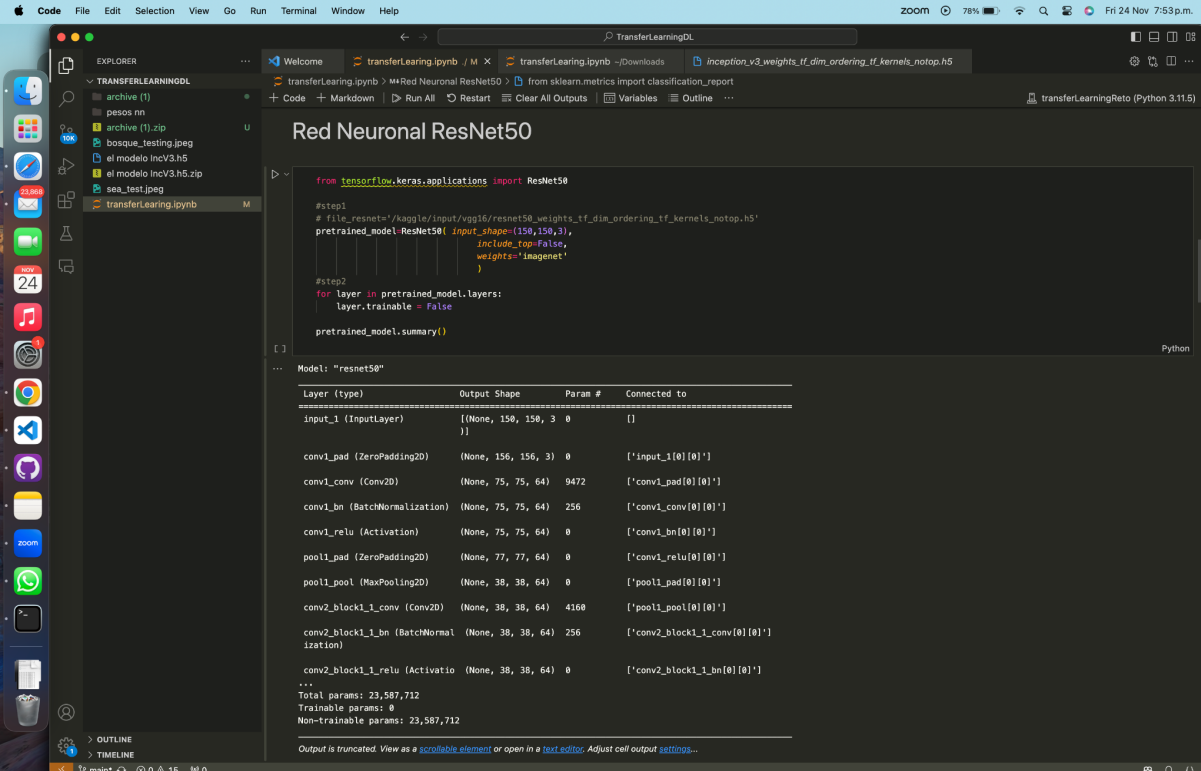


Implementación de un modelo de Deep Learning

Modelos de aprendizaje profundo

ResNet50



```
from tensorflow.keras.applications import ResNet50

#step1
# file_resnet="/kaggle/input/vgg16/resnet50_weights_tf_dim_ordering_tf_kernels_notop.h5"
pretrained_model=ResNet50(input_shape=(150,150,3),
                           include_top=False,
                           weights='imagenet')

#step2
for layer in pretrained_model.layers:
    layer.trainable = False

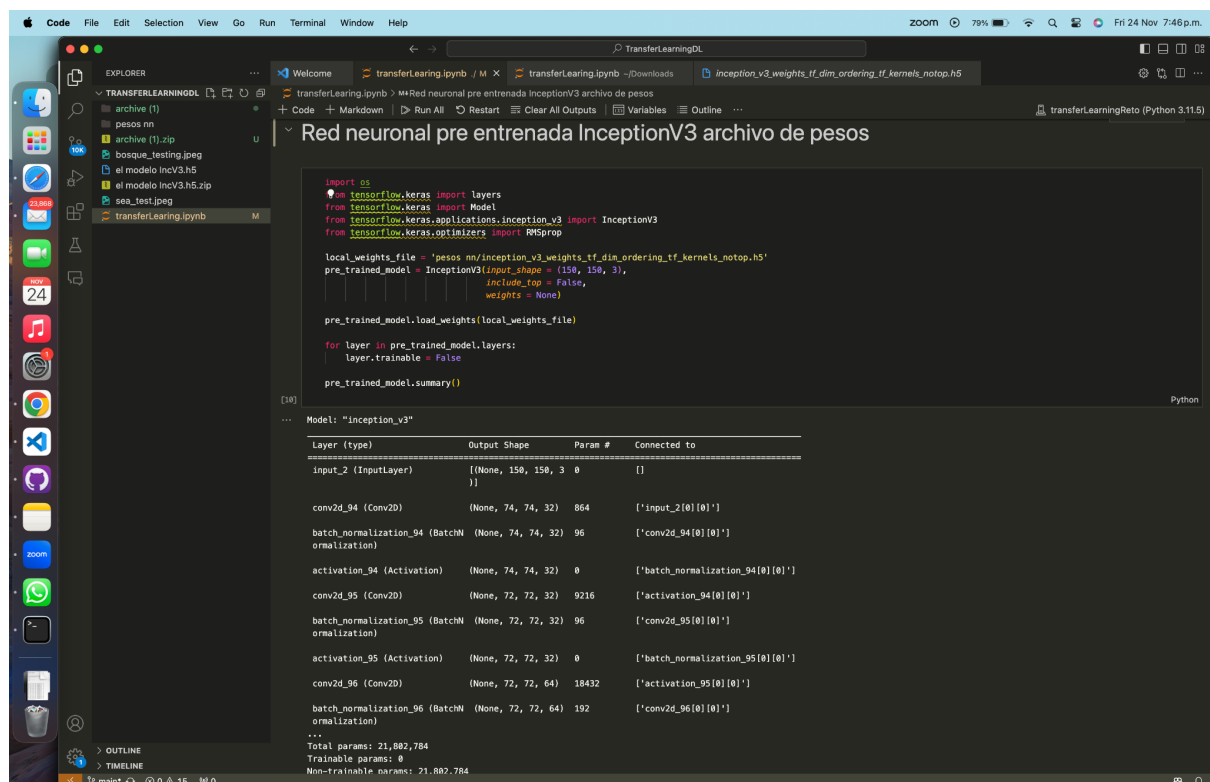
pretrained_model.summary()
```

Layer (type)	Output Shape	Param #	Connected to
input_1 (InputLayer)	(None, 150, 150, 3)	0	input_1[0][0]
conv1_pad (ZeroPadding2D)	(None, 156, 156, 3)	0	conv1_conv[0][0]
conv1_conv (Conv2D)	(None, 75, 75, 64)	9472	conv1_pad[0][0]
conv1_bn (BatchNormalization)	(None, 75, 75, 64)	256	conv1_conv[0][0]
conv1_relu (Activation)	(None, 75, 75, 64)	0	conv1_bn[0][0]
pool1_pad (ZeroPadding2D)	(None, 77, 77, 64)	0	conv1_relu[0][0]
pool1_pool (MaxPooling2D)	(None, 38, 38, 64)	0	pool1_pad[0][0]
conv2_block1_1_conv (Conv2D)	(None, 38, 38, 64)	4360	pool1_pool[0][0]
conv2_block1_1_bn (BatchNormalization)	(None, 38, 38, 64)	256	conv2_block1_1_conv[0][0]
conv2_block1_1_relu (Activation)	(None, 38, 38, 64)	0	conv2_block1_1_bn[0][0]
...
Total params:	23,587,712		
Trainable params:	0		
Non-trainable params:	23,587,712		

La red neuronal convolucional ResNet50 es un modelo profundo ampliamente reconocido en el campo de la visión por computadora. Se destaca por su estructura única que aborda el problema del desvanecimiento del gradiente en redes más profundas a través de conexiones residuales. Estas conexiones permiten que los datos y gradientes fluyan más fácilmente a través de las capas, facilitando el entrenamiento de redes excepcionalmente profundas.

