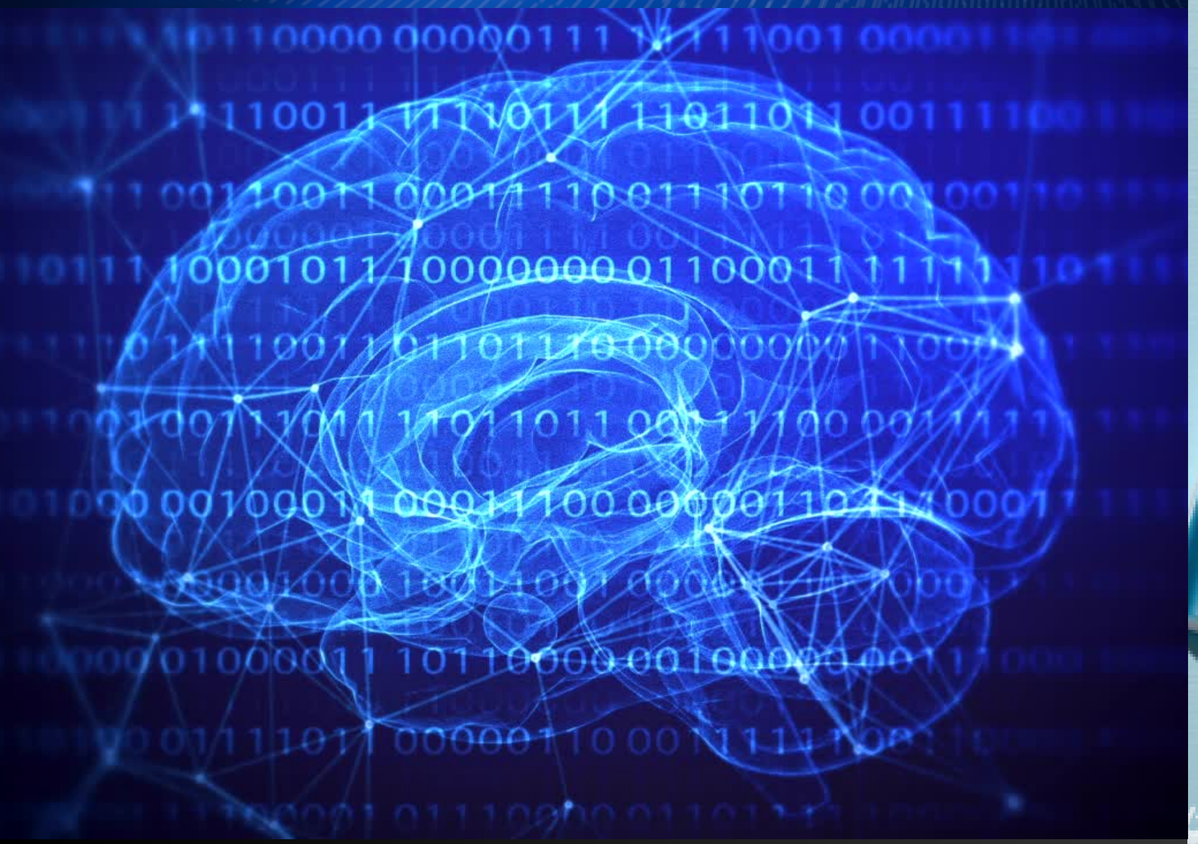
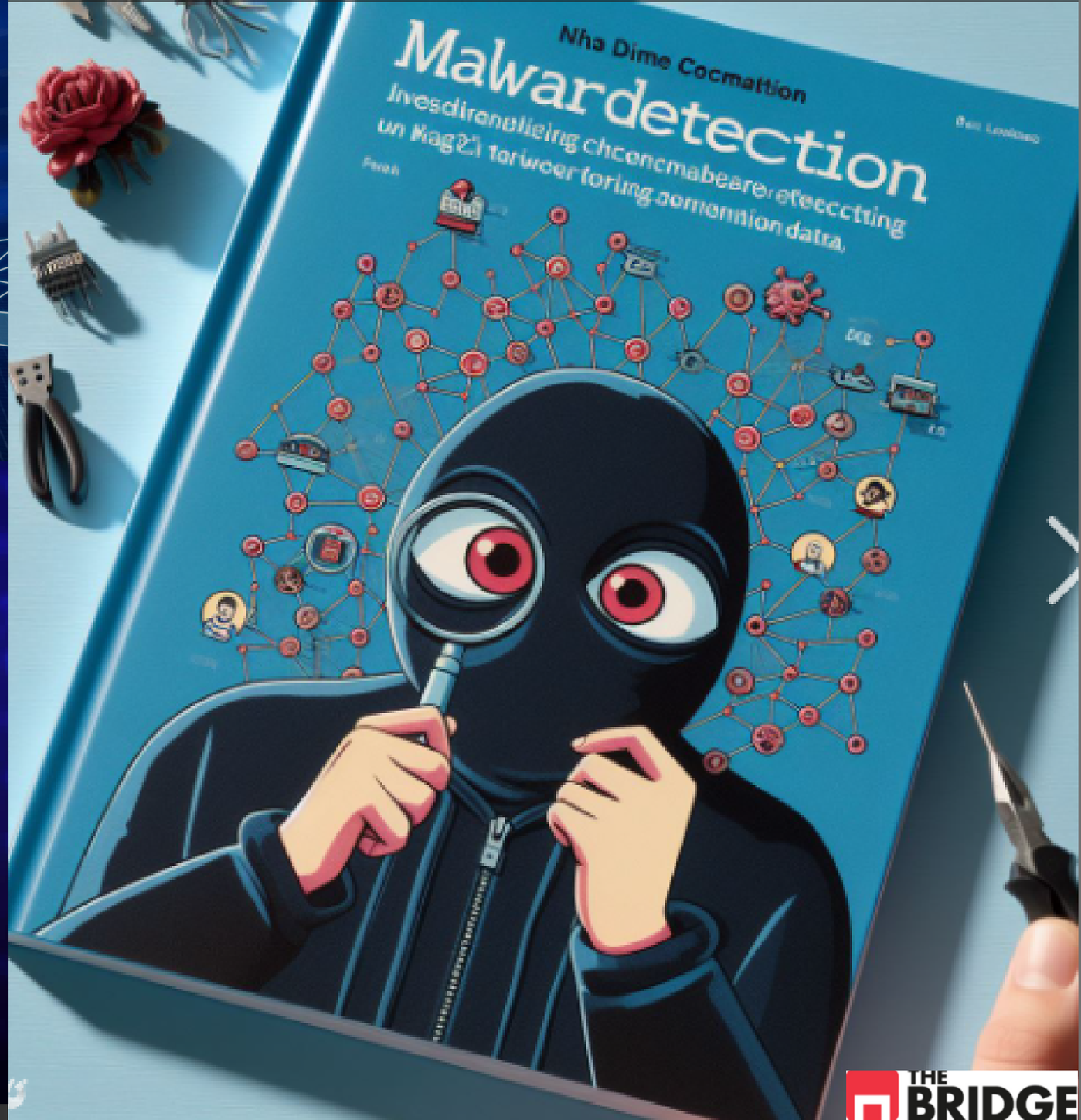
EL APRENDIZAJE AUTOMATICO

ESTUDIO DE LA DETECCIÓN

ATAQUES MALWARE

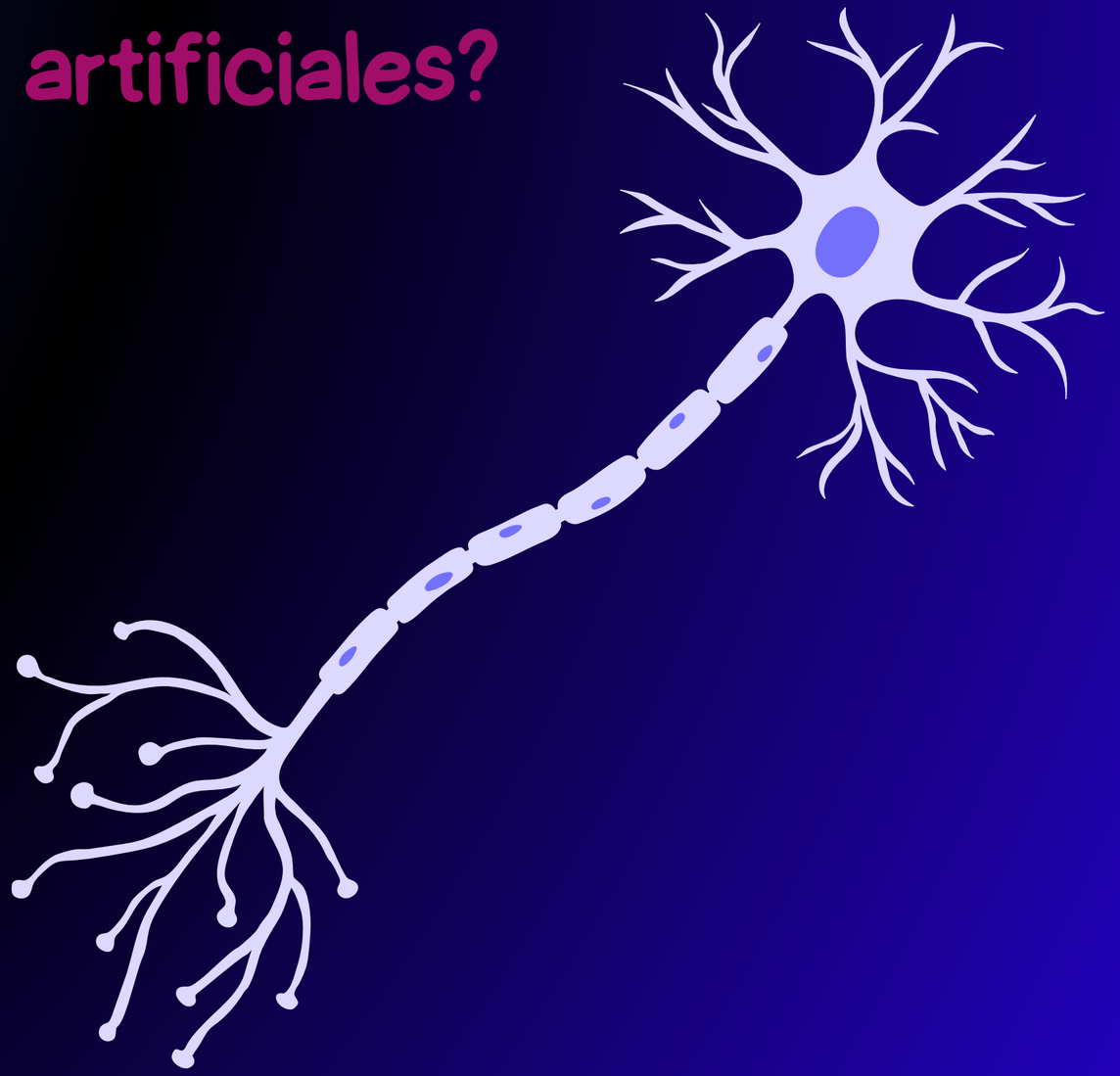




INTRODUCCIÓN

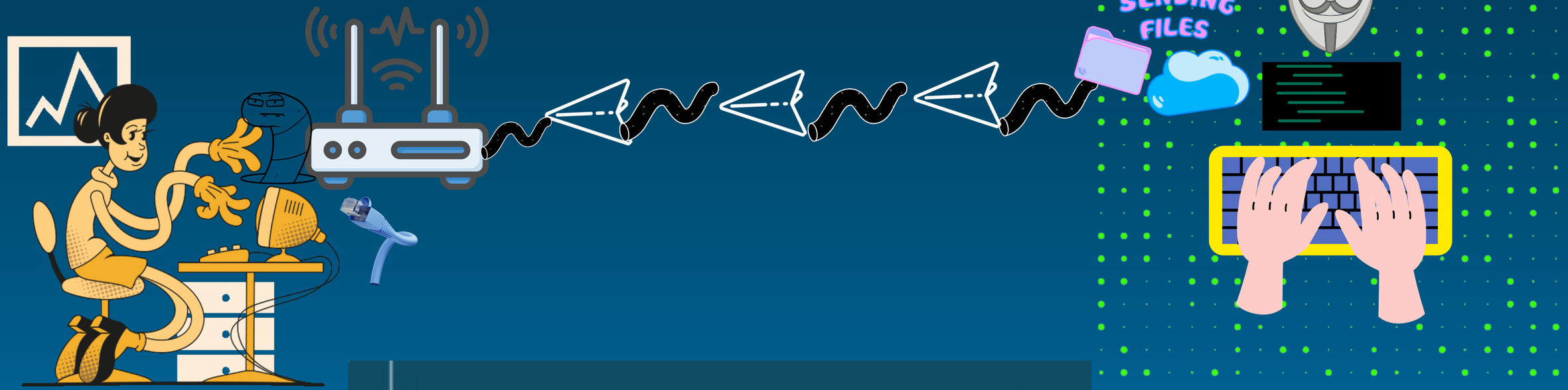
¿Qué son las neuronas?

¿Qué son las neuronas artificiales?



Breve historia de la IA

¿Qué es el malware?



PROCESO

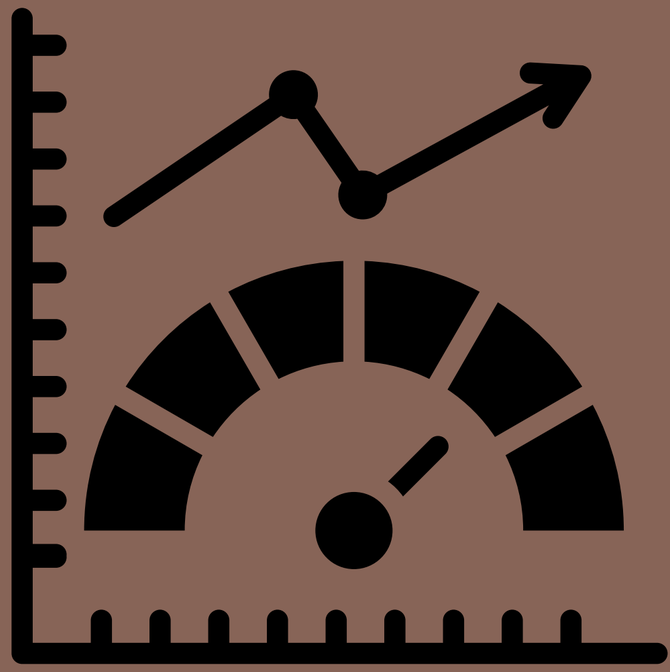
NEGOCIO, RECOPILACIÓN, PREPARACIÓN ANÁLISIS Y MODELADO DE DATOS E INFORME RESULTADOS

COMENZAMOS

PROBLEMA DE NEGOCIO

ESTE PROYECTO BUSCA LA EFICIENCIA EN LA DETECCION DE MALWARE TENIENDO EN CUENTA LOS DATOS OBRANTES EN EL DATASET OBJETO DE ESTUDIO, BUSCANDO QUE DETECTE TODOS LOS ATAQUES DE ESTE TIPO

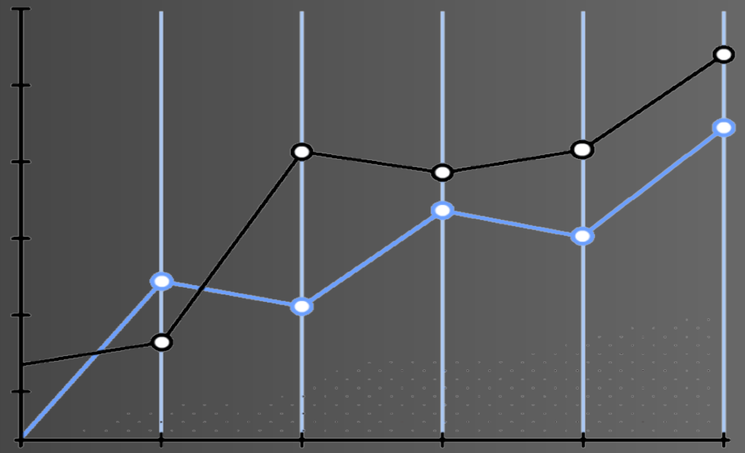
METRICA= RECALL

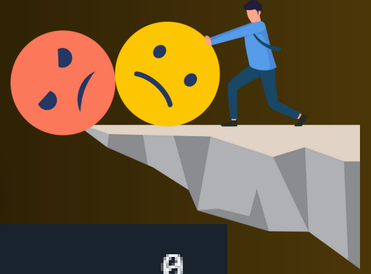


EL DATASET

Columna:	Descripción:	Tipo:	Orden de entrada:	Información posterior a la etiqueta "label"
id.resp_h	Identificador único del host de destino	Categórico	1	No
proto	Protocolo utilizado (TCP, UDP, ICMP, etc.)	Categórico	2	No
conn_state	Estado de la conexión (SYN, ACK, FIN, etc.)	Categórico	3	No
orig_pkts	Número de paquetes enviados por el host de origen	Numérico	4	No
orig_ip_bytes	Número de bytes enviados por el host de origen	Numérico	5	No
resp_pkts	Número de paquetes recibidos por el host de destino	Numérico	6	No
resp_ip_bytes	Número de bytes recibidos por el host de destino	Numérico	7	No
anyo	Año de la conexión	Numérico	8	No
mes	Mes de la conexión	Numérico	9	No
dia	Día de la conexión	Numérico	10	No
hora	Hora de la conexión	Numérico	11	No
id.resp_p	Identificador único del puerto de destino	Categórico	12	No
missed_bytes	Número de bytes perdidos en la conexión	Numérico	13	No
id.orig_p	Identificador único del puerto de origen	Categórico	14	No
hostory	conexiones y patrones detectados de conexiones anteriores	Categórico	15	No
label	Etiqueta que indica si la conexión es normal o anómala	Categórica	Target	

