


## ▼ Introduccion

A partir de los conocimientos vistos en la unidad de formación se tratará de analizar un dataset para poder llegar a concluir si un paciente está infectado o no de Malaria, todo esto implementando diversas técnicas en redes neuronales. Los datos que se pretenden resolver es la detección temprana de malaria en pacientes a través de un modelo de regresión y de clasificación aplicando una red neuronal que pueda detectarlo de la manera más correcta y precisa posible. Algunas de las características que se presentan en el dataset son imagenes de diversa tonalidad de color lo cual sería nuestro indicador de presencia o no de esta enfermedad, cuenta con aproximadamente 43 000 imagenes divididas entre training\_set y test\_set al igual que las dos posibles predicciones del modelo que sería una imagen con la presencia del parásito y otra sin presencia de este.

## ▼ Carga de datos

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

 Mounted at /content/drive

```
1 from google.colab import files
2 files.upload()
```

Ninguno archivo selec. Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.  
Saving kaggle.json to kaggle.json  
{'kaggle.json': h'{"username": "a01275404", "key": "2ee41129e58633b8ch0053e288a27ch0"}'}

```
1 !mkdir ~/.kaggle
2 !cp kaggle.json ~/.kaggle/
3 !chmod 600 ~/.kaggle/kaggle.json
```

```
1 !kaggle datasets download -d miracle9to9/files1
```

```
Downloading files1.zip to /content
100% 523M/525M [00:16<00:00, 42.8MB/s]
100% 525M/525M [00:16<00:00, 33.8MB/s]
```

```
1 !unzip -qq files1.zip
```

```
1 import cv2
2 from google.colab.patches import cv2_imshow
3 img= cv2.imread('/content/Malaria Cells/single_prediction/Parasitised.png')
4 img2= cv2.imread('/content/Malaria Cells/single_prediction/Uninfected.png')
5 print("Prueba de muestra infectada")
6 cv2_imshow(img)
7
```

Prueba de muestra infectada



```
1 print("Prueba de muestra no infectada")
2 cv2_imshow(img2)
```

Prueba de muestra no infectada



1



## ▼ Procesamiento de datos para trabajar con keras pasando de png a narray

```
1 import tensorflow as tf
2 from tensorflow import keras
3 from sklearn.datasets import load_sample_image
4 import matplotlib.pyplot as plt
5 import numpy as np
6 import pandas as pd
7 from PIL import Image
8 %matplotlib inline
```

```
1 from sklearn.preprocessing import LabelEncoder
```

Al realizar las cargas de las imágenes, colab requiere mucha Ram por lo que el entorno se desconectaba al usar los modelos. Es por ello que se decidió solo trabajar con una muestra de 1000 datos parasitados y no infectados. Esto también nos ayuda con el balanceo de clases

```
1 import os
2 ruta="/content/Malaria Cells/testing_set/Parasitized"
3 patest=[]
4 labelpatest=[]
5 img_size=80
6 contador=0
7 for img in os.listdir(ruta):
8     img=cv2.imread(os.path.join(ruta,img))
9     img=cv2.resize(img,(128,128),interpolation=cv2.INTER_AREA)
10    patest.append(img)
11    labelpatest.append(1)
12    contador+=1
13    if contador==2000:
14        break
15 patest=np.array(patest)
16 print(len(patest))
17
```

2000

```
1 ruta="/content/Malaria Cells/testing_set/Uninfected"
2 unfetest=[]
3 labelunfetest=[]
4 img_size=80
5 contador=0
6 for img in os.listdir(ruta):
7     img=cv2.imread(os.path.join(ruta,img))
8     img=cv2.resize(img,(128,128),interpolation=cv2.INTER_AREA)
9     unfetest.append(img)
10    labelunfetest.append(0)
11    contador+=1
12    if contador==2000:
13        break
14 unfetest=np.array(unfetest)
15 print(len(unfetest))
```

2000

## ► Resultado de nuestro proceso:

```
[ ] ↳ 1 celda oculta
```

## ▼ Union de los datos en test y train ya que al venir de carpetas separadas estan divididos

```

1 test= np.concatenate((patest,unfptest))
2 labels_test=np.concatenate((labelpatest, labelunfptest))
3 print(test.shape)
4 print(labels_test.shape)

(4000, 128, 128, 3)
(4000,)

```

▼ Dada la gran cantidad de datos trabajaremos con los datos de test y haremos la correspondiente division de prueba y entrenamiento a continuacion:

```

1

1 from sklearn.model_selection import train_test_split

1 imagenes_train, imagenes_test, labels_train, labels_testf=train_test_split(test,labels_test,test_size=.3)

1
2 len(imagenes_train)

2800

1
2 print(len(imagenes_test))
3 print(len(labels_testf))

1200
1200

1 from keras.models import Sequential
2 from keras.layers import Dense, Flatten, Conv2D, MaxPooling2D, Dropout, BatchNormalization
3 from tensorflow.keras.layers import Rescaling

```

▼ Ya que usaremos las mismas imagenes, aqui se hara la normalizacion de nuestras imagenes

```

1 train_images = imagenes_train/255.0
2 test_images = imagenes_test/255.0

1 train_images[0].shape

(128, 128, 3)

```

▼ Metodos

▼ Jesus Yair Ramirez Islas A01275404

Primer metodo

▼ Descripcion e implementacion

Para la arquitectura de este primer modelo usaremos una capa convolucional 2D con 8 filtros con tamaño de kernel 3x3, función de activación relu. Una capa de Max-Pooling2D con valor de 2x2 para ayudar a prevenir el overfitting, seguida de una capa de flatten para “aplanar nuestro datos”, después se usará dos capas densas de 16 y 64 neuronas ambas con función de activación relu. Finalmente una capa de salida con la función “Softmax” ya que buscamos resultado como probabilidades, como solo hay dos clases entonces esta capa solo tendrá 2 salidas. El modelo tendrá un optimizador de Adam con los valores por defecto, la función de pérdida será sparse categorical cross entropy, y usaremos el accuracy para evaluar el modelo. Usaremos un conjunto de validación correspondiente al 10%, un batch size de 150 para que el proceso sea rápido y 8 épocas.

```
1 keras.backend.clear_session()

1 model = Sequential([
2     Conv2D(32, kernel_size=3, padding='same', activation="relu", input_shape=train_images[0].shape),
3     MaxPooling2D(2),
4     Flatten(),
5     Dense(32, activation='relu'),
6     Dense(128, activation='relu'),
7     Dense(256, activation='relu'),
8     Dense(2, activation='softmax')
9 ])

1 model.summary()

Model: "sequential"
-----
Layer (type)                Output Shape              Param #
-----
conv2d (Conv2D)              (None, 128, 128, 32)     896
max_pooling2d (MaxPooling2D) (None, 64, 64, 32)       0
flatten (Flatten)            (None, 131072)           0
dense (Dense)                 (None, 32)               4194336
dense_1 (Dense)               (None, 128)              4224
dense_2 (Dense)               (None, 256)              33024
dense_3 (Dense)               (None, 2)                514
-----
Total params: 4,232,994
Trainable params: 4,232,994
Non-trainable params: 0

1 model.compile(loss='sparse_categorical_crossentropy', optimizer=tf.keras.optimizers.Adam(),
2               metrics=['accuracy'])

1 history = model.fit(train_images,labels_train, epochs=40, validation_split=0.1,batch_size=150)

Epoch 1/40
17/17 [=====] - 10s 88ms/step - loss: 0.8287 - accuracy: 0.5091 - val_loss: 0.7080 - val_accuracy: 0.4357
Epoch 2/40
17/17 [=====] - 1s 47ms/step - loss: 0.6249 - accuracy: 0.6599 - val_loss: 0.5492 - val_accuracy: 0.7214
Epoch 3/40
17/17 [=====] - 1s 47ms/step - loss: 0.5874 - accuracy: 0.6841 - val_loss: 0.5596 - val_accuracy: 0.7107
Epoch 4/40
17/17 [=====] - 1s 47ms/step - loss: 0.5575 - accuracy: 0.7123 - val_loss: 0.5611 - val_accuracy: 0.7143
Epoch 5/40
17/17 [=====] - 1s 48ms/step - loss: 0.4913 - accuracy: 0.7718 - val_loss: 0.5761 - val_accuracy: 0.7000
Epoch 6/40
17/17 [=====] - 1s 48ms/step - loss: 0.3831 - accuracy: 0.8397 - val_loss: 0.7386 - val_accuracy: 0.6429
Epoch 7/40
17/17 [=====] - 1s 48ms/step - loss: 0.2708 - accuracy: 0.8960 - val_loss: 0.7900 - val_accuracy: 0.7000
Epoch 8/40
17/17 [=====] - 1s 47ms/step - loss: 0.1708 - accuracy: 0.9409 - val_loss: 0.9218 - val_accuracy: 0.6786
Epoch 9/40
17/17 [=====] - 1s 47ms/step - loss: 0.0973 - accuracy: 0.9683 - val_loss: 1.1488 - val_accuracy: 0.6643
Epoch 10/40
17/17 [=====] - 1s 48ms/step - loss: 0.0563 - accuracy: 0.9845 - val_loss: 1.3971 - val_accuracy: 0.6321
Epoch 11/40
17/17 [=====] - 1s 48ms/step - loss: 0.0283 - accuracy: 0.9956 - val_loss: 1.4225 - val_accuracy: 0.6893
Epoch 12/40
17/17 [=====] - 1s 47ms/step - loss: 0.0125 - accuracy: 0.9980 - val_loss: 1.7236 - val_accuracy: 0.6357
Epoch 13/40
17/17 [=====] - 1s 47ms/step - loss: 0.0099 - accuracy: 0.9996 - val_loss: 1.6762 - val_accuracy: 0.7000
Epoch 14/40
17/17 [=====] - 1s 47ms/step - loss: 0.0070 - accuracy: 0.9996 - val_loss: 1.7971 - val_accuracy: 0.6643
Epoch 15/40
17/17 [=====] - 1s 47ms/step - loss: 0.0074 - accuracy: 0.9988 - val_loss: 1.9360 - val_accuracy: 0.6714
Epoch 16/40
17/17 [=====] - 1s 47ms/step - loss: 0.0037 - accuracy: 1.0000 - val_loss: 1.9900 - val_accuracy: 0.6893
Epoch 17/40
17/17 [=====] - 1s 47ms/step - loss: 0.0018 - accuracy: 1.0000 - val_loss: 2.1409 - val_accuracy: 0.6786
```

```

Epoch 18/40
17/17 [=====] - 1s 48ms/step - loss: 0.0011 - accuracy: 1.0000 - val_loss: 2.2136 - val_accuracy: 0.6750
Epoch 19/40
17/17 [=====] - 1s 47ms/step - loss: 7.8416e-04 - accuracy: 1.0000 - val_loss: 2.2884 - val_accuracy: 0.6820
Epoch 20/40
17/17 [=====] - 1s 47ms/step - loss: 5.8754e-04 - accuracy: 1.0000 - val_loss: 2.3725 - val_accuracy: 0.6780
Epoch 21/40
17/17 [=====] - 1s 48ms/step - loss: 4.7102e-04 - accuracy: 1.0000 - val_loss: 2.4312 - val_accuracy: 0.6780
Epoch 22/40
17/17 [=====] - 1s 47ms/step - loss: 3.9387e-04 - accuracy: 1.0000 - val_loss: 2.4585 - val_accuracy: 0.6850
Epoch 23/40
17/17 [=====] - 1s 47ms/step - loss: 3.3955e-04 - accuracy: 1.0000 - val_loss: 2.5032 - val_accuracy: 0.6820
Epoch 24/40
17/17 [=====] - 1s 48ms/step - loss: 2.9551e-04 - accuracy: 1.0000 - val_loss: 2.5385 - val_accuracy: 0.6820
Epoch 25/40
17/17 [=====] - 1s 47ms/step - loss: 2.6159e-04 - accuracy: 1.0000 - val_loss: 2.5717 - val_accuracy: 0.6850
Epoch 26/40
17/17 [=====] - 1s 47ms/step - loss: 2.3444e-04 - accuracy: 1.0000 - val_loss: 2.5940 - val_accuracy: 0.6890
Epoch 27/40
17/17 [=====] - 1s 47ms/step - loss: 2.0745e-04 - accuracy: 1.0000 - val_loss: 2.6354 - val_accuracy: 0.6820
Epoch 28/40
17/17 [=====] - 1s 48ms/step - loss: 1.8478e-04 - accuracy: 1.0000 - val_loss: 2.6814 - val_accuracy: 0.6850
Epoch 29/40

```

## ▼ Resultados

```

1 # Run this cell to load the model history into a pandas DataFrame
2
3 frame = pd.DataFrame(history.history)

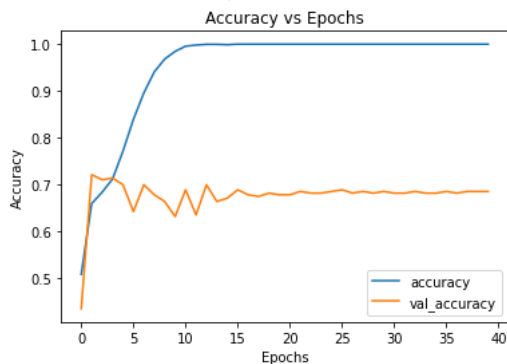
```

```

1 # Run this cell to make the Accuracy vs Epochs plot
2
3 acc_plot = frame.plot(y=["accuracy", "val_accuracy"], title="Accuracy vs Epochs", legend=True)
4 acc_plot.set(xlabel="Epochs", ylabel="Accuracy")

```

```
[Text(0, 0.5, 'Accuracy'), Text(0.5, 0, 'Epochs')]
```

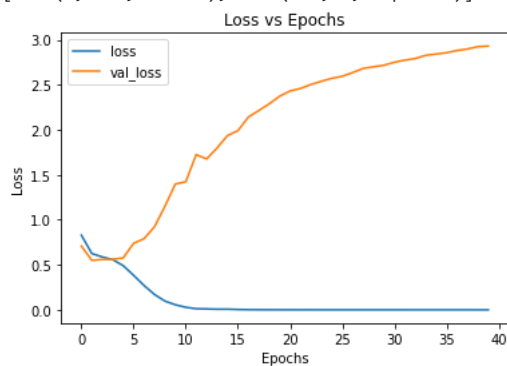


```

1 # Run this cell to make the Loss vs Epochs plot
2
3 acc_plot = frame.plot(y=["loss", "val_loss"], title = "Loss vs Epochs", legend=True)
4 acc_plot.set(xlabel="Epochs", ylabel="Loss")

```

```
[Text(0, 0.5, 'Loss'), Text(0.5, 0, 'Epochs')]
```



```

1 loss, accuracy = model.evaluate(test_images, labels_testf)
2 print(f"Test loss: {loss}")
3 print(f"Test accuracy: {accuracy}")

38/38 [=====] - 0s 8ms/step - loss: 2.6796 - accuracy: 0.6650
Test loss: 2.679617166519165
Test accuracy: 0.665000214576721

```

```

1 # Define the labels for easier interpretation
2 labels = [
3     'Uninfected',
4     'Parasited',
5 ]

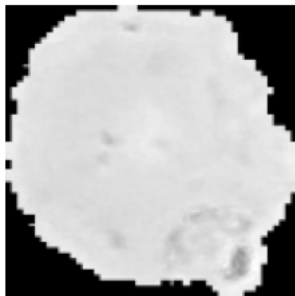
```

```

1 random_inx = np.random.choice(test_images.shape[0])
2 print(random_inx)
3 test_image = test_images[random_inx]
4
5 plt.axis('off')
6
7 plt.imshow(test_image[:, :, 0], cmap='gray')
8 plt.show()
9 print(f"Label: {labels[labels_testf[random_inx]]}")

```

55



Label: Parasited

```

1 pred = model.predict(test_image[np.newaxis,...])
2 print("Model: ", labels[np.argmax(pred)])

```

```

1/1 [=====] - 0s 97ms/step
Model: Uninfected

```

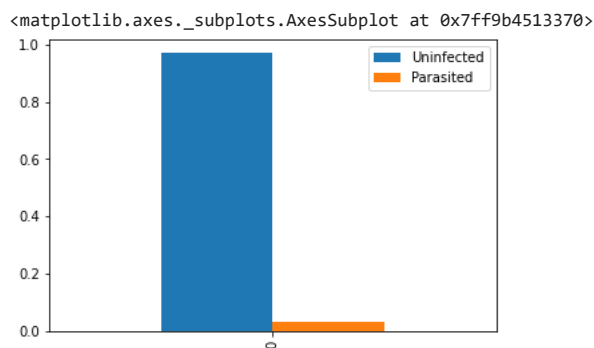
```

1 df = pd.DataFrame(pred, columns=labels)
2 df

