

**Encriptación de imágenes**

Modelado y procesamiento de imágenes  
Mtro. Emmanuel Ovalle Magallanes  
Tercer parcial  
Proyecto final  
512

**Autores:**

Diego Camarena Gutiérrez

Juan Pablo Hernández Ponce de León

Iván Vivas García

Jared Yahir Hernández Delgado

Contenido

[Introducción 3](#_Toc183562336)

[Objetivo principal 3](#_Toc183562337)

[Objetivos secundarios 3](#_Toc183562338)

[Glosario 4](#_Toc183562339)

[Explicación del modelo utilizado 4](#_Toc183562340)

[Ejemplo 4](#_Toc183562341)

[esteganografía.py 5](#_Toc183562342)

[ocultar\_imagen 5](#_Toc183562343)

[extraer\_imagen 8](#_Toc183562344)

[calcular\_metricas 11](#_Toc183562345)

[utils.py 14](#_Toc183562346)

[normalizar\_imagen 14](#_Toc183562347)

[guardar\_imagen 14](#_Toc183562348)

[cargar\_imagen 15](#_Toc183562349)

[mostrar\_metricas 15](#_Toc183562350)

[Interfaz 16](#_Toc183562351)

[Referencias 19](#_Toc183562352)

# Introducción

Este proyecto consiste en un programa desarrollado en Python con la funcionalidad de encriptar una imagen dentro de otra, es decir, encriptar una imagen oculta dentro de una imagen “señuelo”, permitiendo almacenar la información de una imagen dentro de otra, modificando los bits menos significativos con los más significativos de la imagen oculta.

# Objetivo principal

El objetivo de este proyecto es generar una solución que permita ocultar y mostrar imágenes en función de los bits utilizados.

# Objetivos secundarios

1. Permitir aplicar la esteganografía a imágenes de diferentes dimensiones.

2. Permitir aplicar la esteganografía a imágenes con diferentes modos de color como RGB y blanco y negro.

3. Mostrar las métricas de error (MSE y PSNR) sobre la imagen.

# Glosario

**(LSB) Bits menos significativos:**

Los bits menos significativos son los últimos n bits de un numero binario leyéndolo de derecha a izquierda.

**(MSB) Bits más significativos:**

Los [bits más significativos](#Bit_mas_significativo) son los primeros n bits de un numero binario leyéndolo de derecha a izquierda.

# Explicación del modelo utilizado

El modelo utilizado para esta aplicación se basa en el manejo de [los bits menos significativos](#Bit_menos_significativo) de la imagen de portada y los [bits más significativos](#Bit_mas_significativo) de la imagen oculta, es decir, toma los valores de los bits menos significativos de cada píxel y los sustituye con los valores más significativos de la imagen a ocultar.

## Ejemplo

*Considerando como valor un píxel de una imagen a color con 4 canales.*

**n = 4**

**Imagen portada (128, 25, 239):**Binario(10000000, 00011001, 11101111)  
[Bits menos significativos:](#Bit_menos_significativo) (0000, 1001, 1111)

**Imagen oculta (23, 95, 115):**Binario(00010111, 01011111, 01110011)  
[Bits más significativos:](#Bit_mas_significativo) (0001, 0101, 0111)

# esteganografía.py

## ocultar\_imagen

Esta función se encarga de ocultar la imagen secreta dentro de la imagen de portada, a continuación se explican todos los procesos que la integran.

1. **Conversión**

Para poder aplicar la esteganografía se necesita que la imagen este en 8 bits, para esto se utiliza una función integrada de la librería skimage llamadas img\_as\_ubyte.

    # Asegurar que las imágenes sean de 8 bits

    portada = img\_as\_ubyte(imagen\_portada.copy())

    secreta = img\_as\_ubyte(imagen\_secreta.copy())

Sustitución de valores:

En la imagen de portada se sustituyen los bits más relevantes en las posiciones de los bits menos relevantes.