MINI- EDA





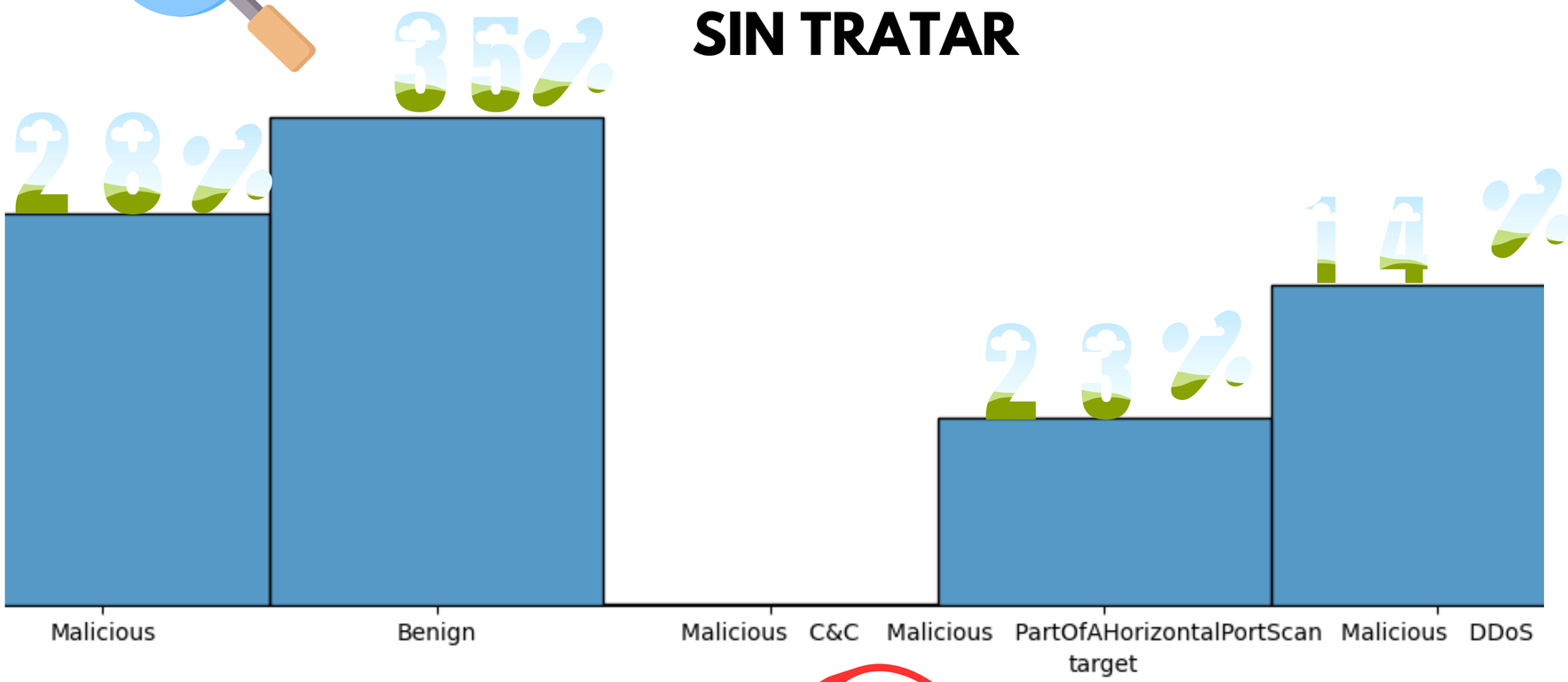
NULOS

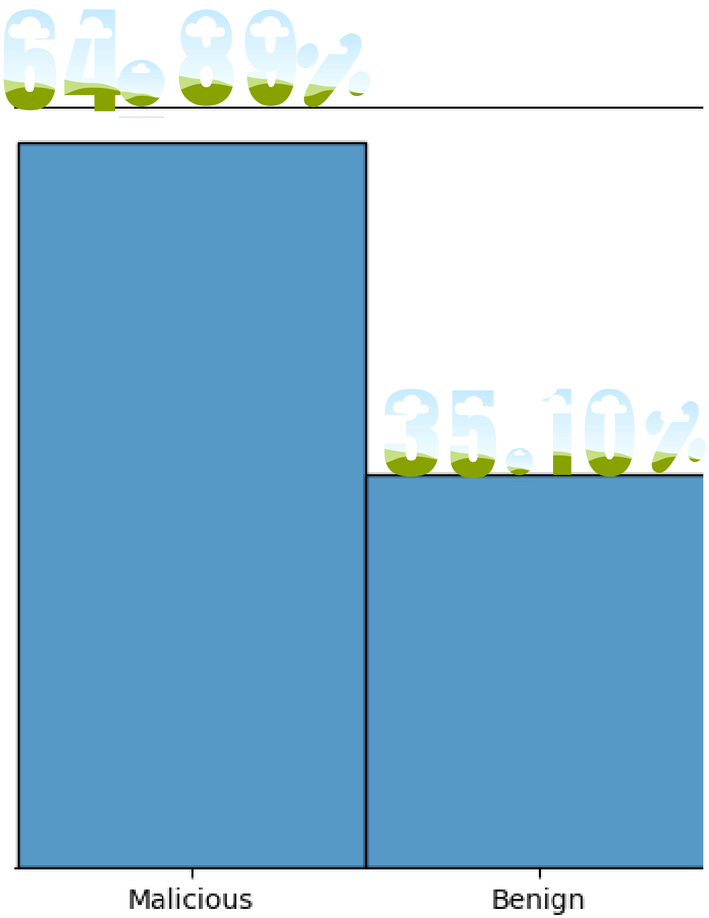
ts	0
uid	0
id.orig_h	0
id.orig_p	0
id.resp_h	0
id.resp_p	0
proto	0
service	24993006
duration	15272073
orig_bytes	15272073
resp_bytes	15272073
conn_state	0
local_orig	25011003
local_resp	25011003
missed_bytes	0
history	25116
orig_pkts	0
orig_ip_bytes	0
resp_pkts	0
resp_ip_bytes	0
tunnel_parents	25011003
label	0
detailed-label	17954112

TARTAMIENTO DE NULOS

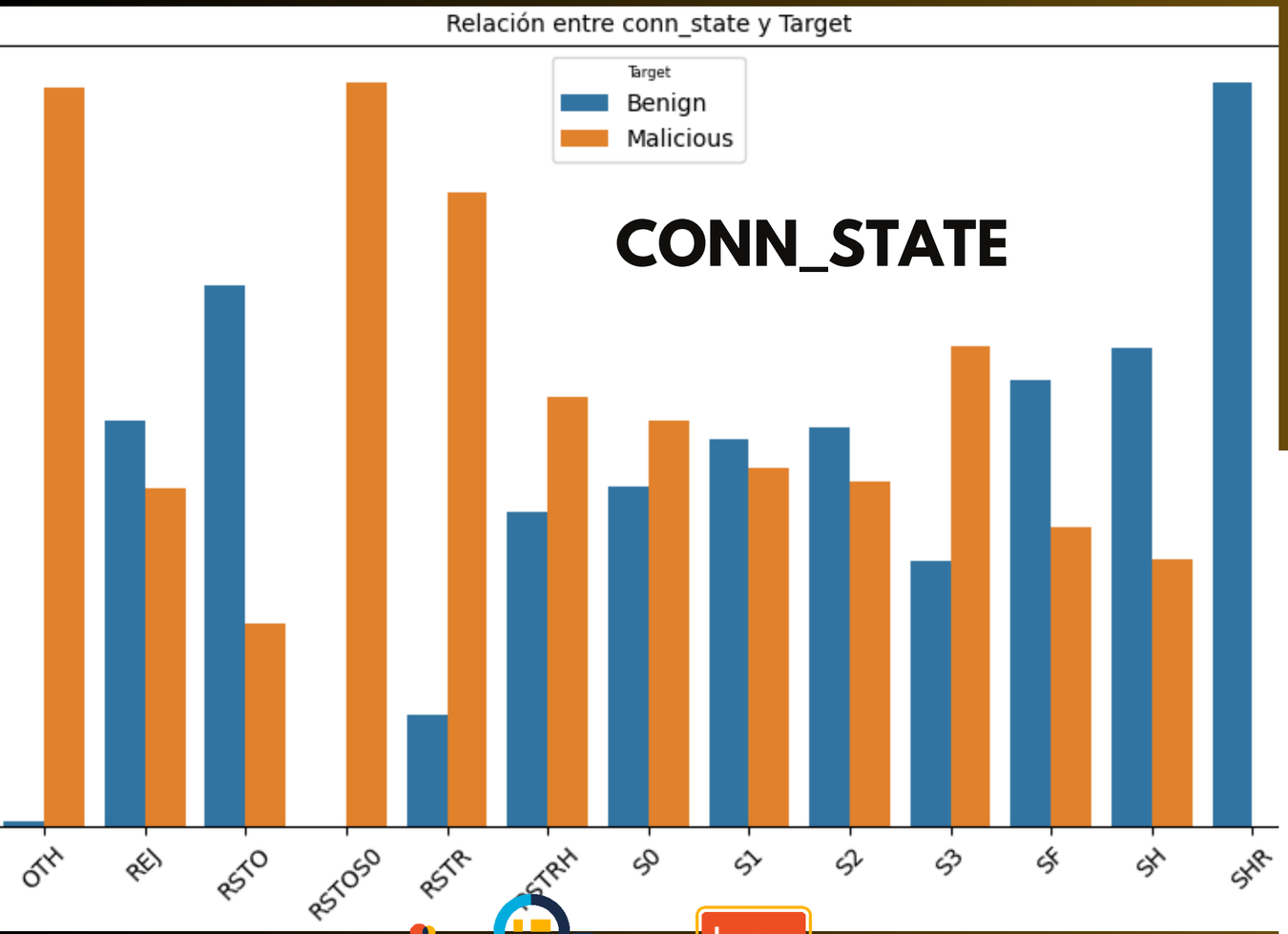
TARGET

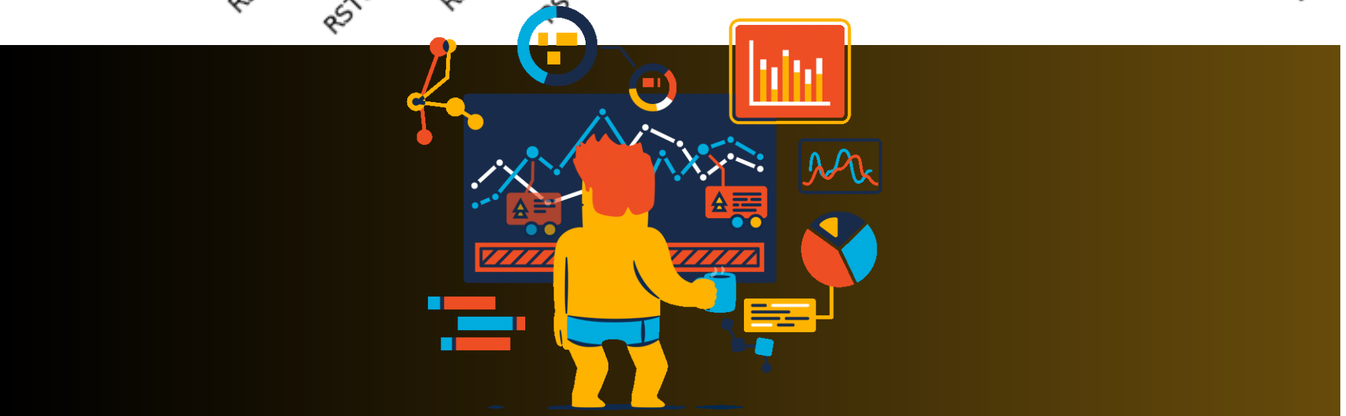
SIN TRATAR

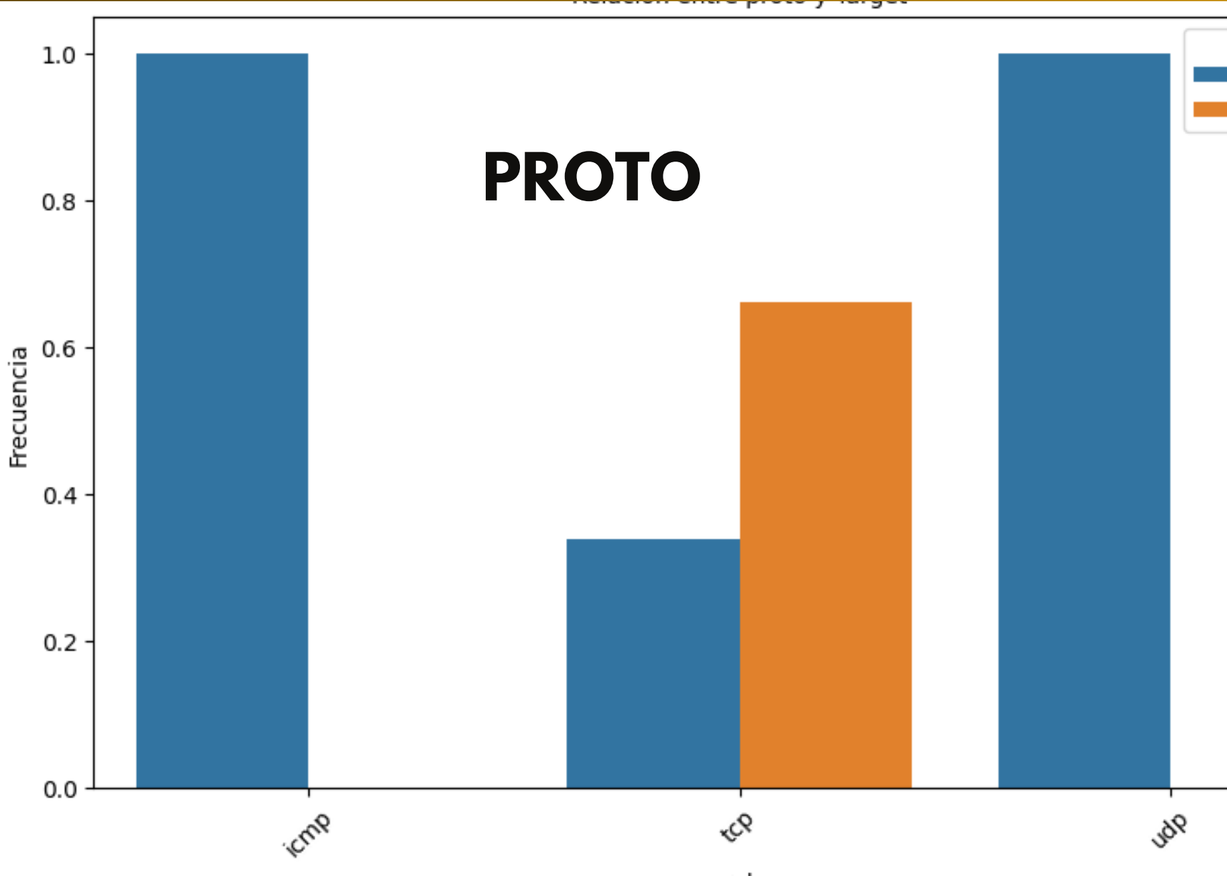




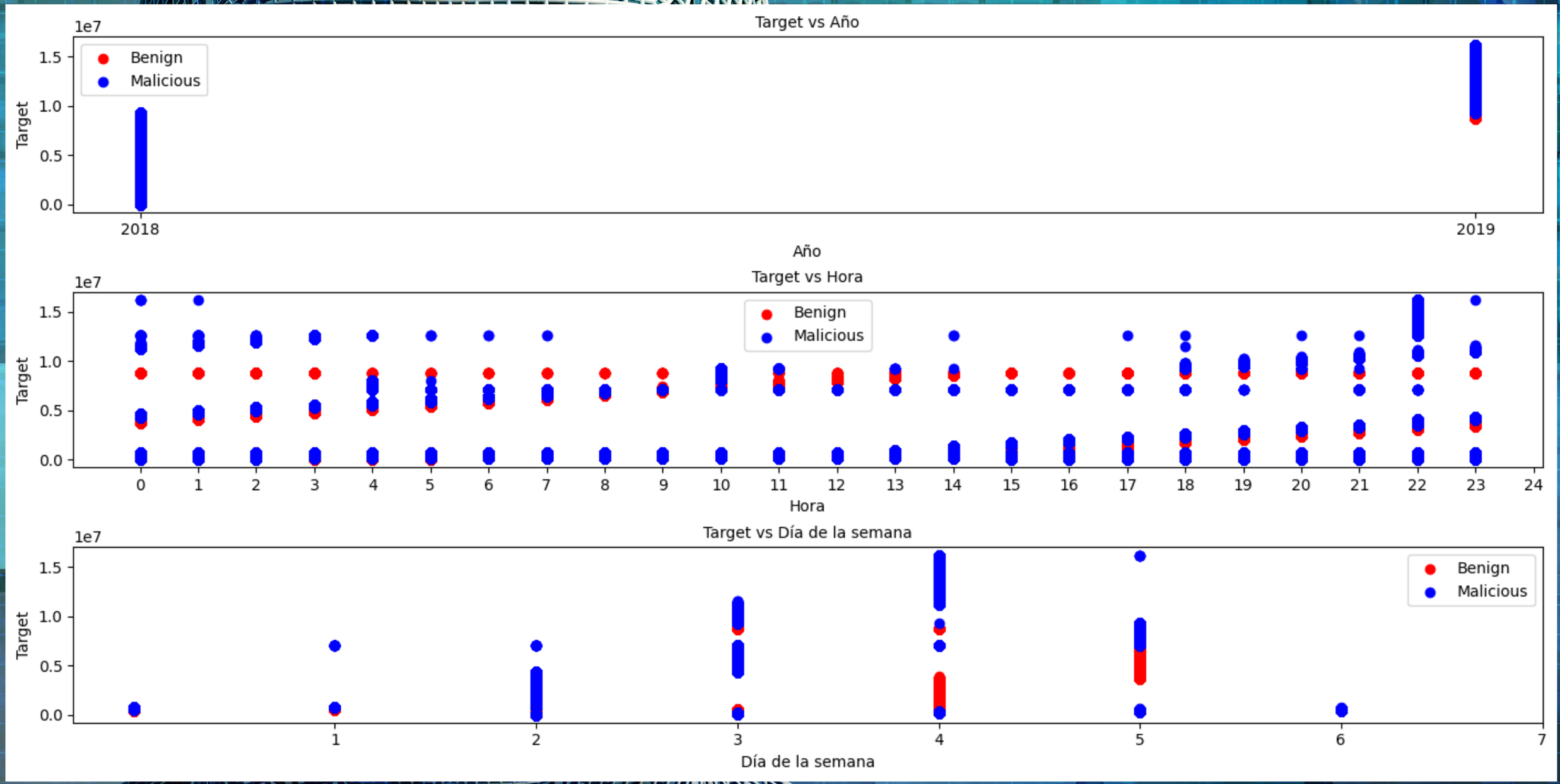
RELACIONES TARGET CON COLUMNAS CONN_STATE Y PROTO



