Tarea 3: Utilizar una arquitectura base para practicar Transfer Learning

Maximiliano Martinez Marquez A01251527

Luis Cano Irigoyen A00827178

Aylin Camacho Reyes A01379272

Gilberto Ramos Salinas A01734128

```
from keras.layers import Input, Lambda, Dense, Flatten
from keras.models import Model
from keras.applications.vgg16 import VGG16
from keras.applications.vgg16 import preprocess_input
from keras.preprocessing import image
from keras.preprocessing.image import ImageDataGenerator
from keras.models import Sequential
import numpy as np
from glob import glob
import matplotlib.pyplot as plt
import warnings
warnings.filterwarnings("ignore", category=FutureWarning)


IMAGE_SIZE = [224, 224]


from google.colab import drive
drive.mount('/content/drive')
```

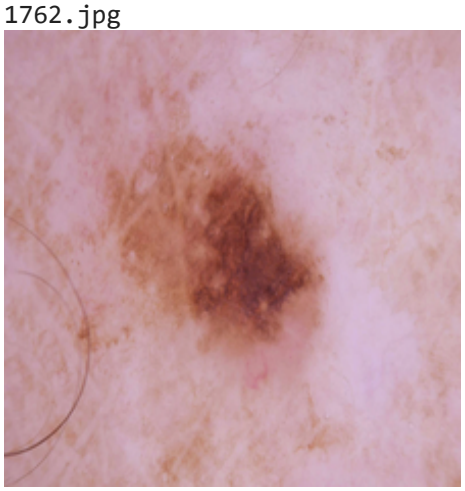
```
     Mounted at /content/drive
```

```
!ls '/content/drive'
```

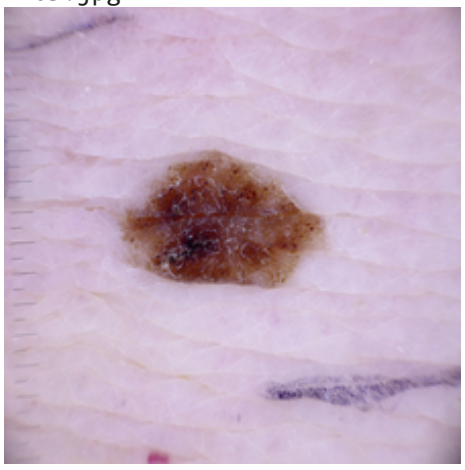
```
     MyDrive  Shareddrives
```

```
#Give dataset path
train_path = '/content/drive/MyDrive/skincancer/train'
test_path = '/content/drive/MyDrive/skincancer/test'


from PIL import Image
import os
from IPython.display import display
from IPython.display import Image as _Imgdis
# creating a object
folder = train_path+'/benign'

onlybenignfiles = [f for f in os.listdir(folder) if os.path.isfile(os.path.join(folder, f))]
print("Working with {0} images".format(len(onlybenignfiles)))
print("Image examples: ")

for i in range(10):
    print(onlybenignfiles[i])
    display(_Imgdis(filename=folder + "/" + onlybenignfiles[i], width=240, height=240))
```

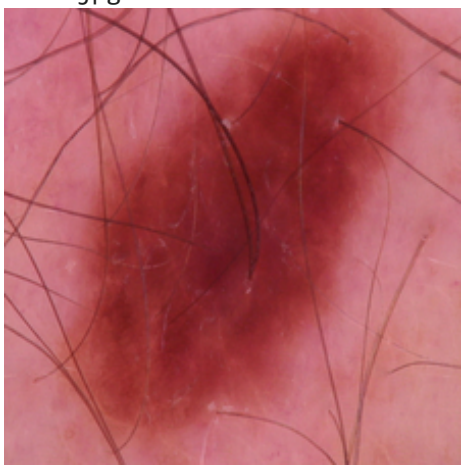
Working with 1440 images
Image examples:
1762.jpg



