import tensorflow as tf

tf.\_\_version\_\_

# CNN params

img\_dimensions = 500

img\_channels = 1

img\_mode = 'grayscale'

import os

from tensorflow.keras import layers

from tensorflow.keras import Model

!wget --no-check-certificate \

https://storage.googleapis.com/mledu-datasets/inception\_v3\_weights\_tf\_dim\_ordering\_tf\_kernels\_notop.h5 \

-O /content/inception\_v3\_weights\_tf\_dim\_ordering\_tf\_kernels\_notop.h5

from tensorflow.keras.applications.inception\_v3 import InceptionV3

local\_weights\_file = '/content/inception\_v3\_weights\_tf\_dim\_ordering\_tf\_kernels\_notop.h5'

pre\_trained\_model = InceptionV3(input\_shape = (img\_dimensions, img\_dimensions, 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()

last\_layer = pre\_trained\_model.get\_layer('mixed7')

print('last layer output shape: ', last\_layer.output\_shape)

last\_output = last\_layer.output

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(1, activation='sigmoid')(x)

model = Model(pre\_trained\_model.input, x)

model.compile(optimizer='adam',

loss='binary\_crossentropy',

metrics=['accuracy'])

## Download dataset

!pip install kaggle

!KAGGLE\_CONFIG\_DIR=/content/kaggle kaggle datasets download -d pcbreviglieri/pneumonia-xray-images

## Prepare for ingest

from tensorflow.keras.preprocessing.image import ImageDataGenerator

import os

import zipfile

local\_zip = '/content/pneumonia-xray-images.zip'

zip\_ref = zipfile.ZipFile(local\_zip, 'r')

zip\_ref.extractall('/content/pneumonia-xray-images')

zip\_ref.close()

# Define our example directories and files

base\_dir = '/content/pneumonia-xray-images'

train\_dir = os.path.join(base\_dir, 'train')

validation\_dir = os.path.join(base\_dir, 'val')

train\_normal\_dir = os.path.join(train\_dir, 'normal')

train\_pneumonia\_dir = os.path.join(train\_dir, 'opacity')

validation\_normal\_dir = os.path.join(validation\_dir, 'normal')

validation\_pneumonia\_dir = os.path.join(validation\_dir, 'opacity')

train\_normal\_fnames = os.listdir(train\_normal\_dir)

train\_pneumonia\_fnames = os.listdir(train\_pneumonia\_dir)

# Add our data-augmentation parameters to ImageDataGenerator

train\_datagen = ImageDataGenerator(rescale = 1./255,

shear\_range = 0.2,

zoom\_range = 0.2,

vertical\_flip = True)

# Note that the validation data should not be augmented!

test\_datagen = ImageDataGenerator(rescale = 1./255)

# Flow training images in batches of 20 using train\_datagen generator

train\_generator = train\_datagen.flow\_from\_directory(train\_dir,

target\_size=(img\_dimensions,

img\_dimensions),

batch\_size=20,

class\_mode='binary',

shuffle=False)

# Flow validation images in batches of 20 using test\_datagen generator

validation\_generator = test\_datagen.flow\_from\_directory(validation\_dir,

target\_size=(img\_dimensions,

img\_dimensions),

batch\_size=20,

class\_mode='binary')

## Img samples

import matplotlib.pyplot as plt

from keras.preprocessing.image import array\_to\_img

image\_batch = train\_generator[0][0]

plt.figure(figsize=(20,5))

for i in range(len(image\_batch)):

plt.subplot(3,8,i + 1)

pil\_img = array\_to\_img(image\_batch[i])

plt.imshow(pil\_img,cmap='gray')

plt.axis('off')

plt.tight\_layout()

plt.show()

train\_generator.reset()

Reduce overfitting

def scheduler(epoch,learning\_rate):

if epoch < 10:

return learning\_rate

else:

return learning\_rate \* tf.math.exp(-0.1)

callback = tf.keras.callbacks.LearningRateScheduler(scheduler)

history = model.fit(

train\_generator,

validation\_data=validation\_generator,

steps\_per\_epoch=40,

epochs=100,

validation\_steps=50,

verbose=1

)

## Accuracy of generated model

import matplotlib.pyplot as plt

acc = history.history['accuracy']

val\_acc = history.history['val\_accuracy']

loss = history.history['loss']

val\_loss = history.history['val\_loss']

epochs = range(len(acc))

plt.plot(epochs, acc, 'r', label='Training accuracy')

plt.plot(epochs, val\_acc, 'b', label='Validation accuracy')

plt.title('Training and validation accuracy')

plt.legend(loc=0)

plt.ylim([0, 1])

plt.figure()

plt.show()

## Tests

import random

test\_dir = os.path.join(base\_dir, 'test')

test\_normal\_dir = os.path.join(train\_dir, 'normal')

test\_pneumonia\_dir = os.path.join(train\_dir, 'opacity')

test\_normal\_fnames = os.listdir(test\_normal\_dir)

test\_opacity\_fnames = os.listdir(test\_pneumonia\_dir)

test\_gen = ImageDataGenerator(rescale = 1./255)

test\_generator = test\_gen.flow\_from\_directory( test\_dir,

class\_mode=None,

target\_size=(img\_dimensions, img\_dimensions),

shuffle=False)

import numpy as np

import pandas as pd

predictions\_md = model.predict(test\_generator, verbose=1)

predictions\_md[predictions\_md <= 0.5] = 0

predictions\_md[predictions\_md > 0.5] = 1

from sklearn.metrics import confusion\_matrix

cm = pd.DataFrame(data=confusion\_matrix(test\_generator.classes, predictions\_md, labels=[0, 1]),

index=["Actual Normal", "Actual Pneumonia"],

columns=["Predicted Normal", "Predicted Pneumonia"])

cm

Archivo de colab:

