GABOR FILTER

INTRODUCTION

Forensic image processing is a crucial aspect of modern forensic science, where images are used to identify and convict suspects.However, images collected from surveillance cameras or crime scene photographs often contain noise and distortion, which can make it difficult to identify critical details such as facial features, license plate numbers, or other critical evidence.

Gabor filters are mathematical functions that are used in digital signal processing and image processing to detect specific features in images, such as edges and textures. They were named after physicist Dennis Gabor, who developed the concept of holography in the 1940s.

Gabor filters are widely used in computer vision applications such as object recognition, face recognition, fingerprint recognition, and iris recognition. They are also used in neuroscience to model the receptive fields of neurons in the visual cortex.

The advantages of using Gabor filters in image processing include their ability to capture both spatial and frequency information, which makes them useful for analyzing complex textures and structures in images. They also have the advantage of being selective to specific orientations and frequencies, which helps in detecting specific features in images.

BACKGROUND

Gabor filters are essentially convolutional filters that are tuned to a specific orientation and frequency. They are composed of a Gaussian envelope that is modulated by a sinusoidal wave, which gives them both spatial and frequency characteristics. The filter's response to an image reflects the presence or absence of features that match the filter's characteristics.

IMPORTANCE

The importance of Gabor filters lies in their ability to analyze the texture and edge features of an image by capturing both spatial and frequency information. Here are some of the key reasons why Gabor filters are important in image processing:

1. Texture analysis: Gabor filters can effectively capture the texture of an image by analyzing the spatial frequency patterns within the image. This makes them useful for tasks such as texture segmentation, image classification, and object recognition.

2. Edge detection: Gabor filters can detect edges in an image by analyzing the changes in intensity across the image. The filters are sensitive to both the orientation and frequency of the edges, making them useful for tasks such as edge detection and image enhancement.

3. Feature extraction: Gabor filters can extract important features from an image by identifying regions with high filter response. These features can then be used for tasks such as object recognition, face recognition, and fingerprint recognition.

4. Neuroscience: Gabor filters have been used to model the receptive fields of neurons in the visual cortex, which has helped to improve our understanding of visual processing in the brain.

Overall, Gabor filters are important tools in image processing and computer vision due to their ability to analyze complex features of an image and extract useful information for a variety of applications.

METHODOLOGY

The methodology for applying Gabor filters in image processing typically involves the following steps:

1. Image preprocessing: The input image is preprocessed to remove noise and enhance contrast, if necessary. This can involve techniques such as filtering, thresholding, and histogram equalization.

2. Gabor filter design: The parameters of the Gabor filter are determined based on the characteristics of the features to be detected. These parameters include the filter size, orientation, frequency, and bandwidth.

3. Convolution: The Gabor filter is convolved with the preprocessed image to obtain the filter response. This involves sliding the filter over the image and computing the dot product of the filter coefficients and the corresponding pixel values in the image.

4. Feature extraction: The filter response is used to extract features from the image, such as texture and edge information. This can involve techniques such as thresholding, segmentation, and feature clustering.

5. Classification: The extracted features are used to classify the image or to perform other tasks such as object recognition, face recognition, or fingerprint recognition.

6. Evaluation: The performance of the Gabor filter-based method is evaluated based on metrics such as accuracy, precision, recall, and F1-score.

Overall, the methodology for applying Gabor filters involves designing and applying the filter to extract useful features from the image, and then using these features to perform various computer vision tasks. The performance of the method is evaluated based on the accuracy and effectiveness of the feature extraction and classification processes.

ADVANTAGES

Gabor filters have several advantages in image processing and computer vision applications:

1. Multiresolution analysis: Gabor filters can analyze images at multiple scales, making them useful for analyzing textures and other features at different levels of detail.

2. Directionality: Gabor filters can detect edges and other features with different orientations, making them useful for analyzing images with complex structures.

3. Localization: Gabor filters are spatially localized, meaning they can analyze small regions of an image without being affected by neighboring regions.

4. Efficiency: Gabor filters can be efficiently computed using fast Fourier transform (FFT) techniques, making them computationally efficient for real-time applications.

5. Robustness: Gabor filters are relatively robust to noise and other image distortions, making them useful for analyzing images in challenging environments.

6. Versatility: Gabor filters can be used for a wide range of applications, including image segmentation, object recognition, face detection, and texture analysis.