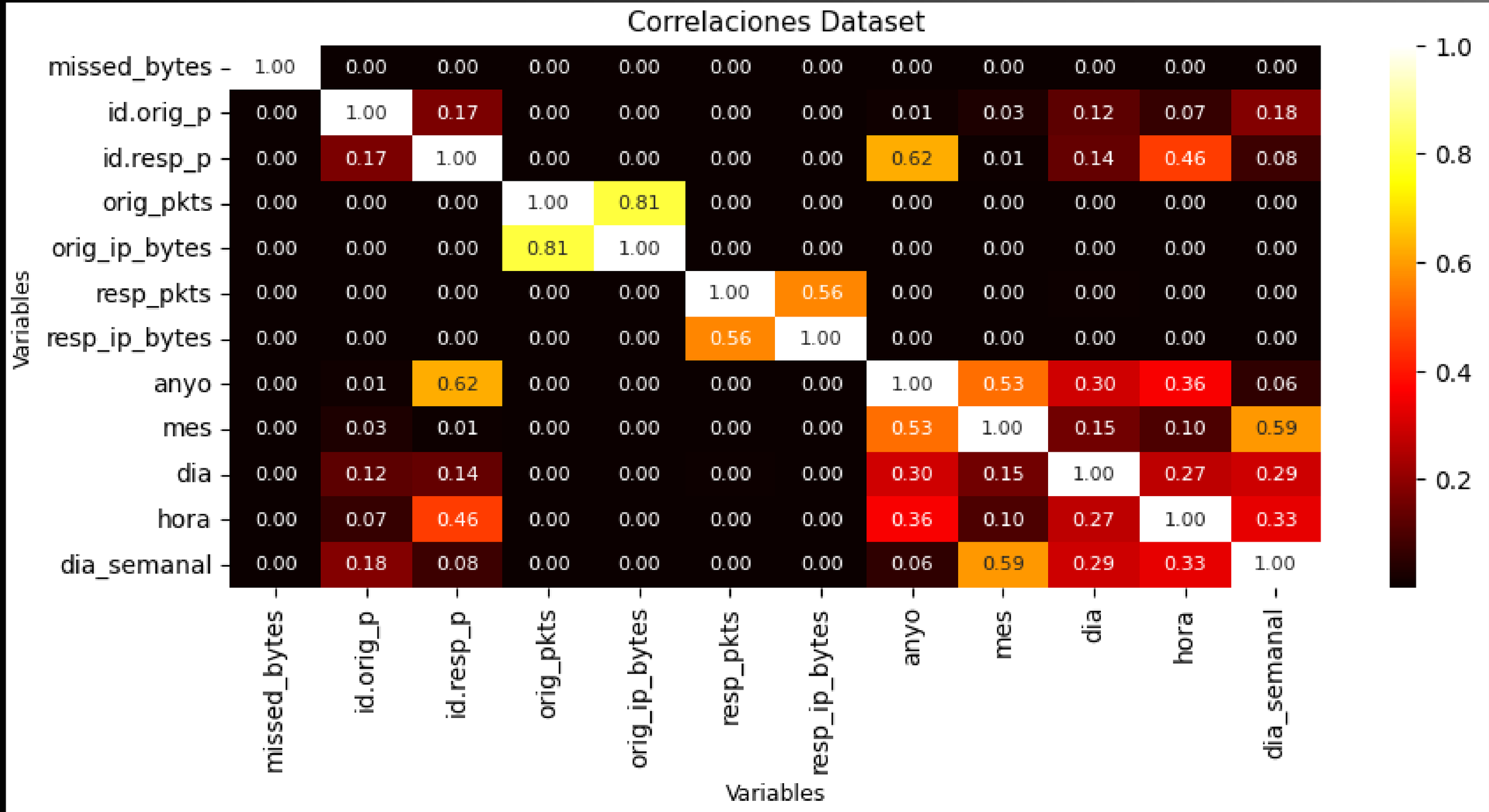


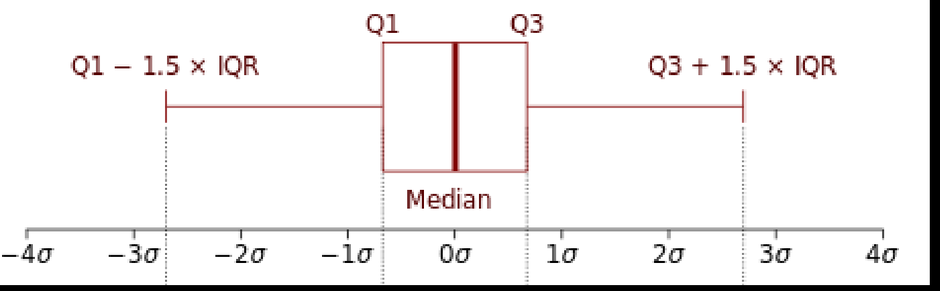


INTERVALOS DE ATAQUES POR HORA , AÑO Y DIA SEMANAL



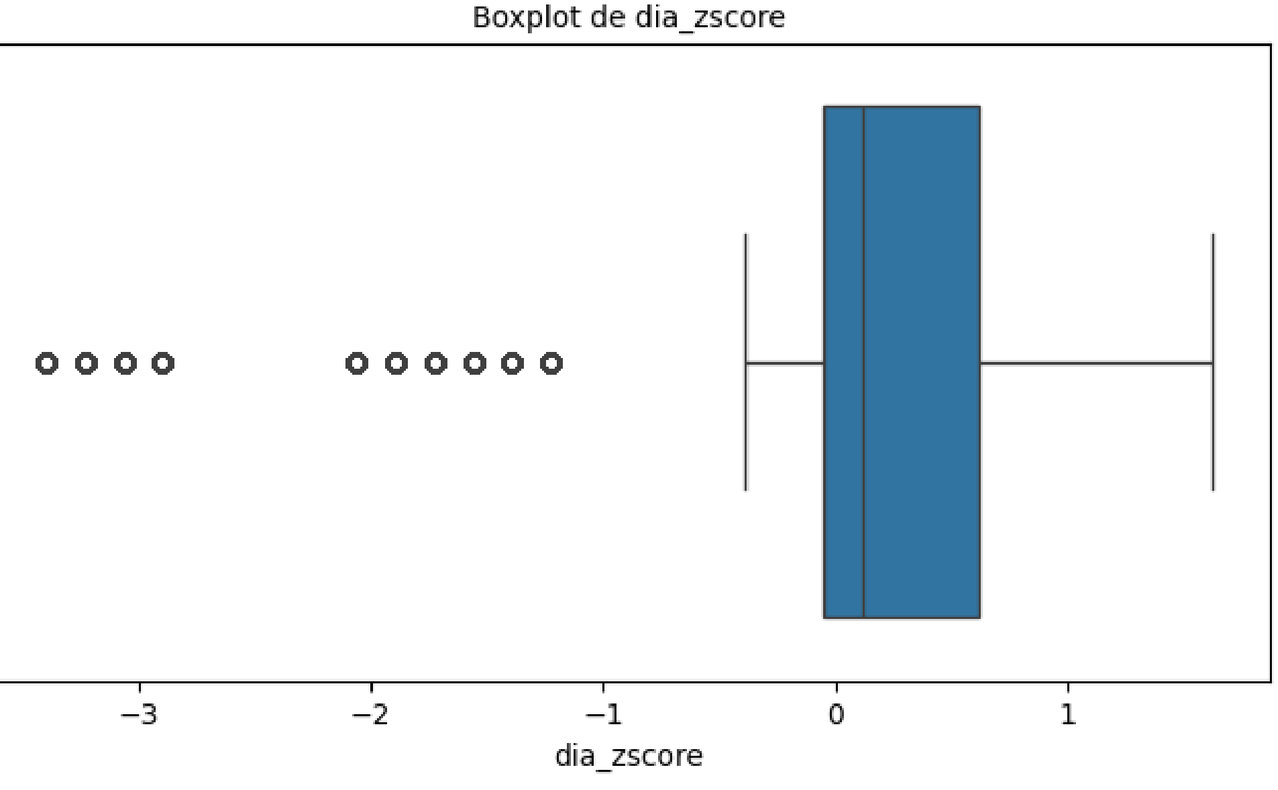
ESTUDIO DE LA CORRELACION



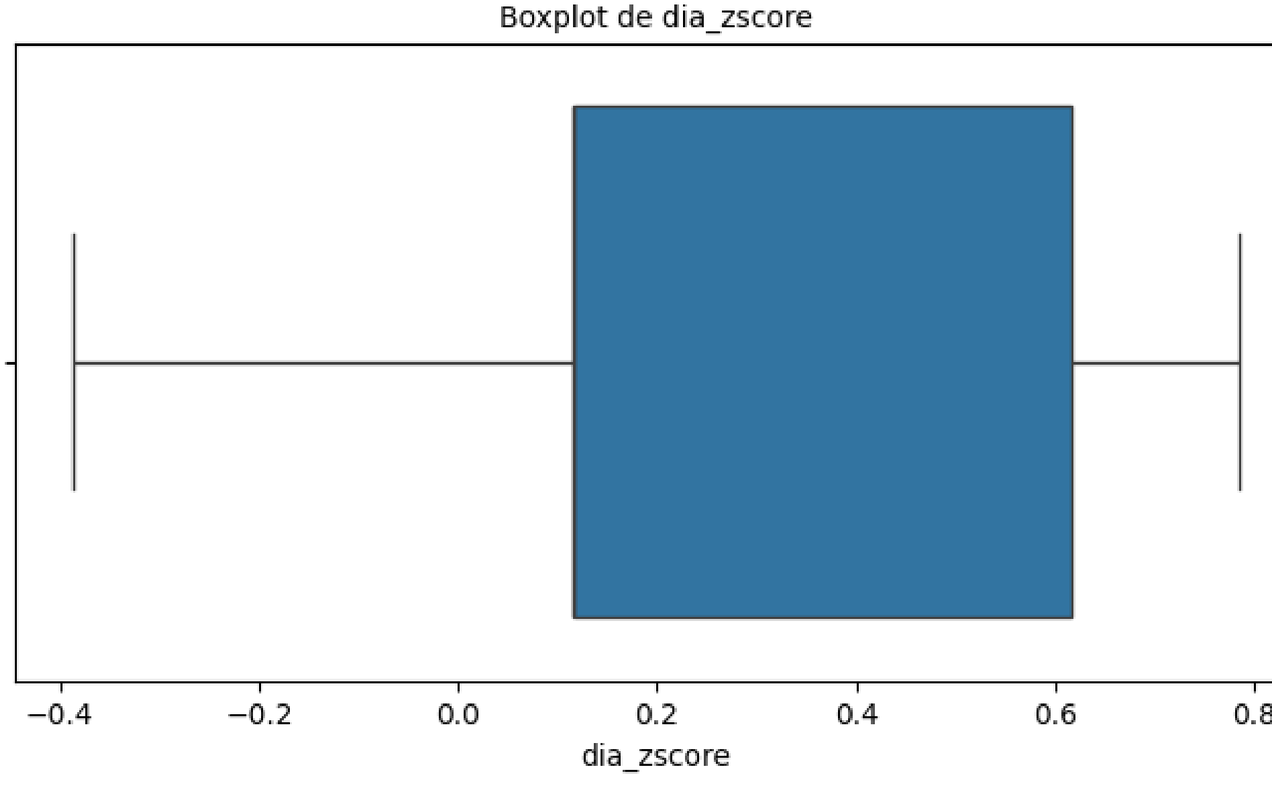


BUSQUEDA Y ELIMINACION DE OUTLIERS

ANTES DE ELIMINACION DE OUTLIERS



TRAS LA ELIMINACION DE OUTLIERS



FIN DEL EDA Y CUADRO DE COLUMNAS FINALES

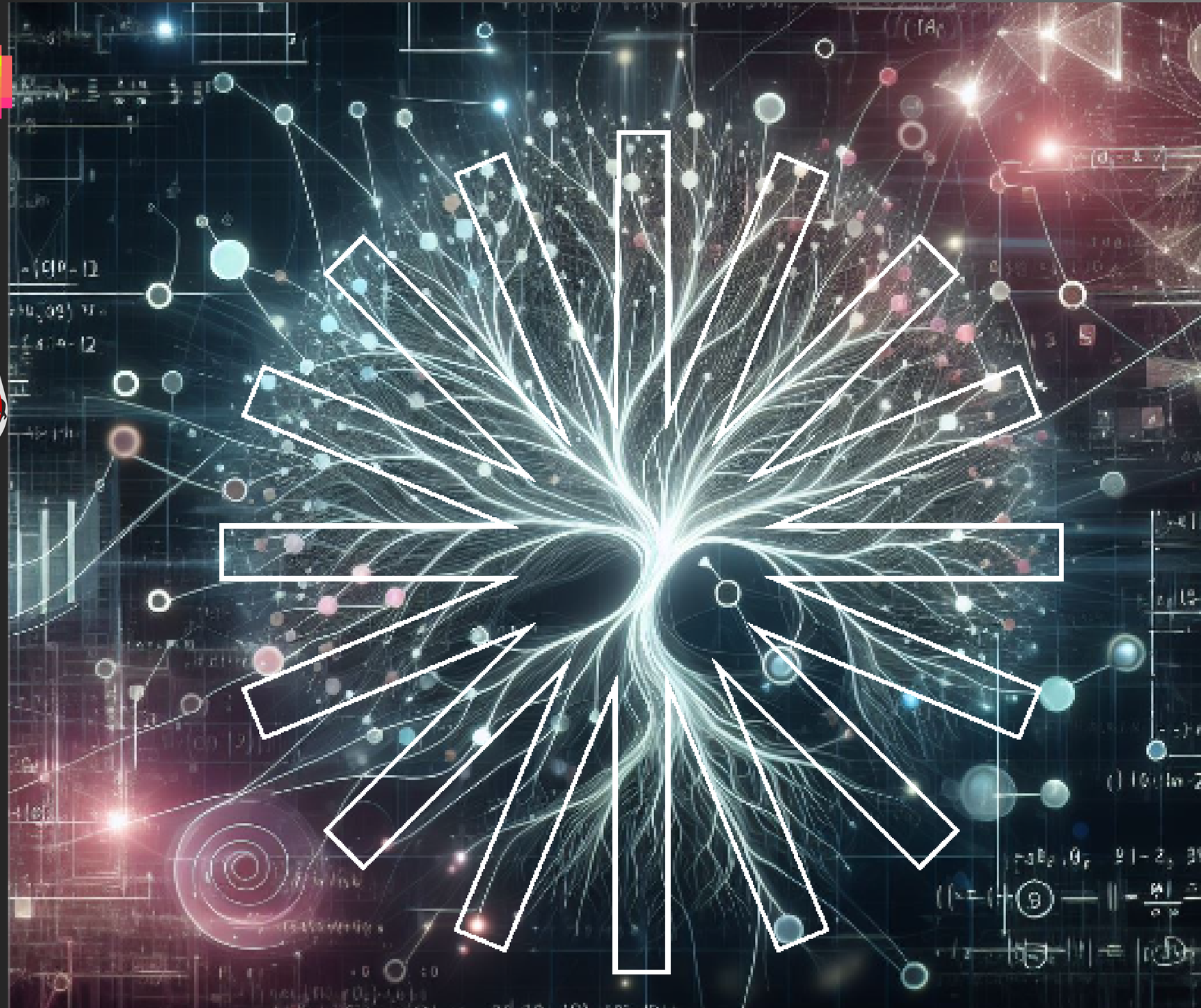
The end

Columna:	Descripción:	Tipo:	Orden de entrada:	Información posterior a la etiqueta "label"
id.resp_h	Identificador único del host de destino	Categórico	1	No
proto	Protocolo utilizado (TCP, UDP, ICMP, etc.)	Categórico	2	No
conn_state	Estado de la conexión (SYN, ACK, FIN, etc.)	Categórico	3	No
history	Historial de tráfico relacionado con esta conexión o sesión, pudiendo obtener patrones de actividad	Categórico	4	No
id.orig_p	Mes de la conexión	Numérico	5	No
orig_pkts	Número de paquetes enviados por el host de origen	Numérico	6	No
resp_pkts	Número de paquetes recibidos por el host de destino	Numérico	7	No
anyo	Año de la conexión	Numérico	8	No
hora	Hora de la conexión	Numérico	9	No
Target	Etiqueta que indica si la conexión es normal o anómala	Categórica	10	Target

