



INSTITUTO POLITECNICO NACIONAL
ESCUELA SUPERIOR DE COMPUTO

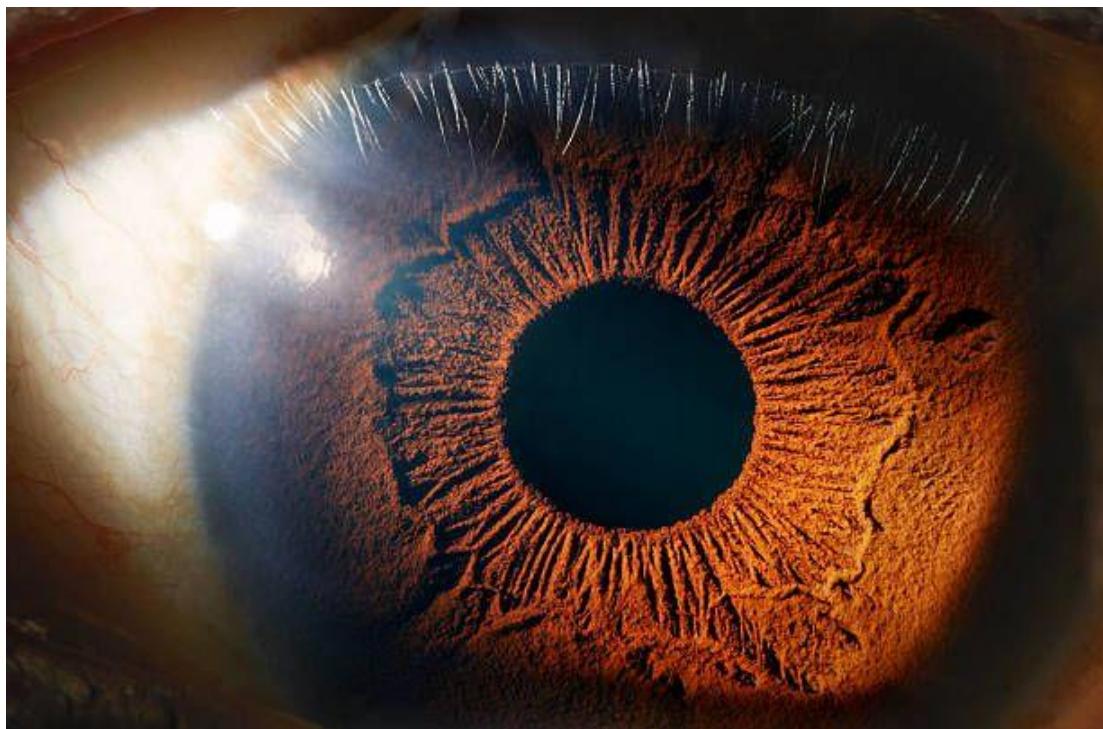


IMAGE ANALYSIS
MODELOS DE COLOR

TABLA COMPARATIVA

ESCUADRON 241:

- PORTOCARRERO RODRIGUEZ HABID
- MUZQUIS PALACIO ERNESTO
- MARTINEZ LOPEZ GERARDO ESTEBAN
- SOLANO MUÑOZ ARTURO



Contenido

Introducción	4
Pillow.....	4
NumPy.....	4
Matplotlib.....	4
Energía.....	5
Entropía.....	5
Asímetria.....	5
Media.....	5
Varianza.....	6
Los siguientes son propiedades del histograma:	7
Resultados Obtenidos:	8
Desarrollo.....	9
1) Resumen Ejecutivo.....	9
2) Objetivo(s) y Alcance	9
3) Contexto y Motivación.....	9
4) Dataset y Fuentes de Datos	9
5) Metodología / Pipeline	10
Diagrama simple	10
6) Arquitectura del Proyecto (Estructura de Carpetas).....	10
7) Instalación y Requisitos.....	10
8) Configuración.....	11
9) Ejecución / Uso	12
10) API Pública (Funciones/Clases Clave)	12
11) Evaluación y Métricas	13
12) Experimentos	13
13) Resultados / Visualizaciones	14
14) Rendimiento y Optimización.....	14
15) Pruebas (Testing).....	14