```

	Uninfected	Parasited
0	0.969768	0.030233

```
1 df.plot.bar()
```



## ▼ Segundo método

▼ Descripción e implementación

Observando los resultados del anterior modelo podemos darnos cuenta que existe mucho overfitting por lo tanto, hay que implementar medidas para disminuirlo. Para ello se agregó una segunda capa convolucional con 64 neuronas después de la ya existente. Según cierta documentación, para disminuir el overfitting se puede optar por un modelo menos complejo, por lo que se quitó una y se modificaron el resto de las capas densas. El primer cambio fue de pasar de 4 capas incluyendo la de salida, a únicamente 3. Se aumentó el número de neuronas a 128 y se utilizó la misma función de activación, adicionalmente se añadió una capa de Dropout con rate de .1 muy útil para reducir el overfitting ya que "apaga" ciertas neuronas en algunas fases del entrenamiento y así sus coeficientes no cambian. Se modificó la penúltima capa reduciendo las neuronas a 32 y finalmente se agregó otra capa de dropout pero con rate de .2. Para el modelo se añadió el callback de early stopping para detener el entrenamiento si se observa un aumento del error de validación con un parámetro de paciencia de 10. Así mismo se aumentó el batch size a 250 pues como recordaremos un número más grande en este parámetro significa que el modelo tomará más tiempo pero hará un mejor aprendizaje

```
1 keras.backend.clear_session()

1 model = Sequential([
2     Conv2D(32, kernel_size=3, padding='same', activation="relu", input_shape=train_images[0].shape),
3     MaxPooling2D(2),
4     Conv2D(64, kernel_size=3, padding='same', activation="relu"),
5     MaxPooling2D(2),
6     Flatten(),
7     Dense(128, activation='relu'),
8     Dropout(rate=0.1),
9     Dense(32, activation='relu'),
10    Dropout(rate=0.2),
11    Dense(2, activation='softmax')
12    ])

1 model.summary()

Model: "sequential"
-----
Layer (type)                Output Shape              Param #
-----
conv2d (Conv2D)              (None, 128, 128, 32)     896

max_pooling2d (MaxPooling2D) (None, 64, 64, 32)       0

conv2d_1 (Conv2D)            (None, 64, 64, 64)       18496

max_pooling2d_1 (MaxPooling2D) (None, 32, 32, 64)       0

flatten (Flatten)            (None, 65536)             0

dense (Dense)                 (None, 128)               8388736

dropout (Dropout)             (None, 128)               0

dense_1 (Dense)               (None, 32)                4128

dropout_1 (Dropout)           (None, 32)                0

dense_2 (Dense)               (None, 2)                 66

Total params: 8,412,322
Trainable params: 8,412,322
Non-trainable params: 0
-----

1 model.compile(loss='sparse_categorical_crossentropy', optimizer=tf.keras.optimizers.Adam(),
2               metrics=['accuracy'])

1 early_stopping_cb = keras.callbacks.EarlyStopping(patience=10, restore_best_weights=True)
2 history = model.fit(train_images,labels_train, epochs=40, validation_split=0.1,batch_size=250, callbacks=[early_stopping_cb])

Epoch 1/40
11/11 [=====] - 2s 133ms/step - loss: 1.1016 - accuracy: 0.5111 - val_loss: 0.6936 - val_accuracy: 0.4679
Epoch 2/40
```

```

11/11 [=====] - 1s 109ms/step - loss: 0.6910 - accuracy: 0.5242 - val_loss: 0.6887 - val_accuracy: 0.5643
Epoch 3/40
11/11 [=====] - 1s 111ms/step - loss: 0.6888 - accuracy: 0.5599 - val_loss: 0.6842 - val_accuracy: 0.5643
Epoch 4/40
11/11 [=====] - 1s 111ms/step - loss: 0.6681 - accuracy: 0.6091 - val_loss: 0.6096 - val_accuracy: 0.6286
Epoch 5/40
11/11 [=====] - 1s 109ms/step - loss: 0.6270 - accuracy: 0.6310 - val_loss: 0.6488 - val_accuracy: 0.5964
Epoch 6/40
11/11 [=====] - 1s 112ms/step - loss: 0.6186 - accuracy: 0.6397 - val_loss: 0.5965 - val_accuracy: 0.6464
Epoch 7/40
11/11 [=====] - 1s 109ms/step - loss: 0.5921 - accuracy: 0.6877 - val_loss: 0.5934 - val_accuracy: 0.6607
Epoch 8/40
11/11 [=====] - 1s 117ms/step - loss: 0.5596 - accuracy: 0.7230 - val_loss: 0.5532 - val_accuracy: 0.6893
Epoch 9/40
11/11 [=====] - 1s 110ms/step - loss: 0.5333 - accuracy: 0.7484 - val_loss: 0.5195 - val_accuracy: 0.7214
Epoch 10/40
11/11 [=====] - 1s 110ms/step - loss: 0.5021 - accuracy: 0.7706 - val_loss: 0.5164 - val_accuracy: 0.7107
Epoch 11/40
11/11 [=====] - 1s 109ms/step - loss: 0.4497 - accuracy: 0.8048 - val_loss: 0.5682 - val_accuracy: 0.7143
Epoch 12/40
11/11 [=====] - 1s 110ms/step - loss: 0.4087 - accuracy: 0.8266 - val_loss: 0.4893 - val_accuracy: 0.7643
Epoch 13/40
11/11 [=====] - 1s 107ms/step - loss: 0.3232 - accuracy: 0.8782 - val_loss: 0.5497 - val_accuracy: 0.7429
Epoch 14/40
11/11 [=====] - 1s 109ms/step - loss: 0.2699 - accuracy: 0.8940 - val_loss: 0.4375 - val_accuracy: 0.8000
Epoch 15/40
11/11 [=====] - 1s 108ms/step - loss: 0.2096 - accuracy: 0.9238 - val_loss: 0.4752 - val_accuracy: 0.7893
Epoch 16/40
11/11 [=====] - 1s 109ms/step - loss: 0.1401 - accuracy: 0.9528 - val_loss: 0.4463 - val_accuracy: 0.8357
Epoch 17/40
11/11 [=====] - 1s 109ms/step - loss: 0.1117 - accuracy: 0.9615 - val_loss: 0.4308 - val_accuracy: 0.8357
Epoch 18/40
11/11 [=====] - 1s 107ms/step - loss: 0.0979 - accuracy: 0.9663 - val_loss: 0.5064 - val_accuracy: 0.8429
Epoch 19/40
11/11 [=====] - 1s 107ms/step - loss: 0.0686 - accuracy: 0.9790 - val_loss: 0.5424 - val_accuracy: 0.8393
Epoch 20/40
11/11 [=====] - 1s 107ms/step - loss: 0.0574 - accuracy: 0.9821 - val_loss: 0.6040 - val_accuracy: 0.8464
Epoch 21/40
11/11 [=====] - 1s 108ms/step - loss: 0.0481 - accuracy: 0.9881 - val_loss: 0.5187 - val_accuracy: 0.8536
Epoch 22/40
11/11 [=====] - 1s 107ms/step - loss: 0.0321 - accuracy: 0.9937 - val_loss: 0.6284 - val_accuracy: 0.8179
Epoch 23/40
11/11 [=====] - 1s 109ms/step - loss: 0.0349 - accuracy: 0.9889 - val_loss: 0.6879 - val_accuracy: 0.8214
Epoch 24/40
11/11 [=====] - 1s 109ms/step - loss: 0.0234 - accuracy: 0.9944 - val_loss: 0.7018 - val_accuracy: 0.8429
Epoch 25/40
11/11 [=====] - 1s 107ms/step - loss: 0.0174 - accuracy: 0.9960 - val_loss: 0.6683 - val_accuracy: 0.8357
Epoch 26/40
11/11 [=====] - 1s 107ms/step - loss: 0.0095 - accuracy: 0.9984 - val_loss: 0.7119 - val_accuracy: 0.8393
Epoch 27/40
11/11 [=====] - 1s 110ms/step - loss: 0.0110 - accuracy: 0.9980 - val_loss: 0.7296 - val_accuracy: 0.8393

```

## ▼ Resultados

```

1 # Run this cell to load the model history into a pandas DataFrame
2
3 frame2 = pd.DataFrame(history.history)

1 # Run this cell to make the Accuracy vs Epochs plot
2
3 acc_plot = frame2.plot(y=["accuracy", "val_accuracy"], title="Accuracy vs Epochs", legend=True)
4 acc_plot.set(xlabel="Epochs", ylabel="Accuracy")

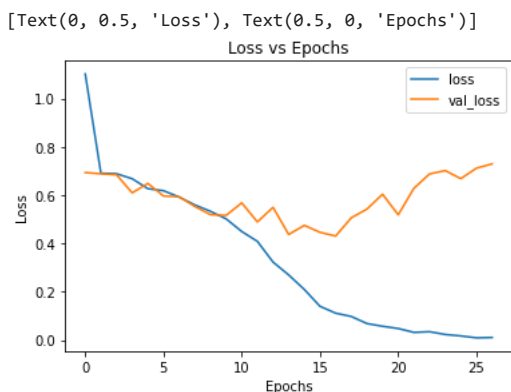
```



```

[Text(0, 0.5, 'Accuracy'), Text(0.5, 0, 'Epochs')]
1 # Run this cell to make the Loss vs Epochs plot
2
3 acc_plot = frame2.plot(y=["loss", "val_loss"], title = "Loss vs Epochs", legend=True)
4 acc_plot.set(xlabel="Epochs", ylabel="Loss")

