```
# Gerardo Herrera... ann: (1 capa oculta con 15 neuronas, activation = 'relu', epoch=325) con


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

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mour
```

```
# This Python 3 environment comes with many helpful analytics libraries installed
# It is defined by the kaggle/python Docker image: https://github.com/kaggle/docker-python
# For example, here's several helpful packages to load

import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)

# Input data files are available in the read-only "../input/" directory
# For example, running this (by clicking run or pressing Shift+Enter) will list all files und

import os
for dirname, _, filenames in os.walk('/kaggle/input'):
    for filename in filenames:
        print(os.path.join(dirname, filename))

# You can write up to 5GB to the current directory (/kaggle/working/) that gets preserved as
# You can also write temporary files to /kaggle/temp/, but they won't be saved outside of the


# https://medium.com/@randerson112358/build-your-own-artificial-neural-network-using-python-f


#Load libraries
from keras.models import Sequential
from keras.layers import Dense
import pandas as pd
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt
plt.style.use('fivethirtyeight')


import time


#sensor77 = pd.read_csv('../input/vombas/sensor_procesado.csv')
#sensor77 = pd.read_csv('../input/10ks25/s25balanced10k.csv')
#sensor77 = pd.read_csv('../input/28k-s24-balan-vombas/sensor2-ordenado_status_sin_broken_bal

sensor77 = pd.read_csv('/content/drive/My Drive/datasets/sensor2-ordenado_status_sin_broken_b
```

```
#Show the shape (number of rows & columns)
sensor77.shape
```

```
(28002, 27)
```

```
#Show the number of missing (NAN, NaN, na) data for each column
sensor77.isnull().sum()
```

```
Unnamed: 0           0
timestamp            0
sensor_00            0
sensor_01           30
sensor_02            0
sensor_03            0
sensor_04            0
sensor_11            0
sensor_14            0
sensor_16            0
sensor_17            0
sensor_18            0
sensor_19            0
sensor_20            0
sensor_21            0
sensor_22            0
sensor_23            0
sensor_25            0
sensor_26            0
sensor_27            0
sensor_28            0
sensor_30            0
sensor_31            0
sensor_44            3
sensor_50        14004
sensor_51         2996
machine_status       0
dtype: int64
```

```
cleanup_nums = {"machine_status":     {"NORMAL": 0, "RECOVERING": 1,"BROKEN": 2}}
```

```
sensor77.replace(cleanup_nums, inplace=True)
```

```
sensor77.fillna(sensor77.mean(), inplace=True)
```

```
#Show the number of missing (NAN, NaN, na) data for each column
sensor77.isnull().sum()
```

```
Unnamed: 0       0
timestamp        0
sensor_00        0
sensor_01        0
sensor_02        0
```

```
sensor_03         0
sensor_04         0
sensor_11         0
sensor_14         0
sensor_16         0
sensor_17         0
sensor_18         0
sensor_19         0
sensor_20         0
sensor_21         0
sensor_22         0
sensor_23         0
sensor_25         0
sensor_26         0
sensor_27         0
sensor_28         0
sensor_30         0
sensor_31         0
sensor_44         0
sensor_50         0
sensor_51         0
machine_status    0
dtype: int64
```

```
#sensor77.drop('sensor_15', axis=1, inplace=True)
sensor77.drop('timestamp', axis=1, inplace=True)
```

```
#sensor77.drop('100000', axis=1, inplace=True)
```

```
sensor77.drop('Unnamed: 0', axis=1, inplace=True)
```

```
sensor77.isnull().sum()
```

```
sensor_00         0
sensor_01         0
sensor_02         0
sensor_03         0
sensor_04         0
sensor_11         0
sensor_14         0
sensor_16         0
sensor_17         0
sensor_18         0
sensor_19         0
sensor_20         0
sensor_21         0
sensor_22         0
sensor_23         0
sensor_25         0
sensor_26         0
sensor_27         0
sensor_28         0
sensor_30         0
```

```
sensor_31         0
sensor_44         0
sensor_50         0
sensor_51         0
machine_status    0
dtype: int64
```

```
#Convert the data into an array
dataset = sensor77.values
dataset
```

```
array([[2.46539400e+00, 4.70920100e+01, 5.32118000e+01, ...,
        4.81174107e+02, 1.77951400e+02, 0.00000000e+00],
       [2.46539400e+00, 4.70920100e+01, 5.32118000e+01, ...,
        4.81174107e+02, 1.78530100e+02, 0.00000000e+00],
       [2.44473400e+00, 4.73524300e+01, 5.32118000e+01, ...,
        4.81174107e+02, 1.77662000e+05, 0.00000000e+00],
       ...,
       [2.40538200e+00, 4.95659714e+01, 5.38194400e+01, ...,
        3.21180573e+01, 3.15393524e+01, 1.00000000e+00],
       [2.40046300e+00, 4.95659700e+01, 5.37760400e+01, ...,
        3.21180573e+01, 3.15393500e+01, 1.00000000e+00],
       [2.40144700e+00, 4.95225700e+01, 5.37760391e+01, ...,
        3.21180573e+01, 3.18287000e+01, 1.00000000e+00]])
```

```
sensor77.shape
```

```
(28002, 25)
```

```
# Get all of the rows from the first eight columns of the dataset
#X = dataset[:,0:51]
X = dataset[:,0:24]
# Get all of the rows from the last column
#y = dataset[:,51]
y = dataset[:,24]
```

```
print(y)
```

```
[0. 0. 0. ... 1. 1. 1.]
```

```
print(X)
```

```
[[2.46539400e+00 4.70920100e+01 5.32118000e+01 ... 4.36921300e+01
  4.81174107e+02 1.77951400e+02]
 [2.46539400e+00 4.70920100e+01 5.32118000e+01 ... 4.45601800e+01
  4.81174107e+02 1.78530100e+02]
 [2.44473400e+00 4.73524300e+01 5.32118000e+01 ... 4.60069400e+01
  4.81174107e+02 1.77662000e+05]
 ...
 [2.40538200e+00 4.95659714e+01 5.38194400e+01 ... 3.15393524e+01
  3.21180573e+01 3.15393524e+01]
```

```
       [2.40046300e+00 4.95659700e+01 5.37760400e+01 ... 3.15393524e+01
        3.21180573e+01 3.15393500e+01]
       [2.40144700e+00 4.95225700e+01 5.37760391e+01 ... 3.15393524e+01
        3.21180573e+01 3.18287000e+01]]
```

```
from sklearn import preprocessing
min_max_scaler = preprocessing.MinMaxScaler()
X_scale = min_max_scaler.fit_transform(X)
X_scale
```

```
     array([[1.16018541e-03, 1.51254935e-04, 2.97595181e-04, ...,
             7.32072243e-05, 1.01425222e-03, 3.84471811e-04],
            [1.16018541e-03, 1.51254935e-04, 2.97595181e-04, ...,
             7.67494867e-05, 1.01425222e-03, 3.85953388e-04],
            [1.15046306e-03, 1.56160565e-04, 2.97595181e-04, ...,
             8.26532983e-05, 1.01425222e-03, 4.54775949e-01],
            ...,
            [1.13194447e-03, 1.97857886e-04, 3.09041171e-04, ...,
             2.36152335e-05, 1.03499268e-05, 9.63031373e-06],
            [1.12962965e-03, 1.97857860e-04, 3.08223654e-04, ...,
             2.36152335e-05, 1.03499268e-05, 9.63030754e-06],
            [1.13009271e-03, 1.97040318e-04, 3.08223638e-04, ...,
             2.36152335e-05, 1.03499268e-05, 1.03710962e-05]])
```

```
X_train, X_test, y_train, y_test = train_test_split(X_scale, y, test_size=0.2, random_state =
```

```
model = Sequential([
    Dense(24, activation='relu', input_shape=( 24 ,)),
    #Dense(12, activation='relu', input_shape=( 51 ,)),
    Dense(15, activation='relu'),
    #Dense(15, activation='relu'),
    #Dense(15, activation='relu'),
    Dense(1, activation='sigmoid')

])
```

```
#model.compile(optimizer='sgd',
#              loss='binary_crossentropy',
#              metrics=['accuracy'])

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

```
start = time.time()
hist = model.fit(X_train, y_train,
          batch_size=20, epochs=25, validation_split=0.2)
```

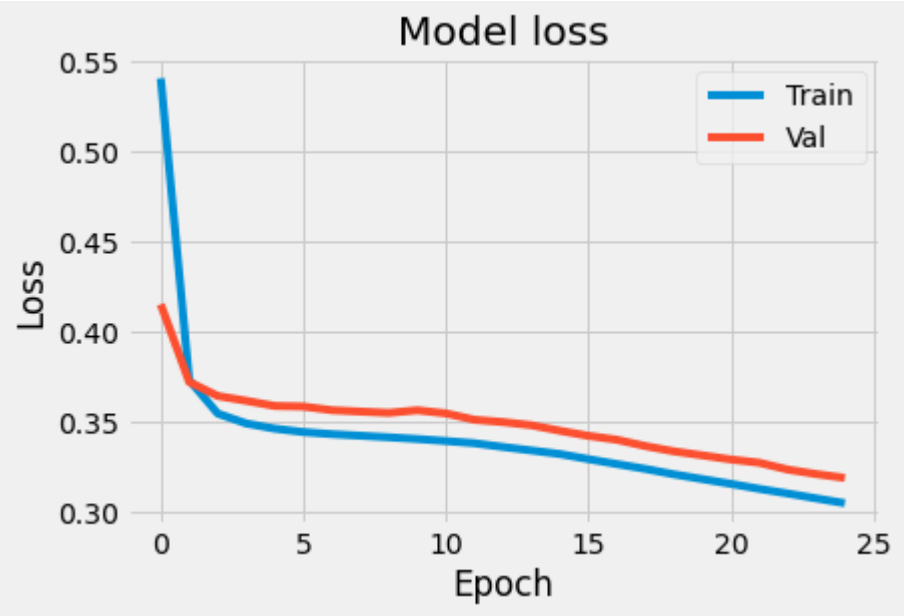
```
stop = time.time()
print(f"Training time: {stop - start}s")
# prints: Training time: 0.20307230949401855s

