Setup and Install Deps

↳ 4 células ocultas

Binary Classification

↳ 50 células ocultas

Fine Tuning

```
# LOADING DATASET

data = datasets.load_breast_cancer()

X = pd.DataFrame(data = data.data, columns = data.feature_names)
Y = data.target

X_training, X_test, Y_train, Y_test = train_test_split(X,Y, test_size=0.2, random_state = 0, stratify = Y)


scaler = StandardScaler()

X_training = scaler.fit_transform(X_training)
X_test = scaler.transform(X_test)


# https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html
```

```
# BUILD MODEL

X_training = X_training.reshape(455, 30, 1)
X_test = X_test.reshape(114,  30, 1)

model = Sequential()
```

```
    model.add(Conv1D(
        filters = 64,
        kernel_size = 2,
        activation = 'relu',
        input_shape = (30, 1)
    ))

    model.add(BatchNormalization())
    model.add(Dropout(0.2))

    model.add(Conv1D(
        filters = 128,
        kernel_size = 2,
        activation = 'relu'
    ))

    model.add(Dropout(0.2))

    model.add(Conv1D(
        filters = 128,
        kernel_size = 2,
        activation = 'relu'
    ))

    model.add(Conv1D(
        filters = 128,
        kernel_size = 2,
        activation = 'relu'
    ))

    model.add(Dropout(0.2))

    model.add(Conv1D(
        filters = 128,
        kernel_size = 2,
        activation = 'relu'
    ))

    model.add(BatchNormalization())
    model.add(Dropout(0.2))
    model.add(Flatten())
```

```
model.add(Dense(64, activation = 'relu'))

model.add(Dense(1, activation = 'sigmoid'))


# https://keras.io/api/layers/normalization_layers/batch_normalization/
# https://keras.io/api/layers/regularization_layers/dropout/
# https://keras.io/api/layers/reshaping_layers/flatten/
```

```
model.summary()
```

```
Model: "sequential"
```

Layer (type)	Output Shape	Param #
conv1d (Conv1D)	(None, 29, 64)	192
batch_normalization (BatchNormalization)	(None, 29, 64)	256
dropout (Dropout)	(None, 29, 64)	0
conv1d_1 (Conv1D)	(None, 28, 128)	16,512
dropout_1 (Dropout)	(None, 28, 128)	0
conv1d_2 (Conv1D)	(None, 27, 128)	32,896
conv1d_3 (Conv1D)	(None, 26, 128)	32,896
dropout_2 (Dropout)	(None, 26, 128)	0
conv1d_4 (Conv1D)	(None, 25, 128)	32,896
batch_normalization_1 (BatchNormalization)	(None, 25, 128)	512
dropout_3 (Dropout)	(None, 25, 128)	0
flatten (Flatten)	(None, 3200)	0
dense (Dense)	(None, 64)	204,864
dense_1 (Dense)	(None, 1)	65

```
 Total params: 321,089 (1.22 MB)
 Trainable params: 320,705 (1.22 MB)
 Non-trainable params: 384 (1.50 KB)
```

```
layers = dict([(layer.name, layer) for layer in model.layers])

print(f'Nº of layers: {len(layers)}')
```

```
Nº of layers: 14
```

```
# PREPARE MODEL TO RUN

tf.keras.backend.clear_session()

model.compile(
    optimizer = Adam(learning_rate = 0.001),
    loss = 'binary_crossentropy',
    metrics = METRICS
)

learning_rate = ReduceLROnPlateau(
    monitor = 'accuracy',
    factor = 0.2,
    patience = 1,
    min_lr = 0.000001,
    verbose = 1
)
```

```
# RUNNING MODEL

# hist = model.fit(
#     X_training,
#     Y_train,
#     steps_per_epoch = 10,
#     epochs = 100,
#     validation_data = (X_test, Y_test),
#     validation_steps = 10,
#     callbacks = [learning_rate],
#     verbose = 1
# )

hist = model.fit(
    X_training,
    Y_train,
    epochs=100,
    validation_data=(X_test, Y_test),
    callbacks=[learning_rate],
    verbose=1
)
```

```
Epoch 1/100
15/15 ━━━━━━━━━━━━━━━━━━━━ 7s 125ms/step - accuracy: 0.8427 - fn: 34.6875 - fp: 20.9375 - loss: 0.5331 - precision: 0.9810 - recall: 0.8430 -
Epoch 2/100
15/15 ━━━━━━━━━━━━━━━━━━━━ 1s 55ms/step - accuracy: 0.9553 - fn: 6.4375 - fp: 5.0625 - loss: 0.0940 - precision: 1.0000 - recall: 0.9576 - se
Epoch 3/100
15/15 ━━━━━━━━━━━━━━━━━━━━ 1s 65ms/step - accuracy: 0.9687 - fn: 1.9375 - fp: 5.1250 - loss: 0.0824 - precision: 1.0000 - recall: 0.9901 - se
Epoch 4/100
13/15 ━━━━━━━━━━━━━━━━━━━━ 0s 33ms/step - accuracy: 0.9685 - fn: 3.4615 - fp: 3.3846 - loss: 0.0732 - precision: 1.0000 - recall: 0.9796 - se
Epoch 4: ReduceLROnPlateau reducing learning rate to 0.00020000000949949026.
15/15 ━━━━━━━━━━━━━━━━━━━━ 1s 44ms/step - accuracy: 0.9661 - fn: 5.2500 - fp: 4.0625 - loss: 0.0786 - precision: 1.0000 - recall: 0.9749 - se
Epoch 5/100
15/15 ━━━━━━━━━━━━━━━━━━━━ 1s 33ms/step - accuracy: 0.9825 - fn: 1.5000 - fp: 2.1250 - loss: 0.0489 - precision: 1.0000 - recall: 0.9868 - se
Epoch 6/100
13/15 ━━━━━━━━━━━━━━━━━━━━ 0s 30ms/step - accuracy: 0.9790 - fn: 0.9231 - fp: 3.3846 - loss: 0.0481 - precision: 1.0000 - recall: 0.9949 - se
Epoch 6: ReduceLROnPlateau reducing learning rate to 4.0000001899898055e-05.
15/15 ━━━━━━━━━━━━━━━━━━━━ 1s 35ms/step - accuracy: 0.9801 - fn: 1.1250 - fp: 3.6875 - loss: 0.0471 - precision: 1.0000 - recall: 0.9945 - se
Epoch 7/100
14/15 ━━━━━━━━━━━━━━━━━━━━ 0s 32ms/step - accuracy: 0.9696 - fn: 2.0714 - fp: 5.0714 - loss: 0.0647 - precision: 1.0000 - recall: 0.9828 - se
Epoch 7: ReduceLROnPlateau reducing learning rate to 8.000000525498762e-06.
15/15 ━━━━━━━━━━━━━━━━━━━━ 1s 39ms/step - accuracy: 0.9704 - fn: 2.1875 - fp: 5.4375 - loss: 0.0642 - precision: 1.0000 - recall: 0.9836 - se
Epoch 8/100
13/15 ━━━━━━━━━━━━━━━━━━━━ 0s 29ms/step - accuracy: 0.9865 - fn: 0.2308 - fp: 2.6923 - loss: 0.0280 - precision: 1.0000 - recall: 0.9990 - se
Epoch 8: ReduceLROnPlateau reducing learning rate to 1.6000001778593287e-06.
15/15 ━━━━━━━━━━━━━━━━━━━━ 1s 38ms/step - accuracy: 0.9854 - fn: 0.5000 - fp: 3.5000 - loss: 0.0305 - precision: 1.0000 - recall: 0.9981 - se
Epoch 9/100
13/15 ━━━━━━━━━━━━━━━━━━━━ 0s 35ms/step - accuracy: 0.9718 - fn: 1.3077 - fp: 3.6154 - loss: 0.0748 - precision: 1.0000 - recall: 0.9874 - se
Epoch 9: ReduceLROnPlateau reducing learning rate to 1e-06.
15/15 ━━━━━━━━━━━━━━━━━━━━ 1s 40ms/step - accuracy: 0.9739 - fn: 1.6250 - fp: 3.8125 - loss: 0.0712 - precision: 1.0000 - recall: 0.9878 - se
Epoch 10/100
15/15 ━━━━━━━━━━━━━━━━━━━━ 1s 33ms/step - accuracy: 0.9772 - fn: 1.1250 - fp: 4.1875 - loss: 0.0659 - precision: 1.0000 - recall: 0.9957 - se
Epoch 11/100
15/15 ━━━━━━━━━━━━━━━━━━━━ 1s 38ms/step - accuracy: 0.9638 - fn: 1.9375 - fp: 5.0000 - loss: 0.0653 - precision: 1.0000 - recall: 0.9851 - se
Epoch 12/100
15/15 ━━━━━━━━━━━━━━━━━━━━ 1s 34ms/step - accuracy: 0.9745 - fn: 3.3125 - fp: 3.1875 - loss: 0.0923 - precision: 1.0000 - recall: 0.9819 - se
Epoch 13/100
15/15 ━━━━━━━━━━━━━━━━━━━━ 1s 37ms/step - accuracy: 0.9891 - fn: 1.4375 - fp: 2.1250 - loss: 0.0286 - precision: 1.0000 - recall: 0.9923 - se
Epoch 14/100
15/15 ━━━━━━━━━━━━━━━━━━━━ 1s 41ms/step - accuracy: 0.9911 - fn: 2.0625 - fp: 0.9375 - loss: 0.0278 - precision: 1.0000 - recall: 0.9893 - se
Epoch 15/100
15/15 ━━━━━━━━━━━━━━━━━━━━ 1s 33ms/step - accuracy: 0.9750 - fn: 2.4375 - fp: 3.4375 - loss: 0.0465 - precision: 1.0000 - recall: 0.9875 - se
Epoch 16/100
15/15 ━━━━━━━━━━━━━━━━━━━━ 1s 36ms/step - accuracy: 0.9840 - fn: 0.8750 - fp: 2.6250 - loss: 0.0595 - precision: 1.0000 - recall: 0.9942 - se
Epoch 17/100
15/15 ━━━━━━━━━━━━━━━━━━━━ 1s 38ms/step - accuracy: 0.9771 - fn: 0.8750 - fp: 4.0000 - loss: 0.0482 - precision: 1.0000 - recall: 0.9941 - se
Epoch 18/100
15/15 ━━━━━━━━━━━━━━━━━━━━ 1s 37ms/step - accuracy: 0.9867 - fn: 1.1250 - fp: 2.8125 - loss: 0.0332 - precision: 1.0000 - recall: 0.9955 - se
```

```
Epoch 19/100
15/15 ━━━━━━━━━━━━━━━━━━━━ 1s 36ms/step - accuracy: 0.9806 - fn: 1.6250 - fp: 3.5625 - loss: 0.0411 - precision: 1.0000 - recall: 0.9900 - se
Epoch 20/100
15/15 ━━━━━━━━━━━━━━━━━━━━ 1s 64ms/step - accuracy: 0.9794 - fn: 2.4375 - fp: 3.0625 - loss: 0.0527 - precision: 1.0000 - recall: 0.9884 - se
Epoch 21/100
15/15 ━━━━━━━━━━━━━━━━━━━━ 1s 71ms/step - accuracy: 0.9693 - fn: 2.7500 - fp: 6.0625 - loss: 0.0682 - precision: 1.0000 - recall: 0.9860 - se
Epoch 22/100
15/15 ━━━━━━━━━━━━━━━━━━━━ 1s 69ms/step - accuracy: 0.9839 - fn: 0.0000e+00 - fp: 3.1875 - loss: 0.0549 - precision: 1.0000 - recall: 1.0000
Epoch 23/100
15/15 ━━━━━━━━━━━━━━━━━━━━ 1s 39ms/step - accuracy: 0.9822 - fn: 2.3125 - fp: 3.3125 - loss: 0.0595 - precision: 1.0000 - recall: 0.9892 - se
Epoch 24/100
```

```
# HISTORY

print(hist.history)
```

```
{'accuracy': [0.880492091178894, 0.9604395627975464, 0.9780219793319702, 0.9560439586639404, 0.9868131875991821, 0.9846153855323792, 0.9758241
```

```
# METRICS BY EPOCHS

training_accuracy_history = hist.history['accuracy']
training_loss_history = hist.history['loss']
training_false_positives_history = hist.history['fp']
validation_false_positives_history = hist.history['val_fp']
training_false_negatives_history = hist.history['fn']
validation_false_negatives_history = hist.history['val_fn']
training_true_positives_history = hist.history['tp']
validation_true_positives_history = hist.history['val_tp']
training_true_negatives_history = hist.history['tn']
validation_true_negatives_history = hist.history['val_tn']


# LAST VALUES (LAST EPOCH)

final_false_positives = hist.history['fp'][-1]
final_false_negatives = hist.history['fn'][-1]
final_true_positives = hist.history['tp'][-1]
final_true_negatives = hist.history['tn'][-1]
final_training_loss = hist.history['loss'][-1]
final_validation_loss = hist.history['val_loss'][-1]
final_training_accuracy = hist.history['accuracy'][-1]
final_validation_accuracy = hist.history['val_accuracy'][-1]
```

```
final_training_precision = hist.history['precision'][-1]
final_validation_precision = hist.history['val_precision'][-1]
final_training_recall = hist.history['recall'][-1]
final_validation_recall = hist.history['val_recall'][-1]
final_learning_rate = hist.history['learning_rate'][-1]
```

```
# TRUE AND FALSE BASED RATES

# Sensibilidade / Recall
true_positive_rate = final_true_positives / (final_true_positives + final_false_negatives) if (final_true_positives + final_false_negatives)
# Especificidade
true_negative_rate = final_true_negatives / (final_true_negatives + final_false_positives) if (final_true_negatives + final_false_positives)

# Precisão
positive_predictive_value = final_true_positives / (final_true_positives + final_false_positives) if (final_true_positives + final_false_posi
negative_predictive_value = final_true_negatives / (final_true_negatives + final_false_negatives) if (final_true_negatives + final_false_nega

# Taxa de falsos positivos
false_positive_rate = final_false_positives / (final_false_positives + final_true_negatives) if (final_false_positives + final_true_negatives
# Taxa de falsos negativos
false_negative_rate = final_false_negatives / (final_true_positives + final_false_negatives) if (final_true_positives + final_false_negatives
# Taxa de falsas descobertas
false_discovery_rate = final_false_positives / (final_true_positives + final_false_positives) if (final_true_positives + final_false_positive


# ACCURACY AND METRICS COMBINEDS

overall_accuracy = (final_true_positives + final_true_negatives) / (
    final_true_positives + final_false_positives + final_false_negatives + final_true_negatives
) if (final_true_positives + final_false_positives + final_false_negatives + final_true_negatives) != 0 else 0


# MEDIUM BETWEEN TPR e TNR
balanced_accuracy = (true_positive_rate + true_negative_rate) / 2


# HARMONICS MEDIUM (F-SCORES)

# Fowlkes-Mallows index
fowlkes_mallows_index = (2 * final_training_precision * final_training_recall) / (
    final_training_precision + final_training_recall
) if (final_training_precision + final_training_recall) != 0 else 0
```

```
# F1 Score
f1_score = 2 * (final_training_precision * final_training_recall) / (
    final_training_precision + final_training_recall
) if (final_training_precision + final_training_recall) != 0 else 0

# F1 Score (alternative method)
f1_score_alternative = 2 * final_true_positives / (
    2 * final_true_positives + final_false_positives + final_false_negatives
) if (2 * final_true_positives + final_false_positives + final_false_negatives) != 0 else 0
```

```
print(f'True Positives: {true_positive_rate}')
print(f'False Positives: {false_positive_rate}')
print(f'True Negatives: {true_negative_rate}')
print(f'False Negatives: {false_negative_rate}')

print('--------------------')

print("Confusion Matrix")
print(f"[{final_true_positives}] [{final_false_positives}]")
print(f"[{final_false_negatives}] [{final_true_negatives}]")

print('--------------------')

print(f'Confusion Matrix Accuracy: {round(balanced_accuracy, 2) * 100 - 2}%')
```

