[[1]](#footnote-1)

Filtros, características y descriptores en el Procesamiento Digital de Imágenes

Santiago Correa Marulanda, Juan Esteban Aristizabal Aristizabal.

Resumen -Este proyecto presenta métodos de procesamiento digital de imágenes, practicas de ciencias de datos y aplicaciones de machine learning básicas con el objetivo de predecir a partir de imágenes de radiografías cerebrales si alguno de estos presenta algún tumor o si por el contrario se encuentra totalmente sano entrenando varios modelos para finalmente encontrar el mas preciso y que menos riesgos conlleve. Se utilizaron técnicas en Python con apoyo de las librerías OpenCV, Matplotlib, Numpy, Scikit-Learn, Path, Os, Joblib y demás. Se aplicaron conceptos de machine learning y reconocimiento de patrones de imágenes, matemáticos y tratamiento de señales aplicado a imágenes.

**Índice de Términos – Problema de clasificación, OpenCV, Procesamiento digital de imágenes, SVM, machine learning, HOG, LBP**

# introducción

El rápido y efectivo diagnóstico de tumores cerebrales es pieza fundamental para mejorar la tasa de supervivencia y optimizar el tratamiento médico de los pacientes. Anteriormente, la interpretación de las imágenes en rayos X dependía únicamente del radiólogo, un proceso susceptible a errores humanos. La aplicación del procesamiento digital de imágenes y *machine learning* permite la automatización, reduciendo el tiempo de análisis y optimizando la toma de decisiones.

El procesamiento digital de imágenes constituye una herramienta fundamental para la extracción de información visual en áreas como la medicina, seguridad y robótica. En este contexto, en la primera parte de este trabajo se exploran ocho filtros fundamentales: media, mediana, logarítmico, cuadro normalizado, gaussiano, Laplace, Sobel y Canny, junto con su implementación en OpenCV, lo cual sirve como base para el preprocesamiento y realce de características relevantes en las imágenes médicas.

En este proyecto se entrenó un modelo para un problema de clasificación binaria, basado en la teoría de los filtros y máscaras y aplicando descriptores con diferentes características (HOG y LBP) para analizar cuál es el más óptimo para esta problemática. Ambos descriptores fueron utilizados para entrenar una SVM con cada uno.

Finalmente, se presenta el proyecto con una simple interfaz gráfica con dos botones que permite probar cada modelo con las imágenes de testeo que anteriormente fueron particionadas. En la parte inferior se muestra la imagen seleccionada y, finalmente, el diagnóstico (sano o tumor) se presenta en la zona superior de la ventana.

Como último paso, se realizó un análisis de las métricas para finalmente decidirse por el mejor modelo posible, dando una vista profunda de la interpretación de las diferentes métricas calculadas.

# FILTROS

## Filtro de Media

Formula:

Descripción: Promedia los valores de un vecindario para suavizar la imagen.

Ventajas: Simple, reduce ruido suave.

Desventajas: Difumina bordes y detalles finos.

## Filtro de Mediana

Proceso: Se ordenan los valores del vecindario y se elige el valor medio.

Ventajas: Muy efectivo contra ruido impulsivo (sal y pimienta).

Desventajas: Costoso computacionalmente, puede eliminar texturas.

## Filtro Logarítmico

Formula:

Ventajas: Mejora detalles en regiones oscuras.

Desventajas: Aumenta el ruido en zonas bajas.

## Filtro de Cuadro Normalizado

Kernel: Todos los valores iguales, normalizados por la suma total.

Ventajas: Suavizado simple.

Desventajas: Menos eficaz que el Gaussiano, afecta bordes.

## Filtro Gaussiano

Formula:

Ventajas: Suavizado efectivo, preserva bordes mejor que la media.

Desventajas: Difumina detalles si σ es alto.

## Filtro Laplace

Kernel:

Ventajas: Detección de bordes cerrados.

Desventajas: Muy sensible al ruido, requiere suavizado previo.

## Filtro Sobel

Kernels: Gx y Gy detectan bordes horizontales y verticales.

Ventajas: Detecta bordes direccionales.

Desventajas: No detecta bien bordes diagonales, genera bor- des gruesos.

## Filtro Canny

Etapas: Suavizado Gaussiano, gradiente Sobel, supresión de no-máximos, umbral con histéresis.

Ventajas: Precisión y robustez en detección de bordes.