**Imagen de portada** (10000000, 00011001, 11101111) **Imagen resultante** (10000001, 00010101, 11100111)

1. **Creación de máscaras para sustituir los valores correspondientes**

   # Crea la máscara para los bits menos significativos elegidos

    mascara\_portada = 256 - (2\*\*bits)

    # Crea la máscara para los bits más significativos

    mascara\_secreta = 2\*\*bits - 1

Básicamente crea una máscara basada en los bits a utilizar para los bits menos significativos y para los más significativos, dejando libres los bits correspondientes en cada caso.

Ejemplo de uso

nBits = 4

**Mascara para los bits menos significativos.**  
11111111 - (2^nBits) = 11110000

**Mascara para los** [bits más significativos](#Bit_mas_significativo)**.**

2^nBits – 1= 00001111

1. **Limpieza de los bits menos significativos de la portada**

Se aplica la operación and & bit a bit, en este caso entre la portada y la máscara, para limpiar los bits menos significativos en cada píxel en la portada y dejarlos utilizables para ingresar los [bits más significativos](#Bit_mas_significativo) de la imagen secreta.

# Limpiar los LSB de la imagen de portada

    portada\_limpia = portada & mascara\_portada

Ejemplo de uso

(128) resultado (128)  
10000000 & 11110000 = 10000000

(25) resultado (16)  
00011001 & 11110000 = 00010000

(239) resultado (224)  
11101111 & 11110000 = 11100000

1. **Extracción de los bits más significativos**

# Preparar los bits de la imagen secreta

    bits\_secreta = (secreta >> (8 - bits)) & mascara\_secreta

En este caso, dependiendo de los bits elegidos, se desplaza dicha cantidad sobre la imagen secreta, y sobre esos bits se aplica la máscara para colocar los valores de los bits más significativos dentro de los menos significativos y así poderlos aplicar sobre la portada.

Ejemplo de uso

*Considerando que desplazamos 4 bits*

(23) resultado (1)  
00010111 >> (4) 00000001  
00000001 & 00001111 = 00000001

(23) resultado (5)  
01011111 >> (4) 00000101  
00000101 & 00001111 = 00000101

(23) resultado (7)  
01110011 >> (4) = 00000111  
00000111 & 00001111 = 00000111

1. **Combinar las imágenes**

# Combinar las imágenes

    estego = portada\_limpia | bits\_secreta

En este caso solo se utiliza el operador OR bit a bit para combinar las dos imágenes con los bits limpios para evitar colisiones de información.

Ejemplo de uso

*Considerando que desplazamos 4 bits*

10000000 | 00000001 = 10000001

00010000 | 00000101 = 00010101

11100000 | 00000111 = 11100111

Imagen inicial (128, 25, 239)

Imagen final (129, 21, 231)

## extraer\_imagen

Esta función se encarga de extraer la imagen oculta dentro de una imagen de portada, para esto se tienen que seguir los siguientes pasos.

1. **Asegurar que la imagen este en 8 bits**

Crucial para poder llevar a cabo las operaciones sobre cada capa de las imágenes.

# Asegurar que la imagen esté en 8bits

    estego = img\_as\_ubyte(imagen\_estego.copy())

1. **Se crea la máscara para extraer los bits menos significativos**

Funciona igual que en la función que utilizamos para codificar la imagen, donde crea una mascara de la cantidad de bits definidos por el usuario.

    # Crear máscara para extraer los LSB

    mascara = 2\*\*bits - 1

Ejemplo de uso

*Considerando que desplazamos 4 bits*

2^4 - 1 = 15

15 = 00001111

1. **Extraer los bits ocultos**

En este caso se aplica el operador AND para poder extraer los bits de la imagen oculta, eliminando los bits que no son los menos significantes seleccionados.

    # Extraer los bits ocultos

    extraida = (estego & mascara)

Ejemplo de uso

*Considerando que desplazamos 4 bits*

10010011 & 00001111 = 00000011

1. **Escalar los valores de los bits menos significativos para convertirlos en los más significativos.**

Esta sección básicamente recorre o escala los bits menos significativos a los [bits más significativos](#Bit_mas_significativo).