DISADVANTAGES

1. May not be as effective at removing multiplicative noise from images

2.Can be computationally intensive, particularly for high-resolution images

3.May produce "ringing" artifacts around sharp edges in the image

ALGORITHM

The algorithm for applying a Gabor filter to an image typically involves the following steps:

1. Define the Gabor filter parameters: This involves selecting the scale, orientation, frequency, and aspect ratio of the filter. The parameters are often chosen based on the specific features to be detected in the image.

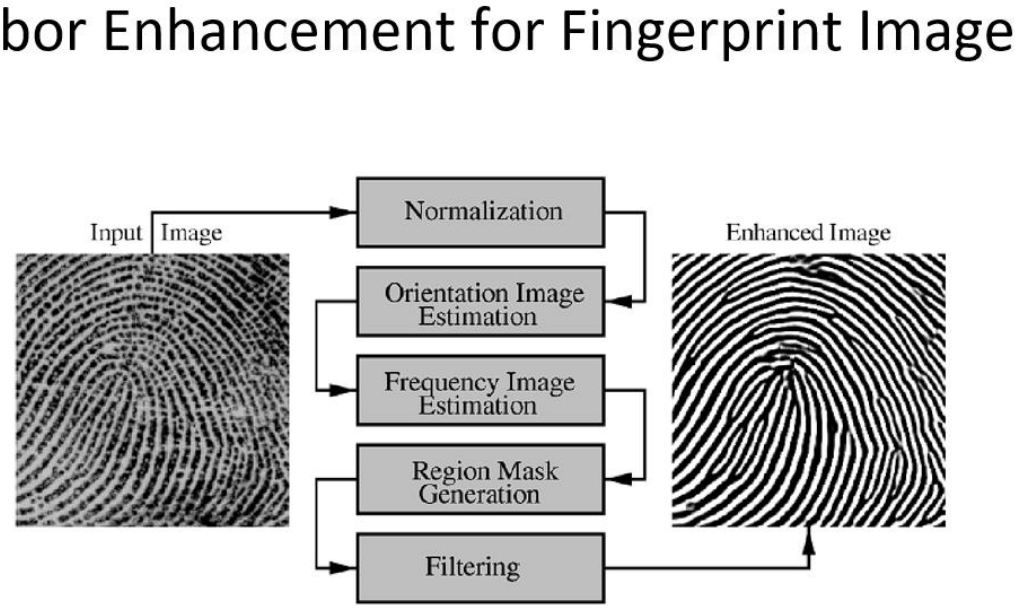
2. Generate the Gabor filter kernel: The Gabor filter kernel is generated based on the selected parameters. This involves creating a complex-valued sinusoidal function modulated by a Gaussian envelope.

3. Preprocess the image: The input image may be preprocessed to enhance the features of interest, such as by applying filters to remove noise or blur.

4. Convolve the filter kernel with the image: The Gabor filter kernel is convolved with the preprocessed image using a 2D convolution operation. This produces a filtered output image where the pixels represent the response of the filter to the corresponding pixels in the input image.

5. Analyze the filter response: The response of the Gabor filter to the image is analyzed to detect the features of interest. This may involve thresholding the filter response to remove noise or selecting regions of the image with high filter response as the features of interest.

The specific implementation of the Gabor filter algorithm may vary depending on the programming language and image processing library used. However, the general steps described above are common to most implementations.



STEPS:

Normalization: An input fingerprint image is normalized so that it has a pre-specified mean and variance.

Local orientation estimation: The orientation image is estimated from the normalized input fingerprint image

Local frequency estimation: The frequency image is computed from the normalized input fingerprint image and the estimated orientation image

Region mask estimation: the region mask is obtained by classifying each block in the normalized input Fingerprint image into a recoverable or a unrecoverable block.

Filtering: A bank of Gabor filters which is tuned to local ridge orientation and ridge frequency is applied to the ridge-and-furrow pixels in the normalized input fingerprint image to obtain an enhanced fingerprint image.