# https://machinelearningmastery.com/evaluate-performance-deep-learning-models-keras/
```

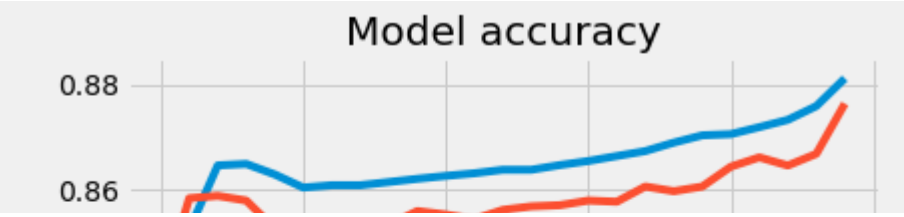
```
Epoch 1/25
896/896 [==============================] - 1s 1ms/step - loss: 0.5406 - accuracy: 0.8000
Epoch 2/25
896/896 [==============================] - 1s 1ms/step - loss: 0.3734 - accuracy: 0.8522
Epoch 3/25
896/896 [==============================] - 1s 1ms/step - loss: 0.3548 - accuracy: 0.8646
Epoch 4/25
896/896 [==============================] - 1s 1ms/step - loss: 0.3493 - accuracy: 0.8648
Epoch 5/25
896/896 [==============================] - 1s 1ms/step - loss: 0.3464 - accuracy: 0.8628
Epoch 6/25
896/896 [==============================] - 1s 1ms/step - loss: 0.3446 - accuracy: 0.8603
Epoch 7/25
896/896 [==============================] - 1s 1ms/step - loss: 0.3434 - accuracy: 0.8608
Epoch 8/25
896/896 [==============================] - 1s 1ms/step - loss: 0.3425 - accuracy: 0.8608
Epoch 9/25
896/896 [==============================] - 1s 1ms/step - loss: 0.3417 - accuracy: 0.8614
Epoch 10/25
896/896 [==============================] - 1s 1ms/step - loss: 0.3407 - accuracy: 0.8620
Epoch 11/25
896/896 [==============================] - 1s 1ms/step - loss: 0.3396 - accuracy: 0.8626
Epoch 12/25
896/896 [==============================] - 1s 1ms/step - loss: 0.3384 - accuracy: 0.8631
Epoch 13/25
896/896 [==============================] - 1s 1ms/step - loss: 0.3363 - accuracy: 0.8637
Epoch 14/25
896/896 [==============================] - 1s 1ms/step - loss: 0.3344 - accuracy: 0.8637
Epoch 15/25
896/896 [==============================] - 1s 1ms/step - loss: 0.3324 - accuracy: 0.8646
Epoch 16/25
896/896 [==============================] - 1s 1ms/step - loss: 0.3296 - accuracy: 0.8654
Epoch 17/25
896/896 [==============================] - 1s 1ms/step - loss: 0.3268 - accuracy: 0.8664
Epoch 18/25
896/896 [==============================] - 1s 1ms/step - loss: 0.3241 - accuracy: 0.8673
Epoch 19/25
896/896 [==============================] - 1s 1ms/step - loss: 0.3211 - accuracy: 0.8689
Epoch 20/25
896/896 [==============================] - 1s 1ms/step - loss: 0.3185 - accuracy: 0.8703
Epoch 21/25
896/896 [==============================] - 1s 1ms/step - loss: 0.3158 - accuracy: 0.8705
Epoch 22/25
896/896 [==============================] - 1s 1ms/step - loss: 0.3131 - accuracy: 0.8719
Epoch 23/25
896/896 [==============================] - 1s 1ms/step - loss: 0.3105 - accuracy: 0.8733
Epoch 24/25
896/896 [==============================] - 1s 1ms/step - loss: 0.3078 - accuracy: 0.8759
Epoch 25/25
```