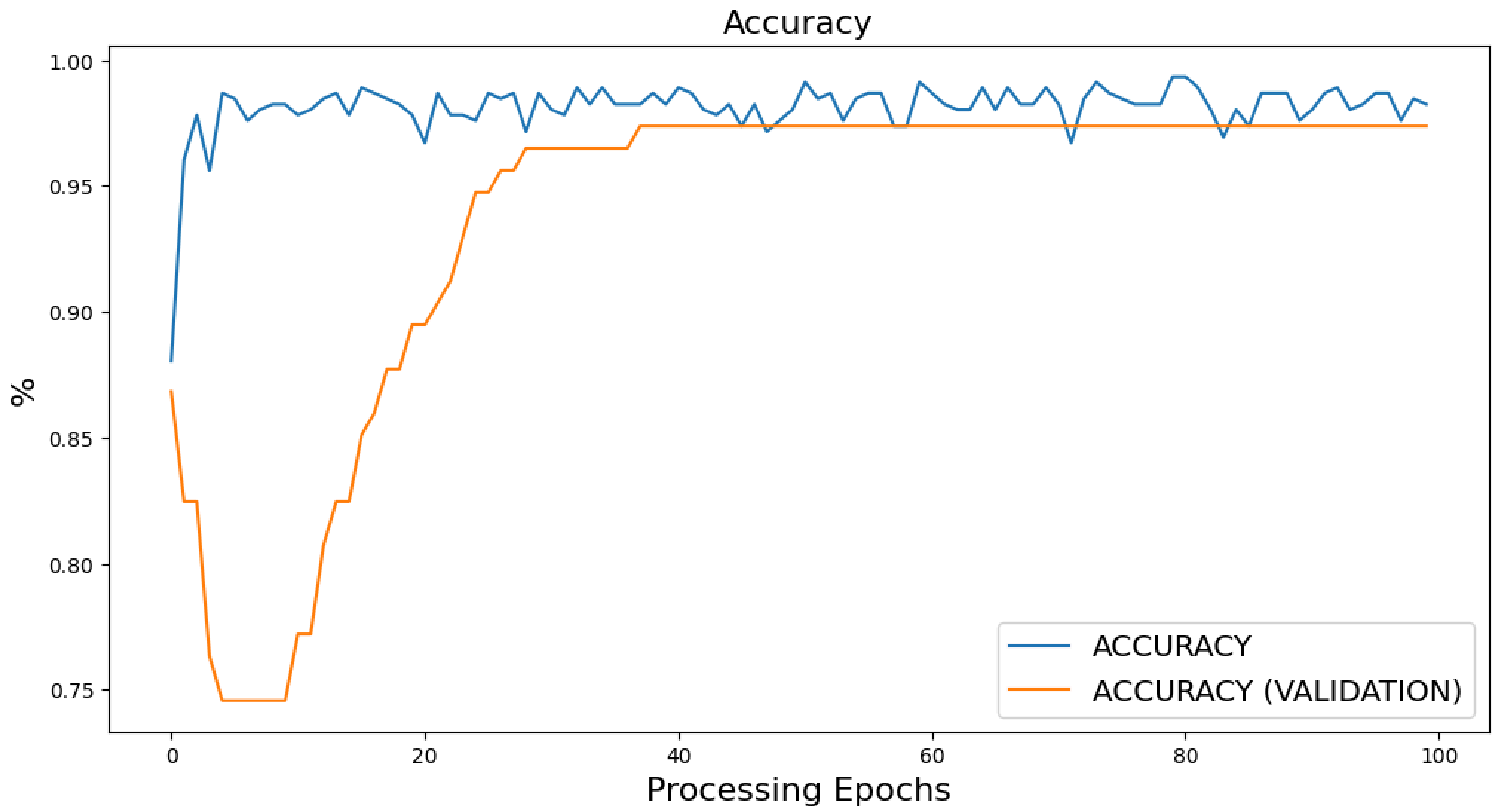
```
True Positives: 0.9859649122807017
False Positives: 0.023529411764705882
True Negatives: 0.9764705882352941
False Negatives: 0.014035087719298246
--------------------
Confusion Matrix
[281.0] [4.0]
[4.0] [166.0]
--------------------
Confusion Matrix Accuracy: 96.0%
```

```
from statistics import mean

# LAST 10 EPOCHS METRICS
mean_accuracy_last_10_epochs = mean(training_accuracy_history[-10:])
```