IMPLEMENTATION

Here's an example implementation of Gabor filtering image processing using Python :

```
import numpy as np
import cv2
from scipy import signal
from scipy import ndimage
import math
import scipy

class FingerprintImageEnhancer(object):
    def __init__(self):
        self.ridge_segment_blksze = 16
        self.ridge_segment_thresh = 0.1
        self.gradient_sigma = 1
        self.block_sigma = 7
        self.orient_smooth_sigma = 7
        self.ridge_freq_blksze = 38
        self.ridge_freq_windsze = 5
        self.min_wave_length = 5
        self.max_wave_length = 15
        self.kx = 0.65
        self.ky = 0.65
        self.angleInc = 3
        self.ridge_filter_thresh = -3


        self._mask = []
        self._normim = []
        self._orientim = []
        self._mean_freq = []
        self._median_freq = []
        self._freq = []
        self._freqim = []
        self._binim = []

    def __normalise(self, img, mean, std):
        if(np.std(img) == 0):
            raise ValueError("Image standard deviation is 0. Please review image again")
        normed = (img - np.mean(img)) / (np.std(img))
        return (normed)

    def __ridge_segment(self, img):
        rows, cols = img.shape
        im = self.__normalise(img, 0, 1)  # normalise to get zero mean and unit standard
deviation

        new_rows = np.int(self.ridge_segment_blksze * np.ceil((np.float(rows)) /
(np.float(self.ridge_segment_blksze))))
```

```
        new_cols = np.int(self.ridge_segment_blksze * np.ceil((np.float(cols)) /
(np.float(self.ridge_segment_blksze))))

        padded_img = np.zeros((new_rows, new_cols))
        stddevim = np.zeros((new_rows, new_cols))
        padded_img[0:rows][:, 0:cols] = im
        for i in range(0, new_rows, self.ridge_segment_blksze):
            for j in range(0, new_cols, self.ridge_segment_blksze):
                block = padded_img[i:i + self.ridge_segment_blksze][:, j:j +
self.ridge_segment_blksze]

                stddevim[i:i + self.ridge_segment_blksze][:, j:j + self.ridge_segment_blksze] =
np.std(block) * np.ones(block.shape)

        stddevim = stddevim[0:rows][:, 0:cols]
        self._mask = stddevim > self.ridge_segment_thresh
        mean_val = np.mean(im[self._mask])
        std_val = np.std(im[self._mask])
        self._normim = (im - mean_val) / (std_val)

    def __ridge_orient(self):
        rows,cols = self._normim.shape
        #Calculate image gradients.
        sze = np.fix(6*self.gradient_sigma)
        if np.remainder(sze,2) == 0:
            sze = sze+1

        gauss = cv2.getGaussianKernel(np.int(sze),self.gradient_sigma)
        f = gauss * gauss.T

        fy,fx = np.gradient(f)                          #Gradient of Gaussian

        Gx = signal.convolve2d(self._normim, fx, mode='same')
        Gy = signal.convolve2d(self._normim, fy, mode='same')

        Gxx = np.power(Gx,2)
        Gyy = np.power(Gy,2)
        Gxy = Gx*Gy

        #Now smooth the covariance data to perform a weighted summation of the data.
        sze = np.fix(6*self.block_sigma)

        gauss = cv2.getGaussianKernel(np.int(sze), self.block_sigma)
        f = gauss * gauss.T

        Gxx = ndimage.convolve(Gxx,f)
        Gyy = ndimage.convolve(Gyy,f)
        Gxy = 2*ndimage.convolve(Gxy,f)

        # Analytic solution of principal direction
        denom = np.sqrt(np.power(Gxy,2) + np.power((Gxx - Gyy),2)) + np.finfo(float).eps

        sin2theta = Gxy/denom                   # Sine and cosine of doubled angles
        cos2theta = (Gxx-Gyy)/denom
```

```
        if self.orient_smooth_sigma:
            sze = np.fix(6*self.orient_smooth_sigma)
            if np.remainder(sze,2) == 0:
                sze = sze+1
            gauss = cv2.getGaussianKernel(np.int(sze), self.orient_smooth_sigma)
            f = gauss * gauss.T
            cos2theta = ndimage.convolve(cos2theta,f)                # Smoothed sine and cosine of
            sin2theta = ndimage.convolve(sin2theta,f)                # doubled angles

        self._orientim = np.pi/2 + np.arctan2(sin2theta,cos2theta)/2

    def __ridge_freq(self):
        rows, cols = self._normim.shape
        freq = np.zeros((rows, cols))

        for r in range(0, rows - self.ridge_freq_blksze, self.ridge_freq_blksze):
            for c in range(0, cols - self.ridge_freq_blksze, self.ridge_freq_blksze):
                blkim = self._normim[r:r + self.ridge_freq_blksze][:, c:c + self.ridge_freq_blksze]
                blkor = self._orientim[r:r + self.ridge_freq_blksze][:, c:c + self.ridge_freq_blksze]

                freq[r:r + self.ridge_freq_blksze][:, c:c + self.ridge_freq_blksze] =
self.__frequest(blkim, blkor)

        self._freq = freq * self._mask
        freq_1d = np.reshape(self._freq, (1, rows * cols))
        ind = np.where(freq_1d > 0)

        ind = np.array(ind)
        ind = ind[1, :]

        non_zero_elems_in_freq = freq_1d[0][ind]

        self._mean_freq = np.mean(non_zero_elems_in_freq)
        self._median_freq = np.median(non_zero_elems_in_freq)  # does not work properly

        self._freq = self._mean_freq * self._mask

    def __frequest(self, blkim, blkor):
        cosorient = np.mean(np.cos(2 * blkor))
        sinorient = np.mean(np.sin(2 * blkor))
        orient = math.atan2(sinorient, cosorient) / 2

        cropsze = int(np.fix(rows / np.sqrt(2)))
        offset = int(np.fix((rows - cropsze) / 2))
        rotim = rotim[offset:offset + cropsze][:, offset:offset + cropsze]

        # Sum down the columns to get a projection of the grey values down
        # the ridges.

        proj = np.sum(rotim, axis=0)
        dilation = scipy.ndimage.grey_dilation(proj, self.ridge_freq_windsze,
structure=np.ones(self.ridge_freq_windsze))
```