Este código utiliza la funcionalidad de Keras para importar ResNet50 junto con sus pesos pre entrenados en el conjunto de datos ImageNet. Esta red pre entrenada puede ser utilizada directamente para tareas de clasificación de imágenes o puede ser adaptada a un problema específico mediante técnicas como transfer learning.

InceptionV3



The screenshot shows a Jupyter Notebook titled 'Red neuronal pre entrenada InceptionV3 archivo de pesos' in a VS Code environment. The notebook contains Python code to load a pre-trained InceptionV3 model using TensorFlow and Keras. The code imports necessary modules, defines the local weights file path, creates an InceptionV3 model with specific input shape and weights, and then loads the weights. Finally, it prints the model summary.

```
import os
from tensorflow.keras import layers
from tensorflow.keras import Model
from tensorflow.keras.applications.inception_v3 import InceptionV3
from tensorflow.keras.optimizers import RMSprop

local_weights_file = "pesos nn/inception_v3_weights_tf_dim_ordering_tf_kernels_notop.h5"
pre_trained_model = InceptionV3(input_shape = (150, 150, 3),
                                include_top = False,
                                weights = None)

pre_trained_model.load_weights(local_weights_file)

for layer in pre_trained_model.layers:
    layer.trainable = False

pre_trained_model.summary()
```

The output of the summary() function is displayed below the code:

```
[10]
Model: "inception_v3"

```

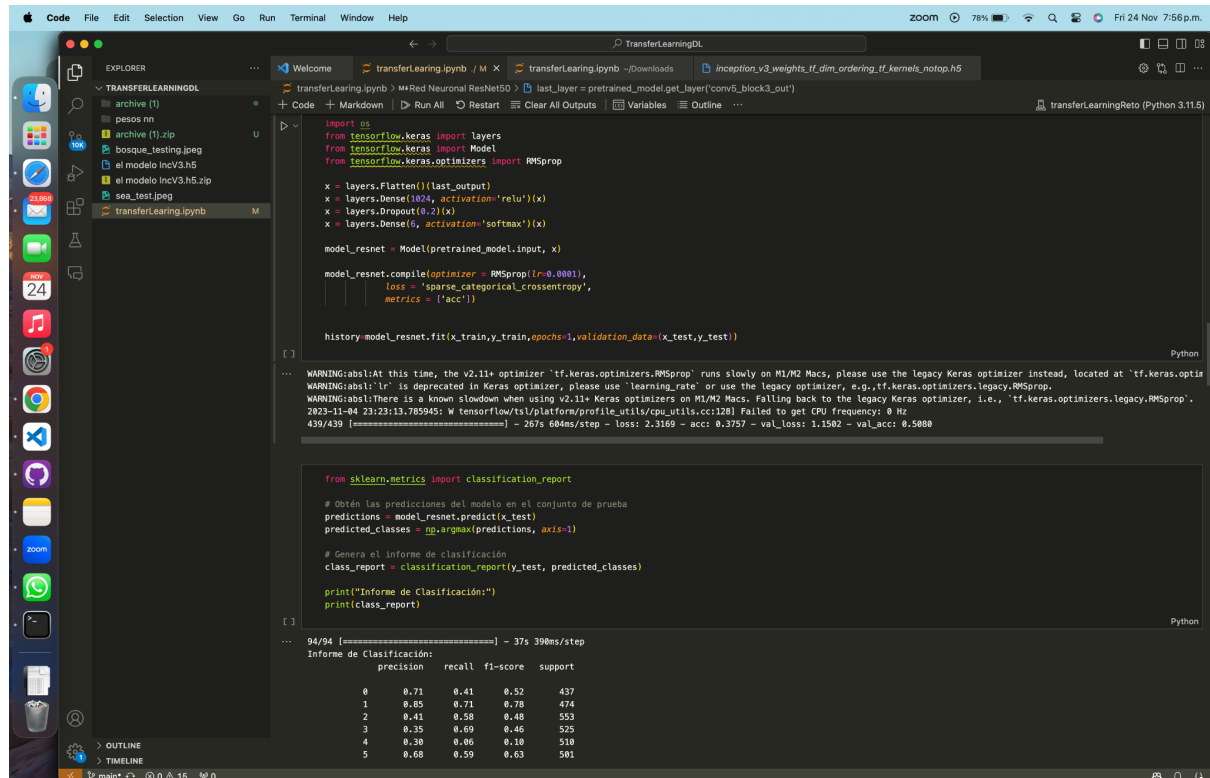
Layer (type)	Output Shape	Param #	Connected to
input_2 (InputLayer)	(None, 150, 150, 3)	0	['input_2[0][0]']
conv2d_94 (Conv2D)	(None, 74, 74, 32)	864	['conv2d_94[0][0]']
batch_normalization_94 (Batch Normalization)	(None, 74, 74, 32)	96	['batch_normalization_94[0][0]']
activation_94 (Activation)	(None, 74, 74, 32)	0	['activation_94[0][0]']
conv2d_95 (Conv2D)	(None, 72, 72, 32)	9216	['conv2d_95[0][0]']
batch_normalization_95 (Batch Normalization)	(None, 72, 72, 32)	96	['batch_normalization_95[0][0]']
activation_95 (Activation)	(None, 72, 72, 32)	0	['activation_95[0][0]']
conv2d_96 (Conv2D)	(None, 72, 72, 64)	18432	['conv2d_96[0][0]']
batch_normalization_96 (Batch Normalization)	(None, 72, 72, 64)	192	['batch_normalization_96[0][0]']
Total params: 21,882,784			
Trainable params: 0			
Non-trainable params: 21,882,784			