```



```

1 loss2, accuracy2 = model.evaluate(test_images, labels_testf)
2 print(f"Test loss: {loss2}")
3 print(f"Test accuracy: {accuracy2}")

38/38 [=====] - 0s 10ms/step - loss: 0.4620 - accuracy: 0.8383
Test loss: 0.461984246969223
Test accuracy: 0.8383333086967468

```

```

1 # Define the labels for easier interpretation
2 labels = [
3     'Uninfected',
4     'Parasited',
5 ]

```

```

1 random_inx = np.random.choice(test_images.shape[0])
2 test_image = test_images[random_inx]
3
4 plt.axis('off')
5
6 plt.imshow(test_image[:, :, 0], cmap='gray')
7 plt.show()
8 print(f"Label: {labels[labels_testf[random_inx]]}")

```



Label: Uninfected

```

1 pred = model.predict(test_image[np.newaxis,...])
2 print("Model: ", labels[np.argmax(pred)])

```

```

WARNING:tensorflow:5 out of the last 5 calls to <function Model.make_predict_function.<locals>.predict_function at 0x7ff9b4ae8ca0> trig
1/1 [=====] - 0s 63ms/step
Model: Uninfected

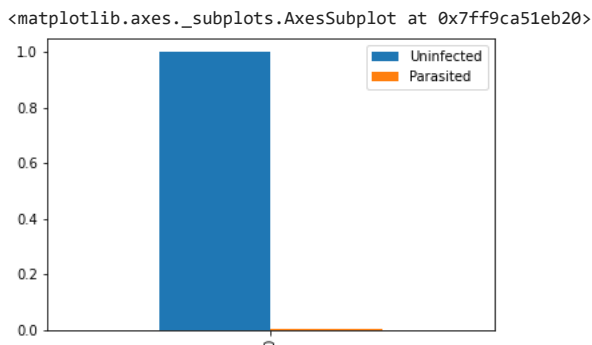
```

```

1 df = pd.DataFrame(pred, columns=labels)
2 df

```

```
1 df.plot.bar()
```



Observamos que mejoró el resultado enormemente al evaluar al modelo al llegar al 82%. Si bien en las épocas en comparación el accuracy en los datos de validación no presenta una enorme diferencia entre este modelo y el anterior, si comparamos en conjunto esta métrica y el accuracy del modelo observamos que en el segundo, el accuracy se aleja del 100% mientras que el accuracy en la validacion aumento, lo cual nos habla de que en efecto el overfitting ha disminuido.

Luis Gerardo Lagunes Najera A01275215

## ▼ Método 1

Para el siguiente método implementaremos una red neuronal usaremos una capa densa de 128 neuronas usando la función de activación "elu" ya que esta nos va permitir acercar las activaciones de unidades medias a cero,, una vez aplicada esta capa aplanaremos los datos con Flatten() y aplicaremos una función de BatchNormalization para normalizar los datos, una vez hecho esto agregaremos 5 capas densas con 128,64,32,16,8 neuronas y la función de activación elu y a las primeras dos capas se le agregará la función Dropout a las dos primeras ya que son las que mayor cantidad de neuronas tienen, al final quedaría una capa de 2 neuronas con la función de activación "softmax". Para compilar el modelo utilizaremos una pérdida "sparse\_categorical\_crossentropy", con un optimizador SGD con un lr = 0.05, momentum de 0.5, nesterov = true y la métrica que sería en este caso la accuracy, con un total de épocas de 10 para no hacer tan tardado nuestro modelo y un split de validación de 0.1 para los datos.

```
1 keras.backend.clear_session()
```

```
1 model = Sequential([
2     Conv2D(128, kernel_size=3, padding='same', activation="elu", input_shape=train_images[0].shape),
3     MaxPooling2D(2),
4     Flatten(),
5     BatchNormalization(),
6     Dense(128, activation='elu'),
7     Dropout(rate=0.1),
8     Dense(64, activation='elu'),
9     Dropout(rate=0.1),
10    Dense(32, activation='elu'),
11    Dense(16, activation='elu'),
12    Dense(8, activation='elu'),
13    BatchNormalization(),
14    Dense(2, activation='softmax')
15    ])
```

```
1 model.summary()
```

Model: "sequential"

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 128, 128, 128)	3584
max_pooling2d (MaxPooling2D)	(None, 64, 64, 128)	0

flatten (Flatten)	(None, 524288)	0
batch_normalization (Batch Normalization)	(None, 524288)	2097152
dense (Dense)	(None, 128)	67108992
dropout (Dropout)	(None, 128)	0
dense_1 (Dense)	(None, 64)	8256
dropout_1 (Dropout)	(None, 64)	0
dense_2 (Dense)	(None, 32)	2080
dense_3 (Dense)	(None, 16)	528
dense_4 (Dense)	(None, 8)	136
batch_normalization_1 (Batch Normalization)	(None, 8)	32
dense_5 (Dense)	(None, 2)	18

```

=====
Total params: 69,220,778
Trainable params: 68,172,186
Non-trainable params: 1,048,592

```

```

1 keras.optimizers.SGD(lr=0.001, momentum=0.9, nesterov=True)
2 model.compile(loss='sparse_categorical_crossentropy', optimizer='sgd',
3               metrics=['accuracy'])

```

```

/usr/local/lib/python3.8/dist-packages/keras/optimizers/optimizer_v2/gradient_descent.py:108: UserWarning: The `lr` argument is deprecated. Use `learning_rate` instead.
super(SGD, self).__init__(name, **kwargs)

```

```
1 history = model.fit(train_images, labels_train, epochs=30, validation_split=0.1)
```

```

Epoch 1/30
79/79 [=====] - 6s 61ms/step - loss: 0.8096 - accuracy: 0.6357 - val_loss: 0.6944 - val_accuracy: 0.5714
Epoch 2/30
79/79 [=====] - 4s 53ms/step - loss: 0.6126 - accuracy: 0.6710 - val_loss: 0.6737 - val_accuracy: 0.5929
Epoch 3/30
79/79 [=====] - 4s 55ms/step - loss: 0.5868 - accuracy: 0.6849 - val_loss: 0.6304 - val_accuracy: 0.6536
Epoch 4/30
79/79 [=====] - 4s 54ms/step - loss: 0.5747 - accuracy: 0.6905 - val_loss: 0.6205 - val_accuracy: 0.6607
Epoch 5/30
79/79 [=====] - 4s 53ms/step - loss: 0.5632 - accuracy: 0.7012 - val_loss: 0.6006 - val_accuracy: 0.6607
Epoch 6/30
79/79 [=====] - 4s 54ms/step - loss: 0.5408 - accuracy: 0.7258 - val_loss: 0.5904 - val_accuracy: 0.6679
Epoch 7/30
79/79 [=====] - 4s 53ms/step - loss: 0.5197 - accuracy: 0.7357 - val_loss: 0.6068 - val_accuracy: 0.6321
Epoch 8/30
79/79 [=====] - 4s 57ms/step - loss: 0.5007 - accuracy: 0.7480 - val_loss: 0.5944 - val_accuracy: 0.6250
Epoch 9/30
79/79 [=====] - 5s 63ms/step - loss: 0.4628 - accuracy: 0.7782 - val_loss: 0.5894 - val_accuracy: 0.6214
Epoch 10/30
79/79 [=====] - 5s 66ms/step - loss: 0.4346 - accuracy: 0.7952 - val_loss: 0.6093 - val_accuracy: 0.5929
Epoch 11/30
79/79 [=====] - 5s 64ms/step - loss: 0.3899 - accuracy: 0.8194 - val_loss: 0.6309 - val_accuracy: 0.6179
Epoch 12/30
79/79 [=====] - 4s 55ms/step - loss: 0.3347 - accuracy: 0.8563 - val_loss: 0.8133 - val_accuracy: 0.4393
Epoch 13/30
79/79 [=====] - 4s 52ms/step - loss: 0.2746 - accuracy: 0.8853 - val_loss: 0.5960 - val_accuracy: 0.6964
Epoch 14/30
79/79 [=====] - 4s 52ms/step - loss: 0.2233 - accuracy: 0.9155 - val_loss: 1.1666 - val_accuracy: 0.4393
Epoch 15/30
79/79 [=====] - 4s 52ms/step - loss: 0.2116 - accuracy: 0.9234 - val_loss: 0.8281 - val_accuracy: 0.6857
Epoch 16/30
79/79 [=====] - 4s 52ms/step - loss: 0.1554 - accuracy: 0.9512 - val_loss: 0.9074 - val_accuracy: 0.7036
Epoch 17/30
79/79 [=====] - 4s 53ms/step - loss: 0.1200 - accuracy: 0.9611 - val_loss: 0.8631 - val_accuracy: 0.7107
Epoch 18/30
79/79 [=====] - 4s 52ms/step - loss: 0.0974 - accuracy: 0.9754 - val_loss: 0.9114 - val_accuracy: 0.7214
Epoch 19/30
79/79 [=====] - 4s 52ms/step - loss: 0.0787 - accuracy: 0.9798 - val_loss: 2.2075 - val_accuracy: 0.5036
Epoch 20/30
79/79 [=====] - 4s 52ms/step - loss: 0.0663 - accuracy: 0.9833 - val_loss: 2.4288 - val_accuracy: 0.4786
Epoch 21/30
79/79 [=====] - 4s 52ms/step - loss: 0.0712 - accuracy: 0.9798 - val_loss: 1.1799 - val_accuracy: 0.6679