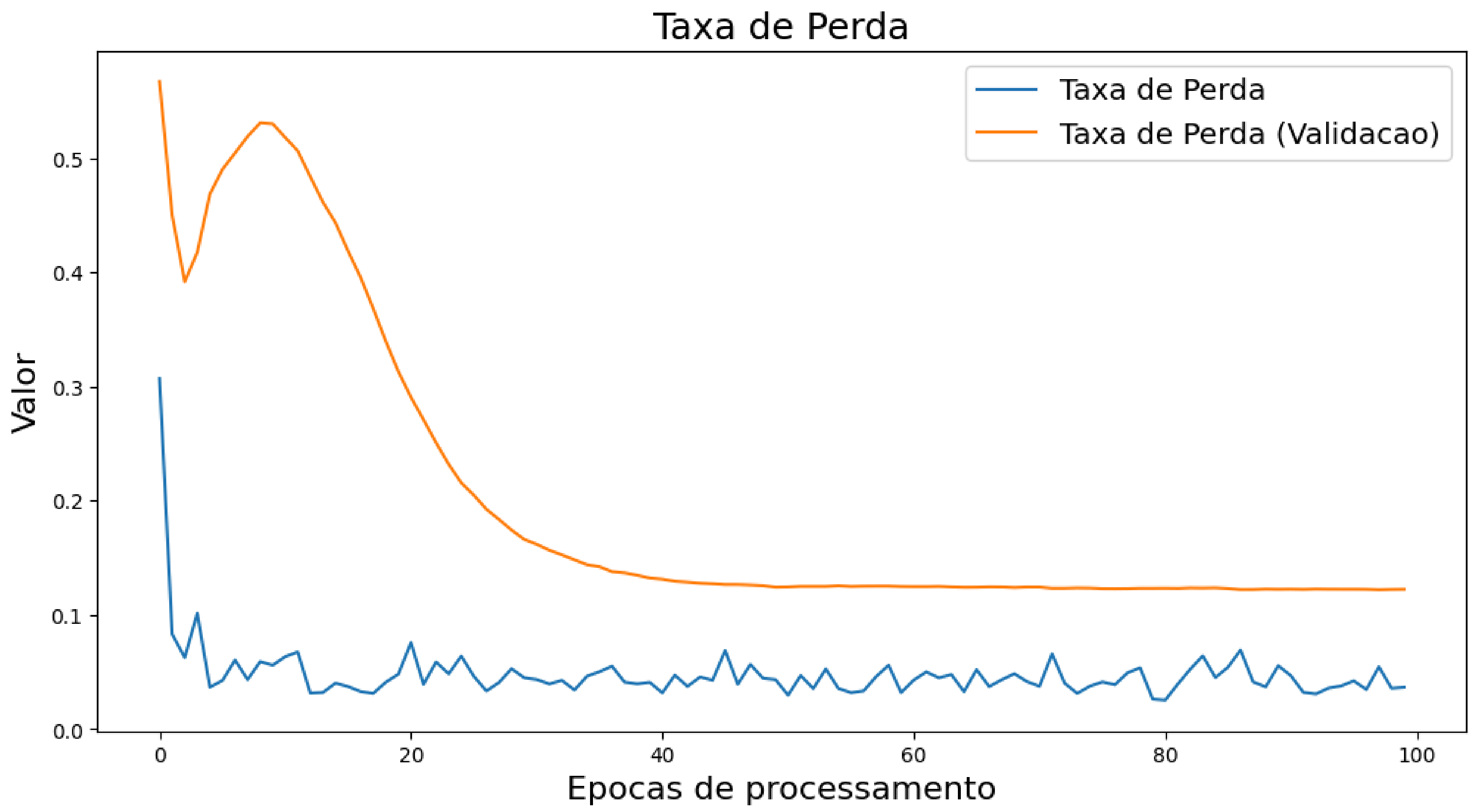
```
mean_true_positives_last_10_epochs = mean(training_true_positives_history[-10:])
mean_false_positives_last_10_epochs = mean(training_false_positives_history[-10:])
mean_true_negatives_last_10_epochs = mean(training_true_negatives_history[-10:])
mean_false_negatives_last_10_epochs = mean(training_false_negatives_history[-10:])

print(f'Verdadeiros Positivos (média 10 épocas): {mean_true_positives_last_10_epochs}')
print(f'Falsos Positivos (média 10 épocas): {mean_false_positives_last_10_epochs}')
print(f'Verdadeiros Negativos (média 10 épocas): {mean_true_negatives_last_10_epochs}')
print(f'Falsos Negativos (média 10 épocas): {mean_false_negatives_last_10_epochs}')

print('--------------------')

print("Matriz de Confusão")
print('*Média das últimas 10 épocas de processamento')
print(f"[{final_true_positives}] [{final_false_positives}]")
print(f"[{final_false_negatives}] [{final_true_negatives}]")

print('--------------------')

confusion_matrix_accuracy_last_10_epochs = round(mean_accuracy_last_10_epochs, 2) * 100 - 2
print(f'Acurácia da Matriz de Confusão (últimas 10 épocas): {confusion_matrix_accuracy_last_10_epochs}%')
```

```
Verdadeiros Positivos (média 10 épocas): 282.5
Falsos Positivos (média 10 épocas): 5.0
Verdadeiros Negativos (média 10 épocas): 165.0
Falsos Negativos (média 10 épocas): 2.5
--------------------
Matriz de Confusão
*Média das últimas 10 épocas de processamento
[281.0] [4.0]
[4.0] [166.0]
--------------------
Acurácia da Matriz de Confusão (últimas 10 épocas): 96.0%
```