InceptionV3 es una red neuronal convolucional (CNN) desarrollada por Google, conocida por su profundidad y eficiencia. Se caracteriza por su utilización de módulos de 'inception', los cuales permiten procesar información en diferentes escalas de manera simultánea, lo que la hace especialmente útil para tareas de visión por computadora, como la clasificación de imágenes y la detección de objetos.

Se empleó la arquitectura InceptionV3 como punto de partida, utilizando Tensorflow como framework principal para el entrenamiento del modelo de aprendizaje profundo. Además de Tensorflow, se hicieron uso de librerías complementarias para la carga de pesos pre-entrenados de la red que se encuentran en el archivo 'inception_v3_weights_tf_dim_ordering_tf_kernels_notop.h5'.

Evalúa el desempeño del modelo en su aproximación inicial y realiza ajustes para mejorar su desempeño

ResNet50



```
import os
from tensorflow.keras import layers
from tensorflow.keras import Model
from tensorflow.keras.optimizers import RMSprop

x = layers.Flatten()(last_output)
x = layers.Dense(1024, activation='relu')(x)
x = layers.Dropout(0.2)(x)
x = layers.Dense(6, activation='softmax')(x)

model_resnet = Model(pretrained_model.input, x)

model_resnet.compile(optimizer = RMSprop(lr=0.0001),
                    loss = 'sparse_categorical_crossentropy',
                    metrics = ['acc'])

history = model_resnet.fit(x_train, y_train, epochs=1, validation_data=(x_test, y_test))

# Dividí las predicciones del modelo en el conjunto de prueba
predictions = model_resnet.predict(x_test)
predicted_classes = np.argmax(predictions, axis=-1)

# Genera el informe de clasificación
class_report = classification_report(y_test, predicted_classes)

print("Informe de Clasificación:")
print(class_report)
```

94/94 [=====] - 37s 390ms/step

Informe de Clasificación:

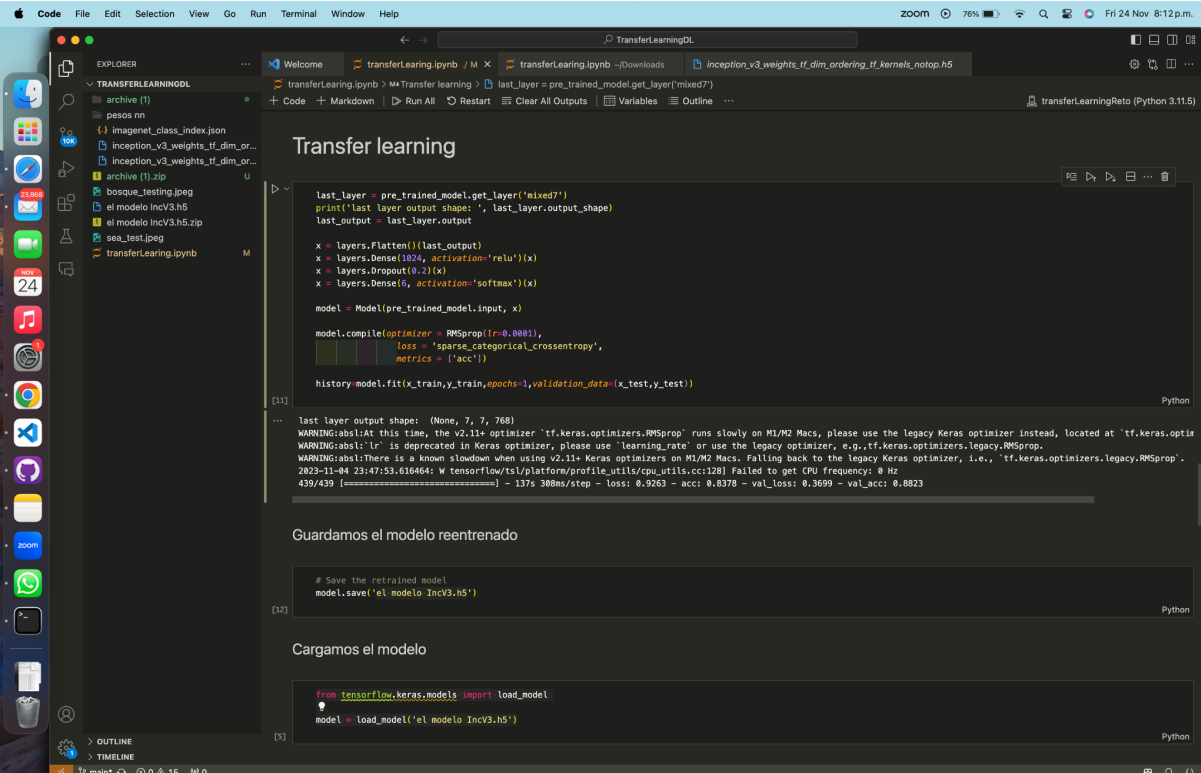
	precision	recall	f1-score	support
0	0.71	0.41	0.52	437
1	0.85	0.71	0.78	474
2	0.41	0.58	0.48	553
3	0.35	0.69	0.46	525
4	0.28	0.46	0.33	519
5	0.68	0.59	0.63	501

En este fragmento de código, se ilustra cómo se añaden capas adicionales a la red ResNet50 para adaptarla a nuestro problema de clasificación en diferentes entornos. El código comienza con la incorporación de una capa de aplanado (Flatten) seguida por dos capas totalmente conectadas (Dense). La primera capa densa tiene 1024 neuronas con una función de activación ReLU, y luego se aplica una capa de regularización (Dropout) para prevenir el sobreajuste. Por último, se añade una capa de salida con 6 neuronas y activación softmax para la clasificación.

El modelo de ResNet50 se compila utilizando el optimizador RMSprop con una tasa de aprendizaje de 0.0001, empleando la función de pérdida 'sparse_categorical_crossentropy' para el entrenamiento. Durante la ejecución del código, se observan advertencias relacionadas con el optimizador RMSprop, lo cual puede impactar el rendimiento en ciertos entornos, específicamente en sistemas M1/M2 Macs.