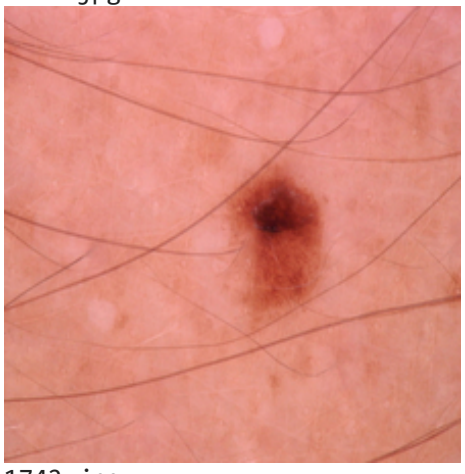
1405.jpg



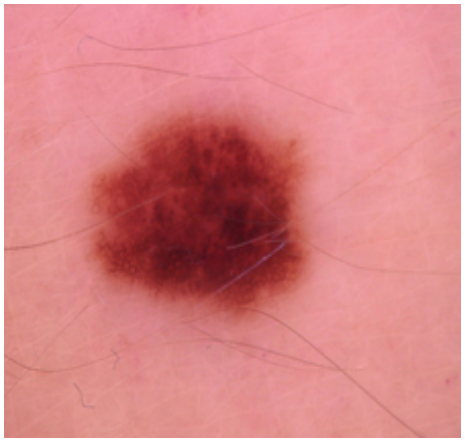
1760.jpg



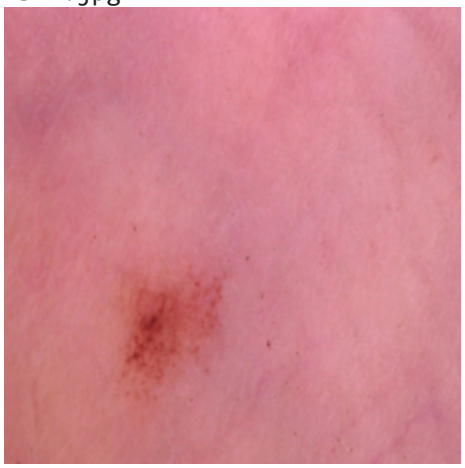
1431.jpg



1742.jpg

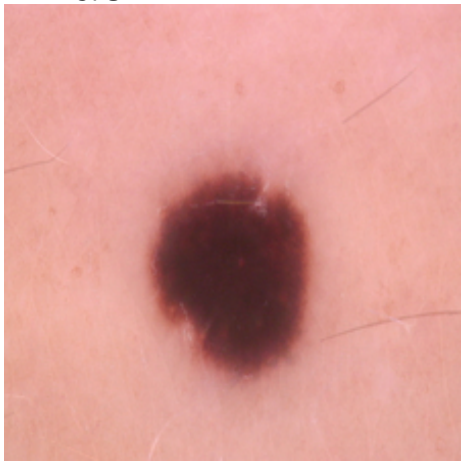


1574.jpg

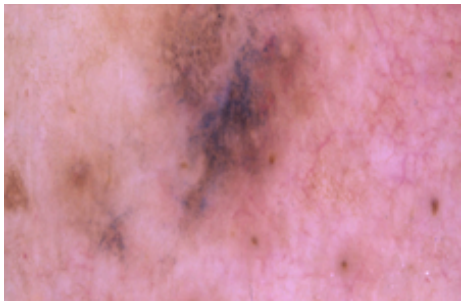


1523.jpg

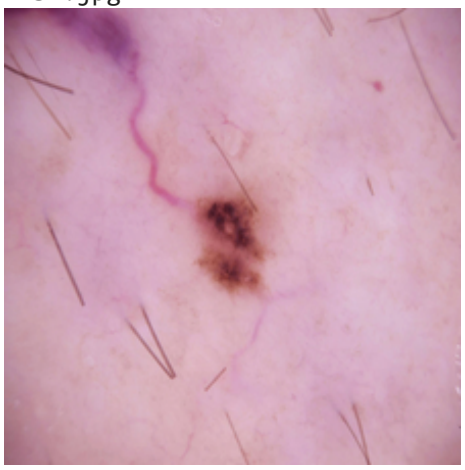
1557.jpg



1430.jpg



1437.jpg



Utilizamos un modelo de CNN preentrenado

```
vgg = VGG16(input_shape=IMAGE_SIZE + [3], weights='imagenet', include_top=False)
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/vgg16/v
58889256/58889256 [==============================] - 0s 0us/step
```

```
vgg.input
```

```
<KerasTensor: shape=(None, 224, 224, 3) dtype=float32 (created by layer 'input_1')>
```

```
for layer in vgg.layers:
  layer.trainable = False
```

```
folders = glob('/content/drive/MyDrive/skincancer/train/*')
print(len(folders))
```

```
2
```

```
x = Flatten()(vgg.output)
prediction = Dense(len(folders), activation='softmax')(x)
model = Model(inputs=vgg.input, outputs=prediction)
model.summary()
```

```
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_1 (InputLayer)        [(None, 224, 224, 3)]     0
```

```
 block1_conv1 (Conv2D)       (None, 224, 224, 64)      1792

 block1_conv2 (Conv2D)       (None, 224, 224, 64)      36928

 block1_pool (MaxPooling2D)  (None, 112, 112, 64)      0

 block2_conv1 (Conv2D)       (None, 112, 112, 128)     73856

 block2_conv2 (Conv2D)       (None, 112, 112, 128)     147584

 block2_pool (MaxPooling2D)  (None, 56, 56, 128)       0

 block3_conv1 (Conv2D)       (None, 56, 56, 256)       295168

 block3_conv2 (Conv2D)       (None, 56, 56, 256)       590080

 block3_conv3 (Conv2D)       (None, 56, 56, 256)       590080

 block3_pool (MaxPooling2D)  (None, 28, 28, 256)       0

 block4_conv1 (Conv2D)       (None, 28, 28, 512)       1180160

 block4_conv2 (Conv2D)       (None, 28, 28, 512)       2359808

 block4_conv3 (Conv2D)       (None, 28, 28, 512)       2359808

 block4_pool (MaxPooling2D)  (None, 14, 14, 512)       0