Desventajas: Costoso y sensible a parámetros.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Filtro** | **Tipo** | **Ruido que**  **atenúa** | **Preserva**  **bordes** | **Complejidad** |
| Media  Mediana Logarítmico Box (Nor- malizado) Gaussiano Laplace Sobel  Canny | Lineal  No lineal Transformación Lineal  Lineal Derivativo Derivativo Derivativo | Suave  Sal y pimienta Contraste bajo Suave  Gaussiano Ninguno Ninguno  Con Gaussiano | No  Parcial No aplica No  Moderada Realza Realza Excelente | Baja  Media Baja Baja  Media Media Media Alta |

Figura 1. Comparación entre las características de los diferentes filtros.

import cv2

import numpy as np

from matplotlib import pyplot as plt img = cv2.imread(’imagen.jpg’, 0)

media = cv2.blur(img, (5,5)) mediana = cv2.medianBlur(img, 5)

img\_float = np.float32(img) + 1

log\_img = cv2.normalize(cv2.log(img\_float), None, 0, 255, cv2.NORM\_MINMAX)

log\_img = np.uint8(log\_img)

box = cv2.boxFilter(img, -1, (5,5), normalize=True)

gauss = cv2.GaussianBlur(img, (5,5), 1) laplace = cv2.Laplacian(img, cv2.CV\_64F)

sobelx = cv2.Sobel(img, cv2.CV\_64F, 1, 0, ksize=3) sobely = cv2.Sobel(img, cv2.CV\_64F, 0, 1, ksize=3) sobel = cv2.magnitude(sobelx, sobely)

canny = cv2.Canny(img, 100, 200)

images = [img, media, mediana, log\_img, box, gauss, laplace, sobel, canny]

titles = [’Original’, ’Media’, ’Mediana’, ’Log’,

’Box’,’Gauss’, ’Laplace’, ’Sobel’, ’Canny’]

for i in range(len(images)): plt.subplot(3, 3, i+1) plt.imshow(images[i], cmap=’gray’) plt.title(titles[i]) plt.xticks([]), plt.yticks([])

plt.tight\_layout() plt.show()

Figura 2. Código de implementación de los diferentes filtros con openCV.

El código puede consultarse en el siguiente enlace:

<https://colab.research.google.com/drive/1mtfNb1tYo6XLtquH8tS7oF23tmHOJt6M?usp=sharing>

Y el informe detallado en:

<https://docs.google.com/document/d/1gehc2-JxEz4fvHpQXiPJGLqMJ8vxPuPaTDWznsEufFc/edit?usp=sharing>

# DESCRIPTORES Y CARACTERÍSTICAS

Las características son valores numéricos que obtenemos de los datos de entrada, en este caso, imágenes. Son utilizadas para obtener “pistas” que ayudan a diferenciar diferentes zonas o regiones por medio de un proceso algorítmico con el objetivo de obtener una representación que distinga los elementos deseados. Los descriptores son el proceso o algoritmo utilizado para obtener las características.

### Descriptor y Características HOG (Histogram of Oriented Gradients)

Las Características HOG son los valores numéricos obtenidos al aplicar el descriptor sobre una imagen. Cada característica o valor, es la magnitud ponderada de los gradientes de intensidad en una determinada dirección (uno de los 9 bins) dentro de una celda.

El descriptor HOG sigue el proceso a continuación: Primero debemos definir que es una celda, las celdas son una partición que se le hará a la imagen, es un cuadrado de pixeles de tamaño fijo, usualmente de 8x8 pixeles. Esto se hace con el objetivo de optimizar, ya que si se realiza pixel por pixel el vector de características final seria inmenso e inviable, cada celda resume la dirección y magnitud de los bordes en ese pequeño parche de la imagen.

Los bloques son un conjunto de celdas, usualmente un bloque se forma por 2x2 celdas, es decir, cada bloque es de 16x16 pixeles.

Se define un sistema de referencia así:

Y para las derivadas parciales con respecto a x,y se calcula con el siguiente kernel:  
GX = [-1, 0, 1] Y GY = GXT

Ejemplo: Supongamos que tenemos la siguiente partición F de pixeles 3x3:

Siendo F[x,y] = 32 el pixel en donde queremos calcula su gradiente, entonces.

GX = F[x+1,y](1) + F[x-1,y](-1) = 25(1) + 20(-1) = 5

GY = F[y+1,x](1) + F[y-1,x](-1) = 22(1) + 20(-1) = 2

.

=

= 21.801

Realizamos el anterior calculo para cada pixel de cada celda y para realizar el histograma, lo ubicaremos en un bin respectivo que estarán divididos de la siguiente forma. 9 bins que separan los grados de 0-180 en 9 partes (0-20, 20-40, …, 160-180) y al finalizar este proceso para cada celda se realiza un proceso de normalización para luego mover el bloque una celda a la derecha o hacia abajo para calcular nuevamente las características concatenándolas una y otra vez hasta formar el vector de características HOG.

El vector de características HOG tendrá:

W: Ancho en pixeles de la imagen original

H: Alto en pixeles de la imagen original

CX: pixeles en el eje x por celda

CY: pixeles en el eje y por celda

BX: Celdas en el eje x por bloque

BY: Celdas en el eje y por bloque

SX: Desplazamiento en el eje x en unidades de celdas

SY: Desplazamiento en el eje y en unidades de celdas

N: número de bins

Cellsx = W/ CX , Cellsy = H/ CY

Blocksx = (Cellsx - BX)/ SX +1

Blocksy = (Cellsy – By)/ Sy +1

Dim\_por\_bloque = BX \* By \* N

Cantidad\_total =

Blocksx \* Blocksy \* Dim\_por\_bloque

Imagen original



Imagen con solo gradientes en x



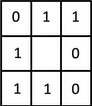
Imagen con solo gradientes en y



### Descriptor y características LBP (Local Binary Pattetns)

Las características LBP son igualmente cada uno de los valores numéricos del vector descriptor LBP, pero estas no buscan bordes, en su lugar son usadas para hallar “texturas” en determinadas regiones, un ejemplo es el reconocimiento facial, ya que la piel tiene texturas que se diferencian del resto del entorno.

Para obtener el vector descriptor se toma de la imagen original un pixel central y lo compararemos con sus 8 pixeles vecinos, si uno de estos es menor al del centro, será reemplazado por 0, si es mayor o igual será codificado como 1

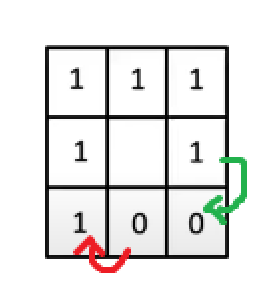


=

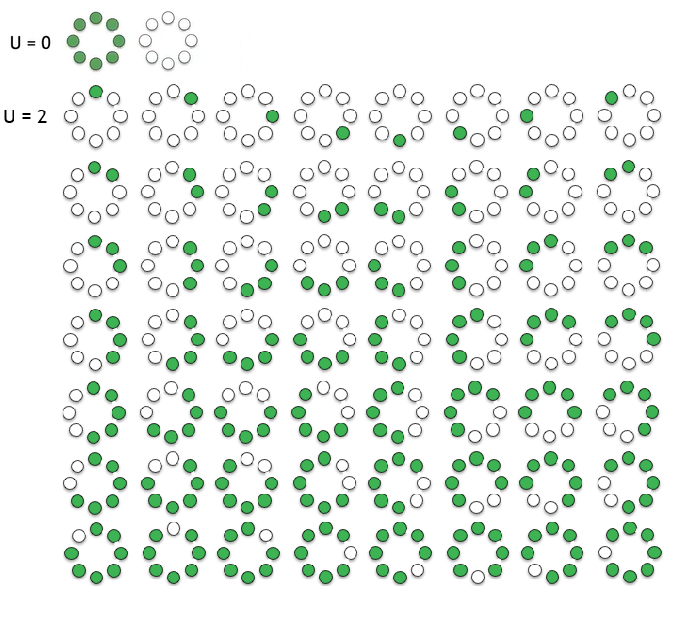
Luego adoptaremos una configuracion de potencias de 2 para multiplicar cada elemento de la matriz resultante, es decir:

0(20)+1(21)+1(22)+0(23)+0(24)+1(25)+1(26)+1(27) = 230  
por lo tanto el nuevo pixel ya no será 6, sino 230

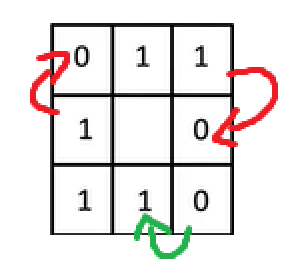
En el siguiente paso, se define el concepto de Uniform Patterns y Non Uniform Patterns. El primero de estos se refiere a aquellos patrones de vecinos que se pueden interpretar como “poco ruidosos” y seria ese patro donde hay 0 o 2 cambios de valores, por ejemplo:



En esta imagen se observa cómo hay 2 cambios, la flecha verde muestra un cambio de 1 a 0 y la roja uno de 0 a 1, llamemos los cambios U, pues este patrón de vecinos es U=2 y será Uniform Pattern todo aquel que U = 0 o U = 2 y Non Uniform Pattern cuando U = 4, U = 6 o U = 8

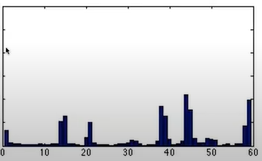


Y será codificado respectivamente como un valor entre [0,57] todo aquel que su patrón de vecinos sea U = 4 o superior será considerado ruido y se le asignará el valor 58. De esta forma podemos calcular el ejemplo anterior:



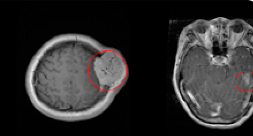
U = 3, por lo tanto, el pixel central que anteriormente era 6, será codificado como 58.

Para el paso del histograma consiste simplemente en contar cuantos pixeles fueron codificados como alguno de los valores entre [0,58] y asignarlo a uno de esos bins respectivamente (58 bins). Juntando estos 59 elementos tendremos un vector de la misma cantidad, el cual podremos comparar con el vector descriptor LBP de otra imagen y calculando la distancia euclidiana entre el uno y el otro, podremos decir si una imagen es igual o la misma que la otra.



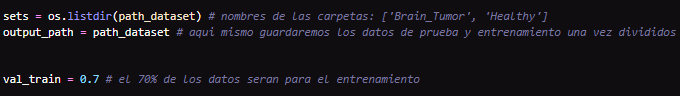
# TRATAMIENTO DATOS Y ENTRENAMIENTO DE MODELOS EN PYTHON

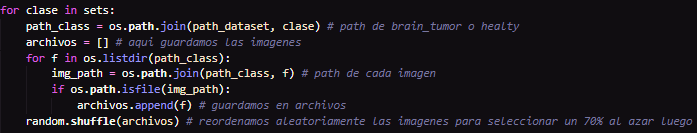
En este proyecto se entrenará una SVM para cada una de las características anteriormente mencionadas, de un dataset seleccionado de Kaggle sobre un problema de clasificación binaria, el set de datos consta de imágenes de radiografías cerebrales con tumores y sin tumor, el modelo debe reconocer y clasificar cada una y lanzar un diagnóstico (posible Cáncer o cerebro sano). En esta ocasión se ha decidido codificar con 1 los cerebros con tumores y 0 aquellos que estén sanos.

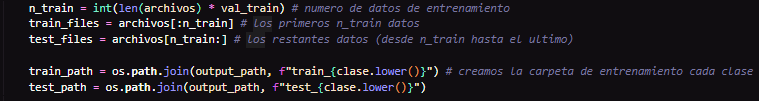


Al descargar el dataset y descomprimirlo, lanza 2 archivos .csv y una carpeta Brain Tumor Data Set, esta tiene dos carpetas con los cerebros sanos y con tumores. Los archivos csv dan información acerca de los metadatos de las imágenes.

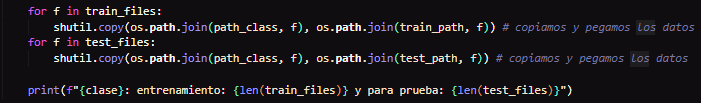
En las diferentes prácticas y actividades realizadas en el curso, se han tratado imágenes de 2 dimensiones (alto, ancho en escala de grises) y 3 dimensiones (alto, ancho y canal que normalmente es RGB o BGR), pero los archivos .csv advierte como hay imágenes que tienen 4 canales RGBA, por lo que estas imágenes deben ser eliminadas. Tomar en cuenta que se debe separar el set de datos en un 70% dedicados al entrenamiento y 30% al testear.







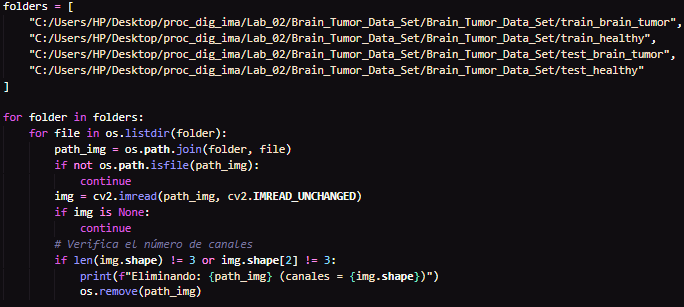


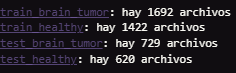




El código anterior, crea la carpeta para entrenamiento y testeo (70% y 30% respectivamente) y copiamos y pegamos los datos a estas carpetas.