El modelo presenta un ajuste deficiente a los datos, reflejado en una precisión (accuracy) del 0.51 en el conjunto de entrenamiento. A pesar de ello, existen oportunidades claras para mejoras. Podríamos explorar diversas funciones de optimización con el objetivo de alcanzar un rendimiento más sólido. Asimismo, considerar la posibilidad de probar otra arquitectura, como InceptionV3, podría ser beneficioso para mejorar el desempeño general del modelo.

InceptionV3



```
last_layer = pre_trained_model.get_layer('mixed7')
print('last layer output shape: ', last_layer.output_shape)
last_output = last_layer.output

x = layers.Flatten()(last_output)
x = layers.Dense(1024, activation='relu')(x)
x = layers.Dropout(0.2)(x)
x = layers.Dense(6, activation='softmax')(x)

model = Model(pre_trained_model.input, x)

model.compile(optimizer = RMSprop(lr=0.0001),
              loss = 'sparse_categorical_crossentropy',
              metrics = ['acc'])

history = model.fit(x_train, y_train, epochs=1, validation_data=(x_test, y_test))

last_layer output shape: (None, 7, 7, 768)
WARNING:absl:At this time, the v2.11+ optimizer 'tf.keras.optimizers.RMSprop' runs slowly on M1/M2 Macs, please use the legacy Keras optimizer instead, located at 'tf.keras.optimizers.absl:lr' is deprecated in Keras optimizer, please use 'learning_rate' or use the legacy optimizer, e.g., 'tf.keras.optimizers.legacy.RMSprop'.
WARNING:absl:There is a known slowdown when using v2.11+ Keras optimizers on M1/M2 Macs. Falling back to the legacy Keras optimizer, i.e., 'tf.keras.optimizers.legacy.RMSprop'.
2023-11-04 23:47:53.616464: W tensorflow/tsl/platform/profile_utils/cpu_utils.cc:128] Failed to get CPU frequency: 0 Hz
439/439 [=====] - 137s 380ms/step - loss: 0.9263 - acc: 0.8378 - val_loss: 0.3699 - val_acc: 0.8823

Guardamos el modelo reentrenado

# Save the retrained model
model.save('el modelo IncV3.h5')

Cargamos el modelo

from tensorflow.keras.models import load_model
#
model = load_model('el modelo IncV3.h5')
```

En este fragmento de código, se muestra cómo se añaden capas adicionales a la red neuronal para adaptarla a nuestro problema de clasificación en diversos entornos. Este proceso implica la extensión de una arquitectura preexistente, como InceptionV3, mediante la incorporación de capas especializadas.

Se utiliza una combinación de capas como 'Flatten', 'Dense' y 'Dropout' para ajustar la red a nuestras necesidades específicas. Se establecen conexiones entre estas capas, seguidas de funciones de activación como 'relu' y 'softmax' para asegurar la adecuada transformación de los datos y la generación de predicciones precisas.

Una vez definidas las nuevas capas, se construye un modelo basado en InceptionV3 con estas modificaciones. Posteriormente, se compila el modelo, especificando el optimizador, la función de pérdida y las métricas relevantes para el problema de clasificación.

Los resultados del modelo InceptionV3 muestran una tendencia favorable en comparación con ResNet50. En InceptionV3, la pérdida de entrenamiento es más baja (0.9263) y la precisión es considerablemente mayor (83.78%) en los datos de entrenamiento. Además, la pérdida de validación es menor (0.3699) y la precisión en los datos de validación es significativamente más alta (88.23%) en comparación con ResNet50.

