```
# Gerardo Herrera... ann: (1 capa oculta con 15 neuronas, activation = 'relu', epoch=10) con
```

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

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

```
# 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)
```

```
#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 =
```

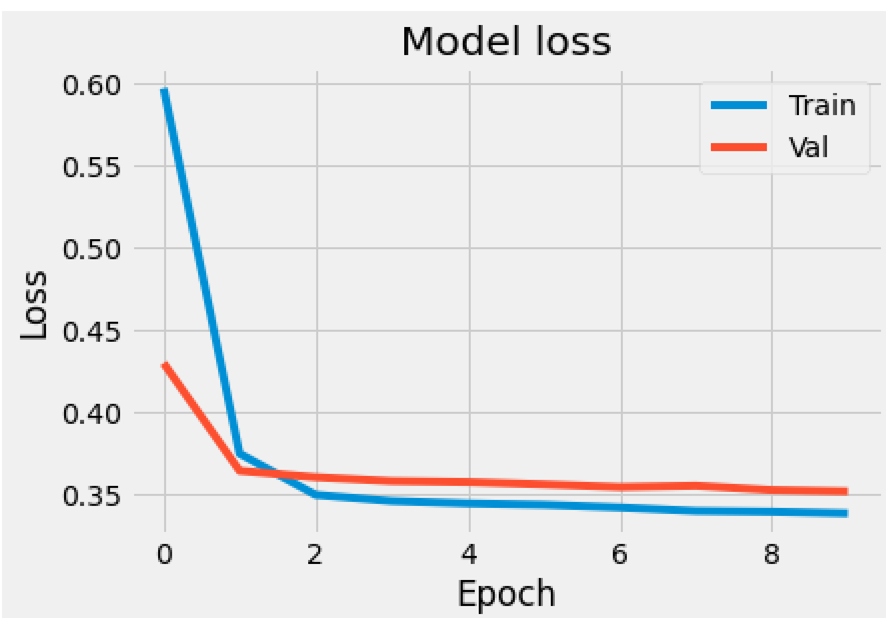
```
model = Sequential([
    Dense(12, activation='relu', input_shape=( 24 ,)),
    #Dense(12, activation='relu', input_shape=( 51 ,)),
    Dense(15, activation='relu'),
    Dense(1, activation='sigmoid')
])
```

```
#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=10, epochs=10, 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/
```

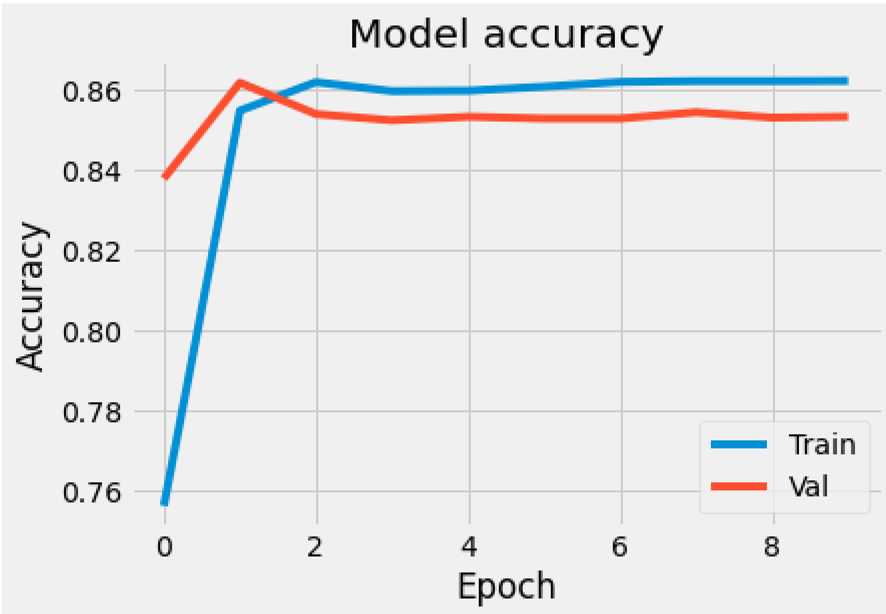
```
Epoch 1/10
1792/1792 [==============================] - 2s 1ms/step - loss: 0.5971 - accuracy: 0.75
Epoch 2/10
1792/1792 [==============================] - 2s 1ms/step - loss: 0.3751 - accuracy: 0.85
Epoch 3/10
1792/1792 [==============================] - 2s 1ms/step - loss: 0.3500 - accuracy: 0.86
Epoch 4/10
1792/1792 [==============================] - 2s 1ms/step - loss: 0.3463 - accuracy: 0.85
Epoch 5/10
1792/1792 [==============================] - 2s 1ms/step - loss: 0.3448 - accuracy: 0.85
Epoch 6/10
1792/1792 [==============================] - 2s 1ms/step - loss: 0.3440 - accuracy: 0.86
Epoch 7/10
1792/1792 [==============================] - 2s 1ms/step - loss: 0.3423 - accuracy: 0.86
Epoch 8/10
1792/1792 [==============================] - 2s 1ms/step - loss: 0.3402 - accuracy: 0.86
Epoch 9/10
1792/1792 [==============================] - 2s 1ms/step - loss: 0.3397 - accuracy: 0.86
Epoch 10/10
1792/1792 [==============================] - 2s 1ms/step - loss: 0.3388 - accuracy: 0.86
Training time: 20.224334716796875s
```

```
#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, 1, 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       0.82      0.92      0.87     11211
         1.0       0.91      0.80      0.85     11190

    accuracy                           0.86     22401
   macro avg       0.87      0.86      0.86     22401
```

```
weighted avg       0.87      0.86      0.86     22401

Confusion Matrix:
 [[10341   870]
 [ 2262  8928]]

Accuracy:  0.8601848131779831
```

```
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       0.82      0.92      0.86      2790
         1.0       0.91      0.80      0.85      2811

    accuracy                           0.86      5601
   macro avg       0.86      0.86      0.86      5601
weighted avg       0.86      0.86      0.86      5601

Confusion Matrix:
 [[2557  233]
 [ 569 2242]]

Accuracy:  0.8568112836993395
```