```
896/896 [==============================] - 1s 1ms/step - loss: 0.3051 - accuracy: 0.8812
Training time: 25.948000192642212s
```

```
#visualize the training loss and the validation loss to see if the model is overfitting
plt.plot(hist.history['loss'])
plt.plot(hist.history['val_loss'])
plt.title('Model loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Train', 'Val'], loc='upper right')
plt.show()
```




```
#visualize the training accuracy and the validation accuracy to see if the model is overfitti
plt.plot(hist.history['accuracy'])
plt.plot(hist.history['val_accuracy'])
plt.title('Model accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Train', 'Val'], loc='lower right')
plt.show()
```




```
#Make a prediction & print the actual values
prediction = model.predict(X_test)
#prediction  = [1 if y>=0.5 else 0 for y in prediction] #Threshold
prediction  = [1 if y>=0.75 else 0 for y in prediction] #Threshold
print(prediction)
print(y_test)
```

```
[1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 1, 1, 1, 0, 0, 1, 0, 0, 0, 0, 0, 1,
[1. 1. 0. ... 0. 0. 1.]
```

```
from sklearn.metrics import classification_report,confusion_matrix, accuracy_score
pred = model.predict(X_train)
#pred  = [1 if y>=0.5 else 0 for y in pred] #Threshold
pred  = [1 if y>=0.5 else 0 for y in pred] #Threshold
print(classification_report(y_train ,pred ))
print('Confusion Matrix: \n',confusion_matrix(y_train,pred))
print()
print('Accuracy: ', accuracy_score(y_train,pred))
print()
```

```
                  precision    recall  f1-score   support

             0.0       0.83      0.97      0.89     11211
             1.0       0.97      0.80      0.87     11190

        accuracy                           0.88     22401
       macro avg       0.90      0.88      0.88     22401
    weighted avg       0.90      0.88      0.88     22401

    Confusion Matrix:
     [[10891   320]
     [ 2273  8917]]

    Accuracy:  0.8842462390071871
```

```
from sklearn.metrics import classification_report,confusion_matrix, accuracy_score
pred = model.predict(X_test)
#pred  = [1 if y>=0.5 else 0 for y in pred] #Threshold
pred  = [1 if y>=0.5 else 0 for y in pred] #Threshold
print(classification_report(y_test ,pred ))
print('Confusion Matrix: \n',confusion_matrix(y_test,pred))
print()
print('Accuracy: ', accuracy_score(y_test,pred))
print()
```

```
                  precision    recall  f1-score   support

             0.0       0.83      0.96      0.89      2790
             1.0       0.96      0.80      0.87      2811

        accuracy                           0.88      5601
       macro avg       0.89      0.88      0.88      5601
    weighted avg       0.89      0.88      0.88      5601

    Confusion Matrix:
     [[2689  101]
     [ 568 2243]]

    Accuracy:  0.8805570433851098
```

```
model.evaluate(X_test, y_test)[1]
```

```
    176/176 [==============================] - 0s 907us/step - loss: 0.3107 - accuracy: 0.88
    0.8805570602416992
```

```
# ann cros vaidacion
# https://medium.com/datadriveninvestor/k-fold-and-dropout-in-artificial-neural-network-ea054
```

```
#builing the neural net
from keras import Sequential
from keras.layers import Dense
from keras.layers import Dropout
from keras.wrappers.scikit_learn import KerasClassifier
from sklearn.model_selection import cross_val_score
```

```
#accuracies =  cross_val_score(estimator=classifier, X= X, y=output_category,cv=10, n_jobs=-1
#accuracies