Como se mencionó anteriormente, los archivos csv nos advertían de imágenes en escala de grises y con 4 canales, por lo que nos encargaremos de pasar carpeta por carpeta, revisar que datos cumplen con lo anterior y eliminarlas.





Una vez eliminados comprobamos que la cantidad suprimida no haya desbalanceado mucho los datos para seguir con la preparación de los dos modelos a entrenar.

*Entrenamiento de modelos*

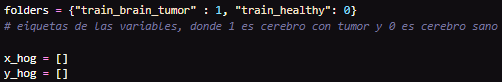
Se planea entrenar una SVM para el descriptor con las características HOG y otra para las LBP. A continuación, se mostrará el proceso con el que se entrenó con HOG, que seguirá prácticamente el mismo proceso con LBP.

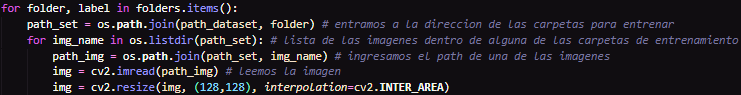
En primer lugar, debemos configurar el descriptor:

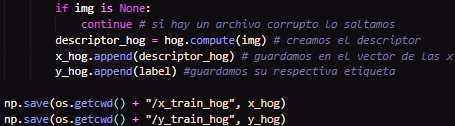


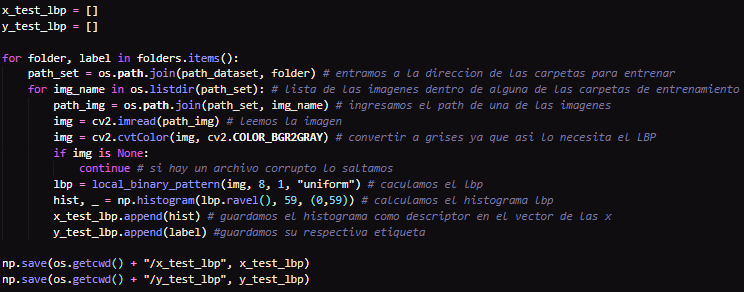


Una vez preparado el descriptor, podemos comenzar a clasificar los datos 1-cerebro con tumo y 0-cerebro sano

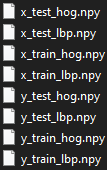




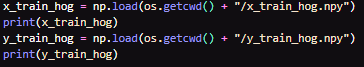




Cuando se han creado los datos de entrenamiento y testeo, no se debe olvidar de verificar que si estén los 8 archivos .npy.



Serán estos los datos que se usaran para el entrenamiento de la SVM. Cargando los anteriores datos, creando el Suporter Vector Classifier y entrenarlo:







Una vez entrenados ambos modelos, es posible analizar su precisión, falsos positivos y demás métricas.

*Accuracy*

Con el accuracy es posible hacer una idea de que tan “bueno” es el modelo en general a la hora de clasificar. Para calcular esta métrica se tienen las siguientes formulas:

. (1)

Donde n: Número total de muestras

: Etiqueta real de la muestra i

: Etiqueta predicha por el modelo

1{…}: Vale 1 si la condición de adentro se cumple, 0 en caso contrario

(2)