16) Registro y Manejo de Errores	14
17) Limitaciones y Trabajo Futuro.....	15
18) Consideraciones Éticas y de Privacidad	15
19) Cómo Reproducir	15
20) FAQ / Troubleshooting	15
21) Licencia y Créditos	16
22) Anexos.....	16
Conclusión.....	17

Introducción

El análisis de imágenes tiene como objetivo extraer información significativa a partir del contenido de una imagen, lo que facilita la comparación y clasificación de imágenes. Este enfoque es valioso en aplicaciones donde la evaluación humana resulta insuficiente por factores como la necesidad de procesamiento masivo, la incapacidad el ojo humano, entre otros.

La importancia de este análisis radica en su capacidad para convertir percepciones visuales en datos matemáticos manipulables. Mientras que el ojo humano puede percibir cualidades como "brillante", "contrastada" o "detallada", el análisis computacional permite asignar valores numéricos específicos a estas características, estableciendo patrones y umbrales que pueden ser utilizados para la toma de decisiones automatizada.

Anteriormente se ha resuelto con librerías el código presentado en la práctica en las cuales se incluyen las siguientes:

Pillow.

Algunas de sus capacidades principales abarcan la lectura y escritura de más de 30 formatos de imagen diferentes, operaciones geométricas como manipulación de píxeles a nivel básico. Pillow es ideal para la etapa de preprocesamiento, permitiendo la extracción de metadatos y la conversión entre modos de color.

NumPy.

Proporciona estructuras de datos eficientes para representar imágenes como arreglos multidimensionales. La importancia de NumPy está en su implementación vectorizada de operaciones matemáticas que permite procesar millones de píxeles simultáneamente sin necesidad de bucles explícitos.

También facilita el cálculo de estadísticas descriptivas mediante funciones optimizadas para media, varianza, desviación estándar, entre otras.

Matplotlib.

Matplotlib se especializa en la visualización de resultados, ofreciendo herramientas comprehensivas para la generación de gráficos. Su módulo pyplot permite crear histogramas interactivos, visualizar distribuciones de probabilidad y comparar múltiples datasets de manera intuitiva.

En las propiedades de las imágenes se tiene lo siguiente:

Energía.

Representa la concentración de la distribución de intensidades. Se calcula como la suma de los cuadrados de las probabilidades de cada nivel de intensidad en el histograma. Una energía elevada indica que la imagen está dominada por pocos tonos específicos. Por el contrario, una energía baja sugiere una distribución equilibrada entre múltiples tonos.

$$E = \sum_{g=0}^{L-1} (P(g))^2$$

Entropía.

La entropía es el desorden presente en una imagen demostrando la impredecibilidad. Se calcula mediante la sumatoria de las probabilidades multiplicadas por el logaritmo de dichas probabilidades. Una entropía alta es una escena rica en detalles y variabilidad. Una entropía baja señala redundancia y predictibilidad.

$$e = -\sum_{g=0}^{L-1} P(g) \log_2 [P(g)]$$

Asimetría

Es el balance de la distribución del histograma respecto a su valor central. Una asimetría cercana a 0 denota una distribución simétrica, donde los valores se distribuyen equilibradamente alrededor de la media. Una asimetría positiva indica que la cola de la distribución se extiende hacia valores altos de intensidad. Una asimetría negativa sugiere concentración en tonos brillantes, común en imágenes sobreexpuestas.

$$a = \sum_{g=0}^{L-1} (g - \bar{g})^3 P(g)$$

Media.

Representa el brillo promedio de la imagen. Valores bajos de media establecida son imágenes oscuras, mientras que valores altos de media son imágenes brillantes. La media sirve como referencia primaria para operaciones de normalización de iluminación y corrección gamma.

$$\bar{g} = \sum_{g=0}^{L-1} g P(g) = \sum_i \sum_j \frac{I(i,j)}{M}$$

Varianza.