2

START

MACHINE LEARNING

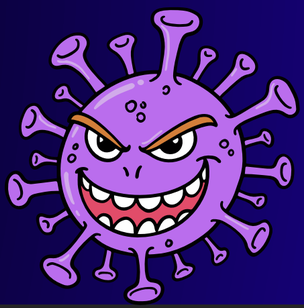


1.- MODELO ML RANDONFOREST CON 115 COLUMNAS

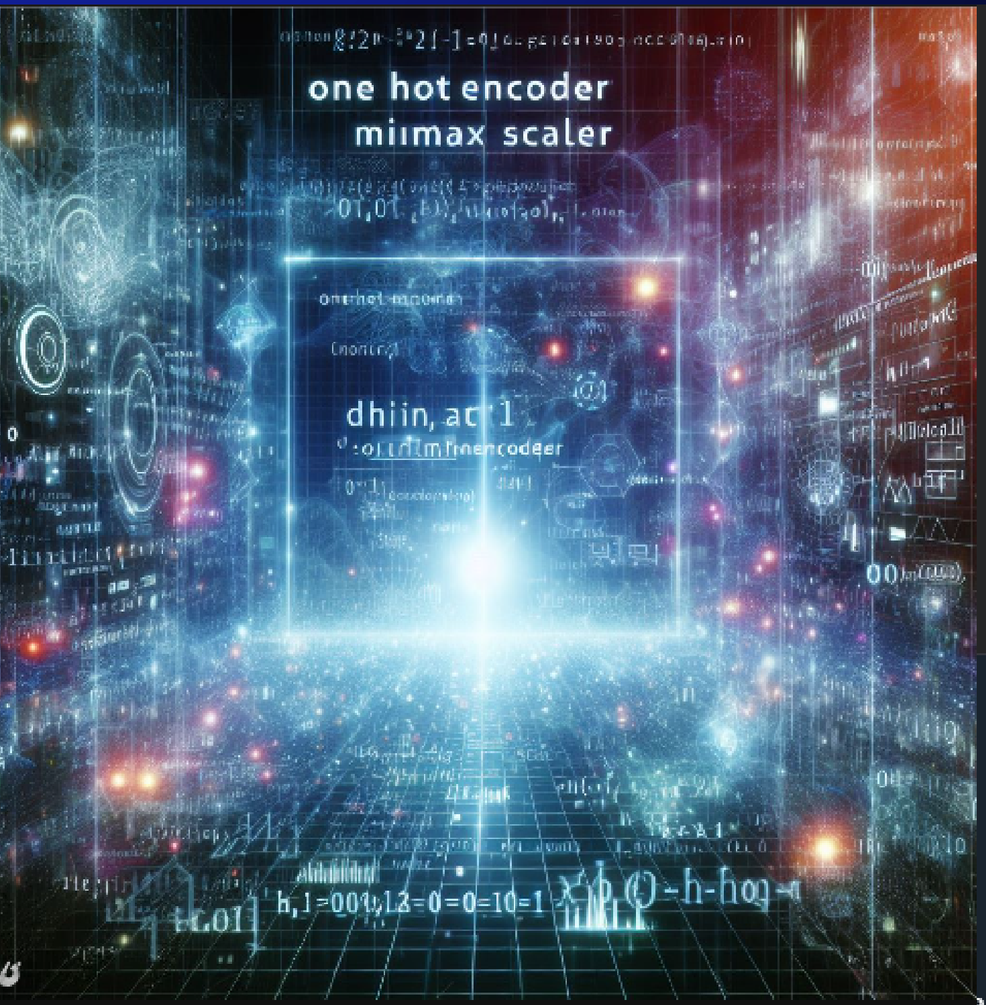
PROCESO TRANSFORMACION

RESULTADO

ALL



60%

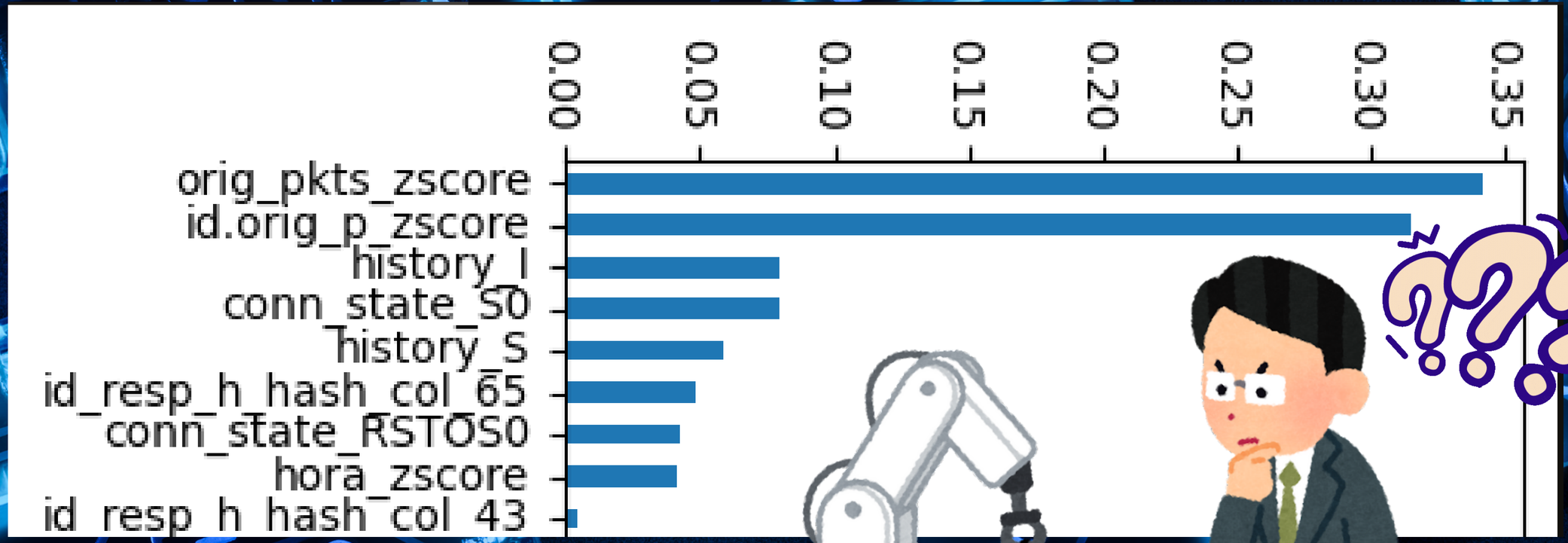



```python
# Definir el modelo Random Forest
model = RandomForestClassifier(n_estimators=100, max_depth=10, random_state=42, n_jobs=-1

# Entrenar el modelo
with parallel_backend('threading'):
    model.fit(X_train, y_train)

# Evaluar el modelo
y_pred = model.predict(X_val)
accuracy = accuracy_score(y_val, y_pred)

print(classification_report(y_pred, y_val))
```

WOW

	precision	recall	f1-score	support
0	0.58	1.00	0.74	808056
1	1.00	0.74	0.85	2228272
accuracy			0.81	3036328
macro avg	0.79	0.87	0.79	3036328
eighted avg	0.89	0.81	0.82	3036328

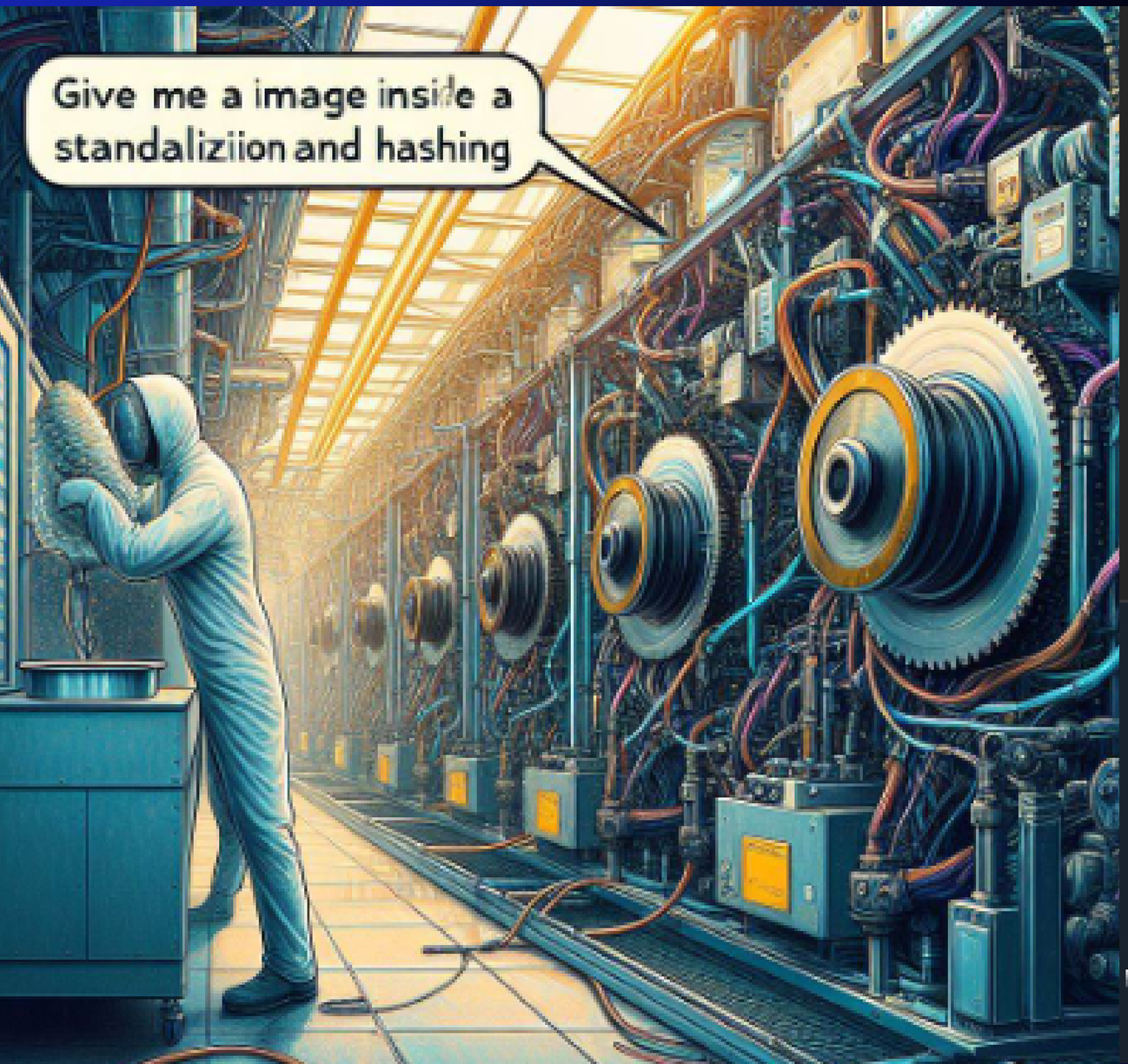
CARACT. MÁS INFLUYENTES DURANTE EL R.FOREST.



....DE 115 COLUMNAS....

2.- MODELO ML LOGISTICREGRESSION CON 115 COLUMNAS

PROCESO TRANSFORMACION



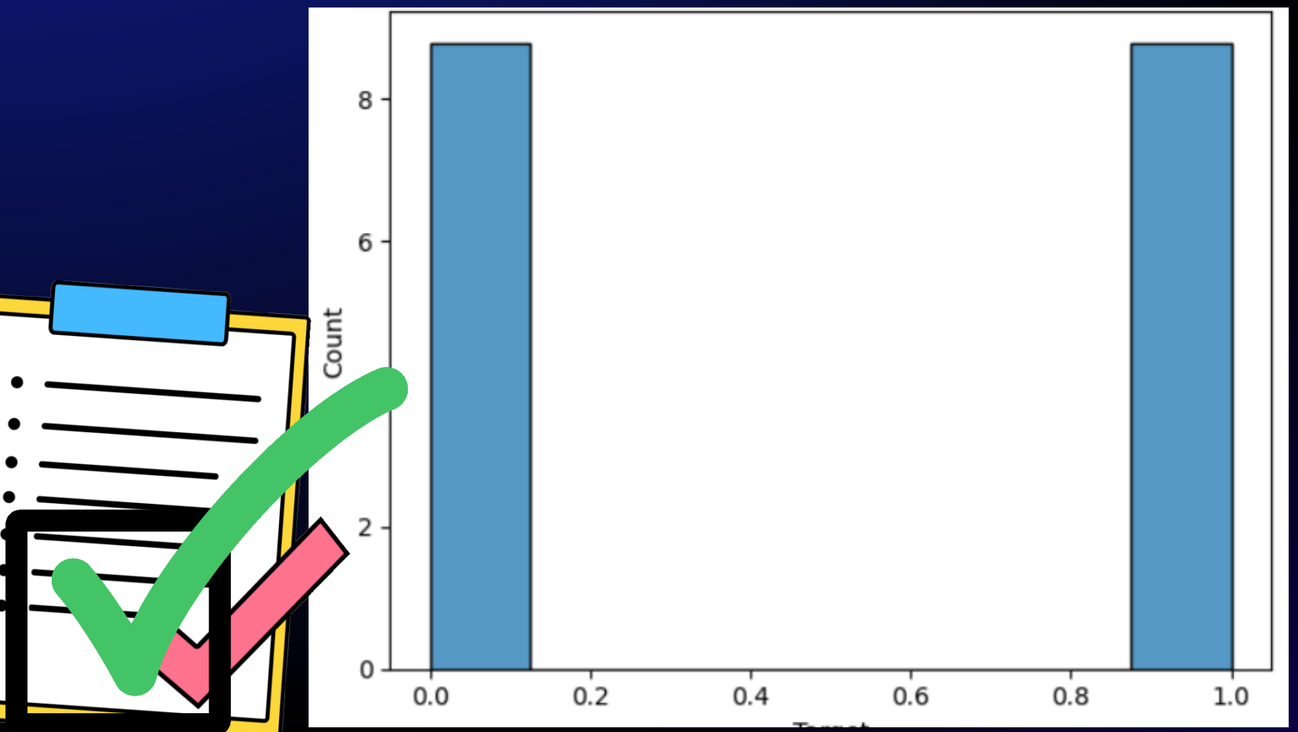
RESULTADO

```python
# Definir el modelo logisctic con C para regule el sobre
model = LogisticRegression(C=10, solver="sag")

# Entrenar el modelo
model.fit(X_train, y_train)

# Evaluar el modelo
y_pred = model.predict(X_val)
accuracy = accuracy_score(y_val, y_pred)

print(classification_report(y_pred, y_val))
```

	precision	recall	f1-score	support
0.0	0.57	0.70	0.62	1124927
1.0	0.79	0.69	0.73	1911401
accuracy			0.69	3036328
macro avg	0.68	0.69	0.68	3036328
weighted avg	0.71	0.69	0.69	3036328

2.-SUBMUESTREO DEL DATASET Y CLASE MINORITARIA RESAMPLED

```python
# Aplica submuestro, para reducir el número de obs
X=df_out.drop(["Target"], axis=1)
y=df_out["Target"]

rus = RandomUnderSampler(random_state=42)
X_resampled, y_resampled = rus.fit_resample(X, y)
```



MODELO LOGISITC REGRESSION

```python
# Definir el modelo
model = LogisticRegression(solver='sag', max_iter=100,

# Entrenar el modelo
with parallel_backend('threading'):
    model.fit(X_train, y_train)

# Predecir
y_pred = model.predict(X_val)

# Evaluar el modelo
print(classification_report(y_val, y_pred))
```

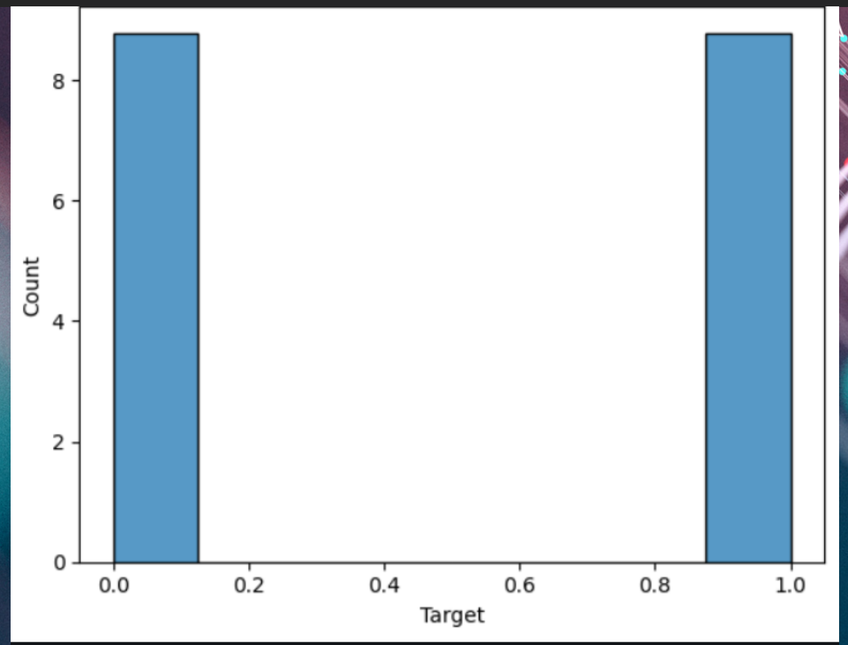
✓ 18m 1.1s

```
c:\Users\victo\anaconda3\envs\tf-gpu\lib\site-packages\skl
  warnings.warn(
              precision    recall  f1-score   support

         0.0       0.73      0.62      0.67   1385714
         1.0       0.67      0.77      0.72   1383801

    accuracy                           0.69   2769515
   macro avg       0.70      0.69      0.69   2769515
weighted avg       0.70      0.69      0.69   2769515
```

3.-SUBMUESTREO DEL DATASET Y CLASE MINORITARIA

RESAMPLED

```
# Aplica submuestro, para reducir el número de obser
X=df_out.drop(["Target"], axis=1)
y=df_out["Target"]

rus = RandomUnderSampler(random_state=42)
X_resampled, y_resampled = rus.fit_resample(X, y)
```



MODELO RANDOMFOREST

```
model_rf = RandomForestClassifier(n_estimators=100, max_depth=

# Entrenar el modelo
# Usar parallel_backend para aprovechar la multiprocesamiento
with parallel_backend('threading'):
    model_rf.fit(X_train, y_train)

# Predecir
y_pred = model_rf.predict(X_val)

# Evaluar el modelo
print(classification_report(y_val, y_pred))
```

✓ 3m 45.0s

	precision	recall	f1-score	support
0.0	0.98	0.60	0.75	1385714
1.0	0.71	0.99	0.83	1383801
accuracy			0.80	2769515
macro avg	0.85	0.80	0.79	2769515
weighted avg	0.85	0.80	0.79	2769515

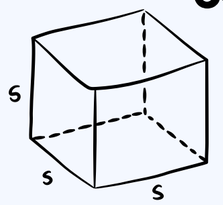
APLICACION DE TECNICAS DE DEEP-LEARNING

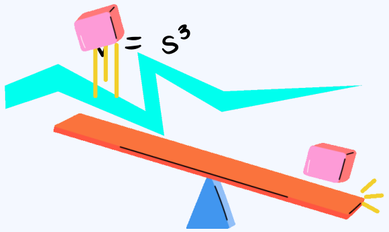
3

MI MODELO DE RED NEURONAL 1

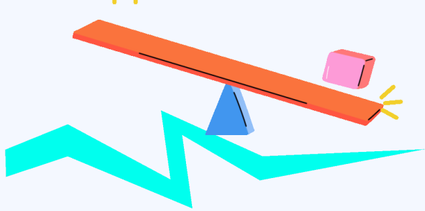
CAPAS OCULTAS
64-32-16-8 Y L2 1 CAPA Y 2 DENSA

CAPA DE ENTRADA
64 UNITS

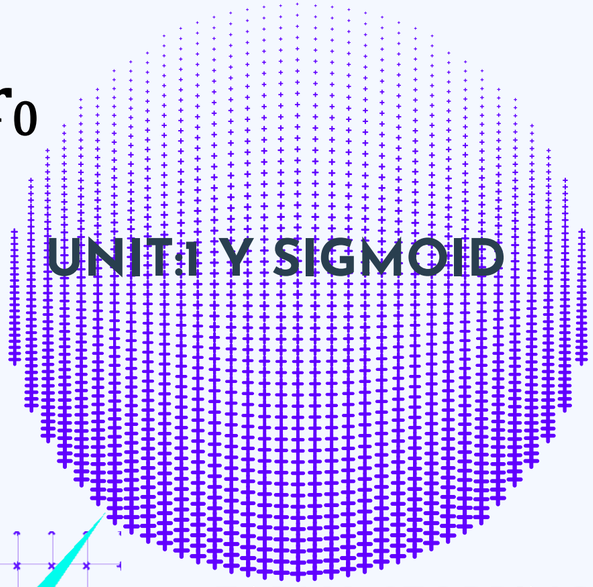




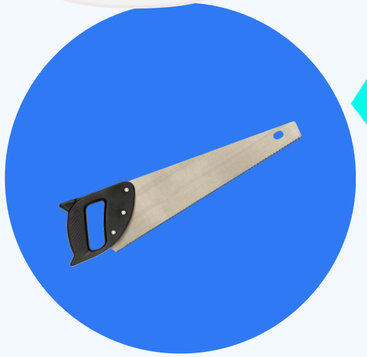
$$\mathbf{r} = \frac{1}{2}\mathbf{a}t^2 + \mathbf{v_0}t + \mathbf{r_0}$$