Donde TP: Verdaderos positivos

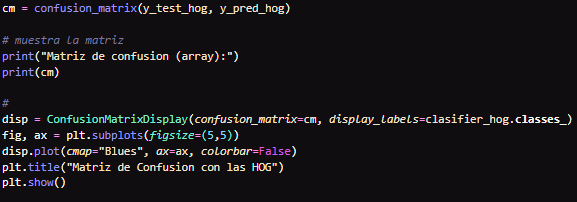
TN: Verdaderos negativos

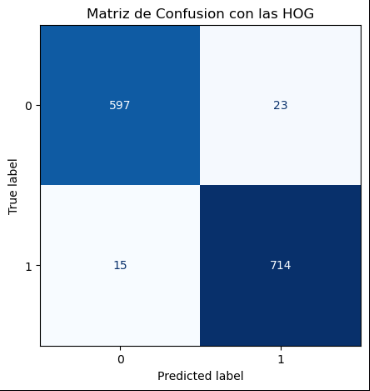
FP: Falsos positivos

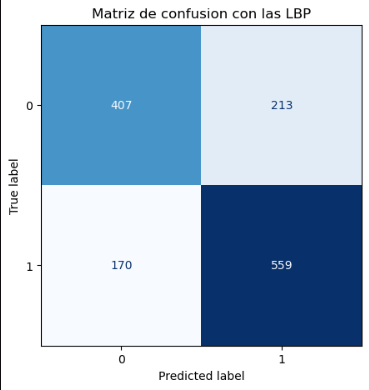
FN: Falsos negativos

Un falso positivo es aquel diagnostico que indica tumor a un cerebro sano y un falso negativo es aquel que predice un cerebro sano cuando en realidad presenta cáncer.

Para el cálculo de esta, es posible apoyarse de la matriz de confusión, la cual nos permite visualizar los valores de la ecuación (2).







La celda superior izquierda indica cuantos datos realmente sanos fueron predichos como sanos por el modelo, la celda a la derecha muestra cuantos datos sin tumor (0) el modelo los predijo como cancerígenos (1), es decir, Falsos positivos. Para las celdas de la segunda fila, permite visualizar cuantas muestras clasificadas con tumor fueron predichas como sanas (Falsos negativos) y cuantos datos cancerígenos el modelo logró predecirlos con exactitud, respectivamente.

Sustituyendo en la fórmula 2 obtenemos:

Accuracy\_hog = 0.97183098 => 97.183098%

Accuracy\_lbp = 0.71608598 => 71.608598%

*Precision*

La precision mide de todas las veces que el modelo predijo un cerebro con tumor (1) cuantas veces acertó.

. (3)

. (4)

Sustituyendo en la fórmula (4), el resultado de precisión de cada modelo es:

Precision\_hog = 0.96879240 => 96.879240%

Precision\_lbp = 0.72409326 => 72.409326%

Observe que, el modelo entrenado con las características HOG logra acertar con más precisión aquellos cerebros con tumores que el entrenado con las LBP.

*Recall*

La interpretación de este puede prestarse a confusiones con la precision, puesto que el Recall se refiere a que, de todos los casos reales de tumor, cuantos detectó el modelo.

. (5)

. (6)

Sustituyendo en la formula (6):

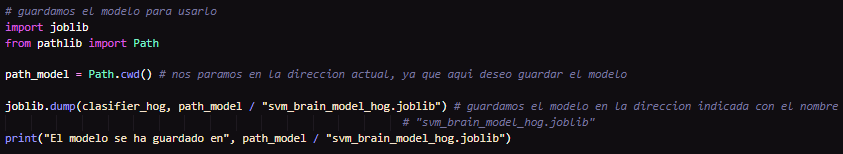
recall\_hog = 0.97942386 => 97.942386%

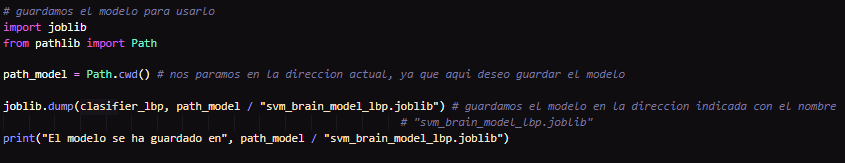
recall\_lbp = 0.76680384 => 76.680384%

Observe que, nuevamente el modelo entrenado con las características HOG tiene mejores resultados con respecto al entrenado con las LBP.

*Guardar modelos*

Una vez entrenados los modelos y luego de analizar sus métricas, estos deben ser guardados con el objetivo de ser utilizados en un software real, de esta manera evitamos entrenar el modelo en cada ocasión que se necesite usarlo.





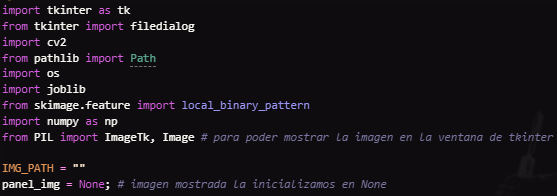
No olvidar que se debe verificar que el archivo para cada modelo se ha guardado correctamente con sus respectivos nombres y extensión .joblib.