La varianza significa la dispersión de los valores de intensidad alrededor de la media, funcionando como indicador de contraste intrínseco. Una varianza alta señala una amplia distribución de valores de intensidad. Una varianza baja indica compresión del rango tonal o bajo contraste.

$$\sigma^2 = \sum_{g=0}^{L-1} (g - \bar{g})^2 P(g)$$

El proceso comienza cargando la imagen con OpenCV, que por defecto la interpreta en formato BGR. Para mostrarla correctamente suele convertirse a RGB, además de revisar que el archivo no esté dañado, ajustar sus dimensiones y, si es necesario, transformarlo a escala de grises.

Cuando la imagen se mantiene en color, el análisis se realiza por separado en cada canal (rojo, verde y azul). Para cada uno se genera un histograma que refleja la frecuencia de los niveles de intensidad entre 0 y 255. Estos histogramas luego se normalizan para obtener distribuciones de probabilidad que servirán como base para los cálculos estadísticos.

Los valores numéricos se interpretan contrastándolos con umbrales previamente definidos. Esto permite clasificar imágenes según sus características.

Un histograma es una representación gráfica de la distribución de frecuencias de un conjunto de datos. Su estructura consiste en dividir el rango total de valores en intervalos llamados, luego se cuentan los elementos de cada conjunto que caen dentro de cada intervalo.

El resultado se visualiza como un conjunto de barras: la base de cada barra corresponde al intervalo de valores, y la altura representa la frecuencia o cantidad de ocurrencias. En este caso, una barra corresponde a la luminiscencia, una al color rojo, otra al color verde y la última al color azul.

Un histograma muestra la distribución de intensidades de los píxeles. En una imagen en escala de grises, cada valor de 0 a 255 (negro y blanco respectivamente) tiene asociado una frecuencia que indica cuántos píxeles presentan ese nivel de brillo. En imágenes a color, el histograma puede calcularse de forma independiente para cada canal en RGB ofreciendo una visión más detallada de la composición cromática de la imagen.

Los siguientes son propiedades del histograma:

- *Distribución*: muestra cómo se reparten los datos en el rango de valores disponibles. Indica si predominan los tonos oscuros o claros. Una concentración hacia la izquierda significa que la imagen es más oscura; hacia la derecha, que es más clara.
- *Rango*: muestra valores desde 0 hasta 255.
- *Frecuencia*: es la distribución de los datos en todo el rango del histograma. Una distribución uniforme sugiere intensidades balanceadas; una sesgada significa predominancia de tonos oscuros o claros; varios picos representan zonas irregulares como lo son muy brillantes o muy oscuras.
- *Eje X*: es el eje horizontal del histograma. Contiene valores. Se establece en función del rango de los valores representados.
- *Eje Y*: es el eje vertical del histograma. Contiene valores. Se establece en función del rango de los valores representados.

En el análisis de imágenes, un histograma sirve para la evaluación de brillo y contraste mostrándose simplemente si la imagen está oscura, clara o balanceada. Eso se puede ver en el canal de luminiscencia que, de acuerdo con ello, mostrará las propiedades de brillo de una imagen. Lo mismo ocurre con la exposición de una imagen si es que la concentración de frecuencias está a la izquierda o a la derecha siendo subexpuestas o sobreexpuesta respectivamente.

Un histograma se puede comparar con otros histogramas para conocer las diferentes frecuencias y concentraciones de datos en distintos histogramas después de someter a la imagen a varias modificaciones (ya sea RGB, binarización, entre otros). También se pueden detectar anomalías como desviaciones inusuales o falta de datos en un rango específico.

Resultados Obtenidos:

- YIQ: La imagen fue separada en sus componentes de luminancia y crominancia. El componente Y mostró la estructura en escala de grises, mientras que I y Q representaron los componentes de color en los rangos de crominancia. Este modelo es útil para la compresión de imágenes y la transmisión de video, ya que puede reducir la cantidad de información sin perder detalles importantes en la luminancia.
- CMY: La conversión a este modelo mostró cómo los colores se pueden representar de manera sustractiva. Al ser un modelo de color usado en impresoras, observamos cómo se genera el color al mezclar los tres colores básicos (Cian, Magenta y Amarillo). Este modelo, aunque no es intuitivo para la visualización, es esencial para la tecnología de impresión.
- HSV: La conversión a HSV permitió observar de manera más visualmente comprensible cómo se estructuran los colores en términos de matiz, saturación y valor. El modelo HSV es ampliamente utilizado en software de edición de imágenes y es muy intuitivo para los usuarios debido a su alineación con la percepción humana del color.
- Binarización: Aplicando un umbral, la imagen fue convertida en una imagen binaria (blanco y negro), lo que permite realizar segmentaciones y análisis de objetos en la imagen de manera más fácil.

Desarrollo

1) Resumen Ejecutivo

Este proyecto carga una imagen de ejemplo (imagen1.jpg, una rosa), la convierte a arreglos NumPy y aplica transformaciones básicas de visión: visualización RGB, conversión a escala de grises, conversión a HSV, cálculo de histogramas y binarización por umbral configurable (por defecto 128). Los resultados se muestran con Matplotlib y sirven como base para prácticas posteriores (segmentación y mejoras de contraste). Se documenta el entorno usado (Anaconda/Nirvana) y la solución a errores típicos de dependencias.

2) Objetivo(s) y Alcance

- Objetivo general: Implementar y documentar un flujo mínimo de análisis de imágenes en Python para cargar, transformar y segmentar una imagen mediante umbralización.
 - Objetivos específicos:
 - Cargar imágenes con Pillow y convertirlas a numpy.ndarray.
 - Visualizar canales y resultados con Matplotlib.
 - Implementar conversión RGB→Grayscale y RGB→HSV.
 - Aplicar umbral fijo (parámetro umbral) y discutir alternativas (Otsu).
 - Dejar instrucciones reproducibles del entorno y dependencias.
 - Alcance: Se trabaja con una imagen local (imagen1.jpg). No incluye entrenamiento de modelos ni procesamiento en lote.
-

3) Contexto y Motivación

- Problema a resolver (por qué es relevante).
 - Casos de uso (laboratorio, seguridad, industria, etc.).
-

4) Dataset y Fuentes de Datos

- Fuente: archivo local imagen1.jpg ubicado en la carpeta Practica1/.

- Formato: JPEG (960x1280 aprox.).
 - Notas: Para reproducir, colocar cualquier imagen JPG con ese nombre en la misma carpeta del script o ajustar la ruta en el código (ver sección Ejecución/ Uso).
-

5) Metodología / Pipeline

1. Carga: PIL.Image.open(path).convert("RGB") → numpy.array.
2. Visualización inicial: mostrar RGB y canales individuales.
3. Conversión a Grayscale: luminancia BT.601 con OpenCV (cv2.cvtColor).
4. Binarización: umbral fijo (cv2.threshold).
5. Modelos de color: YIQ, CMY y HSV con skimage.color.
6. Histogramas interactivos: cada figura incluye un botón "Histogramas" que abre una ventana única con los histogramas de los ítems relevantes de esa figura.
7. Salida: figuras y ventanas de histogramas (bloqueantes) para cada etapa.

Diagrama simple

Imagen1.jpg → Carga (Pillow) → Arrays (NumPy) → RGB/Gris → Umbral → {YIQ, CMY, HSV} → Visualización + Botón de histogramas

6) Arquitectura del Proyecto (Estructura de Carpetas)

vision-proyecto/

```
|-- src/
|   |-- análisis_imagenes.py
|   |-- Documentacio_practica1_escuadron241.docx
|   |-- imagen1.jpg
```

7) Instalación y Requisitos

Hardware: PC con Windows.

Software: VS Code + Anaconda. Entorno: Nirvana (Python 3.9.18).

Paquetes clave y versiones (probadas):

```
numpy==1.26.4  
matplotlib==3.7.3  
pillow==11.1.0  
# opcionales útiles  
opencv-python==4.9.0.80  
scikit-image==0.21.0  
scipy==1.10.1
```

Instalación (usando el Python del entorno Nirvana):

```
"C:\Users\Maria Guadalupe\anaconda3\envs\Nirvana\python.exe" -m pip install --no-cache-dir  
numpy==1.26.4 matplotlib==3.7.3 pillow  
# opcional  
"C:\Users\Maria Guadalupe\anaconda3\envs\Nirvana\python.exe" -m pip install --no-cache-dir  
opencv-python==4.9.0.80 scikit-image==0.21.0 scipy==1.10.1
```