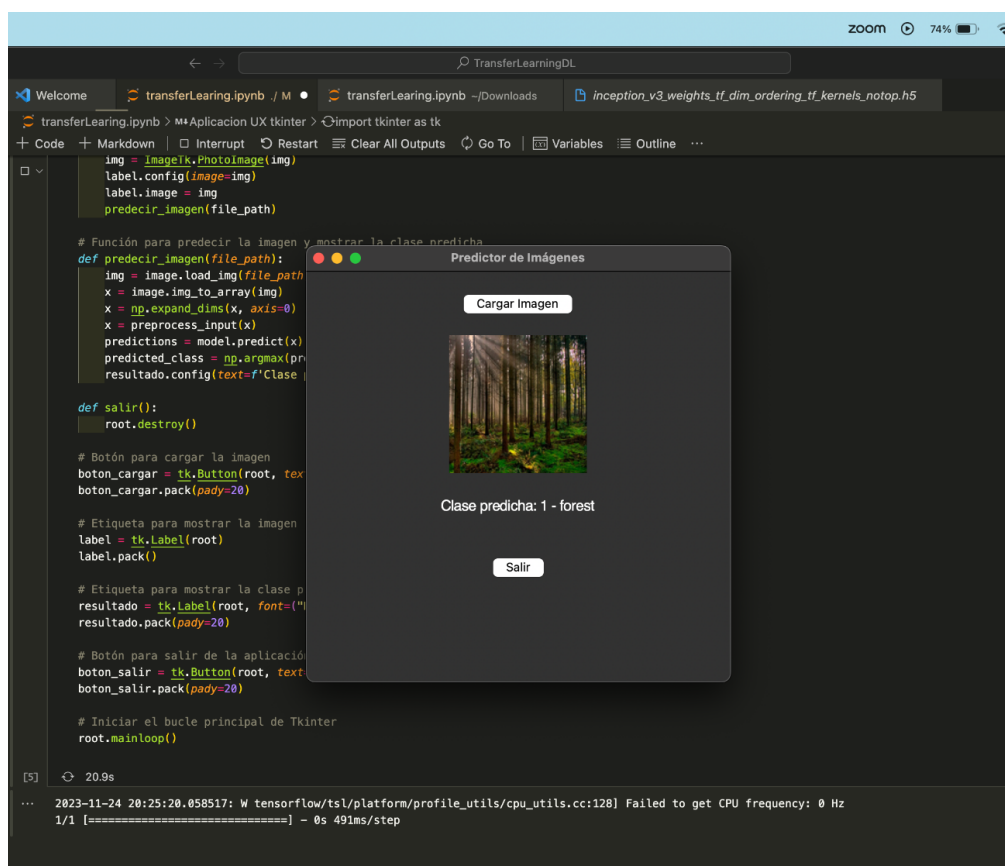
Es por esto que se optó por utilizar InceptionV3 debido a su mejor desempeño en esta evaluación. InceptionV3 parece adaptarse mejor a la tarea de clasificación utilizada en este escenario particular, logrando una menor pérdida y una precisión más alta tanto en los datos de entrenamiento como en los de validación. Esto sugiere que InceptionV3 podría capturar de manera más efectiva los patrones relevantes en los datos y generalizar mejor para hacer predicciones precisas en datos no vistos.

Utiliza un conjunto de datos reales (no ejemplos de clase), para la creación del modelo.

En la carpeta de [Drive](#), se encuentra el archivo 'baseDatos.zip' que contiene las imágenes empleadas para el reentrenamiento de las redes neuronales. También se incluye el archivo de pesos de la red que demostró mejores resultados, en caso de desear ejecutar el código 'transferLearning.ipynb'.

<https://drive.google.com/drive/folders/1q1aYZsa25V7x7Su8rx5lWaSuzdyFwadi?usp=sharing>

El modelo puede generar predicciones o recomendaciones a través de la consola o una interfaz gráfica.



Se desarrolló una interfaz gráfica intuitiva utilizando la librería Tkinter, la cual permite al usuario cargar imágenes de su elección. Posteriormente, el modelo se ejecuta sobre la imagen cargada y muestra al usuario el resultado de la predicción. Este proceso y la interfaz se pueden ver en la imagen de arriba. Además se incluyen 2 imágenes de prueba para que se pruebe el algoritmo si así se desea, ya con el archivo de pesos y las imágenes de prueba se puede correr el código a partir de la sección Aplicación UX tkinter del notebook (transferLearning.ipynb) y ver el algoritmo funcionando.