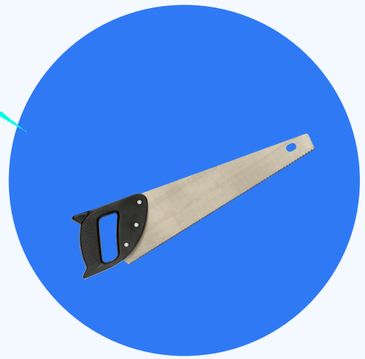


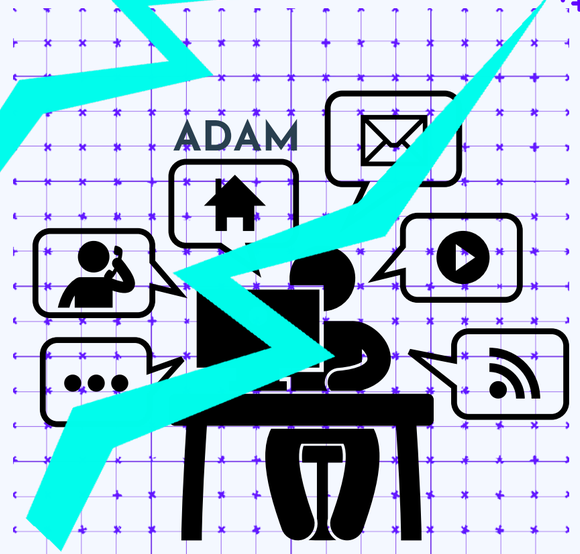
CAPA DE SALIDA



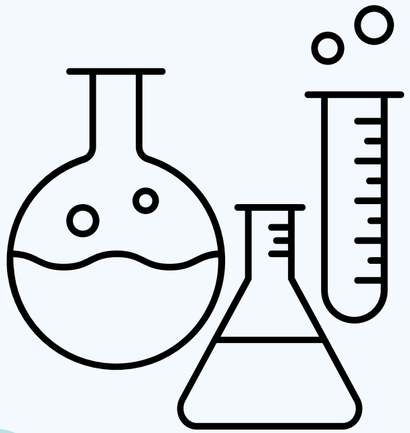
20%







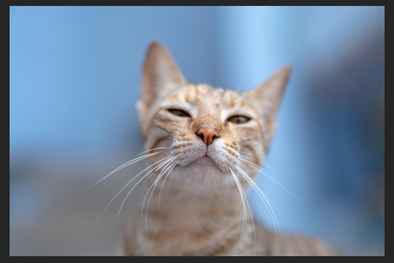
EARLY_STOPPING



REDUCE_LR

START

```python
modelo = tf.keras.Sequential()
# Definir las 4 entradas
modelo.add(tf.keras.layers.Dense(units=64, activation='relu', input_shape=(115,), kernel_regularizer=l2(0.01)))
modelo.add(tf.keras.layers.BatchNormalization())#mejora la tasa de parendizaje, evitando sobreajuste, ya que tb regulariza la funcion

# Capas ocultas
modelo.add(tf.keras.layers.Dense(units=32, activation="relu", input_dim=115))#relu, la mas comun,rapida y eficiente
modelo.add(tf.keras.layers.BatchNormalization())#mejora la tasa de parendizaje, evitando sobreajuste, ya que tb regulariza la funcion

modelo.add(Dropout(0.2))

modelo.add(tf.keras.layers.Dense(units=16, activation="elu", kernel_regularizer=l2(0.01)))
modelo.add(tf.keras.layers.BatchNormalization())

modelo.add(tf.keras.layers.Dense(units=8, activation='elu'))#eLU mejora la convergencia y la velocidad de entrenamiento
modelo.add(tf.keras.layers.BatchNormalization())

# Capa de salida
modelo.add(tf.keras.layers.Dense(units=1, activation='sigmoid'))  # sigmoide para clasificacion binaria
#optimizador
optimizador=Adam( learning_rate=0.01, # rebajo el learning rate por ser mas adecuado para un ajuste fino aunque tarde mas
                  beta_1=0.6, # influye en el gradiente pasado (1 mas al gradiente y 0 nada)
                  beta_2=0.4, #influye en los cuadrados de los gradientes pasados(1 mas a la varianza y 0 nada)
                  epsilon=1e-09, #es para prevenior divisiones entre 0
                  amsgrad=False)#es una variante de Adam y ayuda con la convergencia de ambos a evitar oscilaciones de los pesos durante la optimizacion"""
# Compilar el modelo
modelo.compile(optimizer='adam',
               loss='binary_crossentropy',
               metrics=['accuracy', 'Precision', 'Recall'])
tf.keras.metrics.AUC(name='auc')

#detiene el entrenamiento si la metrica no mejora
early_stopping_callbacks = tf.keras.callbacks.EarlyStopping(patience=5,restore_best_weights=True)

#reduce la tasa de aprendizaje cuando el rendimiento del conj. val no mejora
reduce_lr_callbacks = ReduceLROnPlateau(monitor='val_loss', patience=2, verbose=1)
#Guarda el modelo cuando mejora la métrica de validación
filepath = r'D:\Cursos\REPOSITORIOS\DATASET\malware_total\logs\modelo.h5'
monitor = 'val_loss'
checkpoint_callbacks = ModelCheckpoint(filepath=filepath, monitor=monitor, verbose=1, save_best_only=True)
#visionado_tensorboard
tensorboard_callback = tf.keras.callbacks.TensorBoard(log_dir=log_dir)

callbacks_list = [early_stopping_callbacks, reduce_lr_callbacks, checkpoint_callbacks, tensorboard_callback]
# Entrenar el modelo
historia = modelo.fit(X_train_a, y_train_a, batch_size=32, epochs=10,verbose=1,validation_split=0.1,callbacks=callbacks_list)
```



modelo de red neuronal 1

Adam y desbalaceada

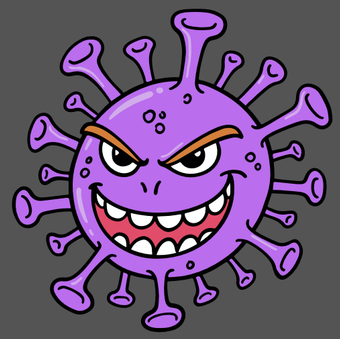
RESULTADO ADAM DESBALANCEADO

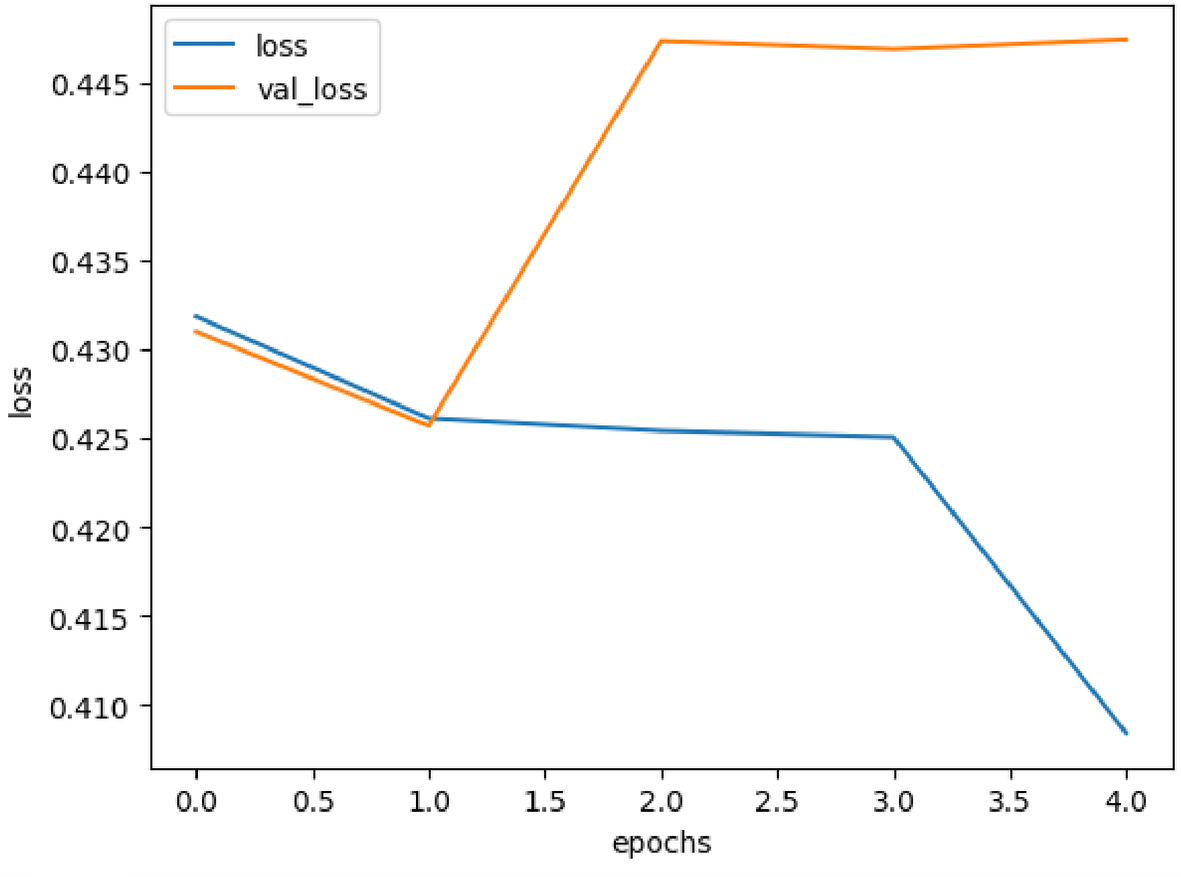
```
Loss: 0.42553311586380005
Accuracy: 0.8055193424224854
precision: 0.7378716468811035
recall: 0.9966943264007568
94886/94886 [==============================] - 75s 787u
c:\Users\victo\anaconda3\envs\tf-gpu\lib\site-packages\
  _warn_prf(average, modifier, msg_start, len(result))
c:\Users\victo\anaconda3\envs\tf-gpu\lib\site-packages\
  _warn_prf(average, modifier, msg_start, len(result))
              precision    recall  f1-score   support

         0.0       0.46      1.00      0.63   1384008
         1.0       0.00      0.00      0.00   1652320

    accuracy                           0.46   3036328
   macro avg       0.23      0.50      0.31   3036328
weighted avg       0.21      0.46      0.29   3036328
```

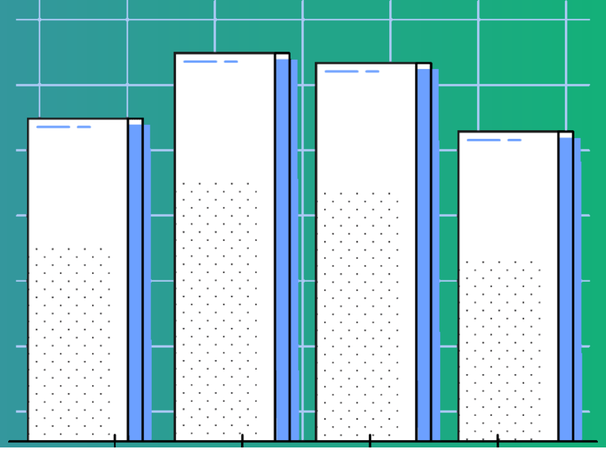
PRUEBA DE VALIDACION

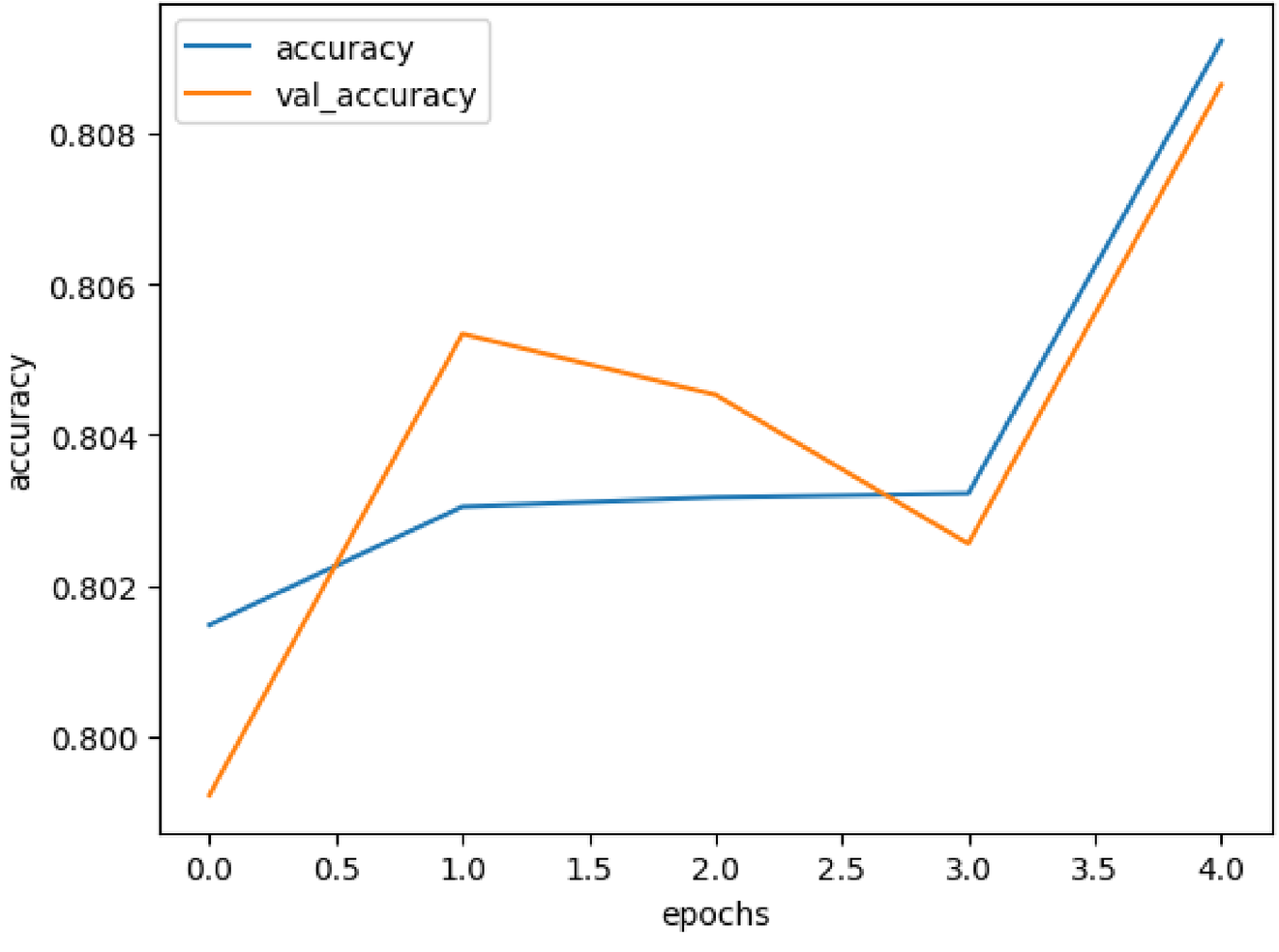
```
# Evaluar el modelo en el conjunto de prueba
loss, accuracy, precision, recall = modelo.evaluate(X_val_a, y_val_
print("Loss:", loss)
print("Accuracy:", accuracy)
print("precision:", precision)
print("recall:", recall)


# Predicción
y_pred = modelo.predict(X_val_a)
y_pred_classes = np.argmax(y_pred, axis=1)

# Métricas de clasificación
print(classification_report(y_val_a, y_pred_classes))
```

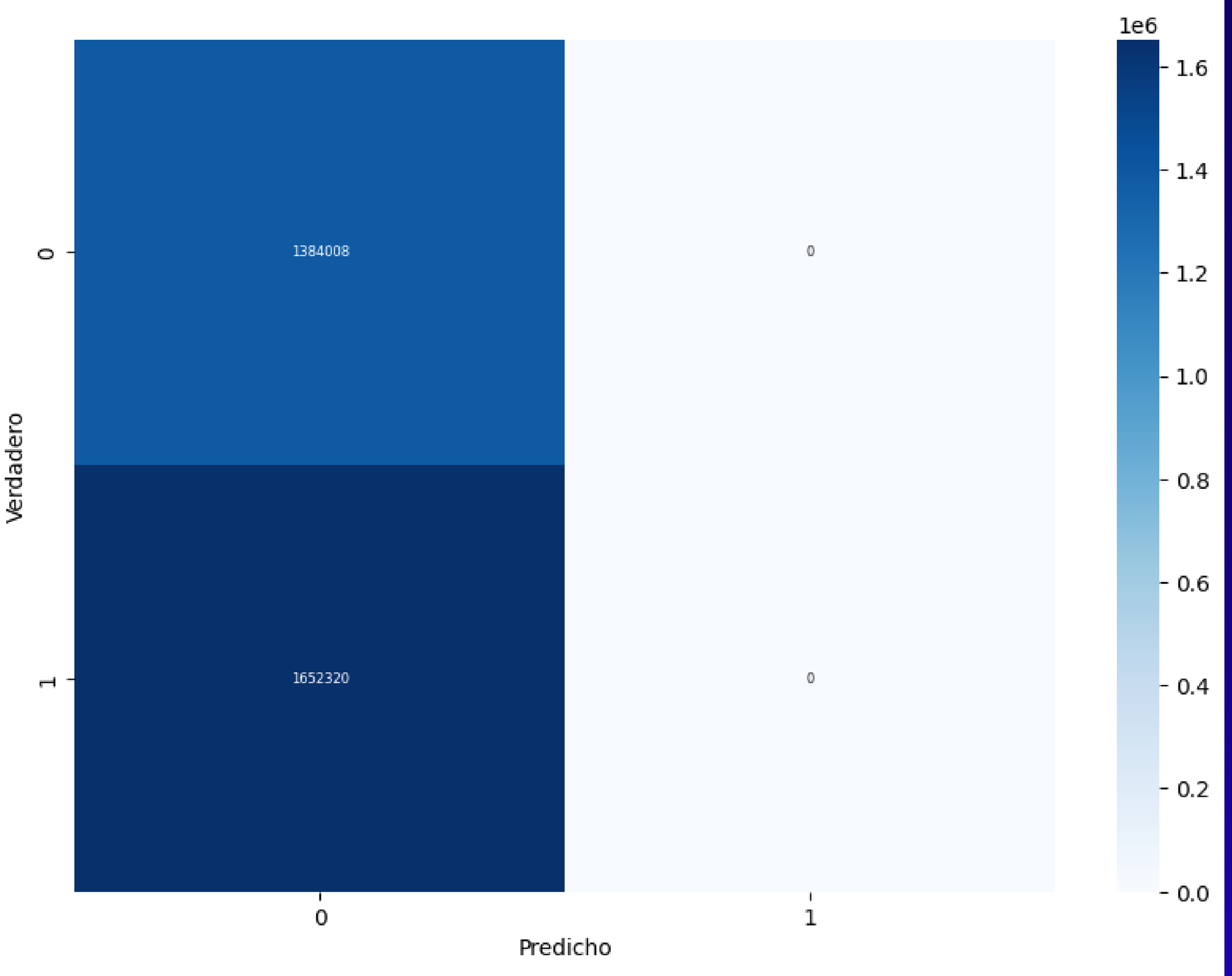
un 46% de Malware acertado de forma certera




loss
val_loss
0.445
0.440
0.435
0.430
0.425
0.420
0.415
0.410
loss
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0
epochs