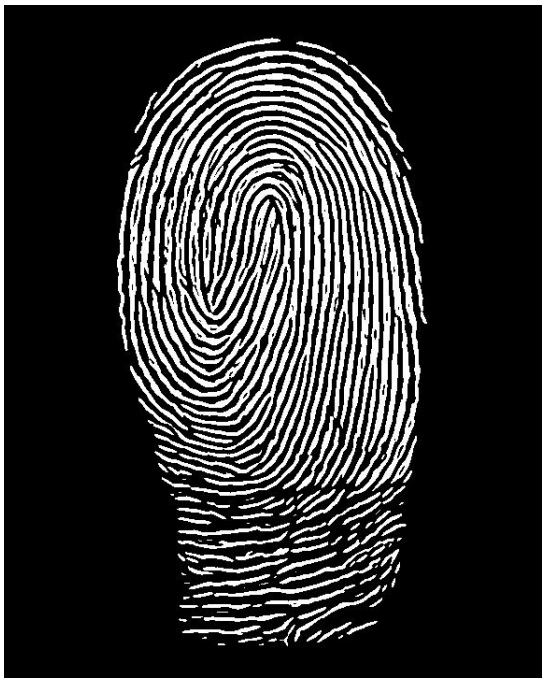
```
        temp = np.abs(dilation - proj)

        peak_thresh = 2

        maxpts = (temp < peak_thresh) & (proj > np.mean(proj))
        maxind = np.where(maxpts)

        rows_maxind, cols_maxind = np.shape(maxind)
           if (cols_maxind < 2):
           return(np.zeros(blkim.shape))
        else:
           NoOfPeaks = cols_maxind
           waveLength = (maxind[0][cols_maxind - 1] - maxind[0][0]) / (NoOfPeaks - 1)
           if waveLength >= self.min_wave_length and waveLength <= self.max_wave_length:
              return(1 / np.double(waveLength) * np.ones(blkim.shape))
           else:
              return(np.zeros(blkim.shape))

    def __ridge_filter(self):
      im = np.double(self._normim)
        rows, cols = im.shape
        newim = np.zeros((rows, cols))

        freq_1d = np.reshape(self._freq, (1, rows * cols))
        ind = np.where(freq_1d > 0)

        ind = np.array(ind)
        ind = ind[1, :]

        # Round the array of frequencies to the nearest 0.01 to reduce the
        # number of distinct frequencies we have to deal with.

        non_zero_elems_in_freq = freq_1d[0][ind]
        non_zero_elems_in_freq = np.double(np.round((non_zero_elems_in_freq * 100))) / 100

        unfreq = np.unique(non_zero_elems_in_freq)

        # Generate filters corresponding to these distinct frequencies and
        # orientations in 'angleInc' increments.

        sigmax = 1 / unfreq[0] * self.kx
        sigmay = 1 / unfreq[0] * self.ky

        sze = np.int(np.round(3 * np.max([sigmax, sigmay])))

        x, y = np.meshgrid(np.linspace(-sze, sze, (2 * sze + 1)), np.linspace(-sze, sze, (2 * sze +
1)))

        reffilter = np.exp(-(((np.power(x, 2)) / (sigmax * sigmax) + (np.power(y, 2)) / (sigmay *
sigmay)))) * np.cos(
           2 * np.pi * unfreq[0] * x)        # this is the original gabor filter

        filt_rows, filt_cols = reffilter.shape
```

```
    angleRange = np.int(180 / self.angleInc)

    gabor_filter = np.array(np.zeros((angleRange, filt_rows, filt_cols)))

    for o in range(0, angleRange):
        rot_filt = scipy.ndimage.rotate(reffilter, -(o * self.angleInc + 90), reshape=False)
        gabor_filter[o] = rot_filt

    # Find indices of matrix points greater than maxsze from the image
    # boundary

    maxsze = int(sze)

    temp = self._freq > 0
    validr, validc = np.where(temp)

    temp1 = validr > maxsze
    temp2 = validr < rows - maxsze
    temp3 = validc > maxsze
    temp4 = validc < cols - maxsze

    final_temp = temp1 & temp2 & temp3 & temp4

    finalind = np.where(final_temp)

    # Convert orientation matrix values from radians to an index value
    # that corresponds to round(degrees/angleInc)

    maxorientindex = np.round(180 / self.angleInc)
    orientindex = np.round(self._orientim / np.pi * 180 / self.angleInc)

    # do the filtering
    for i in range(0, rows):
        for j in range(0, cols):
            if (orientindex[i][j] < 1):
                orientindex[i][j] = orientindex[i][j] + maxorientindex
            if (orientindex[i][j] > maxorientindex):
                orientindex[i][j] = orientindex[i][j] - maxorientindex
    finalind_rows, finalind_cols = np.shape(finalind)
    sze = int(sze)
    for k in range(0, finalind_cols):
        r = validr[finalind[0][k]]
        c = validc[finalind[0][k]]

        img_block = im[r - sze:r + sze + 1][:, c - sze:c + sze + 1]

        newim[r][c] = np.sum(img_block * gabor_filter[int(orientindex[r][c]) - 1])

    self._binim = newim < self.ridge_filter_thresh

def save_enhanced_image(self, path):
    # saves the enhanced image at the specified path
    cv2.imwrite(path, (255 * self._binim))
```