 block5_conv1 (Conv2D)       (None, 14, 14, 512)       2359808

 block5_conv2 (Conv2D)       (None, 14, 14, 512)       2359808

 block5_conv3 (Conv2D)       (None, 14, 14, 512)       2359808

 block5_pool (MaxPooling2D)  (None, 7, 7, 512)         0

 flatten (Flatten)           (None, 25088)             0

 dense (Dense)               (None, 2)                 50178

=================================================================
Total params: 14,764,866
Trainable params: 50,178
Non-trainable params: 14,714,688
_________________________________________________________________
```

Compilamos al modelo

```
from keras import optimizers


adam = optimizers.Adam()
model.compile(loss='binary_crossentropy',
              optimizer=adam,
              metrics=['accuracy'])
```

Preparamos los datos para entrenamiento y prueba

```
train_datagen = ImageDataGenerator(
    preprocessing_function=preprocess_input,
    rotation_range=40,
    width_shift_range=0.2,
    height_shift_range=0.2,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True,
    fill_mode='nearest')

test_datagen = ImageDataGenerator(
    preprocessing_function=preprocess_input,
    rotation_range=40,
    width_shift_range=0.2,
    height_shift_range=0.2,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True,
    fill_mode='nearest')


train_set = train_datagen.flow_from_directory(train_path,
                                                 target_size = (224, 224),
                                                 batch_size = 32,
                                                 class_mode = 'categorical')
```

```
Found 2637 images belonging to 2 classes.
```

```
test_set = test_datagen.flow_from_directory(test_path,
                                            target_size = (224, 224),
                                            batch_size = 32,
                                            class_mode = 'categorical')
```

```
Found 660 images belonging to 2 classes.
```