```

```

Epoch 22/30
79/79 [=====] - 4s 52ms/step - loss: 0.0651 - accuracy: 0.9837 - val_loss: 1.1680 - val_accuracy: 0.7071
Epoch 23/30
79/79 [=====] - 4s 52ms/step - loss: 0.0426 - accuracy: 0.9901 - val_loss: 0.9231 - val_accuracy: 0.7036
Epoch 24/30
79/79 [=====] - 4s 52ms/step - loss: 0.0410 - accuracy: 0.9897 - val_loss: 1.6515 - val_accuracy: 0.6429
Epoch 25/30
79/79 [=====] - 4s 52ms/step - loss: 0.0554 - accuracy: 0.9817 - val_loss: 1.3108 - val_accuracy: 0.6429
Epoch 26/30
79/79 [=====] - 4s 52ms/step - loss: 0.0371 - accuracy: 0.9909 - val_loss: 0.9931 - val_accuracy: 0.7536
Epoch 27/30
79/79 [=====] - 4s 54ms/step - loss: 0.0374 - accuracy: 0.9921 - val_loss: 1.1492 - val_accuracy: 0.7143
Epoch 28/30
79/79 [=====] - 4s 54ms/step - loss: 0.0318 - accuracy: 0.9909 - val_loss: 2.3146 - val_accuracy: 0.5107
Epoch 29/30
79/79 [=====] - 4s 52ms/step - loss: 0.0206 - accuracy: 0.9976 - val_loss: 1.2360 - val_accuracy: 0.6893

```

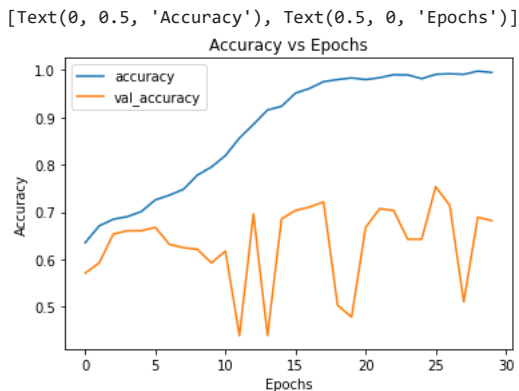
## ▼ Resultados

```

1 # Run this cell to load the model history into a pandas DataFrame
2
3 frame3 = pd.DataFrame(history.history)

1 # Run this cell to make the Accuracy vs Epochs plot
2
3 acc_plot = frame3.plot(y=["accuracy", "val_accuracy"], title="Accuracy vs Epochs", legend=True)
4 acc_plot.set(xlabel="Epochs", ylabel="Accuracy")

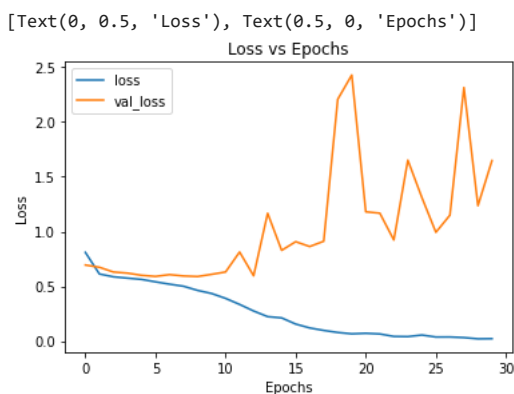
```



```

1 # Run this cell to make the Loss vs Epochs plot
2
3 acc_plot = frame3.plot(y=["loss", "val_loss"], title = "Loss vs Epochs", legend=True)
4 acc_plot.set(xlabel="Epochs", ylabel="Loss")

```



```

1 loss3, accuracy3 = model.evaluate(test_images, labels_testf)
2 print(f"Test loss: {loss3}")
3 print(f"Test accuracy: {accuracy3}")

38/38 [=====] - 1s 18ms/step - loss: 2.3059 - accuracy: 0.5792
Test loss: 2.3059241771698
Test accuracy: 0.5791666507720947

```

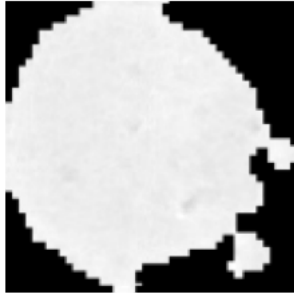
```

1 # Define the labels for easier interpretation
2 labels = [
3     'Uninfected',
4     'Parasited',
5 ]

1 random_inx = np.random.choice(test_images.shape[0])
2 print(random_inx)
3 test_image = test_images[random_inx]
4
5 plt.axis('off')
6
7 plt.imshow(test_image[:, :, 0], cmap='gray')
8 plt.show()
9 print(f"Label: {labels[labels_testf[random_inx]]}")

```

606



Label: Uninfected

```

1 pred = model.predict(test_image[np.newaxis, ...])
2 print("Model: ", labels[np.argmax(pred)])

1/1 [=====] - 0s 123ms/step
Model: Uninfected

```

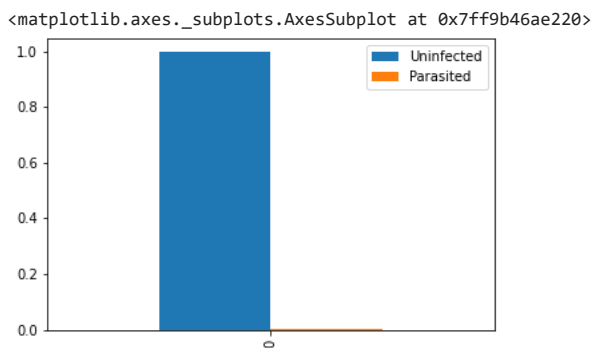
```

1 df = pd.DataFrame(pred, columns=labels)
2 df

```

	Uninfected	Parasited
0	0.99567	0.00433

```
1 df.plot.bar()
```



## ▼ Método 2

Para esta red se usará cuatro capas convolucional 2D con 64 filtros con el tamaño de kernel de 3x3 y la función de activación "relu", al igual que un Max Pooling de 2x2 y una capa de Flatten para el aplanamiento de los datos, después añadiremos 6 capas cada una con una cantidad respectiva de neuronas, todas con la función de activación de relu y la ultima de ella con softmax, las capas quedarían con 32,64,32,16,8,2 neuronas cada una, después de ello en la segunda y tercera se aplicará una capa de Dropout y en la penultima una de Batch Normalization.

```
1 keras.backend.clear_session()
```

```
1 model = Sequential([
2     Conv2D(64, kernel_size=3, padding='same', activation="relu", input_shape=train_images[0].shape),
3     Conv2D(64, kernel_size=3, padding='same', activation="relu"),
4     Conv2D(64, kernel_size=3, padding='same', activation="relu"),
5     Conv2D(64, kernel_size=3, padding='same', activation="relu"),
6     MaxPooling2D(2),
7     Flatten(),
8     Dense(32, activation='relu'),
9     Dense(64, activation='relu'),
10    Dropout(rate=0.1),
11    Dense(32, activation='relu'),
12    Dropout(rate=0.1),
13    Dense(16, activation='relu'),
14    Dense(8, activation='relu'),
15    BatchNormalization(),
16    Dense(2, activation='softmax'),
17    ])
```

```
1 model.summary()
```

Model: "sequential"

Layer (type)	Output Shape	Param #
=====		
conv2d (Conv2D)	(None, 128, 128, 64)	1792
conv2d_1 (Conv2D)	(None, 128, 128, 64)	36928
conv2d_2 (Conv2D)	(None, 128, 128, 64)	36928
conv2d_3 (Conv2D)	(None, 128, 128, 64)	36928
max_pooling2d (MaxPooling2D)	(None, 64, 64, 64)	0
flatten (Flatten)	(None, 262144)	0
dense (Dense)	(None, 32)	8388640
dense_1 (Dense)	(None, 64)	2112
dropout (Dropout)	(None, 64)	0
dense_2 (Dense)	(None, 32)	2080
dropout_1 (Dropout)	(None, 32)	0
dense_3 (Dense)	(None, 16)	528
dense_4 (Dense)	(None, 8)	136
batch_normalization (Batch Normalization)	(None, 8)	32
dense_5 (Dense)	(None, 2)	18
=====		
Total params: 8,506,122		
Trainable params: 8,506,106		
Non-trainable params: 16		

```
1 model.compile(loss='sparse_categorical_crossentropy', optimizer=tf.keras.optimizers.RMSprop(lr=0.001, rho=0.9),
2               metrics=['accuracy'])

/usr/local/lib/python3.8/dist-packages/keras/optimizers/optimizer_v2/rmsprop.py:135: UserWarning: The `lr` argument is deprecated, use
super(RMSprop, self).__init__(name, **kwargs)
```

```
1 early_stopping_cb = keras.callbacks.EarlyStopping(patience=10, restore_best_weights=True)
2 history = model.fit(train_images, labels_train, epochs=50, validation_split=0.1, batch_size=250, callbacks=[early_stopping_cb])

Epoch 1/50
11/11 [=====] - 11s 884ms/step - loss: 0.7576 - accuracy: 0.5452 - val_loss: 4.5569 - val_accuracy: 0.4357
Epoch 2/50
11/11 [=====] - 9s 850ms/step - loss: 0.6440 - accuracy: 0.6282 - val_loss: 0.6596 - val_accuracy: 0.6071
Epoch 3/50
11/11 [=====] - 9s 856ms/step - loss: 0.6256 - accuracy: 0.6794 - val_loss: 0.6830 - val_accuracy: 0.5643
```

```

Epoch 4/50
11/11 [=====] - 10s 862ms/step - loss: 0.6216 - accuracy: 0.6599 - val_loss: 1.0208 - val_accuracy: 0.4357
Epoch 5/50
11/11 [=====] - 10s 865ms/step - loss: 0.6085 - accuracy: 0.7095 - val_loss: 0.5651 - val_accuracy: 0.8250
Epoch 6/50
11/11 [=====] - 10s 871ms/step - loss: 0.5460 - accuracy: 0.7520 - val_loss: 0.5578 - val_accuracy: 0.7821
Epoch 7/50
11/11 [=====] - 10s 880ms/step - loss: 0.5071 - accuracy: 0.7913 - val_loss: 0.5362 - val_accuracy: 0.7393
Epoch 8/50
11/11 [=====] - 10s 889ms/step - loss: 0.4600 - accuracy: 0.8190 - val_loss: 0.4102 - val_accuracy: 0.8571
Epoch 9/50
11/11 [=====] - 10s 881ms/step - loss: 0.3813 - accuracy: 0.8599 - val_loss: 0.4265 - val_accuracy: 0.7679
Epoch 10/50
11/11 [=====] - 10s 876ms/step - loss: 0.3479 - accuracy: 0.8885 - val_loss: 0.2900 - val_accuracy: 0.8857
Epoch 11/50
11/11 [=====] - 10s 871ms/step - loss: 0.2910 - accuracy: 0.9143 - val_loss: 0.3169 - val_accuracy: 0.8964
Epoch 12/50
11/11 [=====] - 10s 875ms/step - loss: 0.3553 - accuracy: 0.8635 - val_loss: 0.2394 - val_accuracy: 0.9536
Epoch 13/50
11/11 [=====] - 10s 877ms/step - loss: 0.2583 - accuracy: 0.9234 - val_loss: 0.2906 - val_accuracy: 0.9143
Epoch 14/50
11/11 [=====] - 10s 884ms/step - loss: 0.2473 - accuracy: 0.9286 - val_loss: 0.2169 - val_accuracy: 0.9357
Epoch 15/50
11/11 [=====] - 10s 888ms/step - loss: 0.2137 - accuracy: 0.9385 - val_loss: 0.1993 - val_accuracy: 0.9607
Epoch 16/50
11/11 [=====] - 10s 886ms/step - loss: 0.1898 - accuracy: 0.9425 - val_loss: 0.2508 - val_accuracy: 0.9286
Epoch 17/50
11/11 [=====] - 10s 890ms/step - loss: 0.1707 - accuracy: 0.9536 - val_loss: 0.2607 - val_accuracy: 0.8929
Epoch 18/50
11/11 [=====] - 10s 889ms/step - loss: 0.1627 - accuracy: 0.9500 - val_loss: 0.4329 - val_accuracy: 0.8357
Epoch 19/50
11/11 [=====] - 10s 891ms/step - loss: 0.1531 - accuracy: 0.9552 - val_loss: 0.1752 - val_accuracy: 0.9536
Epoch 20/50
11/11 [=====] - 10s 879ms/step - loss: 0.1323 - accuracy: 0.9675 - val_loss: 0.1853 - val_accuracy: 0.9357
Epoch 21/50
11/11 [=====] - 10s 881ms/step - loss: 0.0908 - accuracy: 0.9813 - val_loss: 0.2441 - val_accuracy: 0.9393
Epoch 22/50
11/11 [=====] - 10s 884ms/step - loss: 0.0979 - accuracy: 0.9750 - val_loss: 0.5216 - val_accuracy: 0.7643
Epoch 23/50
11/11 [=====] - 10s 907ms/step - loss: 0.0937 - accuracy: 0.9766 - val_loss: 0.3093 - val_accuracy: 0.8679
Epoch 24/50
11/11 [=====] - 10s 913ms/step - loss: 0.0527 - accuracy: 0.9921 - val_loss: 0.2266 - val_accuracy: 0.9214
Epoch 25/50
11/11 [=====] - 10s 905ms/step - loss: 0.0699 - accuracy: 0.9833 - val_loss: 1.0920 - val_accuracy: 0.7107
Epoch 26/50
11/11 [=====] - 10s 898ms/step - loss: 0.0718 - accuracy: 0.9829 - val_loss: 0.1974 - val_accuracy: 0.9321
Epoch 27/50
11/11 [=====] - 10s 893ms/step - loss: 0.0263 - accuracy: 0.9980 - val_loss: 0.2231 - val_accuracy: 0.9321
Epoch 28/50
11/11 [=====] - 10s 889ms/step - loss: 0.0799 - accuracy: 0.9742 - val_loss: 0.2090 - val_accuracy: 0.9321
Epoch 29/50
11/11 [=====] - 10s 898ms/step - loss: 0.0204 - accuracy: 0.9984 - val_loss: 0.2302 - val_accuracy: 0.9250

```

## ▼ Resultados

```

1 # Run this cell to load the model history into a pandas DataFrame
2
3 frame4 = pd.DataFrame(history.history)

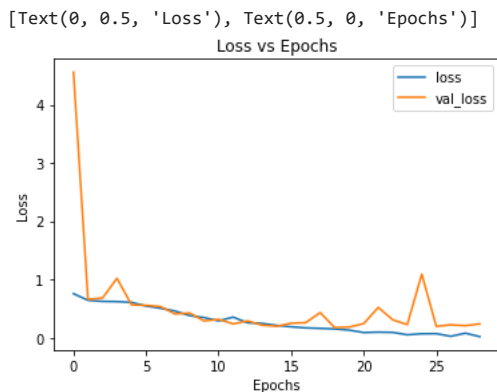
1 # Run this cell to make the Accuracy vs Epochs plot
2
3 acc_plot = frame4.plot(y=["accuracy", "val_accuracy"], title="Accuracy vs Epochs", legend=True)
4 acc_plot.set(xlabel="Epochs", ylabel="Accuracy")

```

```

[Text(0, 0.5, 'Accuracy'), Text(0.5, 0, 'Epochs')]
1 # Run this cell to make the Loss vs Epochs plot
2
3 acc_plot = frame4.plot(y=["loss", "val_loss"], title = "Loss vs Epochs", legend=True)
4 acc_plot.set(xlabel="Epochs", ylabel="Loss")