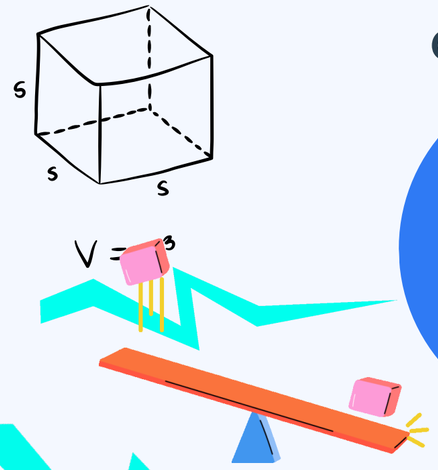



accuracy
val_accuracy
0.808
0.806
0.804
0.802
0.800
accuracy
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0
epochs

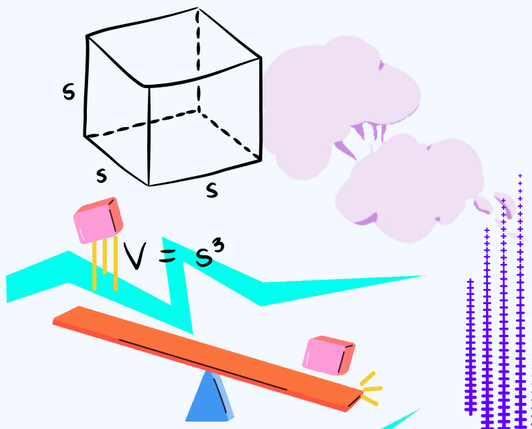

1e6
1.6
1.4
1.2
1.0
0.8
0.6
0.4
0.2
0.0
1384008
0
1652320
0
0
1
Verdadero
Predicho

MI MODELO DE RED NEURONAL 2

CAPA DE ENTRADA



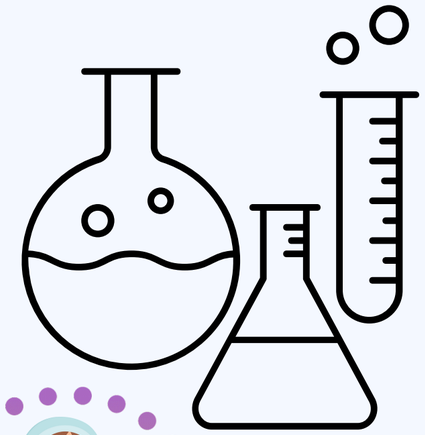
CAPAS OCULTAS



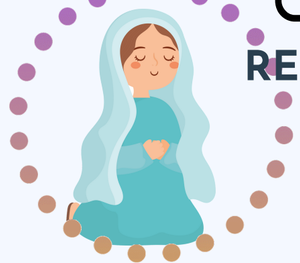
CAPA DE SALIDA

UNIT 1 Y SIGMOID

EARLY_STOPPING



REDUCE_LR



TODAS LAS CAPAS
TIENEN 64 UNITS

SGD

START

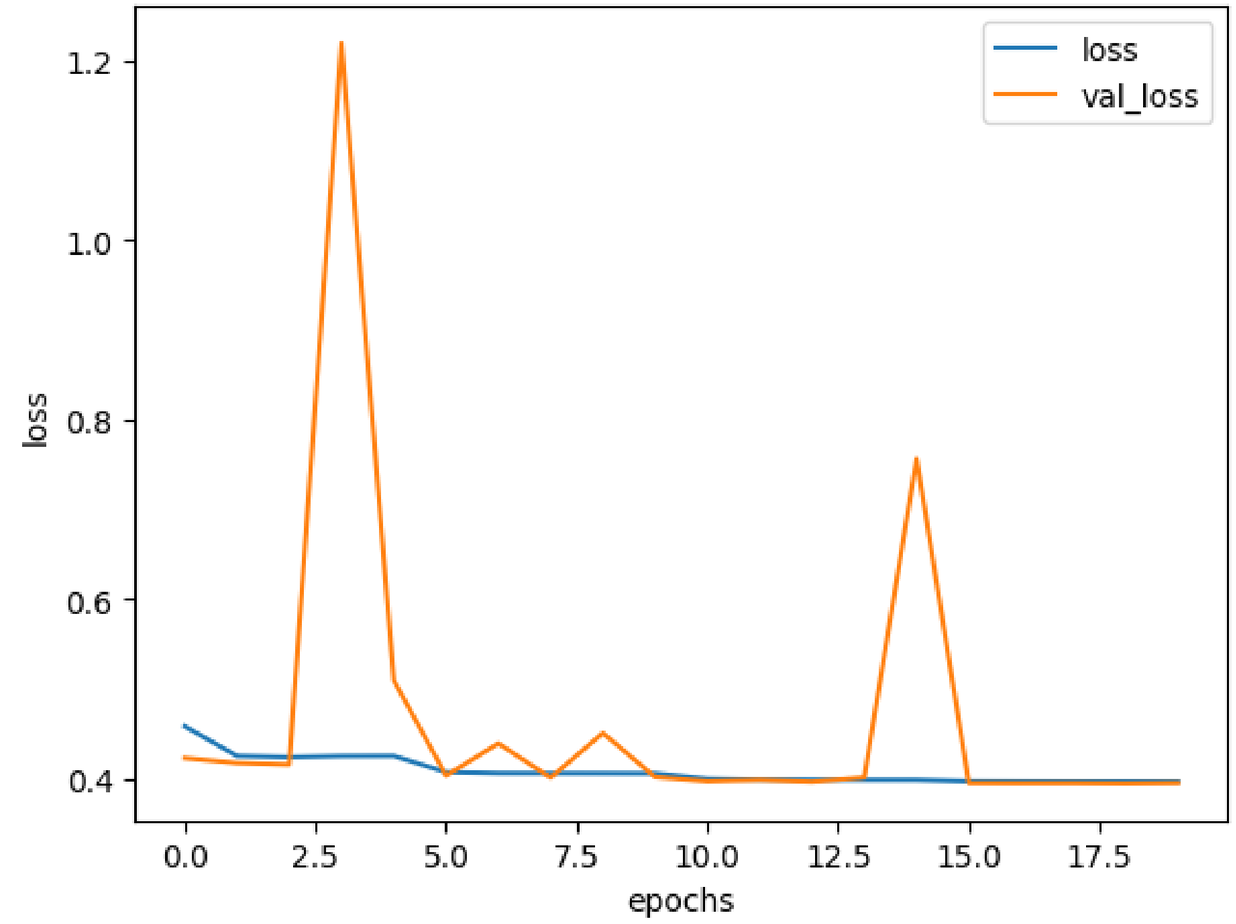
segundo modelo de red neuronal

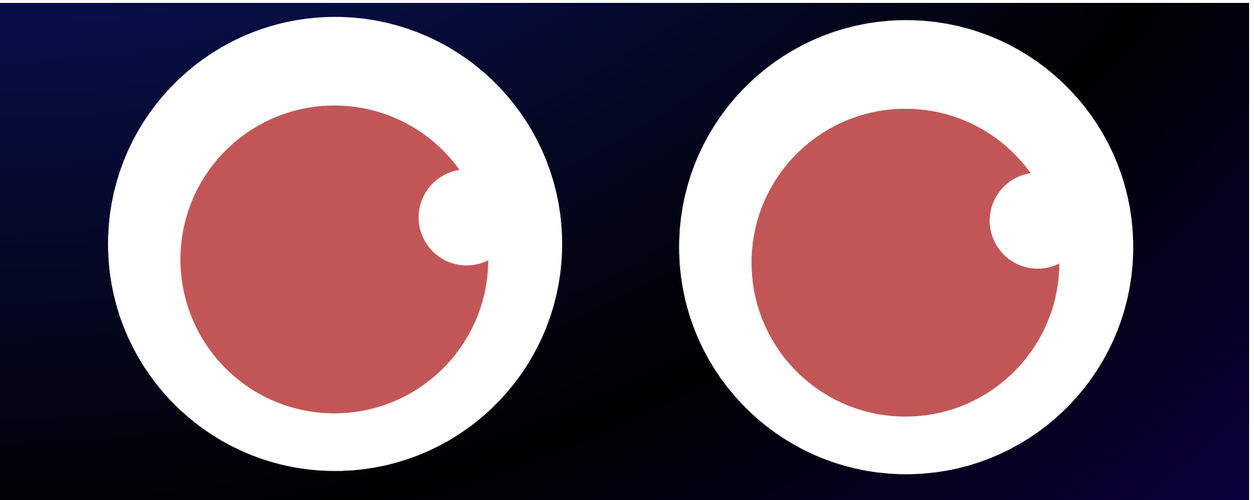
```
modelo = tf.keras.Sequential()
# Definir las 4 entradas

modelo.add(tf.keras.layers.Dense(units=64, activation='relu', input_shape=(115,), kernel_regularizer=l2(0.01)))
modelo.add(tf.keras.layers.BatchNormalization())#mejora la tasa de parendizaje, evitando sobreajuste, ya que tb regulariza la funcion

# Capas ocultas
modelo.add(tf.keras.layers.Dense(units=64, activation="relu", input_dim=115))#relu, la mas comun,rapida y eficiente
modelo.add(tf.keras.layers.BatchNormalization())#mejora la tasa de parendizaje, evitando sobreajuste, ya que tb regulariza la funcion

modelo.add(Dropout(0.2))

modelo.add(tf.keras.layers.Dense(units=64, activation="elu", kernel_regularizer=l2(0.01)))
modelo.add(tf.keras.layers.BatchNormalization())

modelo.add(tf.keras.layers.Dense(units=64, activation='elu'))#eLU mejora la convergencia y la velocidad de entrenamiento
modelo.add(tf.keras.layers.BatchNormalization())

# Capa de salida
modelo.add(tf.keras.layers.Dense(units=1, activation='sigmoid')) # sigmoide para clasificacion binaria
#optimizador
optimizador= tf.keras.optimizers.SGD(learning_rate=0.005,
                                     momentum=0.9, #aporta inercia al proceso de optimizacion, acelerando la convergencia
                                     nesterov=True)# en True actualiza momentum demanera anticipada al gradiente, mejorando la estaboiilidad

# Compilar el modelo
modelo.compile(optimizer='SGD',
               loss='binary_crossentropy',
               metrics=['accuracy', 'Precision', 'Recall'])
tf.keras.metrics.AUC(name='auc')

#detiene el entrenamiento si la metrica no mejora
early_stopping_callbacks = tf.keras.callbacks.EarlyStopping(patience=5,restore_best_weights=True)

#reduce la tasa de aprendizaje cuando el rendimiento del conj. val no mejora
reduce_lr_callbacks = ReduceLROnPlateau(monitor='val_loss', patience=2, verbose=1)

#Guarda el modelo cuando mejora la métrica de validación
filepath = r'D:\Cursos\REPOSITORIOS\DATASET\malware_total\logs\modelo_SGD.h5'
monitor = 'val_loss'
checkpoint_callbacks = ModelCheckpoint(filepath=filepath, monitor=monitor, verbose=1, save_best_only=True)

#visionado_tensorboard
tensorboard_callback = tf.keras.callbacks.TensorBoard(log_dir=log_dir)
```

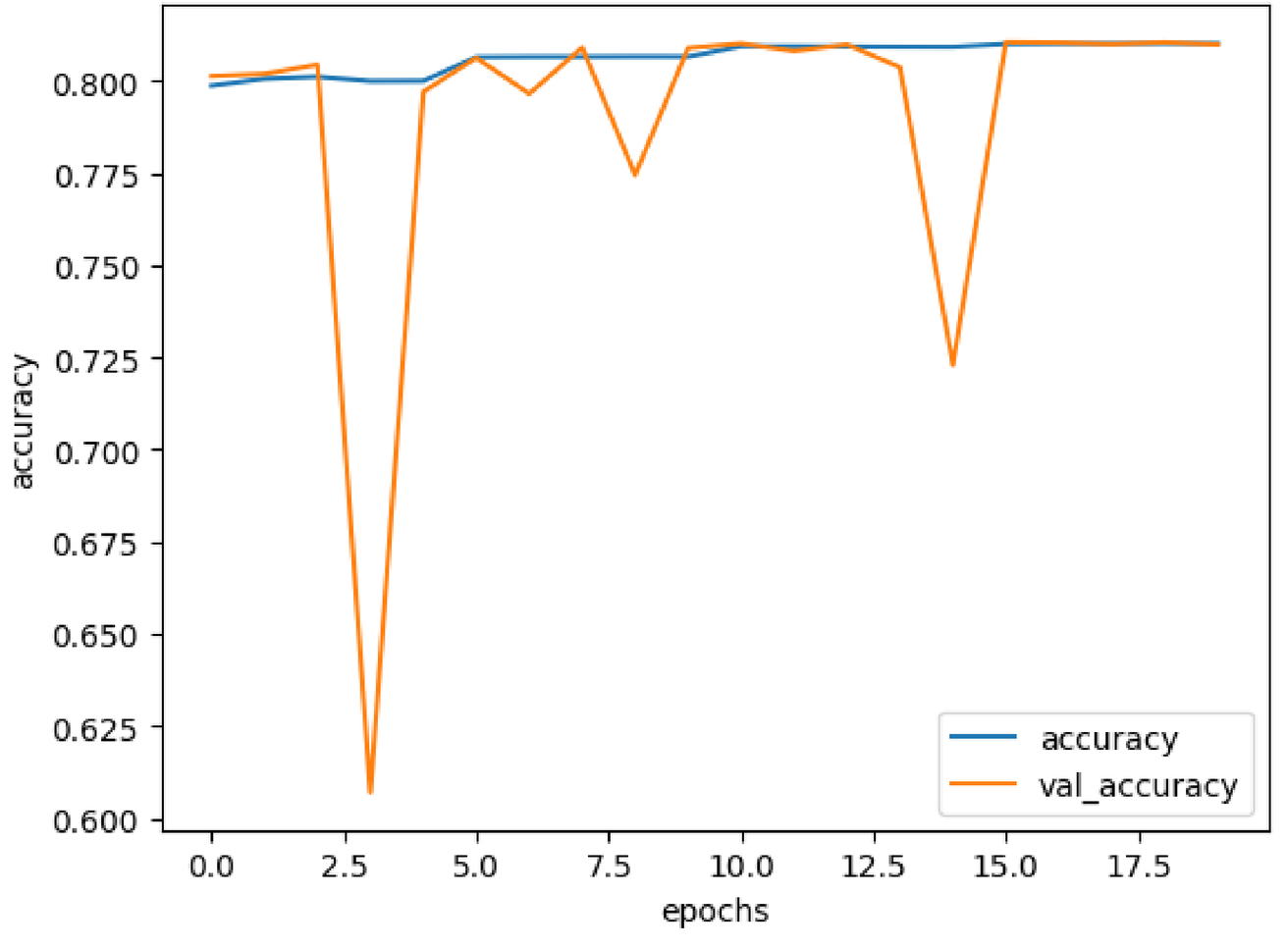
modelo de red neuronal 2
SGD y desbalaceada

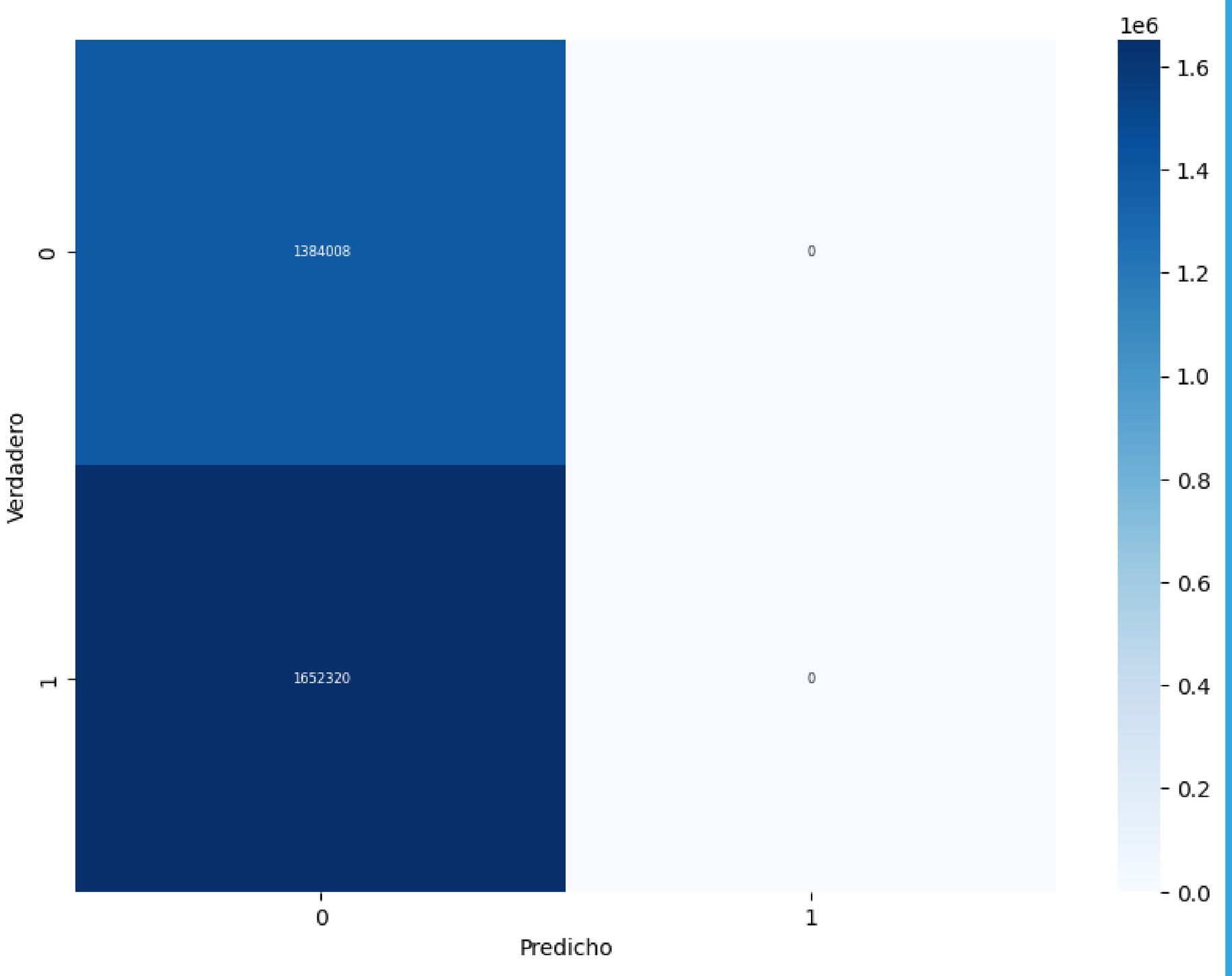
Igual que con optimizador Adam y con distinta configuracion de neuronasl

```
94886/94886 [==============================] - 95s 997us
Loss: 0.40602222084999084
Accuracy: 0.8063921332359314
precision: 0.7384393215179443
recall: 0.9974405765533447
94886/94886 [==============================] - 74s 777us
c:\Users\victo\anaconda3\envs\tf-gpu\lib\site-packages\s
  _warn_prf(average, modifier, f"{metric.capitalize()} i
c:\Users\victo\anaconda3\envs\tf-gpu\lib\site-packages\s
  _warn_prf(average, modifier, f"{metric.capitalize()} i
              precision    recall  f1-score   support

         0.0       0.46      1.00      0.63   1384396
         1.0       0.00      0.00      0.00   1651932

    accuracy                           0.46   3036328
   macro avg       0.23      0.50      0.31   3036328
weighted avg       0.21      0.46      0.29   3036328
```


loss
val_loss
1.2
1.0
0.8
0.6
0.4
loss
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
epochs