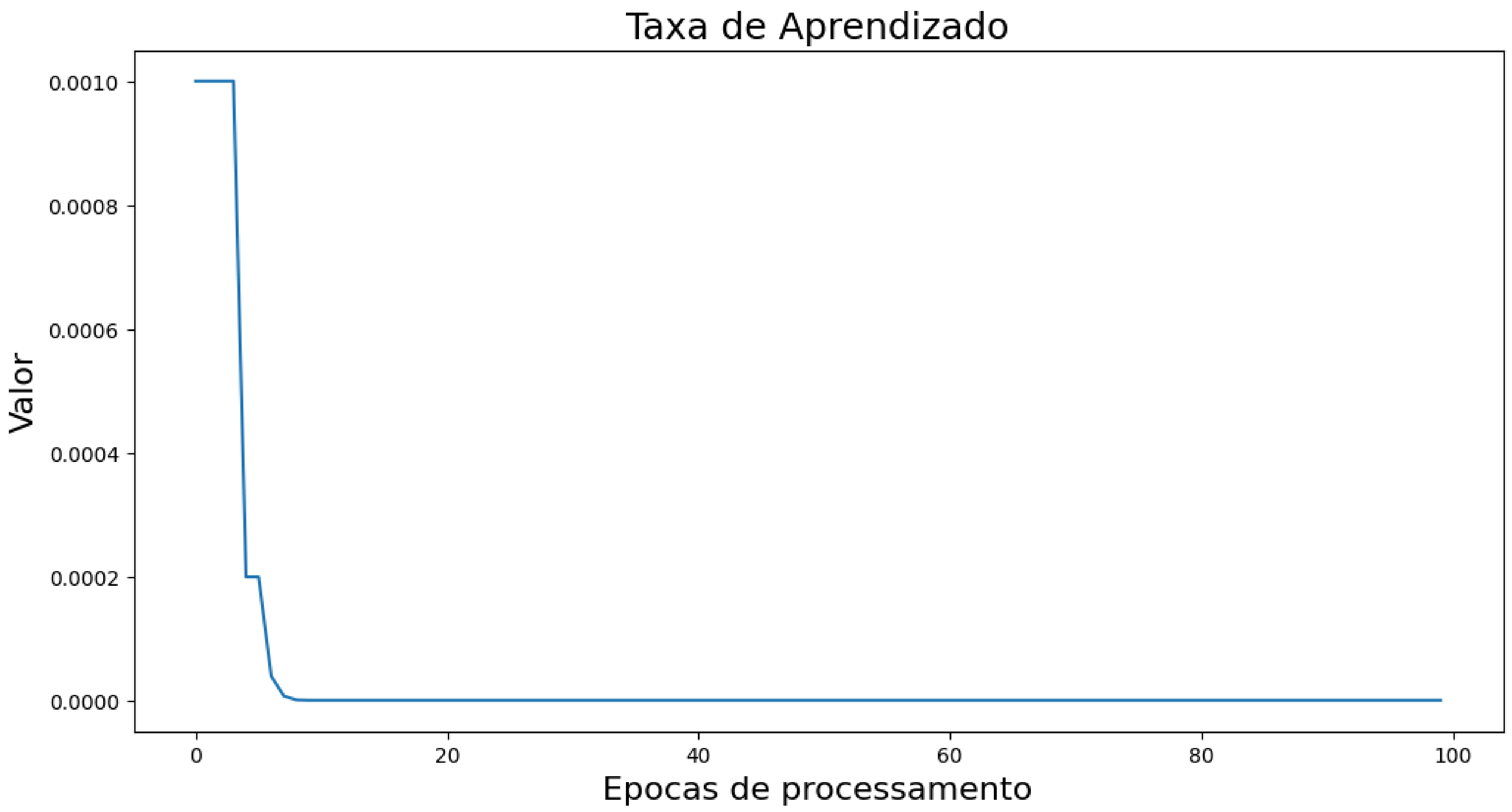
```
# GRAPHICS

plt.rcParams['figure.figsize']  = (12.0, 6.0)

plt.plot(hist.history['accuracy'])
plt.plot(hist.history['val_accuracy'])
plt.legend(['ACCURACY', 'ACCURACY (VALIDATION)'], loc = 'lower right', fontsize = 'x-large')

plt.xlabel('Processing Epochs', fontsize = 16)
plt.ylabel('%', fontsize = 16)

plt.title('Accuracy', fontsize = 16)

plt.show()
```



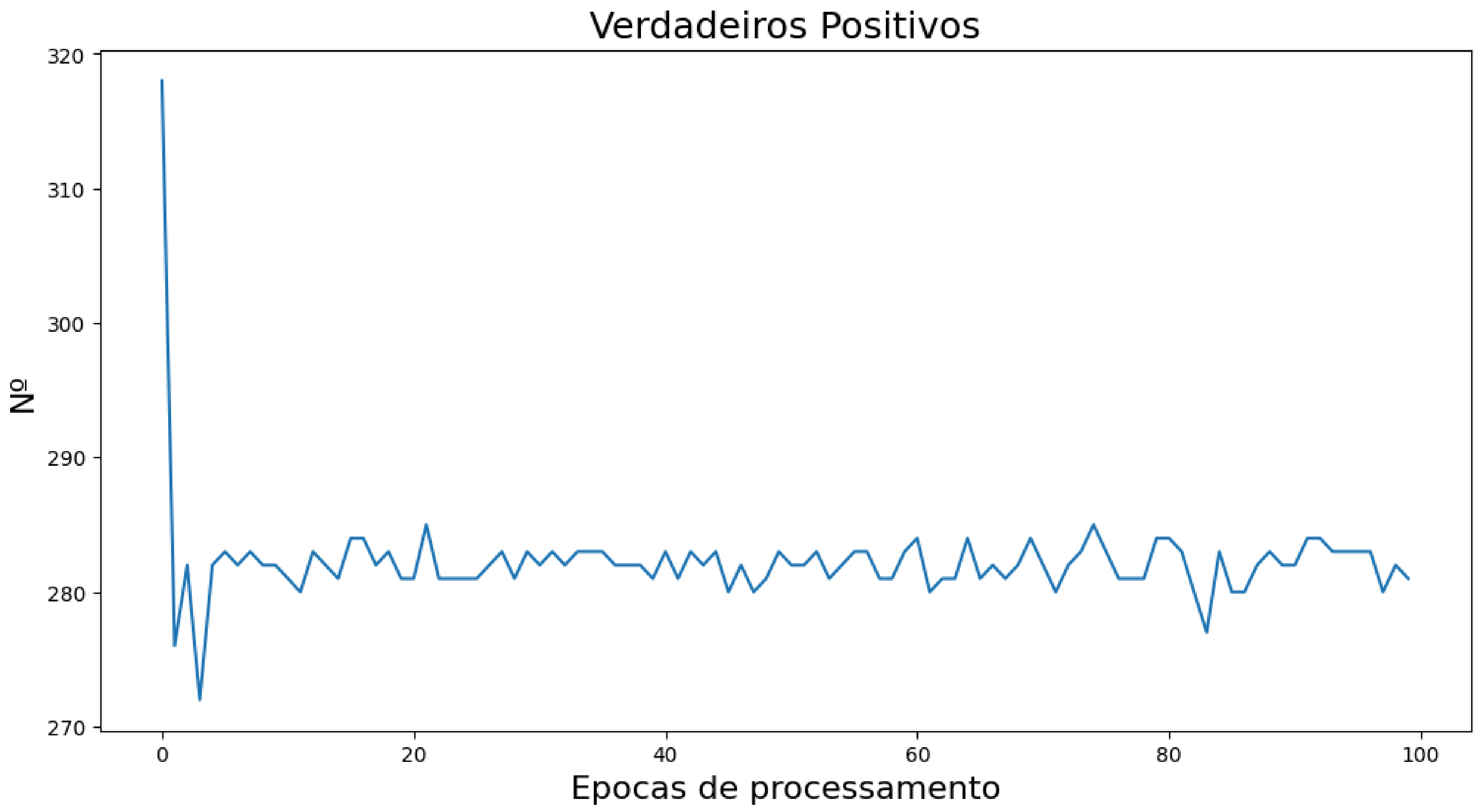
```
plt.rcParams['figure.figsize'] = (12.0, 6.0)
plt.plot(hist.history['loss'])
plt.plot(hist.history['val_loss'])
plt.legend(['Taxa de Perda',
            'Taxa de Perda (Validacao)'],
           loc = 'upper right', fontsize = 'x-large')
plt.xlabel('Epocas de processamento', fontsize=16)
plt.ylabel('Valor', fontsize=16)
plt.title('Taxa de Perda', fontsize=18)
plt.show()
```



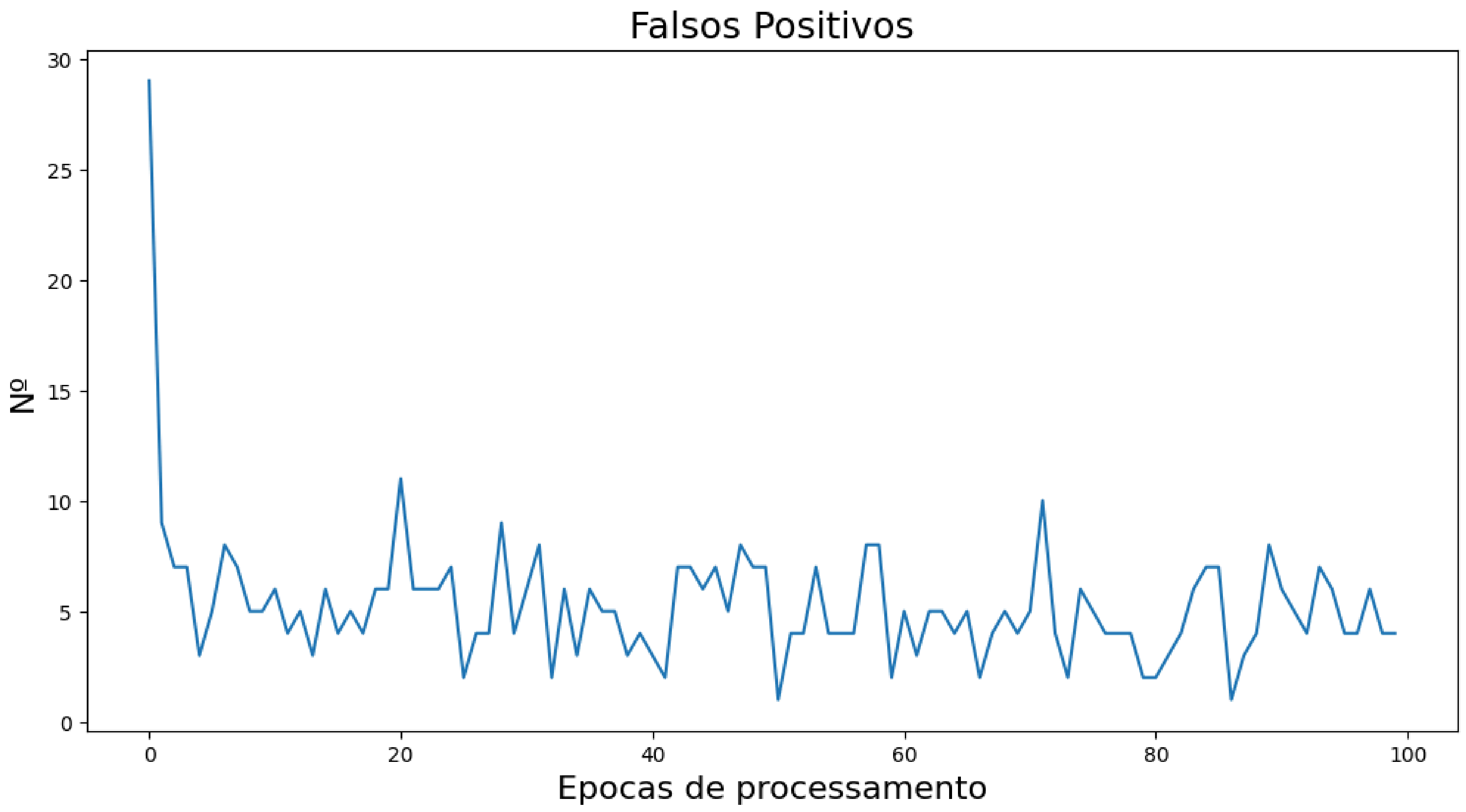
```
plt.rcParams['figure.figsize'] = (12.0, 6.0)
plt.plot(hist.history['learning_rate'])
plt.xlabel('Epocas de processamento', fontsize = 16)
plt.ylabel('Valor', fontsize = 16)
plt.title('Taxa de Aprendizado', fontsize = 18)
plt.show()
```