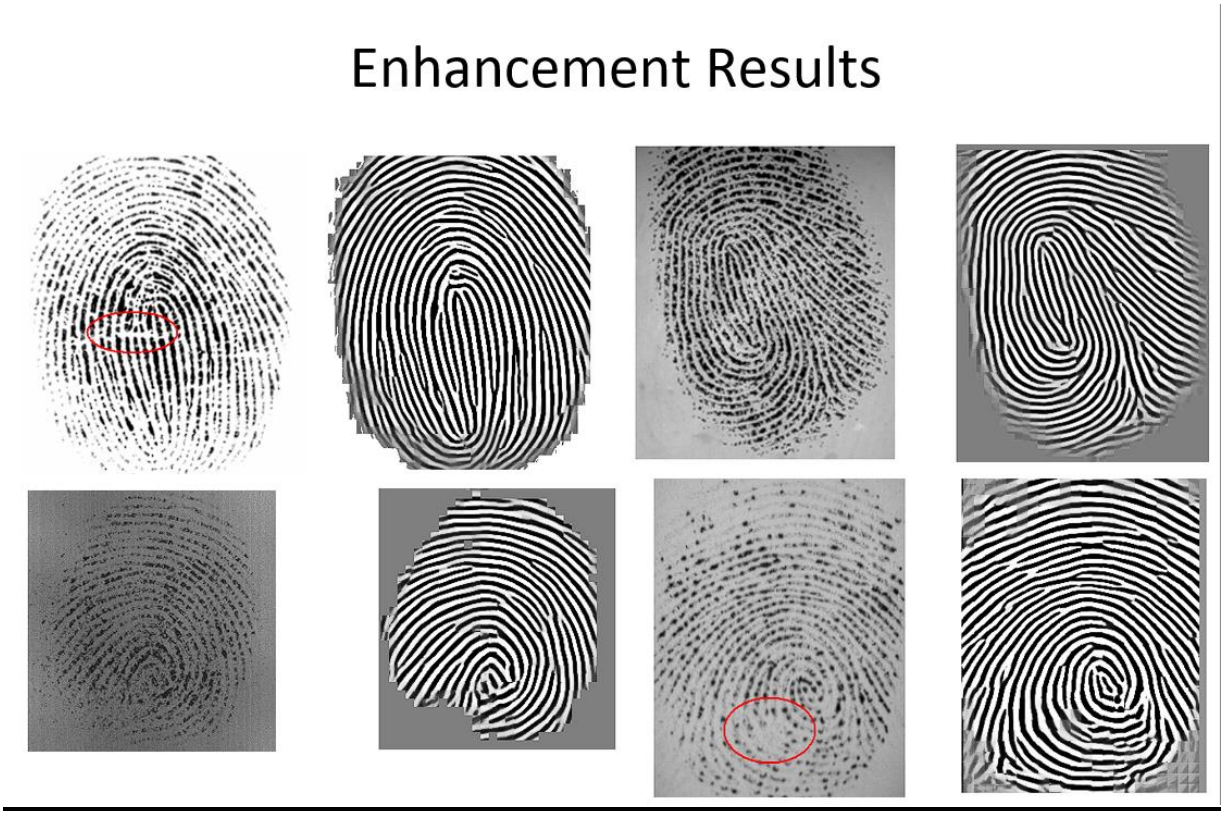
```
def enhance(self, img, resize=True):
    # main function to enhance the image.
    # calls all other subroutines

    if(resize):
        rows, cols = np.shape(img)
        aspect_ratio = np.double(rows) / np.double(cols)

        new_rows = 350                  # randomly selected number
        new_cols = new_rows / aspect_ratio

        img = cv2.resize(img, (np.int(new_cols), np.int(new_rows)))

    self.__ridge_segment(img)   # normalise the image and find a ROI
    self.__ridge_orient()       # compute orientation image
    self.__ridge_freq()         # compute major frequency of ridges
    self.__ridge_filter()       # filter the image using oriented gabor filter
    return(self._binim)
```