8) Configuración

configs/default.yaml (ejemplo)

input:

```
source: "data/samples/video.mp4" # ruta de video o carpeta de imágenes  
img_size: 640
```

model:

```
weights: "models/best.pt"  
conf_thres: 0.25  
iou_thres: 0.45
```

tracking:

```
enabled: true
```

```
max_age: 30
```

```
min_hits: 3
```

output:

```
save_video: true
```

```
save_csv: true
```

```
out_dir: "runs/exp1"
```

9) Ejecución / Uso

Opción A (recomendada): ejecutar desde la carpeta Practica1/:

```
cd "C:\Users\Maria Guadalupe\Desktop\GitHub\vision-proyecto\Practica1"
```

```
"C:\Users\Maria Guadalupe\anaconda3\envs\Nirvana\python.exe" .\nalysis_imagenes.py
```

Opción B: pasar ruta absoluta en el código (ejemplo):

```
demo(r"C:\Users\Maria Guadalupe\Desktop\GitHub\vision-proyecto\Practica1\imagen1.jpg",  
umbral=128)
```

(Opcional) Llamada robusta dentro del script:

```
import os
```

```
BASE = os.path.dirname(os.path.abspath(__file__))
```

```
path_img = os.path.join(BASE, "imagen1.jpg")
```

```
demo(path_img, umbral=128)
```

10) API Pública (Funciones/Clases Clave)

- `to_uint8(img)`: normaliza cualquier imagen a uint8 0–255 sin alterar su apariencia visual.

- `plot_hist_on_axes(ax, img_u8, name)`: dibuja histograma; una curva para gris, tres para RGB.
 - `open_hist_grid(images_dict, title, hist_whitelist)`: abre una sola ventana con un grid de histogramas para claves seleccionadas.
 - `figure_with_hist_button(images_dict, title, layout, cmmaps, hist_whitelist)`: crea figura con imágenes y un botón "Histogramas" que invoca `open_hist_grid`.
 - `cargar_rgb(path)`: carga con Pillow en RGB uint8.
 - `separar_canales(rgb)`: devuelve matrices R, G, B.
 - `a_grises_bt601(rgb)`: conversión a gris por BT.601 usando OpenCV.
 - `binarizar(gray_u8, t)`: umbral fijo con OpenCV.
 - `convertir_yiq(rgb_u8)`: usa skimage para obtener Y, I, Q y un compuesto visual.
 - `convertir_cmy(rgb_u8)`: modelo sustractivo simple CMY (255-R/G/B).
 - `convertir_hsv(rgb_u8)`: HSV con skimage.color y reconstrucción HSV→RGB para vista.
 - `demo(path, umbral)`: orquesta el flujo completo en 7 pasos con figuras consecutivas y sus histogramas.
-

11) Evaluación y Métricas

- Clasificación: accuracy, precision, recall, F1.
- Detección: mAP@[.5:.95], IoU, FPS.
- Tracking: MOTA, MOTP, IDF1, ID switches.
- Curvas: PR, ROC, confusión.

Incluye tablas y gráficas (pega imágenes o enlaza a runs/).

12) Experimentos

- Hipótesis y variables (ablation): *sin* sustracción de fondo vs *con*, img_size 640 vs 960, etc.
- Configuraciones probadas (tabla).
- Resultados por experimento (tabla).

- Discusión breve.
-

13) Resultados / Visualizaciones

- Capturas de pantalla (detecciones, trayectorias).
 - Videos anotados.
 - Ejemplos de fallos (falsos positivos/negativos) y explicación.
-

14) Rendimiento y Optimización

- FPS promedio (CPU/GPU), uso de memoria.
 - Técnicas: batch inferencing, half precision, ONNX/TensorRT, multithreading.
 - Perfilado (line-profiler, cProfile) y hallazgos.
-

15) Pruebas (Testing)

- Unitarias: pytest (I/O, preprocessado, parsers).
- Integración: pipeline end-to-end con un video corto.
- Determinismo: fijar semillas aleatorias.

tests/test_io.py (ejemplo)

```
import pytest  
  
from src.utils.io import load_video
```

```
def test_load_video_not_found():  
    with pytest.raises(FileNotFoundError):  
        load_video("no_existe.mp4")
```

16) Registro y Manejo de Errores

- Logging con niveles (INFO/DEBUG/ERROR).

