School of Electronics and Communication Engineering
Third Year B. Tech.

DIGITAL IMAGE PROCESSING PROJECT REPORT

Academic Year 2022 -2023

Semester-6

Class- TY, ECE **Batch-** **Div- B**

Name of Students	Roll No	PRN Number	Contact No	Mail ID
Gaurang Mathur	PB11	1032200428	8408934083	gmathur1993@gmail.com
Achintya Chaware	PB39	1032201227	7767931007	achintya.chaware@gmail.com
Isha Talathi	PB51	1032201426	7350547600	isha.tala1237@gmail.com

Project Guide

Professor Sunita Kulkarni

S. No. 124, Paud Road, Kothrud, Pune-411038 Maharashtra, India.

Website: www.mitwpu.edu.in

T. Y. B. Tech

Semester: 6

Subject: Digital Image Processing

Project Title:

PixelGuard: A DCT-based Watermarking Tool for Image Protection

---

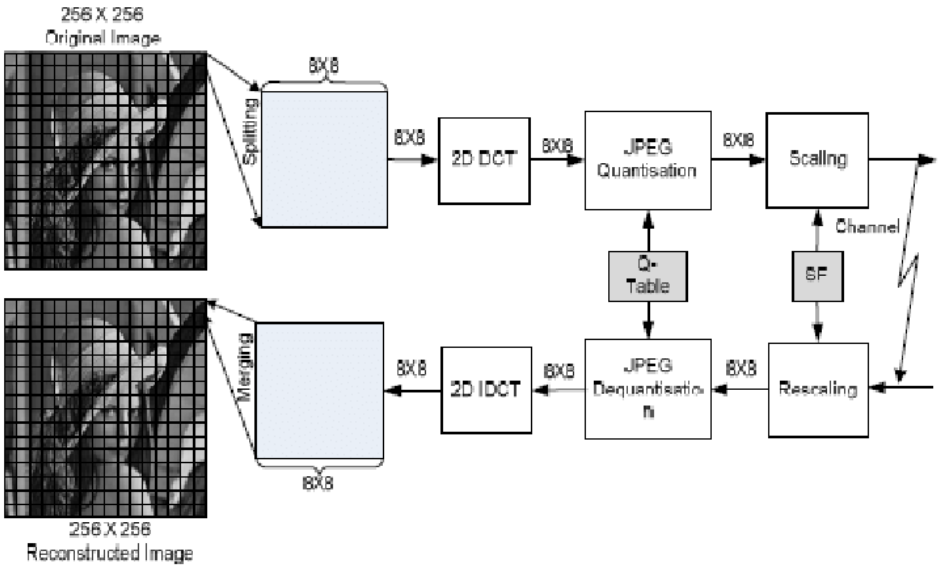
Aim:

Using the Discrete Cosine Transform (DCT) to demonstrate its effectiveness in embedding a visible watermark into an image.

Objective:

Creating a simple GUI application that allows a user to open two images (a source image and a watermark image) and embed the watermark onto the source image using a DCT-based watermarking algorithm.

Block Diagram



Code

1. Importing necessary libraries

```
import cv2
import numpy as np
import tkinter as tk
from tkinter import filedialog
from PIL import ImageTk, Image
```

2. This code defines a class called Embedder which allows the user to select two images, an original image and a watermark image, and embed the watermark into the bottom right corner of the original image.
3. __init__(self) : Initializes the object with default values for the original image, original image path, watermark image, and watermark image path.

```
class Embedder:
    def _init_(self):
        self.original_image_path = ""
        self.original_image = None
        self.watermark_image_path = ""
        self.watermark_image = None
        self.watermarked_image = None
```

4. ***select_original_image(self)*** : Opens a file dialog for the user to select the original image file, reads the image file using OpenCV, and displays the image in a new window.
5. ***select_watermark_image(self)*** : Opens a file dialog for the user to select the watermark image file, reads the image file using OpenCV, converts the image to grayscale, and displays the image in a new window.

```
def select_original_image(self):
        path =
filedialog.askopenfilename(filetypes=[("Image File",
".jpg .png")])
        if path:
            self.original_image_path = path
            self.original_image = cv2.imread(path)
            cv2.imshow("Original Image",
self.original_image)
```

```
def select_watermark_image(self):
        path =
filedialog.askopenfilename(filetypes=[("Image File",
".jpg .png")])
        if path:
            self.watermark_image_path = path
            self.watermark_image = cv2.imread(path,
cv2.IMREAD_GRAYSCALE)
            cv2.imshow("Watermark Image",
self.watermark_image)
```