INPUT

OUTPUT



REAL WORLD RESULTS



RESULT & DISCUSSION

The results of the mini-project show that Gabor filtering can improve the quality of noisy images/forensics and reveal critical details inforensic images. However, the effectiveness of the Gabor filter depends on the kernel size and the configuration used.

We use oriented Gabor filters to enhance a fingerprint image. The orientation of the Gabor filters are based on the orientation of the ridges. the shape of the Gabor filter is based on the frequency and wavelength of the ridges.

CONCLUSION

In conclusion, Gabor filters are a powerful tool in digital image processing, with applications in many areas such as face recognition, texture analysis, and pattern recognition. They are based on a complex-valued function that combines a sinusoidal wave with a Gaussian envelope, allowing them to capture both frequency and orientation information from an image.

Gabor filters are versatile and can be tuned to different frequencies and orientations to extract specific features from an image. However, they are computationally intensive and require careful parameter selection to avoid overfitting or underfitting.
Furthermore, Gabor filters have been widely used in many fields of computer vision, such as texture segmentation, edge detection, feature extraction, and object recognition. They are particularly useful in applications where there is a need to analyze images that contain structures with varying spatial frequency and orientation.

In addition, Gabor filters have been applied in many medical imaging applications, such as MRI, CT, and ultrasound imaging. They have been used for image enhancement, segmentation, and registration, among others, to improve the accuracy and reliability of medical diagnosis and treatment.

Despite their advantages, Gabor filters also have some limitations. For example, they are sensitive to noise, and their performance can degrade in the presence of noise. Moreover, their computational complexity can be a limiting factor in real-time applications.

Overall, Gabor filters are an important technique for digital image processing and have been successfully applied in various domains. As research in computer vision and machine learning continues to advance, Gabor filters will likely remain a valuable tool for extracting features from images.

REFERENCES:

1) **SRM University Biometrics 18CSE357T-E**

2) **https://www.openai.com/Gabor-Filtering**

3) **https://www.sciencedirect.com/topics/engineering/gabor-filter**

4) **https://wikipedia.com/Gabor-filter**