# IMPLEMENTACIÓN DEL PROGRAMA EN PYTHON

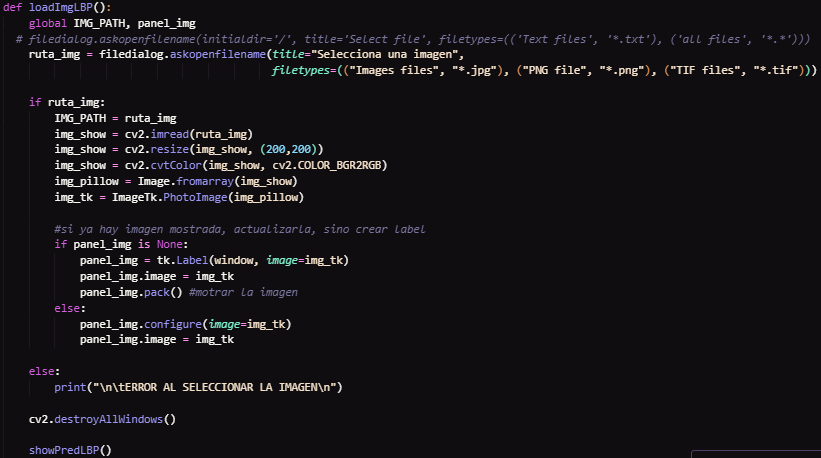
Para el uso de los modelos entrenados se ha creado una pequeña y sencilla interfaz con dos botones, uno para usar el modelo entrenado con las características HOG y el otro con las LBP, ambos permiten seleccionar una imagen del almacenamiento local y probarlo con el modelo deseado, luego de seleccionar la imagen esta se debe mostrar en la misma venta y un texto en la parte superior indicando el diagnostico (con tumor o sano).

Se hizo uso de Tkinter y sus funciones para realizar la interfaz gráfica y demás elementos de la ventana, al igual que la posibilidad de seleccionar imágenes del almacenamiento local.

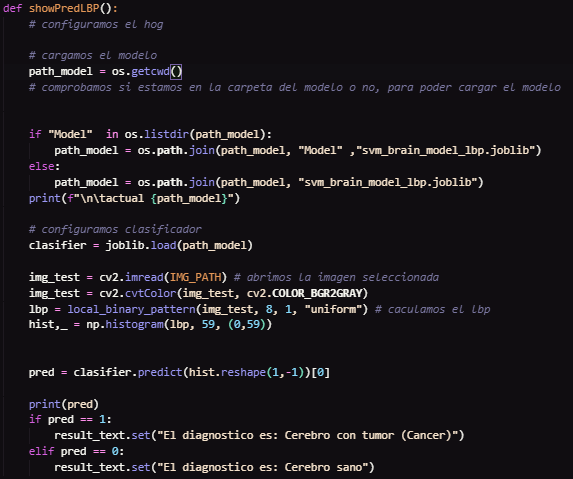


Se importa lo necesario e inicializan las variables que guardan la dirección de la imagen seleccionada y la misma imagen a ser mostrada.

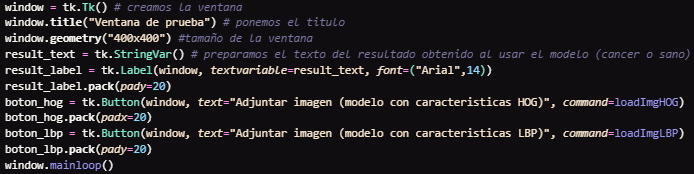
Para cada modelo se crea una función encargada de cargar la imagen seleccionada y entrará a otra función que se encarga de mostrarla e indicar el diagnostico en la ventana.



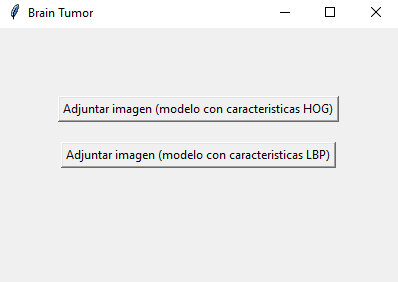
Recordar que se debe indicar que imágenes serán aceptadas para el programa, luego de esto podemos definir la función encargada de mostrar la imagen seleccionada junto con su diagnostico



El siguiente paso es crear la ventana junto con sus elementos y que se muestren al ejecutar el programa, donde cada botón lama a la respectiva función para cargar la imagen usando un modelo u otro.