# Escalar los valores extraídos al rango completo

    extraida = (extraida << (8 - bits))

Ejemplo de uso

*Considerando que desplazamos 4 bits*

00000011 << 4 = 00110000

1. **Convertir al modo de color deseado**

Si el modo de color está en escala de grises se convierte a escala de grises y si es RGB se convierte a escala de grises, si el modo de salida es RGB y la imagen extraída esta en escala de grises se convierte a color.

 # Convertir al modo de color deseado si es necesario

    if modo\_salida == 'L' and len(extraida.shape) == 3:

        extraida = rgb2gray(extraida)

    elif modo\_salida == 'RGB' and len(extraida.shape) == 2:

        extraida = gray2rgb(extraida)

    return extraida

## calcular\_metricas

Esta función se encarga de calcular que tan buena es la calidad de la imagen recuperada.

1. **Convertir las imágenes a flotantes**

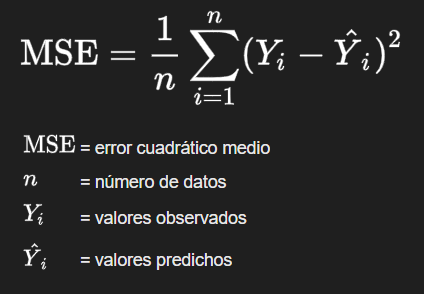
En esta sección del código se convierten las imágenes a flotante para poder hacer los cálculos de las métricas.

    original = img\_as\_float(original)

    recuperada = img\_as\_float(recuperada)

1. **Calcular mse**

Se encarga de medir el error promedio entre la imagen original y la imagen recuperada, mientras más bajo sea el valor, menos error tienen las imágenes.



mse = np.mean((original - recuperada) \*\* 2)

Ejemplo de uso

x Original 120  
y Recuperado 110  
n = 1

1/1\*(120-110)^2=100

El error en este caso es 100, lo cual es bastante y nos deja pensar que el tono puede cambiar mucho.

1. **Calcular PSNR**

Esta métrica se encarga de analizar la diferencia de intensidad entre la imagen original y la imagen recuperada bit a bit.

Valores más altos de PSNR indican que la diferencia es menos perceptible entre las imágenes.

**PSNR alto** (por ejemplo, > 30 dB): Indica que la imagen recuperada es de buena calidad y tiene poca diferencia respecto a la imagen original.

**PSNR bajo** (por ejemplo, < 20 dB): Indica que la calidad de la imagen recuperada es baja y hay una gran diferencia con la original.

Interfaz de usuario gráfica

Descripción generada automáticamente

Ejemplo de uso

MSE = 100  
MAX = 255

PSNR = 20 \* log10(255/RAIZ(100))  
PSNR = 20 \* log10(25.5)  
PSNR = 20 \* 1.407 = **28.14 dB**

# utils.py

En este archivo se encuentran las funciones para realizar procesos sobre las imágenes o sobre nuestra interfaz.

## normalizar\_imagen

Esta funcion tiene como objetivo normalizar el color de la imagen dependiendo el caso, convirtiendo imágenes de RGB a escala de grises o viceversa.

Funciona con la librería skimage.color con las funciones rgb2gray, gray2rgb

def normalizar\_imagen(imagen, modo\_objetivo='RGB'):

    """

    Normaliza el modo de color de una imagen.

    """

    if modo\_objetivo == 'RGB' and len(imagen.shape) == 2:

        # Convierte imágenes en escala de grises a RGB

        return gray2rgb(imagen)

    elif modo\_objetivo == 'L' and len(imagen.shape) == 3:

        # Convierte imágenes RGB a escala de grises

        return rgb2gray(imagen)

    return imagen

## guardar\_imagen

Guarda la imagen extraída con la información en 8bits

def guardar\_imagen(imagen, nombre\_archivo):

    """

    Guarda la imagen asegurando que se preserve la información correctamente.