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

```
176/176 [==============================] - 0s 904us/step - loss: 0.3450 - accuracy: 0.85
0.8568112850189209
```

```
# 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 = 10, epochs = 10)
#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:
[0.91797062 0.92028573 0.9113896  0.91047531 0.91091228 0.91365018
 0.91178275 0.91476923 0.90475673 0.9086701 ]
0.9124662529577628
0.004242643318785688
recall:
[0.80540541 0.79566855 0.80192813 0.79912281 0.79695886 0.80594406
 0.79535299 0.79454545 0.78163993 0.79276896]
0.7969335143011957
0.006661388367289318
f1-score:
[0.89187843 0.88552013 0.88705768 0.87445887 0.88436725 0.89135683
 0.89867841 0.88372093 0.87238285 0.88223749]
0.8851658870026796
0.007476804866656009
accuracy:
[0.90316823 0.90223214 0.89598214 0.89508929 0.9        0.91696429
 0.89375    0.896875   0.88839286 0.89285714]


0.898531108561229


0.007456965145834511
```

```
CV Training time: 709.1969230175018s
```

```
#  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 = 12,  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 = 10, epochs = 10, validation_sp

# neuronas = 12 + 24 + 15 + 1 = 52

#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(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
```

```
1613/1613 [==============================] - 2s 1ms/step - loss: 0.2731 - accuracy: 0
Epoch 9/10
1613/1613 [==============================] - 2s 1ms/step - loss: 0.2704 - accuracy: 0
Epoch 10/10
1613/1613 [==============================] - 2s 1ms/step - loss: 0.2689 - accuracy: 0
Epoch 1/10
1613/1613 [==============================] - 2s 1ms/step - loss: 0.3957 - accuracy: 0
Epoch 2/10
1613/1613 [==============================] - 2s 1ms/step - loss: 0.3314 - accuracy: 0
Epoch 3/10
1613/1613 [==============================] - 2s 1ms/step - loss: 0.3138 - accuracy: 0
Epoch 4/10
1613/1613 [==============================] - 2s 1ms/step - loss: 0.2983 - accuracy: 0
Epoch 5/10
1613/1613 [==============================] - 2s 1ms/step - loss: 0.2912 - accuracy: 0
Epoch 6/10
1613/1613 [==============================] - 2s 1ms/step - loss: 0.2868 - accuracy: 0
Epoch 7/10
1613/1613 [==============================] - 2s 1ms/step - loss: 0.2834 - accuracy: 0
Epoch 8/10
1613/1613 [==============================] - 2s 1ms/step - loss: 0.2805 - accuracy: 0
Epoch 9/10
1613/1613 [==============================] - 2s 1ms/step - loss: 0.2784 - accuracy: 0
Epoch 10/10
1613/1613 [==============================] - 2s 1ms/step - loss: 0.2766 - accuracy: 0
Epoch 1/10
1613/1613 [==============================] - 2s 2ms/step - loss: 0.3933 - accuracy: 0
Epoch 2/10
1613/1613 [==============================] - 2s 1ms/step - loss: 0.3282 - accuracy: 0
Epoch 3/10
1613/1613 [==============================] - 2s 1ms/step - loss: 0.3097 - accuracy: 0
Epoch 4/10
1613/1613 [==============================] - 2s 1ms/step - loss: 0.2971 - accuracy: 0
Epoch 5/10
1613/1613 [==============================] - 2s 1ms/step - loss: 0.2907 - accuracy: 0
Epoch 6/10
1613/1613 [==============================] - 2s 1ms/step - loss: 0.2865 - accuracy: 0
Epoch 7/10
1613/1613 [==============================] - 2s 1ms/step - loss: 0.2831 - accuracy: 0
Epoch 8/10
1613/1613 [==============================] - 2s 1ms/step - loss: 0.2811 - accuracy: 0
```

```
1613/1613 [==============================] - 2s 1ms/step - loss: 0.2811 - accuracy: 0
Epoch 9/10
1613/1613 [==============================] - 2s 1ms/step - loss: 0.2794 - accuracy: 0
Epoch 10/10
1613/1613 [==============================] - 2s 1ms/step - loss: 0.2779 - accuracy: 0
Epoch 1/10
1613/1613 [==============================] - 2s 1ms/step - loss: 0.3984 - accuracy: 0
Epoch 2/10
1613/1613 [==============================] - 2s 1ms/step - loss: 0.3333 - accuracy: 0
Epoch 3/10
1613/1613 [==============================] - 2s 1ms/step - loss: 0.3114 - accuracy: 0
Epoch 4/10
1613/1613 [==============================] - 2s 1ms/step - loss: 0.2938 - accuracy: 0
Epoch 5/10
1613/1613 [==============================] - 2s 1ms/step - loss: 0.2847 - accuracy: 0
Epoch 6/10
1613/1613 [==============================] - 2s 1ms/step - loss: 0.2785 - accuracy: 0
Epoch 7/10
```