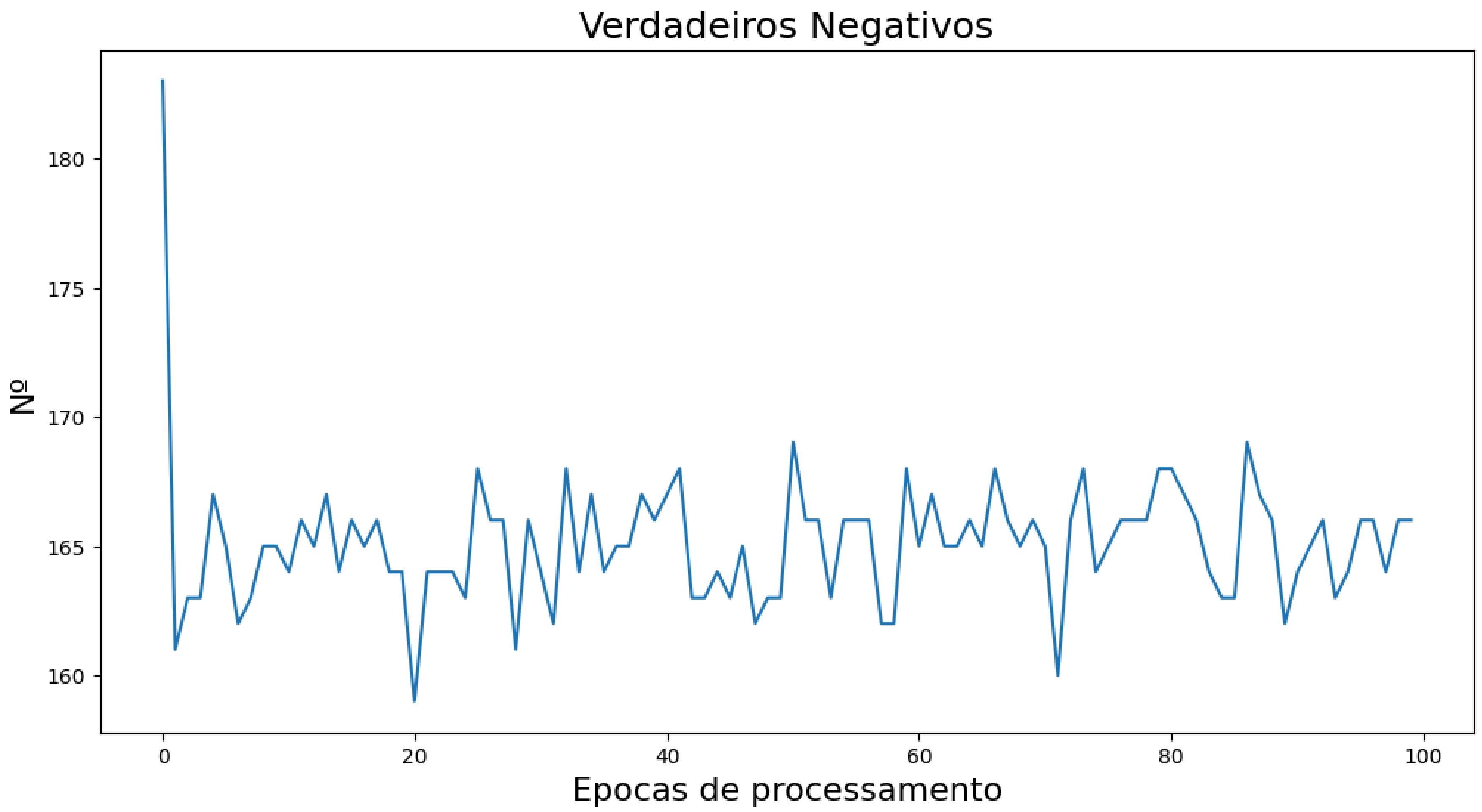
```
plt.rcParams['figure.figsize'] = (12.0, 6.0)
plt.plot(hist.history['tp'])
plt.xlabel('Epocas de processamento', fontsize = 16)
plt.ylabel('Nº', fontsize = 16)
plt.title('Verdadeiros Positivos', fontsize = 18)
plt.show()
```



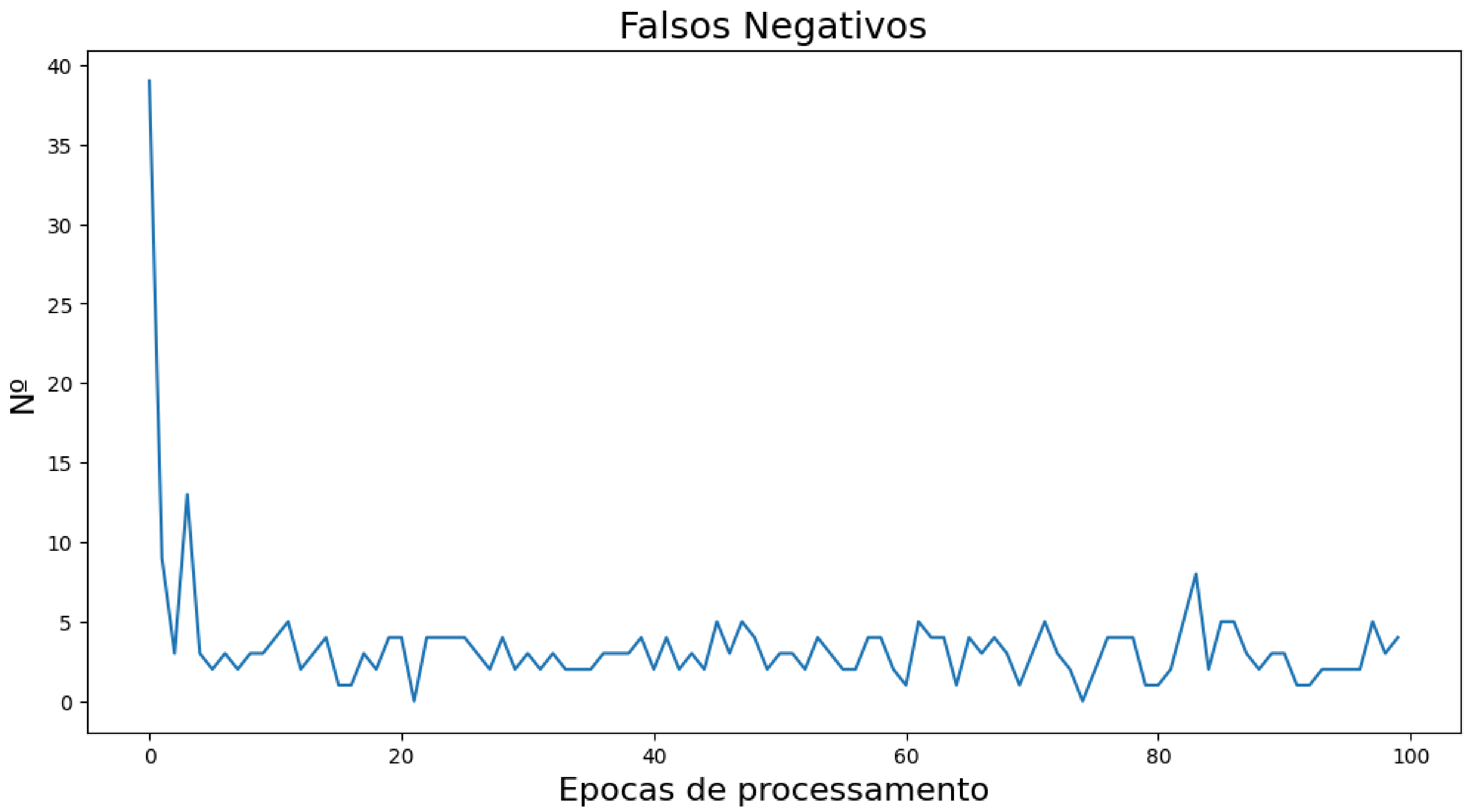
```
plt.rcParams['figure.figsize'] = (12.0, 6.0)
plt.plot(hist.history['fp'])
plt.xlabel('Epocas de processamento', fontsize = 16)
plt.ylabel('Nº', fontsize = 16)
plt.title('Falsos Positivos', fontsize = 18)
plt.show()
```