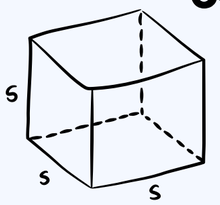

0.800
0.775
0.750
0.725
0.700
0.675
0.650
0.625
0.600
accuracy
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
epochs
accuracy
val_accuracy

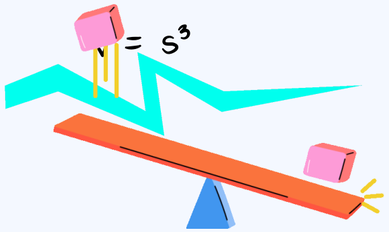

1e6
1.6
1.4
1.2
1.0
0.8
0.6
0.4
0.2
0.0
1384008
0
1652320
0
0
1
Verdadero
Predicho

MI MODELO DE RED NEURONAL RESAMPLING 1

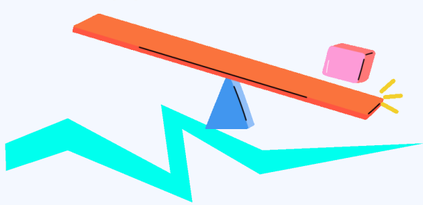
CAPAS OCULTAS
64-32-16-8 Y L2 1 CAPA Y 2 DENSA

CAPA DE ENTRADA
64 UNITS

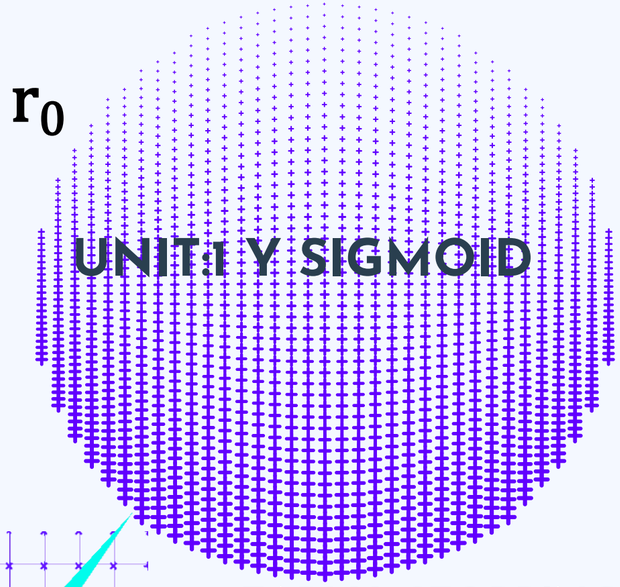




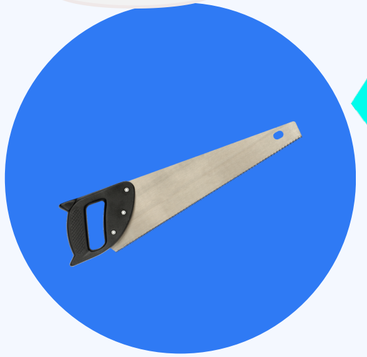
$$\mathbf{r} = \frac{1}{2}\mathbf{a}t^2 + \mathbf{v_0}t + \mathbf{r_0}$$

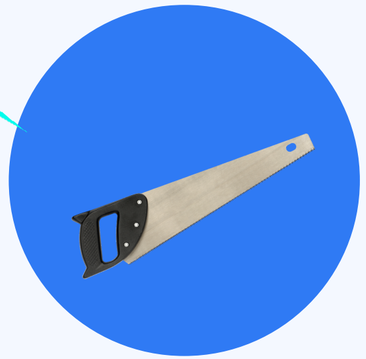


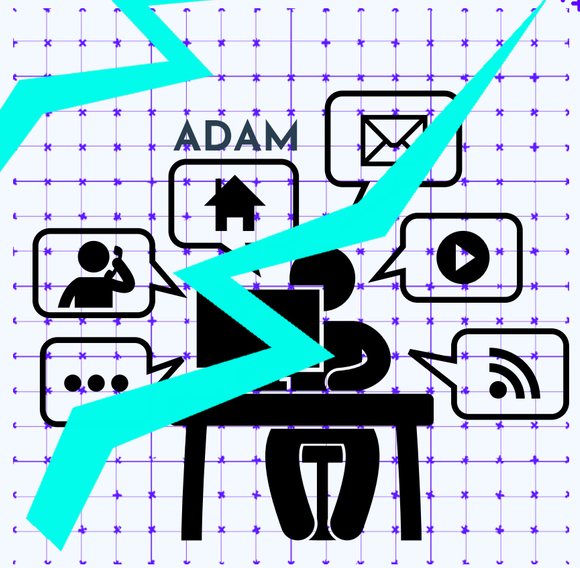
CAPA DE SALIDA



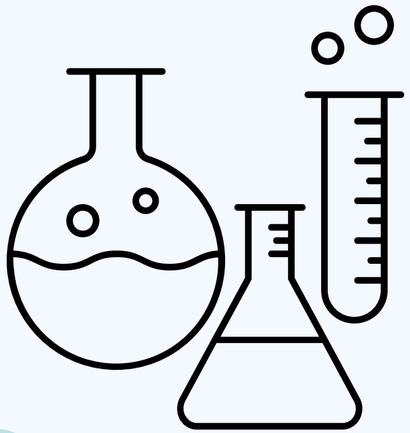
20%







EARLY_STOPPING



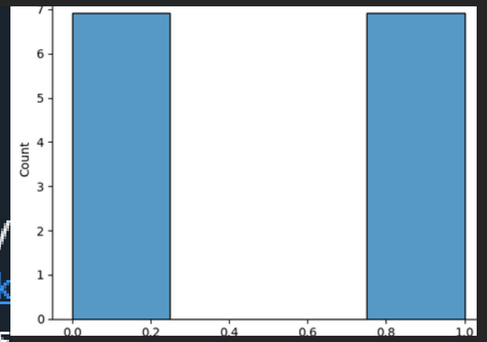
REDUCE_LR

START

modelo de red neuronal 1

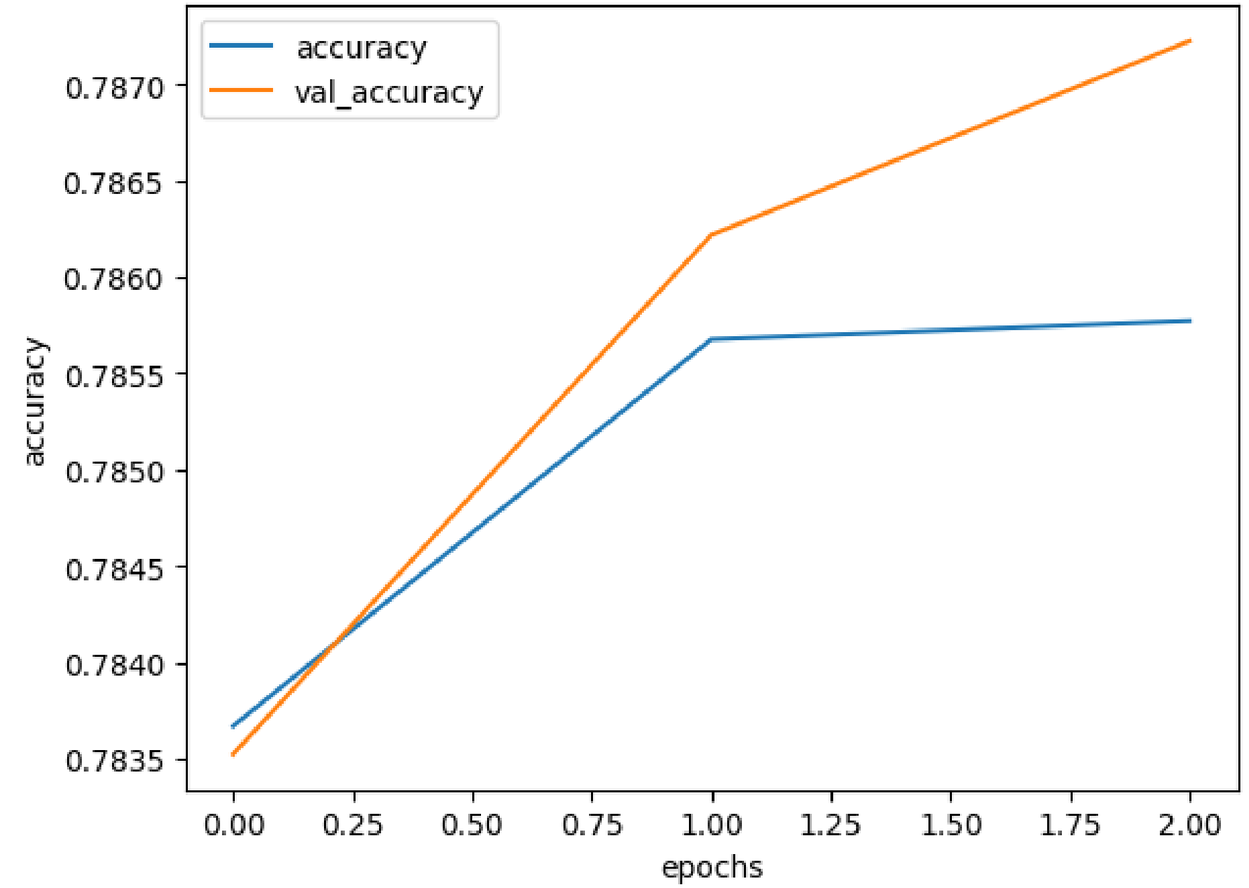
```
# Aplica submuestro, para reducir el número de observaciones de todas las clases menos la clase minoritaria
X=df_rn.drop(["Target"], axis=1)
y=df_rn["Target"]

rus = RandomUnderSampler(random_state=42)
X_resampled, y_resampled = rus.fit_resample(X, y)


X_train, X_temp, y_train, y_temp = train_test_split(X_resampled, y_resampled, test_size=0.2, random_state=4
X_test, X_val, y_test, y_val = train_test_split(X_temp, y_temp, test_size=0.25, random_state=42)
```

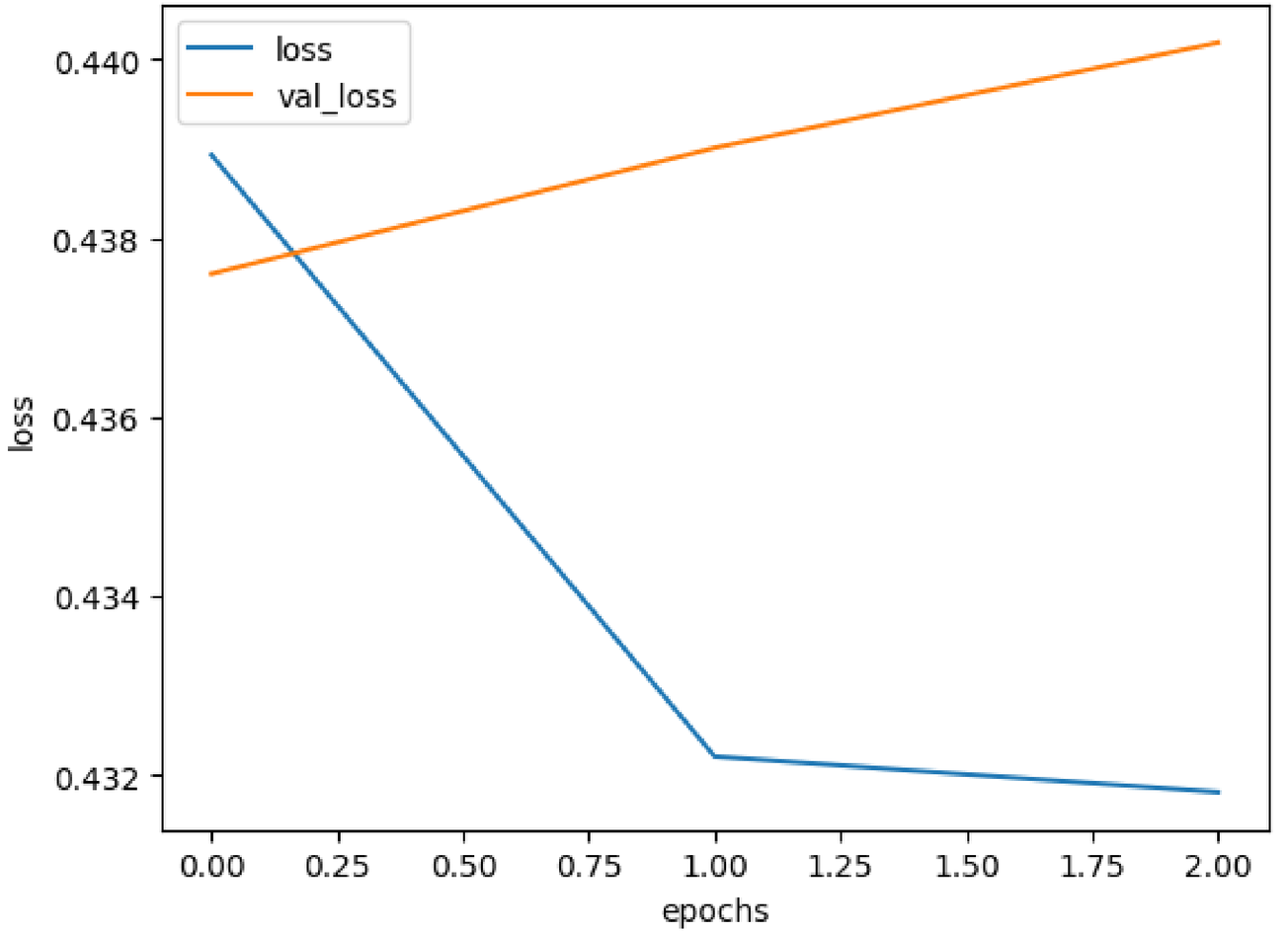
```
Loss: 0.43704232573509216
Accuracy: 0.7845828533172607
precision: 0.7084441184997559
recall: 0.9675958752632141
86548/86548 [==============================] - 63s 731us/
c:\Users\victo\anaconda3\envs\tf-gpu\lib\site-packages\sk
  _warn_prf(average, modifier, f"{metric.capitalize()} is
c:\Users\victo\anaconda3\envs\tf-gpu\lib\site-packages\sk
  _warn_prf(average, modifier, f"{metric.capitalize()} is
              precision    recall  f1-score   support

         0.0       0.50      1.00      0.67   1384044
         1.0       0.00      0.00      0.00   1385471

    accuracy                           0.50   2769515
   macro avg       0.25      0.50      0.33   2769515
weighted avg       0.25      0.50      0.33   2769515
```