    """

    # Asegurar que la imagen esté en uint8

    imagen\_a\_guardar = img\_as\_ubyte(imagen)

    io.imsave(nombre\_archivo, imagen\_a\_guardar)

## cargar\_imagen

Esta función sirve para cargar archivos con skimage.

def cargar\_imagen(archivo):

    """

    Carga la imagen asegurando la correcta interpretación del formato.

    """

    # Leer la imagen directamente desde el archivo usando skimage

    imagen = io.imread(archivo)

    return imagen

## mostrar\_metricas

Esta función muestra las métricas y la interpretación de resultados con base a los resultados de la métrica PSNR.

def mostrar\_metricas(metricas, titulo="Métricas de Calidad", bits\_usados=None):

    """

    Muestra las métricas en Streamlit.

    """

    st.write(f"### {titulo}")

    for metrica, valor in metricas.items():

        st.write(f"{metrica}: {valor:.4f}")

    psnr = metricas['PSNR']

    if bits\_usados is not None:

        if psnr > 40:

            st.success("🟢 Calidad Excelente - La imagen es prácticamente idéntica al original.")

        elif psnr > 30:

            st.info("🟡 Buena Calidad - Hay algunas diferencias menores pero aceptable.")

        elif psnr > 20:

            st.warning("🟠 Calidad Regular - Hay diferencias notables pero la imagen es reconocible.")

        else:

            st.error("🔴 Calidad Baja - La imagen recuperada tiene diferencias significativas.")

# Interfaz

En este caso hicimos la interfaz utilizando streamlit, un servicio que permite hacer aplicaciones en python interactivas rápidamente con una integración fácil e intuitiva.

Integramos varias funciones para eliminar errores en el código como redimensionar imágenes al mismo tamaño, convertir modos de color, etc.

import streamlit as st

from esteganografia import ocultar\_imagen,calcular\_metricas, extraer\_imagen

from utils import cargar\_imagen, normalizar\_imagen, mostrar\_metricas, guardar\_imagen

# Interfaz de Streamlit

st.title('Esteganografía de Imágenes')

tab1, tab2 = st.tabs(["Ocultar Imagen", "Extraer Imagen"])

with tab1:

    st.header("Ocultar una imagen")

    bits\_ocultar = st.slider('Número de bits a utilizar (LSB)', 1, 8, 1, key='hide\_bits',

                             help='Más bits = mejor calidad de la imagen oculta, pero más visible')

    archivo\_portada = st.file\_uploader("Selecciona la imagen de portada", type=['png', 'jpg', 'jpeg','jfif'], key='cover')

    archivo\_secreta = st.file\_uploader("Selecciona la imagen a ocultar", type=['png', 'jpg', 'jpeg', 'jfif'], key='secret')

    if archivo\_portada and archivo\_secreta:

        try:

            # Cargar imágenes usando la nueva función

            imagen\_portada = cargar\_imagen(archivo\_portada)

            imagen\_secreta = cargar\_imagen(archivo\_secreta)

            # Asegurar que las imágenes estén en RGB

            imagen\_portada = normalizar\_imagen(imagen\_portada, 'RGB')

            imagen\_secreta = normalizar\_imagen(imagen\_secreta, 'RGB')

            # Ajustar tamaños

            altura\_minima = min(imagen\_portada.shape[0], imagen\_secreta.shape[0])

            anchura\_minima = min(imagen\_portada.shape[1], imagen\_secreta.shape[1])

            imagen\_portada = imagen\_portada[:altura\_minima, :anchura\_minima]

            imagen\_secreta = imagen\_secreta[:altura\_minima, :anchura\_minima]

            col1, col2 = st.columns(2)

            with col1:

                st.image(imagen\_portada, caption="Imagen de portada")

                st.write(f"Dimensiones: {imagen\_portada.shape[:2]}")

            with col2:

                st.image(imagen\_secreta, caption="Imagen a ocultar")

                st.write(f"Dimensiones: {imagen\_secreta.shape[:2]}")

            if st.button("Ocultar Imagen"):

                imagen\_estego = ocultar\_imagen(imagen\_portada, imagen\_secreta, bits\_ocultar)

                metricas\_portada = calcular\_metricas(imagen\_estego,imagen\_portada,imagen\_secreta,bits\_ocultar)

                st.image(imagen\_estego, caption="Imagen con mensaje oculto")

                mostrar\_metricas(metricas\_portada, "Métricas de calidad - Imagen de portada", bits\_ocultar)