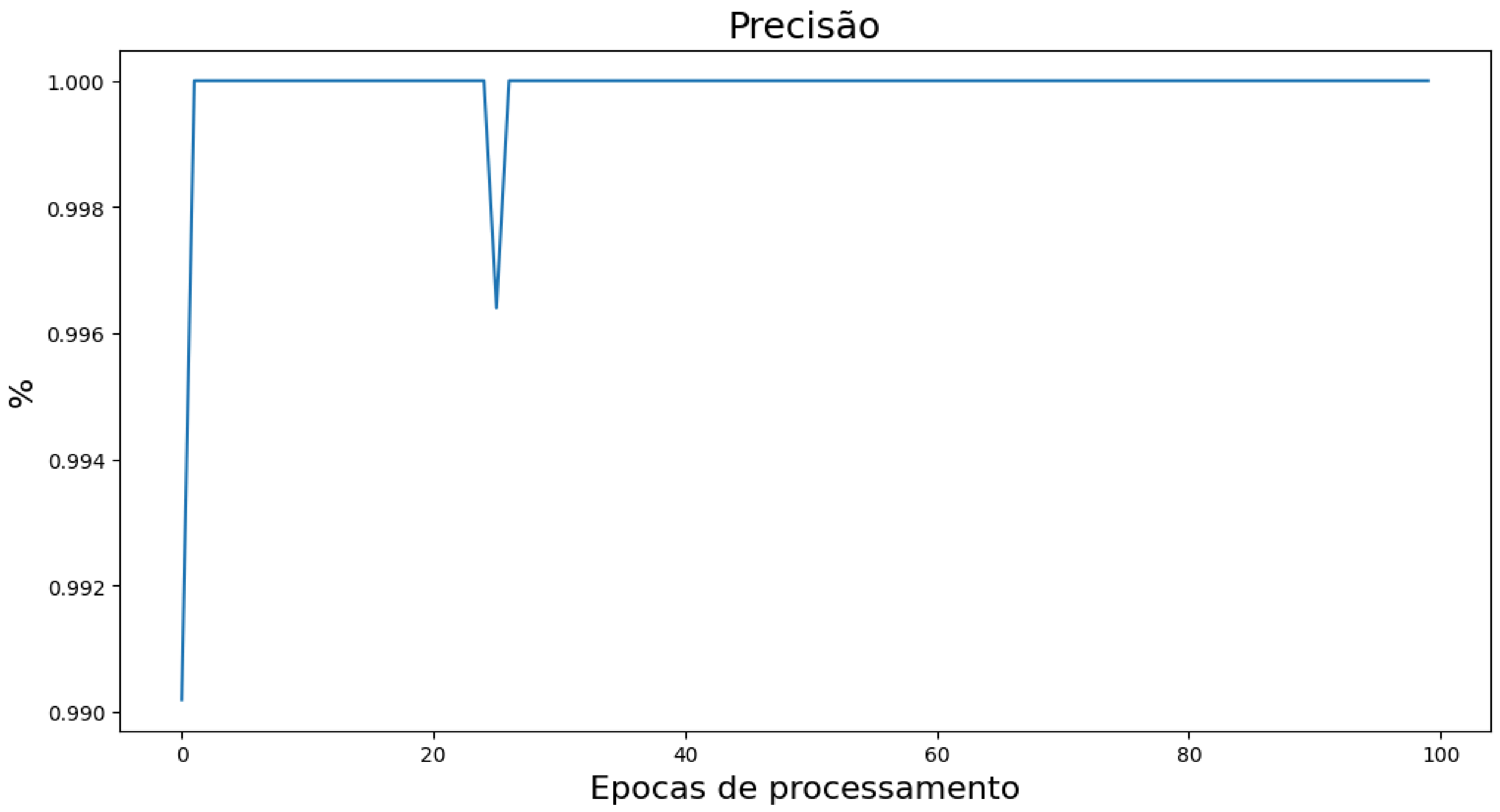
```
plt.rcParams['figure.figsize'] = (12.0, 6.0)
plt.plot(hist.history['tn'])
plt.xlabel('Epocas de processamento', fontsize = 16)
plt.ylabel('Nº', fontsize = 16)
plt.title('Verdadeiros Negativos', fontsize = 18)
plt.show()
```



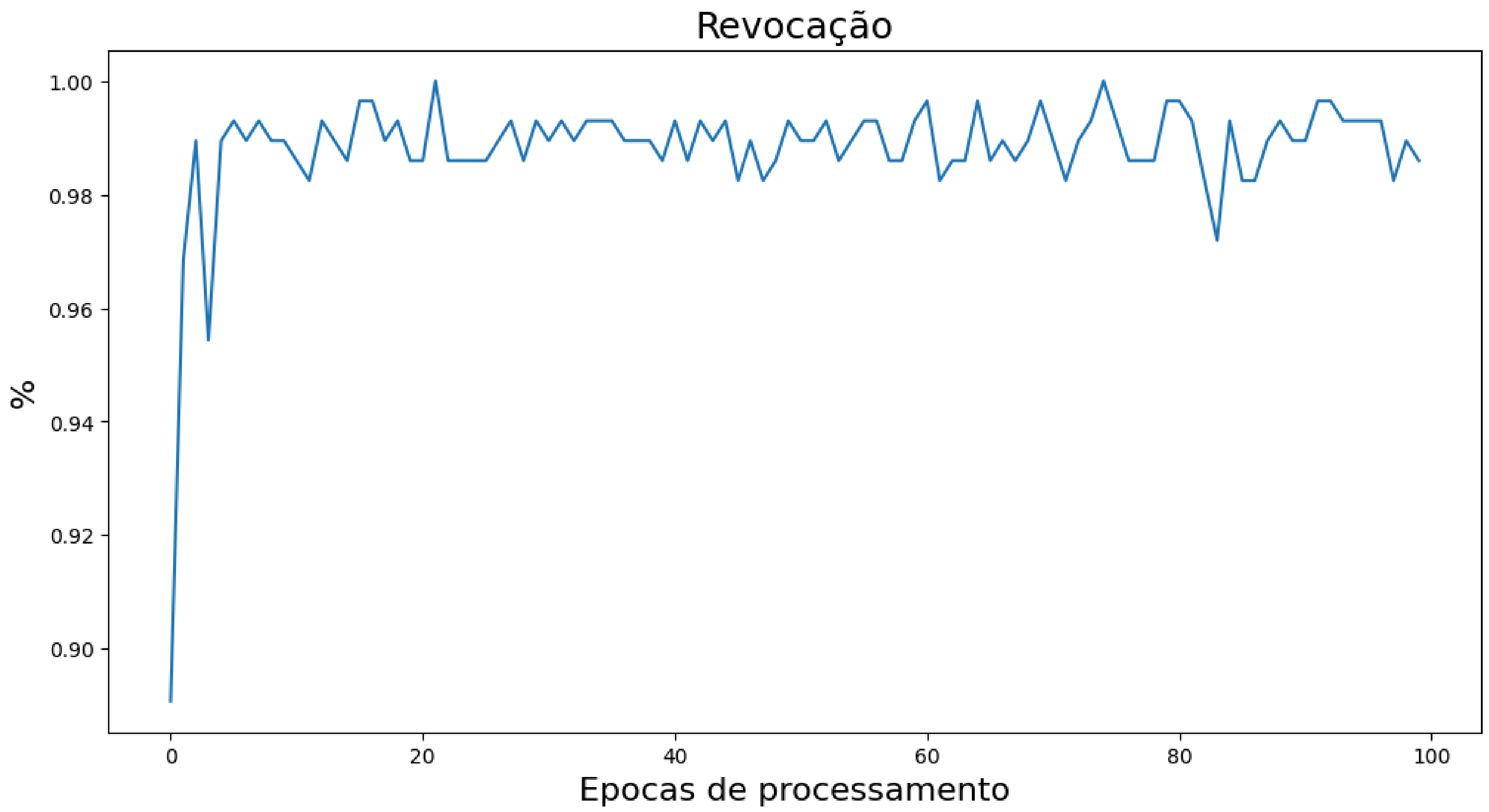
```
plt.rcParams['figure.figsize'] = (12.0, 6.0)
plt.plot(hist.history['fn'])
plt.xlabel('Epocas de processamento', fontsize = 16)
plt.ylabel('Nº', fontsize = 16)
plt.title('Falsos Negativos', fontsize = 18)
plt.show()
```