```



```

1 loss4, accuracy4 = model.evaluate(test_images, labels_testf)
2 print(f"Test loss: {loss4}")
3 print(f"Test accuracy: {accuracy4}")

38/38 [=====] - 1s 37ms/step - loss: 0.2844 - accuracy: 0.9217
Test loss: 0.2844119668006897
Test accuracy: 0.92166668176651

```

```

1 # Define the labels for easier interpretation
2 labels = [
3     'Uninfected',
4     'Parasited',
5 ]

```

```

1 random_inx = np.random.choice(test_images.shape[0])
2 print(random_inx)
3 test_image = test_images[random_inx]
4
5 plt.axis('off')
6
7 plt.imshow(test_image[:, :, 0], cmap='gray')
8 plt.show()
9 print(f"Label: {labels[labels_testf[random_inx]]}")

```

973



Label: Uninfected

```

1 pred = model.predict(test_image[np.newaxis,...])
2 print("Model: ", labels[np.argmax(pred)])

```

```

WARNING:tensorflow:6 out of the last 6 calls to <function Model.make_predict_function.<locals>.predict_function at 0x7ff98f718af0> trig
1/1 [=====] - 0s 106ms/step
Model: Uninfected

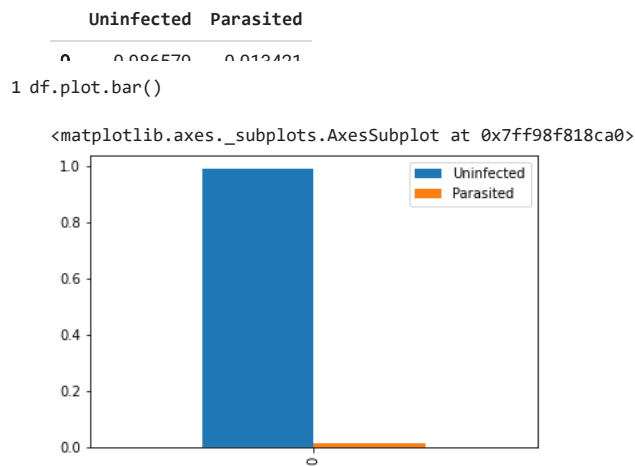
```

```

1 df = pd.DataFrame(pred, columns=labels)
2 df

```



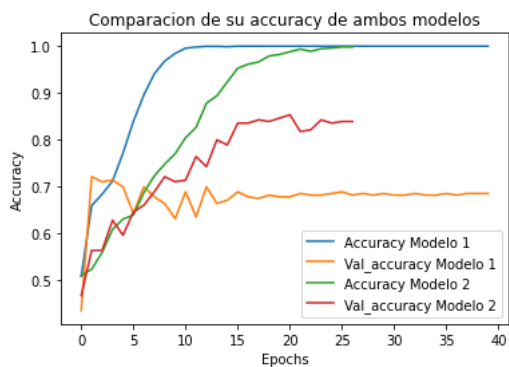


1

## ▼ Comparación

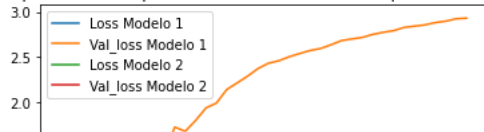
### ▼ Resultados de los primeros modelos

```
1 plt.plot(frame["accuracy"], label="Accuracy Modelo 1")
2 plt.plot(frame["val_accuracy"], label="Val_accuracy Modelo 1")
3 plt.plot(frame2["accuracy"], label="Accuracy Modelo 2")
4 plt.plot(frame2["val_accuracy"], label="Val_accuracy Modelo 2")
5 plt.title("Comparacion de su accuracy de ambos modelos")
6 plt.legend()
7 plt.xlabel("Epochs")
8 plt.ylabel("Accuracy")
9 plt.show()
```



```
1 plt.plot(frame["loss"], label="Loss Modelo 1")
2 plt.plot(frame["val_loss"], label="Val_loss Modelo 1")
3 plt.plot(frame2["loss"], label="Loss Modelo 2")
4 plt.plot(frame2["val_loss"], label="Val_loss Modelo 2")
5 plt.title("Comparacion de la perdida de ambos modelos respecto a las epocas")
6 plt.xlabel("Epochs")
7 plt.ylabel("Loss")
8 plt.legend()
9 plt.show()
```

Comparacion de la perdida de ambos modelos respecto a las epocas



```
1 print(f"Test loss primer modelo: {loss}")
2 print(f"Test accuracy primer modelo: {accuracy}")
```

```
Test loss primer modelo: 2.679617166519165
Test accuracy primer modelo: 0.665000214576721
```

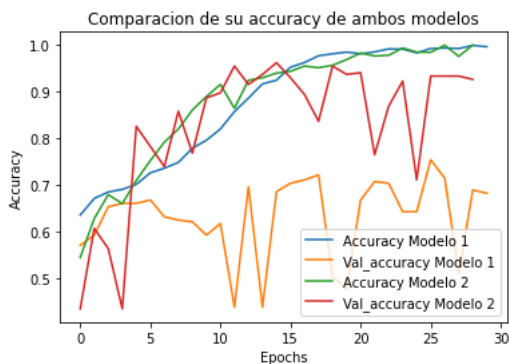
```
0.0 |
```

```
1 print(f"Test loss segundo modelo: {loss2}")
2 print(f"Test accuracy segundo modelo: {accuracy2}")
```

```
Test loss segundo modelo: 0.461984246969223
Test accuracy segundo modelo: 0.8383333086967468
```

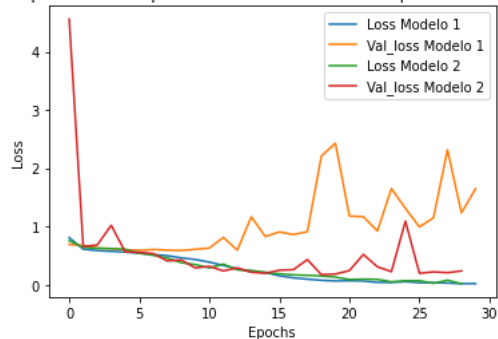
## ▼ Resultados de los restantes modelos

```
1 plt.plot(frame3["accuracy"], label="Accuracy Modelo 1")
2 plt.plot(frame3["val_accuracy"], label="Val_accuracy Modelo 1")
3 plt.plot(frame4["accuracy"], label="Accuracy Modelo 2")
4 plt.plot(frame4["val_accuracy"], label="Val_accuracy Modelo 2")
5 plt.title("Comparacion de su accuracy de ambos modelos")
6 plt.legend()
7 plt.xlabel("Epochs")
8 plt.ylabel("Accuracy")
9 plt.show()
```



```
1 plt.plot(frame3["loss"], label="Loss Modelo 1")
2 plt.plot(frame3["val_loss"], label="Val_loss Modelo 1")
3 plt.plot(frame4["loss"], label="Loss Modelo 2")
4 plt.plot(frame4["val_loss"], label="Val_loss Modelo 2")
5 plt.title("Comparacion de la perdida de ambos modelos respecto a las epocas")
6 plt.xlabel("Epochs")
7 plt.ylabel("Loss")
8 plt.legend()
9 plt.show()
```

Comparacion de la perdida de ambos modelos respecto a las epocas



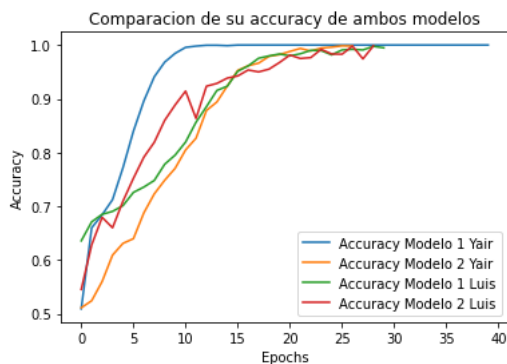
```
1 print(f"Test loss primer modelo: {loss3}")
2 print(f"Test accuracy primer modelo: {accuracy3}")
```

```
Test loss primer modelo: 2.3059241771698
Test accuracy primer modelo: 0.5791666507720947
```