#accuracies =  cross_val_score(estimator=model, X= X_test, y=pred,cv=5, n_jobs=-1)
#accuracies
```

```
# https://medium.com/analytics-vidhya/artificial-neural-network-ann-with-keras-simplified-use
```

```
from sklearn.model_selection import cross_validate
from sklearn.model_selection import cross_val_score
def kera_classifier():
    cf = Sequential()
    cf.add(Dense(units = 12,  activation = 'relu', input_dim = 24))
    #cf.add(Dense(units = 12,  activation = 'relu', input_dim = 51))
    cf.add(Dense(units = 15,  activation = 'relu'))
    cf.add(Dense(units = 1, activation = 'sigmoid'))
```

```
        cf.compile(optimizer = 'adam', loss = 'binary_crossentropy', metrics = ['accuracy'])
        return cf
    start7 = time.time()
    cf = KerasClassifier(build_fn = kera_classifier, batch_size = 1, epochs = 1)
    #cf = KerasClassifier(build_fn = kera_classifier, batch_size = 57, epochs = 100)
    #acuracies = cross_val_score(estimator = cf, X = X_train, y = y_train, cv = 10, n_jobs = -1)
    #accuracies = cross_validate(cf, X = X_train, y = y_train, cv = 10, n_jobs = -1,scoring ='acc
    accuracies = cross_val_score(cf, X = X_train, y = y_train, cv = 10, n_jobs = -1,scoring ='acc
    #ean = accuracies.mean()
    #iance = accuracies.std()
    #
    prec = cross_val_score(cf, X = X_train, y = y_train, cv = 10, n_jobs = -1 ,scoring ='precisio
    f1 = cross_val_score(cf, X = X_train, y = y_train, cv = 10, n_jobs = -1 ,scoring ='f1')
    recal = cross_val_score(cf, X = X_train, y = y_train, cv = 10, n_jobs = -1 ,scoring ='recall'
    #ean1= recal.mean()
    #ariance1= recal.std()


    #print(f"accuracy:")
    #print (accuracies)
    #print (mean)
    #print( variance)
    #print(f"recall:")
    #print (recal)
    #print (mean1)
    #print( variance1)

    #
    print(f"preci:")
    print(prec)
    print(prec.mean())
    print(prec.std())
    #print(variance['test_score'])
    #
    print(f"recall:")
    print(recal)
    print(recal.mean())
    print(recal.std())
    #print(variance['test_score'])
    #
    #
    print(f"f1-score:")
    print(f1)
    print(f1.mean())
    print(f1.std())
    #print(variance['test_score'])
    #
    #
    print(f"accuracy:")
    print(accuracies)
    print("\n")
    print(accuracies.mean())
```

```
print("\n")
print(accuracies.std())
print("\n")
#print(variance['test_score'])
#
stop7 = time.time()
print(f"CV Training time: {stop7 - start7}s")

# 200 epochs
# 0.9936249911785126
# 0.003466360910457571
```

```
preci:
[0.88281216 0.91045902 0.9068215  0.89759051 0.89851648 0.9103238
 0.90897298 0.90790434 0.89977969 0.90286294]
0.9026043416063464
0.008058196402300932
recall:
[0.8018018  0.79378531 0.80105171 0.79736842 0.79695886 0.80332168
 0.79177837 0.79181818 0.7798574  0.79100529]
0.7948747019911875
0.006547228267754053
f1-score:
[0.89032901 0.85237614 0.88394584 0.87403846 0.87766221 0.88781431
 0.87984111 0.87519103 0.87238285 0.88027478]
0.8773855752752342
0.010001501585788207
accuracy:
[0.90004462 0.89732143 0.89508929 0.88526786 0.88839286 0.890625
 0.89196429 0.88973214 0.884375   0.88839286]


0.8911205337221905


0.004810021260618194


CV Training time: 669.8710045814514s
```

```
#  Dense(12, activation='relu', input_shape=( 51 ,)),
#    Dense(15, activation='relu'),
#    Dense(1, activation='sigmoid')


from sklearn.model_selection import cross_validate
from sklearn.model_selection import cross_val_score

from sklearn.metrics import plot_confusion_matrix

#hist = model.fit(X_train, y_train,
#          batch_size=20, epochs=25, validation_split=0.2)
```