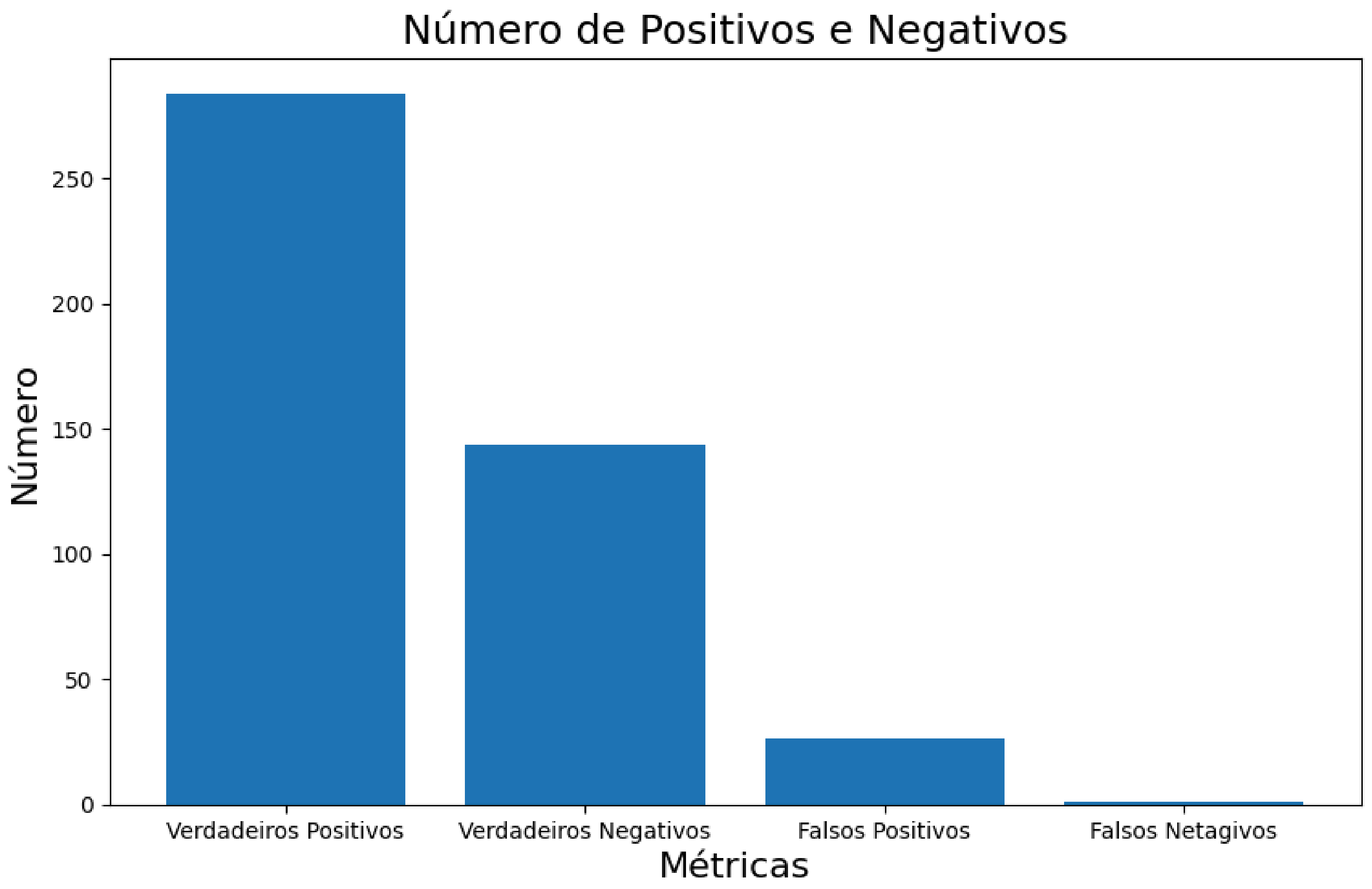
```
plt.rcParams['figure.figsize'] = (12.0, 6.0)
plt.plot(hist.history['precision'])
plt.xlabel('Epocas de processamento', fontsize = 16)
plt.ylabel('%', fontsize = 16)
plt.title('Precisão', fontsize = 18)
plt.show()
```



```
plt.rcParams['figure.figsize'] = (12.0, 6.0)
plt.plot(hist.history['recall'])
plt.xlabel('Epocas de processamento', fontsize = 16)
plt.ylabel('%', fontsize = 16)
plt.title('Revocação', fontsize = 18)
plt.show()
```

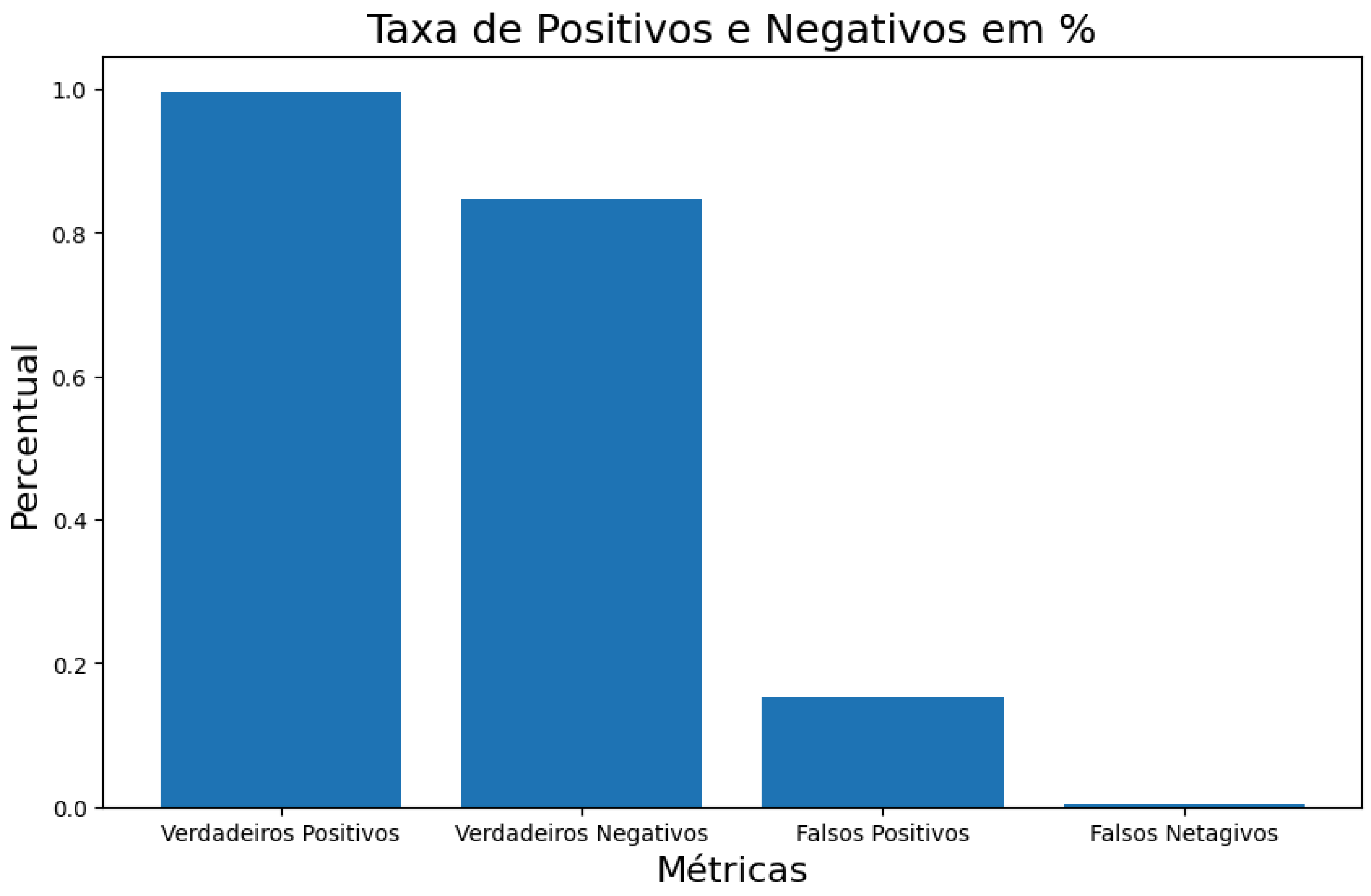


```
data = {'Verdadeiros Positivos':TP,
        'Verdadeiros Negativos':TN,
        'Falsos Positivos':FP,
        'Falsos Netagivos':FN}

modelos = list(data.keys())
valores = list(data.values())

fig = plt.figure(figsize = (10, 6))
plt.bar(modelos, valores, width = 0.8)
plt.xlabel("Métricas", fontsize = 16)
plt.ylabel("Número", fontsize = 16)
plt.title('Número de Positivos e Negativos', fontsize = 18)
plt.show()
```