- Mensajes claros para archivos faltantes, cámaras no disponibles, pesos incompatibles.
- Mecanismos de retry y tiempo de espera.

logging (snippet)

```
from loguru import logger

logger.add("runs/app.log", rotation="10 MB")

logger.info("Inicio de la aplicación")
```

17) Limitaciones y Trabajo Futuro

- Condiciones donde falla (iluminación, occlusiones, velocidad).
 - Mejoras planeadas (más datos, finetuning, nuevos algoritmos, mejores métricas).
-

18) Consideraciones Éticas y de Privacidad

- Tratamiento de datos personales en video.
 - Almacenamiento seguro y retención.
 - Minimización de datos y anonimización (blur de rostros, p.ej.).
-

19) Cómo Reproducir

1. Clonar el repo o copiar la carpeta del proyecto.
 2. En VS Code, seleccionar intérprete: anaconda3/envs/Nirvana/python.exe.
 3. Instalar dependencias probadas (ver Instalación y Requisitos).
 4. Colocar imagen1.jpg en Practica1/ (o ajustar la ruta en el script).
 5. Ejecutar el comando de la sección Ejecución / Uso.
-

20) FAQ / Troubleshooting

- ModuleNotFoundError: No module named 'PIL' → instalar Pillow en el entorno activo:
python -m pip install pillow.

- Choque NumPy 2.x vs Matplotlib ("_ARRAY_API not found" / "multiarray failed to import") → usar numpy==1.26.4 y matplotlib==3.7.3.
 - ImportError de backend Qt5Agg → instala PyQt5 o cambia a TkAgg en la línea de backend.
 - FileNotFoundError: 'imagen1.jpg' → ejecutar desde Practica1 o usar ruta absoluta / solución robusta con os.path.
 - Se abren muchas ventanas de histogramas → el botón abre una sola ventana por figura; ciérrala antes de pasar a la siguiente.
-

21) Licencia y Créditos

- Licencia del código (MIT/GPL/...)
 - Créditos a datasets/librerías.
-

22) Anexos

- Fórmulas (IoU, mAP, MOTA).
 - Diagramas adicionales.
 - Tablas extensas de resultados.
-

(Opcional) Documentación Autogenerada

- Docstrings + Sphinx/MkDocs:

```
# Sphinx
```

```
pip install sphinx sphinx-rtd-theme
```

```
sphinx-quickstart docs
```

```
# Configura theme y autodoc; luego:
```

```
make html
```

```
# MkDocs
```

```
pip install mkdocs mkdocs-material  
mkdocs new .  
mkdocs serve  
(Opcional) Calidad de Código  
pip install black isort flake8 pre-commit  
pre-commit install
```

Conclusión

La práctica ha permitido comprender la utilidad de diferentes modelos de color para representar y procesar imágenes. A través de la conversión de una imagen del espacio de color RGB a otros modelos como YIQ, CMY y HSV, se han observado cómo las imágenes pueden ser descritas y manipuladas de manera más eficiente dependiendo del contexto y del propósito de la tarea.

- **YIQ** es útil en **compresión y transmisión de video**, ya que separa la luminancia de la crominancia, permitiendo reducir el ancho de banda sin perder detalles importantes del brillo.
- **CMY** es esencial en la **impresión a color**, donde se utilizan tintas cian, magenta y amarilla para generar colores.
- **HSV** es particularmente útil en aplicaciones de **edición de imágenes y segmentación**, ya que su estructura es más intuitiva y alineada con la percepción humana de los colores.

El uso de **histogramas** como herramienta de análisis visual de los valores de intensidad en cada uno de los componentes de los modelos de color permite una comprensión más profunda de la distribución y el contraste de los colores. En general, los modelos de color permiten trabajar de manera más eficiente en diversas aplicaciones, desde la compresión de video hasta la edición de imágenes y la impresión.

Esta práctica demuestra cómo los diferentes modelos de color se pueden aplicar dependiendo de las necesidades de procesamiento de la imagen, y cómo cada modelo tiene ventajas y limitaciones según el contexto en el que se utilice.