```
def kera_classifier():
    clf = Sequential()
    clf.add(Dense(units = 24,  activation = 'relu', input_dim = 24))
    #cf.add(Dense(units = 12,  activation = 'relu', input_dim = 51))
    clf.add(Dense(units = 15,  activation = 'relu'))
    clf.add(Dense(units = 1, activation = 'sigmoid'))
    clf.compile(optimizer = 'adam', loss = 'binary_crossentropy', metrics = ['accuracy'])
    return clf
start7 = time.time()
#cf = KerasClassifier(build_fn = kera_classifier, batch_size = 20, epochs = 25)
clf = KerasClassifier(build_fn = kera_classifier, batch_size = 25, epochs = 325, validation_s

#accuracies = cross_val_score(cf, X = X_train, y = y_train, cv = 10, n_jobs = -1,scoring ='ac

#
#prec = cross_val_score(cf, X = X_train, y = y_train, cv = 10, n_jobs = -1 ,scoring ='precisi
#f1 = cross_val_score(cf, X = X_train, y = y_train, cv = 10, n_jobs = -1 ,scoring ='f1')
#recal = cross_val_score(cf, X = X_train, y = y_train, cv = 10, n_jobs = -1 ,scoring ='recall

#scores = cross_validate(clf, X, y, cv=10,scoring =['accuracy','f1','recall','precision'],ret
scores = cross_validate(clf, X = X_train,  y = y_train, cv=10,scoring =['accuracy','f1','reca



#mejora gh
# solo es soportado por clasificadores
#plot_confusion_matrix(clf, X_test, y_test, cmap=plt.cm.Blues)

#plt.show()

gh4 = scores.get("test_accuracy")

print(f"accuracy:")
print(gh4)
print(gh4.mean())

gh3 = scores.get("test_precision")

print(f"precision:")
print(gh3)
print(gh3.mean())

gh = scores.get("test_recall")

print(f"recall:")
print(gh)
print(gh.mean())

gh2 = scores.get("test_f1")

print(f"f1:")
print(gh2)
print(gh2.mean())
```

```
#CM = confusion_matrix(y_test, y_pred)

CM = confusion_matrix(y_test, pred)

print(f"--------")
print(f"matriz de confusion:")
TN = CM[0][0]
FN = CM[1][0]
TP = CM[1][1]
FP = CM[0][1]
print(f"TN={TN}, FP={FP} ")
print(f"FN={FN}, TP={TP} ")

print(f"--------")
print(f"matriz de confusion %:")
total1=(TN+TP+FN+FP)

print(f"TN={100*TN/total1}, FP={100*FP/total1} ")
print(f"FN={100*FN/total1}, TP={100*TP/total1} ")

print(f"--------")
acc1=(TN+TP)/(TN+TP+FN+FP)
print(f"accuracy1={acc1}")

print(f"--------")
re1=(TP)/(TP+FN)
print(f"reca1={re1}")

print(f"--------")
pre1=(TP)/(TP+FP)
print(f"pre1={pre1}")

print(f"--------")
f1s1=(2*pre1*re1)/(pre1+re1)
print(f"f1score={f1s1}")

#mejora gh

stop7 = time.time()
print(f"CV Training time: {stop7 - start7}s")

# 200 epochs
# 0.9936249911785126
# 0.003466360910457571
```

```
WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/tensorflow/python/keras/w
Instructions for updating:
Please use instead:* `np.argmax(model.predict(x), axis=-1)`,   if your model does multi-
accuracy:
[1.         0.996875   0.99821429 0.99821429 0.99866071 0.99642857
 1.         0.98928571 0.99732143 0.99910714]
```

```
0.9974107142857143
precision:
[1.         0.99811143 0.99912204 1.         0.99910474 0.99824561
 1.         0.99539595 0.99821429 1.        ]
0.9988194055784195
recall:
[1.         0.9952919  0.99737073 0.99649123 0.99821109 0.99475524
 1.         0.98272727 0.99643494 0.99823633]
0.9959518735471752
f1:
[1.         0.99669967 0.99824561 0.99824253 0.99865772 0.99649737
 1.         0.98902104 0.99732382 0.99911739]
0.9973805154395816
--------
matriz de confusion:
TN=2689, FP=101
FN=568, TP=2243
--------
matriz de confusion %:
TN=48.009284056418494, FP=1.803249419746474
FN=10.141046241742545, TP=40.046420282092484
--------
accuracy1=0.8805570433851098
--------
reca1=0.7979366773390253
--------
pre1=0.9569112627986348
--------
f1score=0.8702230843840932
CV Training time: 2181.6532554626465s
```