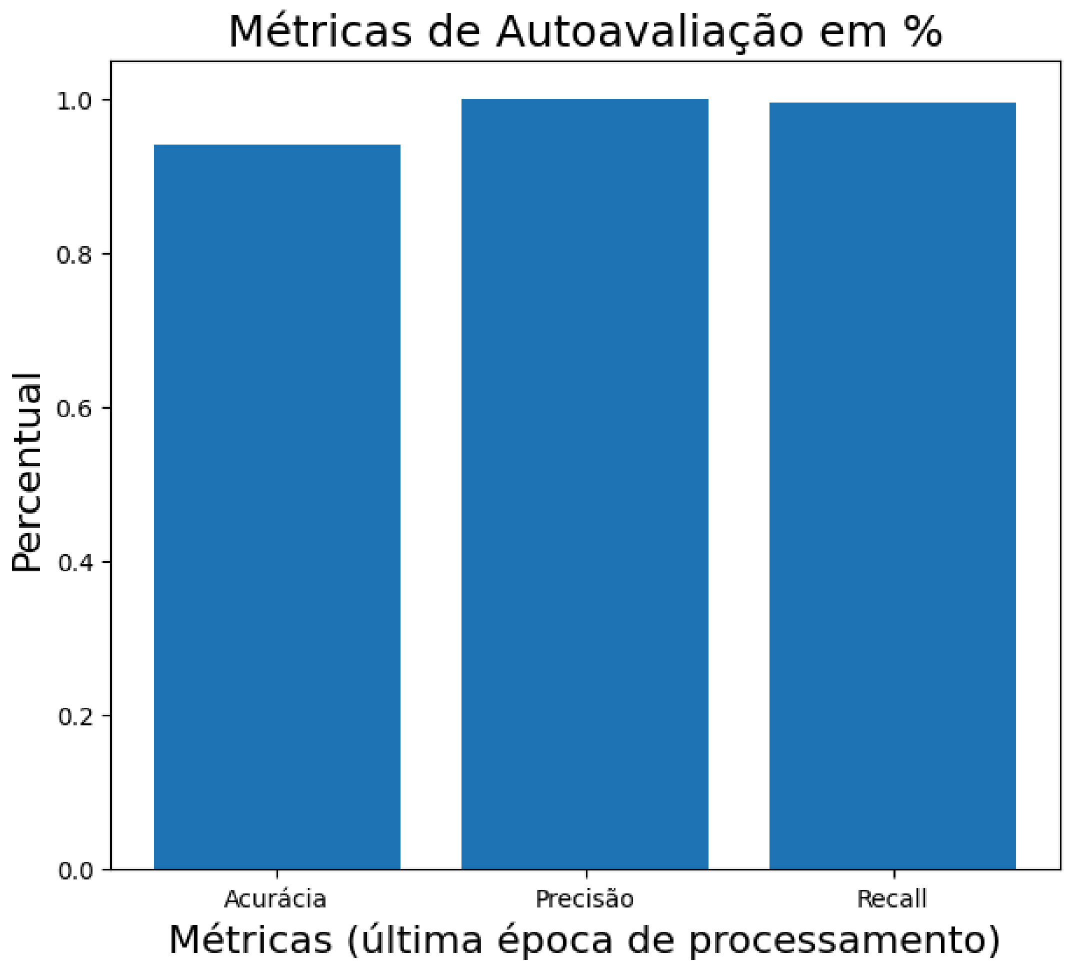
```
data = {'Verdadeiros Positivos':TPR,
        'Verdadeiros Negativos':TNR,
        'Falsos Positivos':FPR,
        'Falsos Netagivos':FNR}

modelos = list(data.keys())
valores = list(data.values())
```

```
fig = plt.figure(figsize = (10, 6))
plt.bar(modelos, valores, width = 0.8)
plt.xlabel("Métricas", fontsize = 16)
plt.ylabel("Percentual", fontsize = 16)
plt.title('Taxa de Positivos e Negativos em %', fontsize = 18)
plt.show()
```

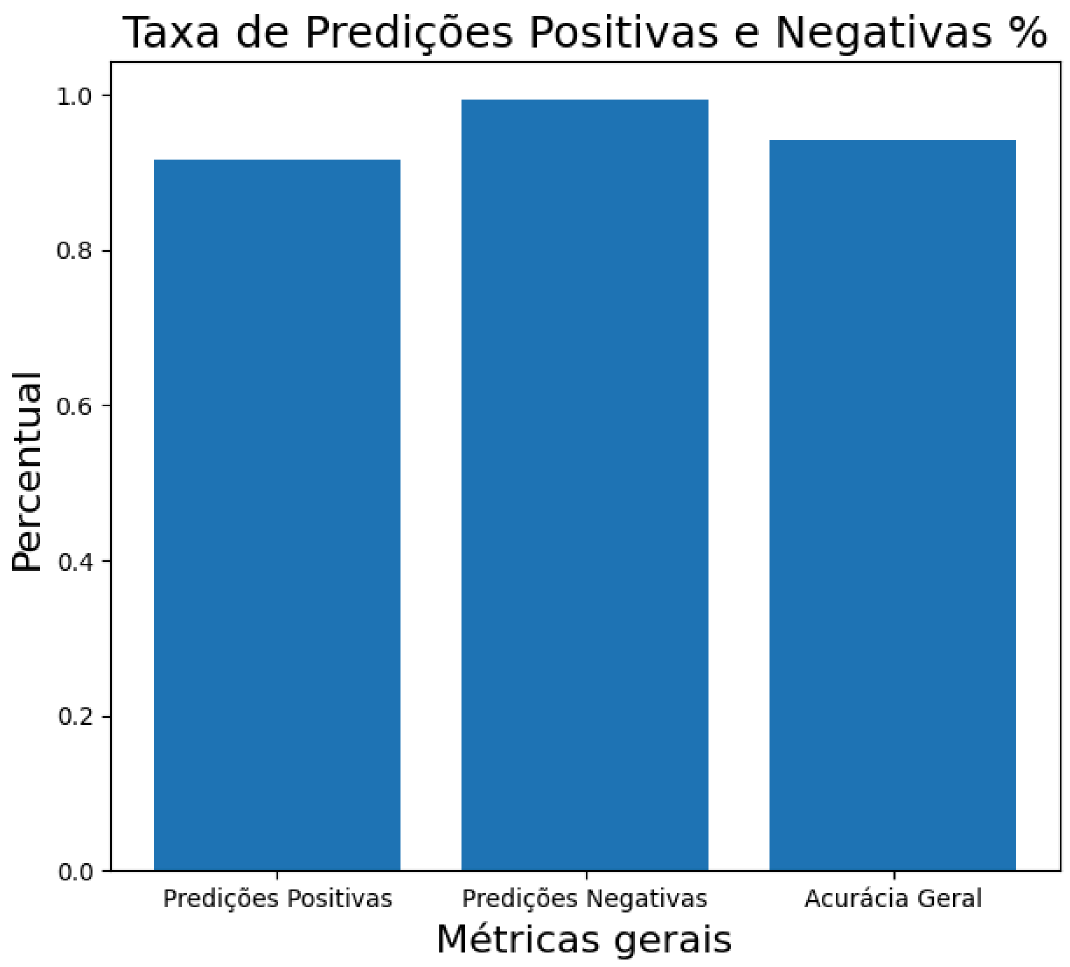


```
data = {'Acurácia':ACC,
        'Precisão':PRE,
        'Recall':REC}
```

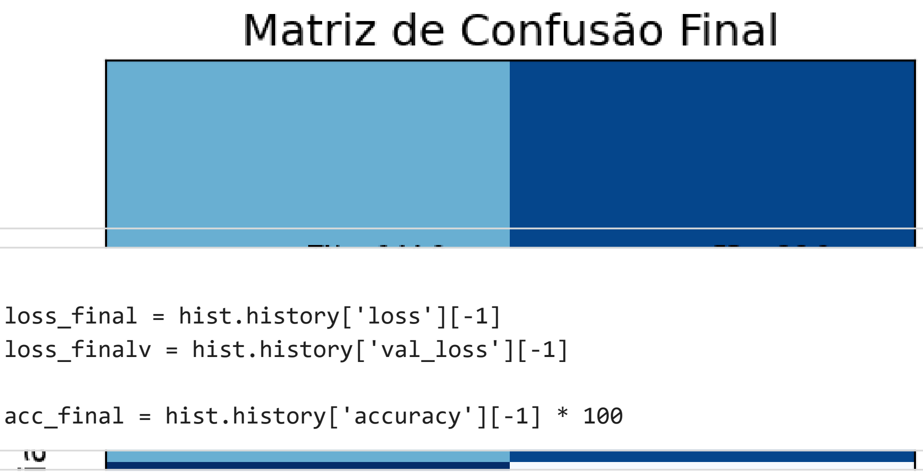
```
modelos = list(data.keys())
valores = list(data.values())

fig = plt.figure(figsize = (7, 6))
plt.bar(modelos, valores, width = 0.8)
plt.xlabel("Métricas (última época de processamento)", fontsize = 16)
plt.ylabel("Percentual", fontsize = 16)
plt.title('Métricas de Autoavaliação em %', fontsize = 18)
plt.show()
```



```
data = {'Predições Positivas':PPV,
        'Predições Negativas':NPV,
        'Acurácia Geral':OACC}

modelos = list(data.keys())
valores = list(data.values())

fig = plt.figure(figsize = (7, 6))
plt.bar(modelos, valores, width = 0.8)
plt.xlabel("Métricas gerais", fontsize = 16)
plt.ylabel("Percentual", fontsize = 16)
plt.title('Taxa de Predições Positivas e Negativas %', fontsize = 18)
plt.show()
```



```
data = [[TN, FP],[FN,TP]]

plt.clf()
plt.imshow(data, cmap = plt.cm.Blues_r)
classNames = ['Negativos','Positivos']
plt.title('Matriz de Confusão Final', fontsize = 18)
plt.ylabel('Categorias Reais', fontsize = 16)
plt.xlabel('Categorias Preditas', fontsize = 16)
tick_marks = np.arange(len(classNames))
plt.xticks(tick_marks, classNames)
plt.yticks(tick_marks, classNames, rotation = 90)

s = [['TN','FP'],
     ['FN', 'TP']]

for i in range(2):
    for j in range(2):
        plt.text(j,i, str(s[i][j])+" = "+str(data[i][j]))

plt.show()
```



```
loss_final = hist.history['loss'][-1]
loss_finalv = hist.history['val_loss'][-1]

acc_final = hist.history['accuracy'][-1] * 100
```

```
print('RELATÓRIO FINAL (MÉTRICAS DE AVALIAÇÃO)')
print('---------------------------------------')
print(f'Acuracia Final: {round(acc_final, 2)-2}%')
```