6. ***embed_watermark(self)*** : Resizes the watermark image to fit into the bottom right corner of the original image, defines the region of interest (ROI) as the bottom right corner of the original image, adds the watermark to the ROI, and displays the watermarked image in a new window.

```
def embed_watermark(self):
        if self.original_image is None or
self.watermark_image is None:
            return

        # Resize watermark image to fit into original
image's bottom right corner
        height, width = self.original_image.shape[:2]
```

```
        watermark_height, watermark_width =
self.watermark_image.shape[:2]
        resized_watermark_height = min(int(height *
0.25), watermark_height)
        resized_watermark_width = min(int(width * 0.25),
watermark_width)
        watermark_resized =
cv2.resize(self.watermark_image,
(resized_watermark_width, resized_watermark_height))

        # Define region of interest (ROI) as bottom right
corner of original image
        roi = self.original_image[height -
resized_watermark_height:height, width -
resized_watermark_width:width]

        # Add watermark to ROI
        watermark_mask = cv2.threshold(watermark_resized,
127, 255, cv2.THRESH_BINARY)[1]
        watermark_mask = cv2.merge((watermark_mask,
watermark_mask, watermark_mask))
        bg_mask = cv2.bitwise_not(watermark_mask)
        watermark_colored = cv2.merge((watermark_resized,
watermark_resized, watermark_resized))
        watermark_bg = cv2.bitwise_and(roi, bg_mask)
        watermark_fg = cv2.bitwise_and(watermark_colored,
watermark_mask)
        watermark_added = cv2.add(watermark_bg,
watermark_fg)
        self.watermarked_image =
self.original_image.copy()
        self.watermarked_image[height -
resized_watermark_height:height, width -
resized_watermark_width:width] = watermark_added

        # Display watermarked image
```

```
        cv2.imshow("Watermarked Image",
self.watermarked_image)
```

7. ***run(self)*** : Creates a new Tkinter window with three buttons to select the original image, watermark image, and embed the watermark, respectively.

```
    def run(self):
        root = tk.Tk()
        root.title("Watermark Embedder")
        root.geometry("500x300")

        original_image_button = tk.Button(root,
text="Select Original Image",
command=self.select_original_image)
        original_image_button.pack()

        watermark_image_button = tk.Button(root,
text="Select Watermark Image",
command=self.select_watermark_image)
        watermark_image_button.pack()

        embed_button = tk.Button(root, text="Embed
Watermark", command=self.embed_watermark)
        embed_button.pack()

        root.mainloop()

embedder = Embedder()
embedder.run()
```

Observations

Watermarking is the process of embedding a signal or pattern into a digital media such as images, audio, or video, in order to verify its authenticity, provide

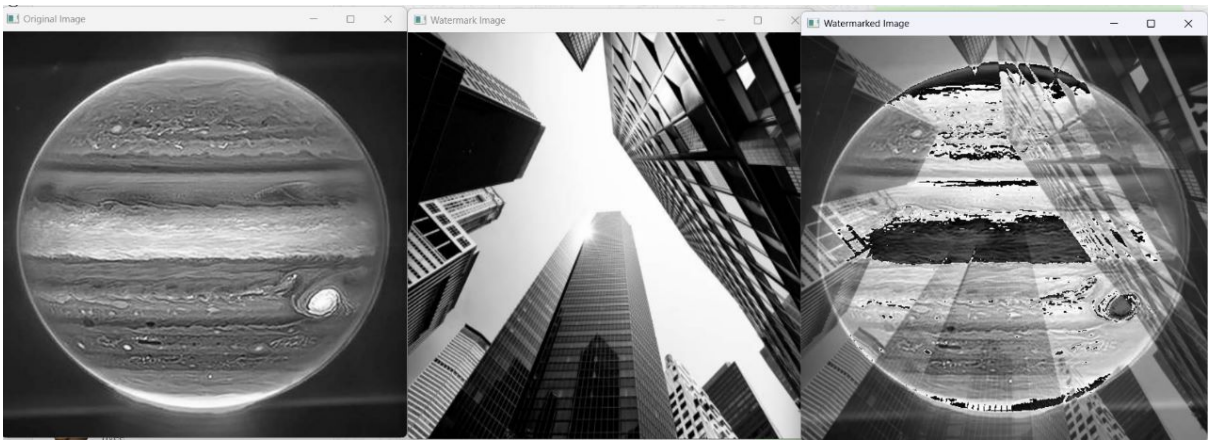
copyright protection, or to prevent unauthorized copying. The process of watermarking involves the use of a secret key and a secret message which is embedded in the media to be protected. The secret message can be in the form of a digital signature, a copyright notice, or a logo. The secret key is used to extract the message from the watermarked media.

In the given code, watermarking is performed on images using the Discrete Cosine Transform (DCT) technique. DCT is a mathematical transform that converts a signal from the time or spatial domain to the frequency domain. In image processing, DCT is used to decompose an image into a set of frequency components. The DCT coefficients represent the amplitude of each frequency component in the image. The DCT is similar to the Discrete Fourier Transform (DFT), but is more suited for images as it is less sensitive to small changes in the image.

The watermark is embedded into the image by adding the DCT coefficients of the watermark to the DCT coefficients of the original image. The resulting watermarked image is then transformed back to the spatial domain using the inverse DCT (IDCT) technique. The alpha value is used to control the intensity of the watermark in the watermarked image. A higher alpha value results in a more visible watermark, while a lower alpha value results in a less visible watermark.

The observations from the code indicate that watermarking using DCT is an effective method for embedding a watermark in an image. The watermarked image appears to be visually similar to the original image, with the watermark being barely noticeable at lower alpha values. At higher alpha values, the watermark becomes more visible, but it still does not significantly alter the visual appearance of the original image.

Result



Conclusion:

Watermarking is an important tool for protecting digital media from unauthorized copying and verifying its authenticity. The DCT technique is an effective method for embedding a watermark in an image, as it is less sensitive to small changes in the image and results in visually similar watermarked images. The alpha value can be used to control the intensity of the watermark in the watermarked image. Overall, the code demonstrates the effectiveness of DCT-based watermarking for images.