                # Guardar la imagen en formato PNG

                nombre\_archivo\_salida = "imagen\_estego.png"

                guardar\_imagen(imagen\_estego, nombre\_archivo\_salida)

                with open(nombre\_archivo\_salida, "rb") as archivo:

                    btn = st.download\_button(

                        label="Descargar imagen con mensaje oculto",

                        data=archivo,

                        file\_name="imagen\_oculta.png",

                        mime="image/png"

                    )

                st.info(f"Esta imagen fue ocultada usando {bits\_ocultar} bits. "

                       f"Asegúrate de recordar este número para la extracción.")

        except Exception as e:

            st.error(f"Error al procesar las imágenes: {str(e)}")

with tab2:

    st.header("Extraer imagen oculta")

    bits\_extraer = st.slider('Número de bits utilizados (LSB)', 1, 8, 1, key='extract\_bits',

                             help='Debe coincidir EXACTAMENTE con el número usado al ocultar')

    modo\_salida = st.selectbox('Modo de color para la imagen extraída',

                               options=['RGB', 'L'],

                               format\_func=lambda x: 'Color (RGB)' if x == 'RGB' else 'Escala de grises',

                               help='Selecciona el modo de color deseado para la imagen extraída')

    archivo\_estego = st.file\_uploader("Selecciona la imagen con mensaje oculto", type=['png'], key='stego')

    if archivo\_estego:

        try:

            # Cargar imagen usando la nueva función

            imagen\_estego = cargar\_imagen(archivo\_estego)

            imagen\_estego = normalizar\_imagen(imagen\_estego, 'RGB')

            st.image(imagen\_estego, caption="Imagen con mensaje oculto")

            st.write(f"Dimensiones: {imagen\_estego.shape[:2]}")

            if st.button("Extraer Imagen"):

                imagen\_extraida = extraer\_imagen(imagen\_estego, bits\_extraer, modo\_salida)

                st.image(imagen\_extraida, caption=f"Imagen extraída usando {bits\_extraer} bits")

                # Guardar la imagen extraída

                nombre\_archivo\_salida = "imagen\_extraida.png"

                guardar\_imagen(imagen\_extraida, nombre\_archivo\_salida)

                with open(nombre\_archivo\_salida, "rb") as archivo:

                    btn = st.download\_button(

                        label="Descargar imagen extraída",

                        data=archivo,

                        file\_name="imagen\_extraida.png",

                        mime="image/png"

                    )

        except Exception as e:

            st.error(f"Error al procesar la imagen: {str(e)}")

# Referencias

*Image Steganography*. (2024). Incoherency.co.uk. <https://incoherency.co.uk/image-steganography/>

Torres, A. (2021, October 14). *Aprendizaje automático: Una introducción al error cuadrático medio y las líneas de regresión.* FreeCodeCamp.org. <https://www.freecodecamp.org/espanol/news/aprendizaje-automatico-una-introduccion-al-error-cuadratico-medio-y-las-lineas-de-regresion/>

*Peak Signal-to-Noise Ratio as an Image Quality Metric*. (2024). Ni.com. <https://www.ni.com/en/shop/data-acquisition-and-control/add-ons-for-data-acquisition-and-control/what-is-vision-development-module/peak-signal-to-noise-ratio-as-an-image-quality-metric.html?srsltid=AfmBOoooFq908C4p4S3T1aoXKO-0D8hKzUzXItaeteTaZN6714nRYGCr>