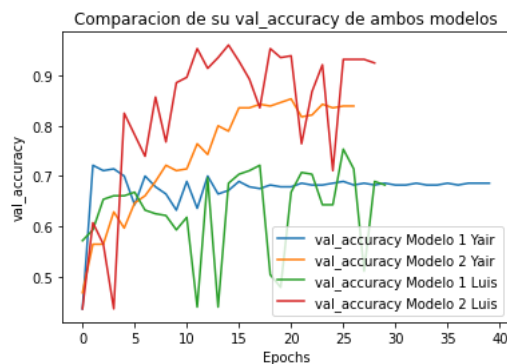
```
1 print(f"Test loss segundo modelo: {loss4}")
2 print(f"Test accuracy segundo modelo: {accuracy4}")
```

```
Test loss segundo modelo: 0.2844119668006897
Test accuracy segundo modelo: 0.92166668176651
```

```
1 plt.plot(frame["accuracy"], label="Accuracy Modelo 1 Yair")
2 plt.plot(frame2["accuracy"], label="Accuracy Modelo 2 Yair")
3 plt.plot(frame3["accuracy"], label="Accuracy Modelo 1 Luis")
4 plt.plot(frame4["accuracy"], label="Accuracy Modelo 2 Luis")
5 plt.title("Comparacion de su accuracy de ambos modelos")
6 plt.legend()
7 plt.xlabel("Epochs")
8 plt.ylabel("Accuracy")
9 plt.show()
```

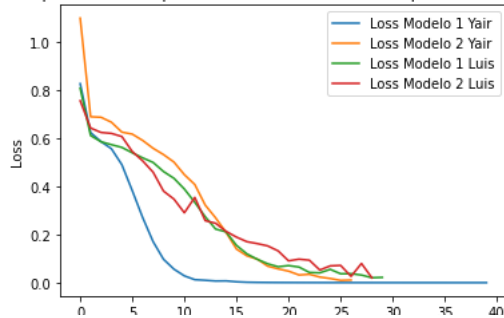


```
1 plt.plot(frame["val_accuracy"], label="val_accuracy Modelo 1 Yair")
2 plt.plot(frame2["val_accuracy"], label="val_accuracy Modelo 2 Yair")
3 plt.plot(frame3["val_accuracy"], label="val_accuracy Modelo 1 Luis")
4 plt.plot(frame4["val_accuracy"], label="val_accuracy Modelo 2 Luis")
5 plt.title("Comparacion de su val_accuracy de ambos modelos")
6 plt.legend()
7 plt.xlabel("Epochs")
8 plt.ylabel("val_accuracy")
9 plt.show()
```



```
1 plt.plot(frame["loss"], label="Loss Modelo 1 Yair")
2 plt.plot(frame2["loss"], label="Loss Modelo 2 Yair")
3 plt.plot(frame3["loss"], label="Loss Modelo 1 Luis")
4 plt.plot(frame4["loss"], label="Loss Modelo 2 Luis")
5 plt.title("Comparacion de la perdida de ambos modelos respecto a las epocas")
6 plt.xlabel("Epochs")
7 plt.ylabel("Loss")
8 plt.legend()
9 plt.show()
```

Comparacion de la perdida de ambos modelos respecto a las epocas

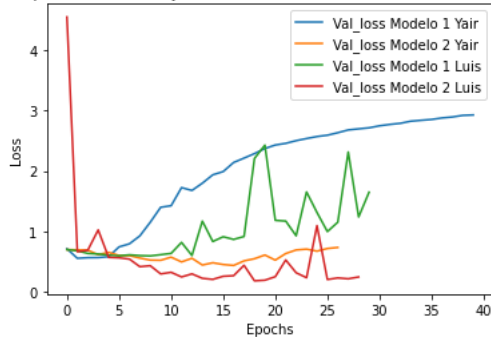


```

1
2 plt.plot(frame["val_loss"], label="Val_loss Modelo 1 Yair")
3 plt.plot(frame2["val_loss"], label="Val_loss Modelo 2 Yair")
4 plt.plot(frame3["val_loss"], label="Val_loss Modelo 1 Luis")
5 plt.plot(frame4["val_loss"], label="Val_loss Modelo 2 Luis")
6 plt.title("Comparacion de la perdida en el conjunto de validacion de ambos modelos respecto a las epocas")
7 plt.xlabel("Epochs")
8 plt.ylabel("Loss")
9 plt.legend()
10 plt.show()

```

Comparacion de la perdida en el conjunto de validacion de ambos modelos respecto a las epocas



Dentro de los 4 métodos utilizados se llegó a la conclusión de que los primeros implementados por cada uno de nosotros no obtuvieron buenos resultados en su entrenamiento ya que se obtuvo un accuracy de 0.66 y 0.82 respectivamente, estos modelos cabe recalcar son sencillos ya que son capas no tan desarrolladas y como se puede ver en las gráficas los valores de accuracy y val\_accuracy son bastantes distantes entre sí por lo cual nos indica que los valores y en las predicciones podemos ver que ambos modelos se equivocan con lo que comprobamos que una red neuronal muy sencilla no puede ser de gran ayuda para la detección de la Malaria.

Ahora bien, viendo los siguientes dos métodos implementados por cada uno de nosotros llegamos a muchos mejores resultados solo en el último con una accuracy del 0.92 y aunque se trató de armar un nuevo método con mejoras se llegó a una accuracy en este de 0.57.

Para el primer método más complejo se añadieron dos Dropout en las últimas capas densas antes de la última de Softmax con una tasa de 0.01 y 0.02 solamente pero no se tuvieron resultados significativos ya que posiblemente esto acertaba los resultados de la penúltima capa al apagar el 2% de las neuronas, al igual que se agregó una capa conv2D al principio para entrenar de mejor manera la red pero no logro ser así ya que el modelo mostraba un poco de deficiencias a la hora de hacer las predicciones ya que contaba con una accuracy del 0.62 lo cual era un poco bajo.

Para el segundo método complejo se decidió aplicar 4 capas conv2D esto para obtener mucho mayor complejidad en el aprendizaje de la red, ahora después de esto se implementaron 6 capas densas 5 con la función de activación relu y una última con softmax, a la antepenúltima se le aplicó una capa de batch normalization para así normalizar los datos y relacionarlos de mejor manera y la antepenúltima y la que le sigue se aplicó un proceso de Dropout ya que eran demasiadas neuronas y se le aplicó un apago del 0.01% de ellas para ver el comportamiento de esta red y rindió resultados. Al correr esta red sus resultados fueron muy buenos obteniendo un test de accuracy del 0.91 aunque la pérdida es un poco alta considerablemente con 0.21, después de lanzar una imagen para probar la posible predicción resulta que si lo logra predecir correctamente aunque la imagen sea difícil de percibir el parásito, esto con una identificación del parásito del 63% lo cual indica una buena predicción con una imagen muy compleja.

## ▼ Conclusiones individuales

Luis Gerardo Lagunes Najera Las posibles mejoras que veo para los dos métodos que implemente podrían ser agregar más capas conv2D a ambos métodos para mejorar el entrenamiento de estos y mejorar el val\_accuracy ya que pude ver que con mayor capas conv2D se podía mejorar mucho mejor el modelo, aunque tambien se le podrían agregar más capas densas de gran cantidad de neuronas que vayan bajando de número hasta llegar a 2 neuronas en la ultima capa que es de la función de activación softmax, aunque las demás capas se podrían cambiar por distintas funciones de activación para ver como se comportan cada una de ellas y ver cuál sería la adecuada para este modelo al igual que probar con las funciones de optimización.

## ▼ Jesus Yair Ramirez Islas

Para concluir puedo decir que si bien logre reducir en cierta medida el overfitting en el segundo modelo, como mejora podría implementar algunos cambios para buscar un aumento en el accuracy que se obtiene al evaluar el modelo. Los cambios podrían ser probar con otro optimizador, añadir capas de batch normalization y ya que el modelo que obtuvo mejores resultados tenía una gran cantidad de capas, podría añadir mas a mi arquitectura, ya que si bien la teoría indica que para reducir el overfitting hay que emplear modelos más sencillos probablemente al ser tan sencillo no es capaz de extraer la información más relevante para poder hacer las predicciones. Tales capas pueden ser una extra convolucional, y el resto densas con más neuronas y algunas capas de dropout de manera intercalada.

## ▼ Anexos

Atecnea. (2021, 11 marzo). Programar Red Neuronal Convolucional (CNN) en Python utilizando Keras [Video]. YouTube. Recuperado 3 de diciembre de 2022, de <https://www.youtube.com/watch?v=K8H1GVWD3hY&feature=youtu.be>  
<https://www.aprendemachinelearning.com/author/user/#author>. (2021, 22 marzo). Qué es overfitting y underfitting y cómo solucionarlo. Aprende Machine Learning. <https://www.aprendemachinelearning.com/que-es-overfitting-y-underfitting-y-como-solucionarlo/> Tutorial Python: ¿Cómo combatir el Overfitting en el Machine Learning? (s. f.). Codificando Bits. <https://www.codificandobits.com/blog/tutorial-overfitting-machine-learning-python/>