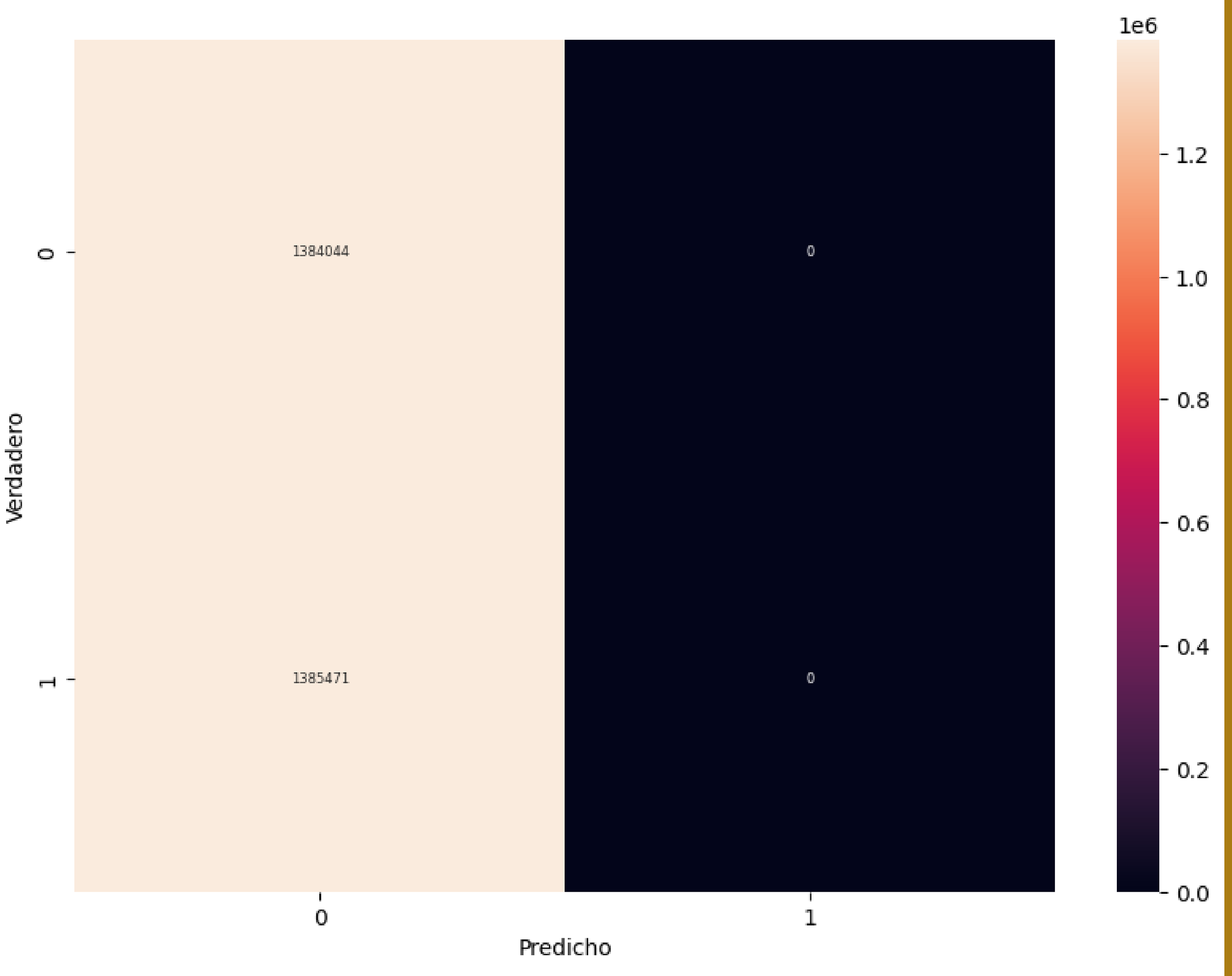


GAME ON

50% ACIERTOS CERTEROS

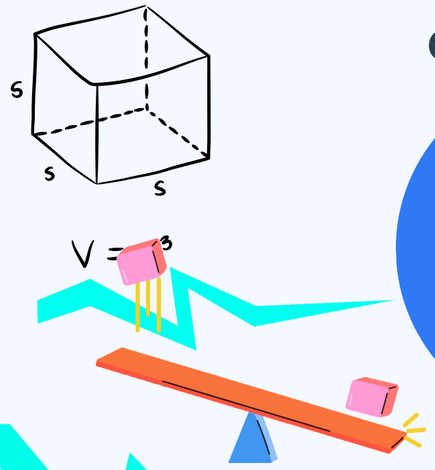

accuracy
val_accuracy
0.7870
0.7865
0.7860
0.7855
0.7850
0.7845
0.7840
0.7835
accuracy
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
2.00
epochs


loss
val_loss
0.440
0.438
0.436
0.434
0.432
loss
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
2.00
epochs

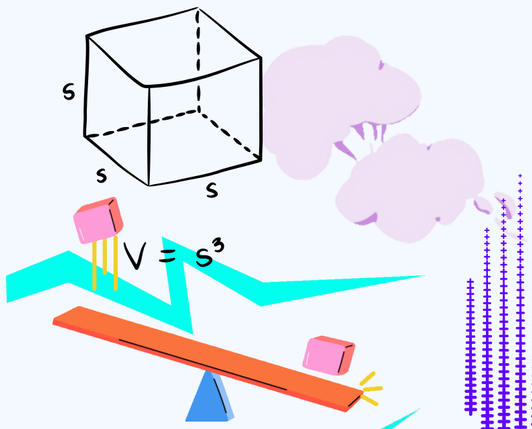

1e6
1.2
1.0
0.8
0.6
0.4
0.2
0.0
1384044
0
1385471
0
0
1
Verdadero
Predicho

MI MODELO DE RED NEURONAL RESAMPLING 2

CAPA DE ENTRADA



CAPAS OCULTAS



CAPA DE SALIDA

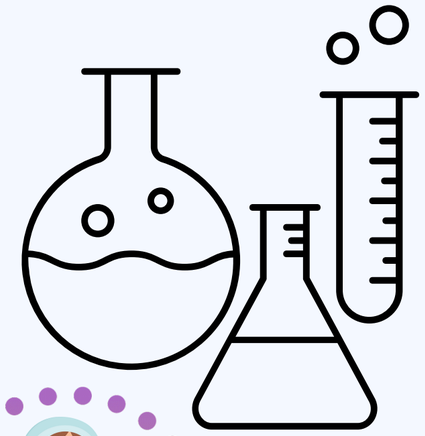
UNIT 1 Y SIGMOID

TODAS LAS CAPAS
TIENEN 64 UNITS

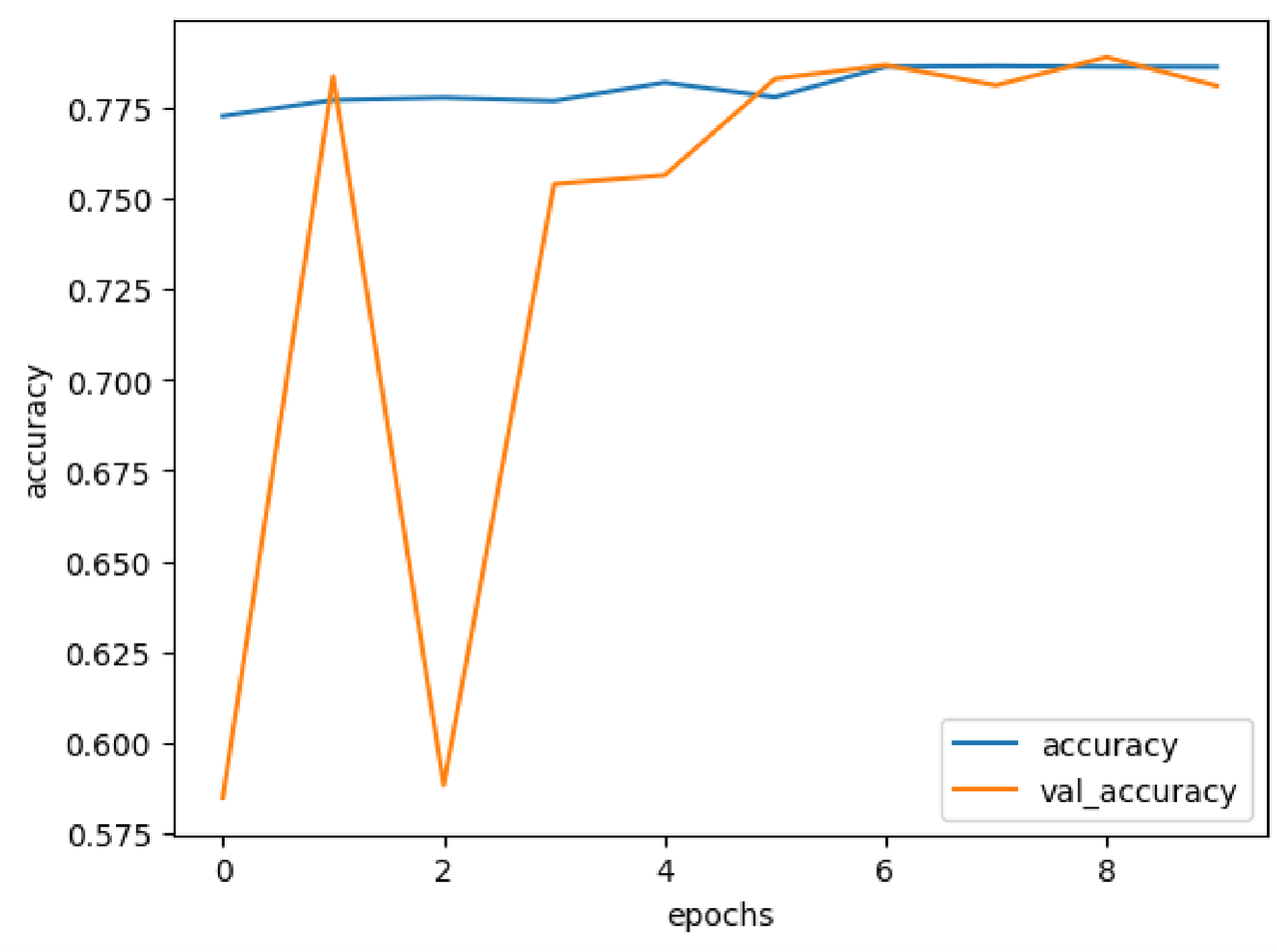
SGD

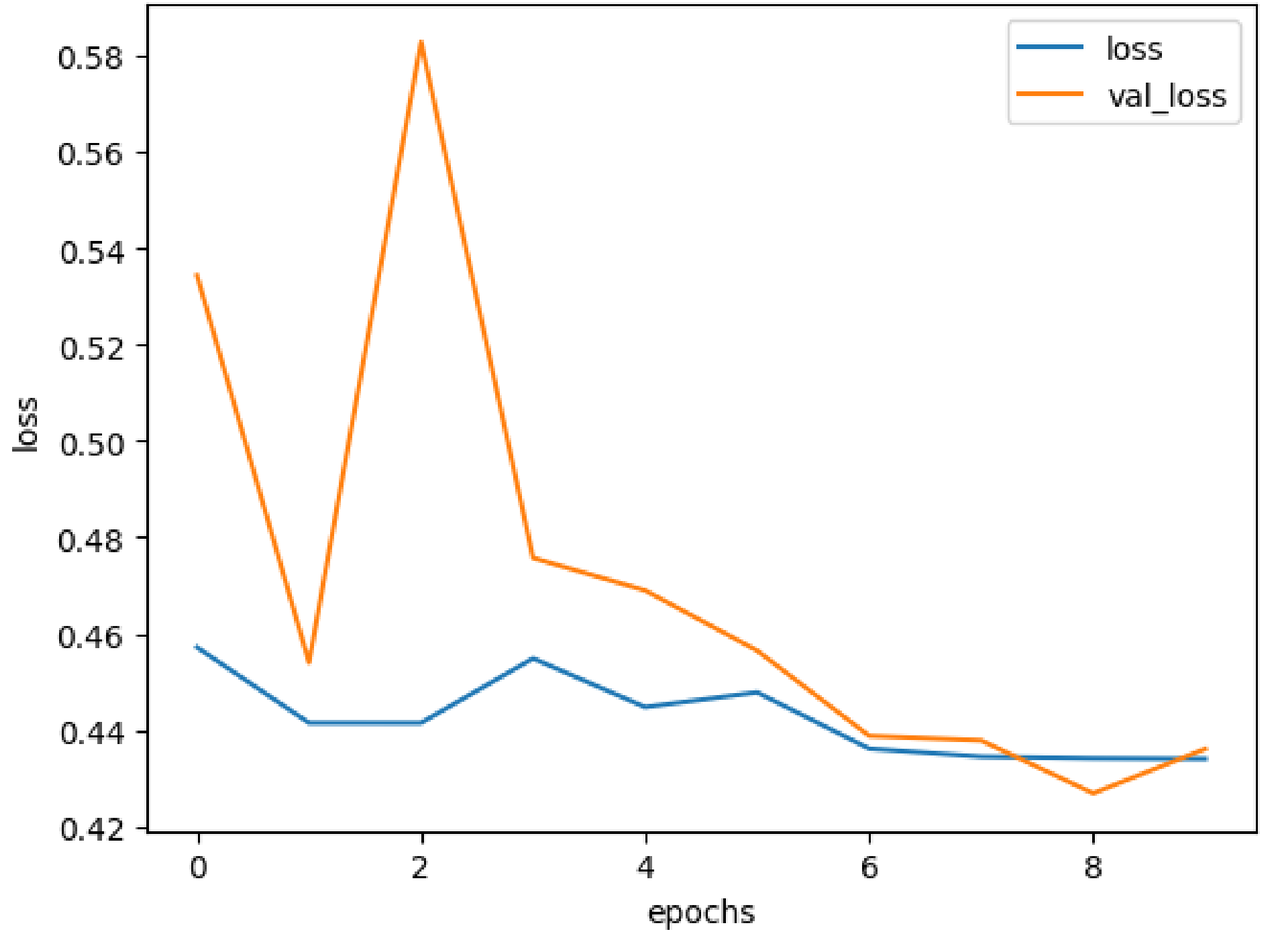
START

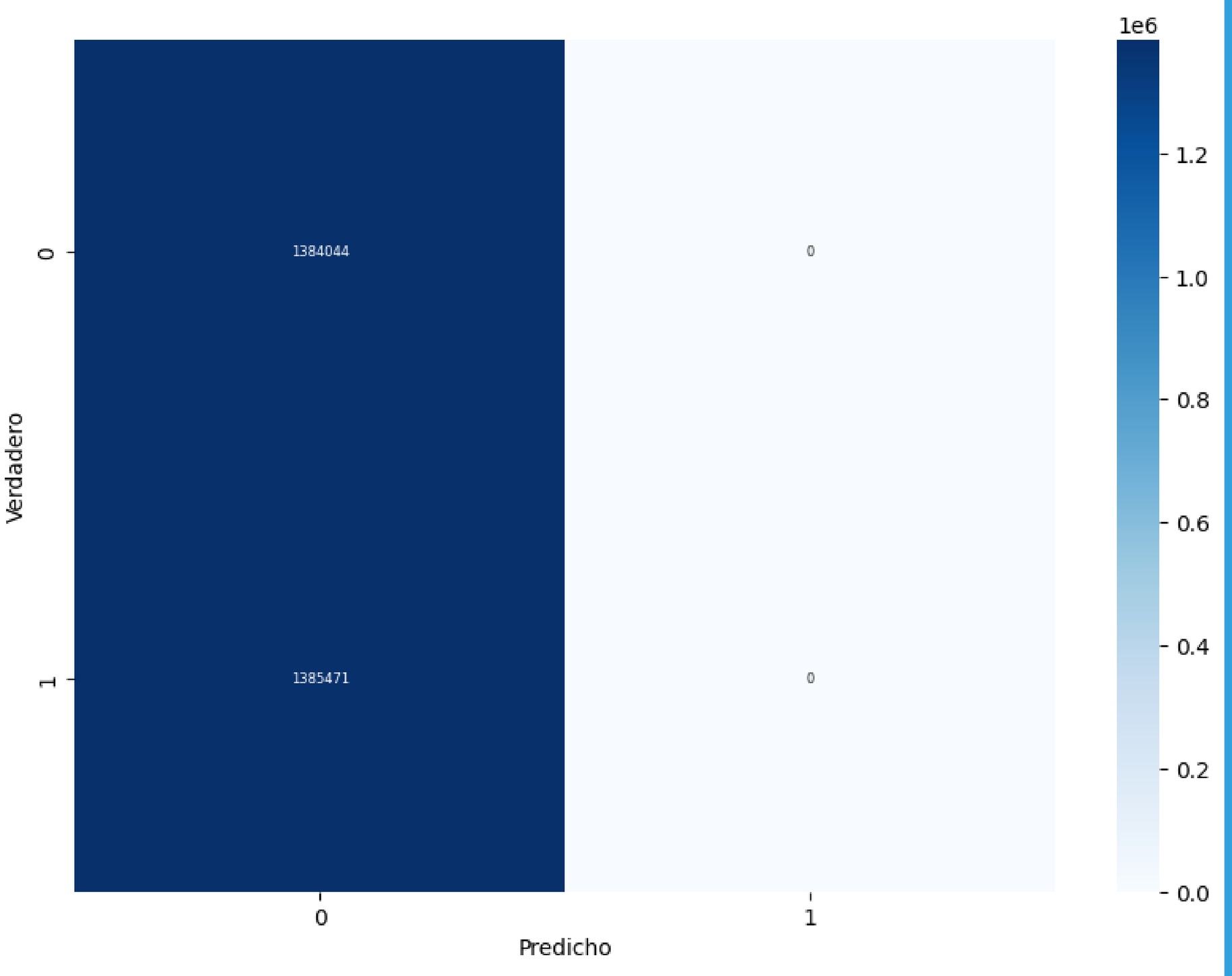
EARLY_STOPPING



```
Loss: 0.4370423257350921
Accuracy: 0.7845828533172607
precision: 0.7084441184997559
recall: 0.9675958752632141
86548/86548 [==============================] - 63s 731us/step
c:\Users\victo\anaconda3\envs\tf-gpu\lib\site-packages\sklearn
  _warn_prf(average, modifier, f"{metric.capitalize()} is", le
c:\Users\victo\anaconda3\envs\tf-gpu\lib\site-packages\sklearn
  _warn_prf(average, modifier, f"{metric.capitalize()} is", le
              precision    recall  f1-score   support

         0.0       0.50      1.00      0.67   1384044
         1.0       0.00      0.00      0.00   1385471

    accuracy                           0.50   2769515
   macro avg       0.25      0.50      0.33   2769515
weighted avg       0.25      0.50      0.33   2769515
```


accuracy
0.775
0.750
0.725
0.700
0.675
0.650
0.625
0.600
0.575
0
2
4
6
8
epochs
accuracy
val_accuracy


loss
0.58
0.56
0.54
0.52
0.50
0.48
0.46
0.44
0.42
0
2
4
6
8
epochs
loss
val_loss


1e6
1.2
1.0
0.8
0.6
0.4
0.2
0.0
1384044
0
1385471
0
0
1
Verdadero
Predicho

CONCLUSIONES FINALES

El modelo que mejor se adapta a nuestro objetivo de negocio es el modelo RandonForest en multiproceso sin resampling con un 58% de aciertos en la clase Malicious, aunque falle en reconocer los archivos benignos en un 25%. Los modelos de DL el mejor ha sido con resampling consiguiendo un 50% de verdaderos positivos sin fallos , lo que ocurre que no podemos valorar la otra clase al no ser reconocida. ,

En resumen, , un casi 60% de positivos verdaderos totalmente fiables, es una buena proporción para nuestro objetivo, ahora le voy a pasar el test de prueba a este modelo para su ultima comrobacion

TEST DE VALIDACION

```python
# Entrenar el modelo
with parallel_backend('threading'):
    model.fit(X_train, y_train)

# Evaluar el modelo
y_pred = model.predict(X_val)
accuracy = accuracy_score(y_val, y_pred)

print(classification_report(y_pred, y_val))
```

	precision	recall	f1-score	support
0	0.58	1.00	0.74	808056
1	1.00	0.74	0.85	2228272
accuracy			0.81	3036328
macro avg	0.79	0.87	0.79	3036328
weighted avg	0.89	0.81	0.82	3036328

TEST DE PRUEBA

```python
with parallel_backend('threading'):
    model.fit(X_train, y_train)

# Evaluar el modelo
y_pred = model.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)

print(classification_report(y_pred, y_test))
```

✓ 4m 28.7s

	precision	recall	f1-score	support
0	0.58	1.00	0.74	808574
1	1.00	0.74	0.85	2227754
accuracy			0.81	3036328
macro avg	0.79	0.87	0.79	3036328
weighted avg	0.89	0.81	0.82	3036328

finally

FINISHED

THANK YOU SO MUCH