Al ejecutar el Código se mostrará la siguiente ventana y se podrá usar perfectamente cada modelo según sea seleccionado



# RESULTADOS.

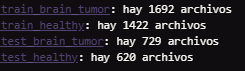
Al repasar las métricas obtenidas con anterioridad para cada modelo y con ayuda de las matrices de confusión podemos obtener con certeza los falsos positivos y negativos y los verdaderos positivos y negativos.

|  |  |
| --- | --- |
|  |  |

A partir de esto podemos realizar una comparación entre ambos modelos entrenados con anterioridad.

|  |  |  |
| --- | --- | --- |
|  | Model con HOG | Modelo con LBP |
| Verdaderos negativos | 597 | 407 |
| Verdaderos positivos | 714 | 559 |
| Falsos negativos | 15 | 170 |
| Falsos positivos | 23 | 213 |
| Accuracy | 0.97183098 | 0.71608598 |
| Presicion | 0.96879240 | 0.72409326 |
| Recall | 0.97942386 | 0.76680384 |

El accuracy en el caso del modelo HOG es mayor que el del LBP, esto nos da a entender que, con las HOG, logró predecir con más precision tanto los cerebros con tumores como los sanos. Es posible que esta métrica pueda ser desbalanceada al tener mas datos de un tipo que el otro (tumores o sanos), pero la diferencia de datos tanto para entrenar como para testeo no es muy grande, por lo que podemos decir con alta confianza que el modelo entrenado con las características HOG es capaz de clasificar correctamente mas cerebros con tumores y sanos en comparación del modelo con las LBP.



La métrica precision mide, de todas las veces que el modelo predijo un tumor, cuantas de esas predicciones fueron correctas, con esta definición es posible saber que tan frecuentes se presentarán falsos positivos. Si la precision es cercana a 1 entonces hay pocos falsos positivos, si es cercano a cero el modelo tendrá una cantidad considerable de falsos positivos.

En este caso, la precision del modelo con las HOG genera mas confianza a la hora de tratar a los falsos positivos.

El recall que tambien es llamado sensibilidad o true positive rate mide, del total de casos reales con tumor, que proporción el modelo logró predecirlos con éxito, esto quiere decir que, un recall cercano a 1 significa que el modelo detecta casi todos los verdaderos positivos y hay una baja tasa de falsos negativos (diagnosticar sano cuando en realidad presenta tumor), mientras que un recall bajo o cercano a cero presentaría muchos falsos negativos ya que se le “escapan” muchos positivos.

En esta última métrica nuevamente el modelo con las HOG es más alto que con LBP, por lo que con HOG nos da una menor tasa de Falsos negativos reduciendo un gran riesgo para los pacientes.

# CONCLUSIONES

En este proyecto se demostró la importancia de realizar un análisis exploratorio y tratamiento de los datos, el planteamiento de un correcto descriptor dependiendo de las características para el entrenamiento de una SVM y el análisis de las diferentes métricas para identificar el mejor de los modelos. Además, el uso de buenas prácticas como guardar los modelos para probarlos en un programa con el objetivo de facilitar el proceso de predicción de cada dato o muestra.

Se logra evidenciar como para el ejercicio de predicción de tumores en imágenes de rayos x de cerebros, entrenando una SVM con características HOG es más preciso y reduce la tasa de falsos negativos a comparación de un modelo con características LBP, con la condición de que ambos sean tratados con los mismos parámetros.

# REFERENCIAS

[1] Domingo Mery “Características de intensidad, LBP (Local Binary Patterns) y reconocimiento facial” https://www.youtube.com/watch?v=7woR1nDc9r0

[2] Scikit Learn “sklearn.svm” https://scikit-learn.org/stable/api/sklearn.svm.html#module-sklearn.svm

[3] Ramon Ariel Ivan Muñoz Corona “

Python OpenCV - Detección de Peatones con HOG Descriptor y SVM” https://www.youtube.com/watch?v=Kz5PMCmLrHg

[4] Datacamp “Tutorial sobre máquinas de vectores de soporte con Scikit-learn” https://www.datacamp.com/es/tutorial/svm-classification-scikit-learn-python

[5] Domingo Mrey “Características de intensidad, LBP (Local Binary Patterns) y reconocimiento facial” https://www.youtube.com/watch?v=7woR1nDc9r0

[6] Pyimagesearch “Local Binary Patterns with Python & OpenCV” https://pyimagesearch.com/2015/12/07/local-binary-patterns-with-python-opencv/

[7] Domingo Mery “

09 Reconocimiento de Patrones: HoG + Introducción a SIFT” https://www.youtube.com/watch?v=mcqkE-gzUHM

1. [↑](#footnote-ref-1)