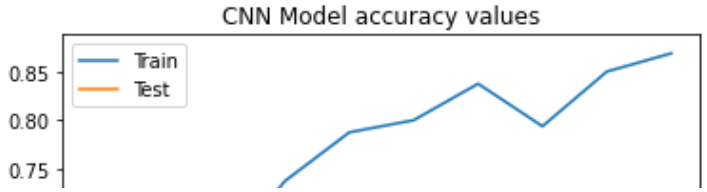
Creamos un checkpoint y entrenamos al modelo de TL con las imágenes de cancer de piel

```
from datetime import datetime
from keras.callbacks import ModelCheckpoint



checkpoint = ModelCheckpoint(filepath='mymodel.h5',
                             verbose=2, save_best_only=True)

callbacks = [checkpoint]

start = datetime.now()

model_history=model.fit_generator(
  train_set,
  validation_data=test_set,
  epochs=10,
  steps_per_epoch=5,
  validation_steps=32,
    callbacks=callbacks ,verbose=2)
```

```
duration = datetime.now() - start
print("Training completed in time: ", duration)
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:19: UserWarning: `Model.fit_g
Epoch 1/10
WARNING:tensorflow:Your input ran out of data; interrupting training. Make sure that your

Epoch 1: val_loss improved from inf to 5.96058, saving model to mymodel.h5
5/5 - 544s - loss: 4.8113 - accuracy: 0.5625 - val_loss: 5.9606 - val_accuracy: 0.4818 - 5
Epoch 2/10
WARNING:tensorflow:Can save best model only with val_loss available, skipping.
5/5 - 38s - loss: 3.5458 - accuracy: 0.6525 - 38s/epoch - 8s/step
Epoch 3/10
WARNING:tensorflow:Can save best model only with val_loss available, skipping.
5/5 - 42s - loss: 4.7592 - accuracy: 0.6562 - 42s/epoch - 8s/step
Epoch 4/10
WARNING:tensorflow:Can save best model only with val_loss available, skipping.
5/5 - 43s - loss: 3.0804 - accuracy: 0.7375 - 43s/epoch - 9s/step
Epoch 5/10
WARNING:tensorflow:Can save best model only with val_loss available, skipping.
5/5 - 36s - loss: 3.3308 - accuracy: 0.7875 - 36s/epoch - 7s/step
Epoch 6/10
WARNING:tensorflow:Can save best model only with val_loss available, skipping.
5/5 - 46s - loss: 1.6956 - accuracy: 0.8000 - 46s/epoch - 9s/step
Epoch 7/10
WARNING:tensorflow:Can save best model only with val_loss available, skipping.
5/5 - 28s - loss: 2.0462 - accuracy: 0.8375 - 28s/epoch - 6s/step
Epoch 8/10
WARNING:tensorflow:Can save best model only with val_loss available, skipping.
5/5 - 33s - loss: 1.6401 - accuracy: 0.7937 - 33s/epoch - 7s/step
Epoch 9/10
WARNING:tensorflow:Can save best model only with val_loss available, skipping.
5/5 - 29s - loss: 1.9187 - accuracy: 0.8500 - 29s/epoch - 6s/step
Epoch 10/10
WARNING:tensorflow:Can save best model only with val_loss available, skipping.
5/5 - 26s - loss: 1.1380 - accuracy: 0.8687 - 26s/epoch - 5s/step
Training completed in time:  0:16:22.781050
```

```
_# Plot training & validation loss values
plt.plot(model_history.history['accuracy'])
plt.plot(model_history.history['val_accuracy'])
plt.title('CNN Model accuracy values')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Train', 'Test'], loc='upper left')
plt.show()
```



Terminamos con una accuracy de 86.9% en las imágenes de cancer de piel