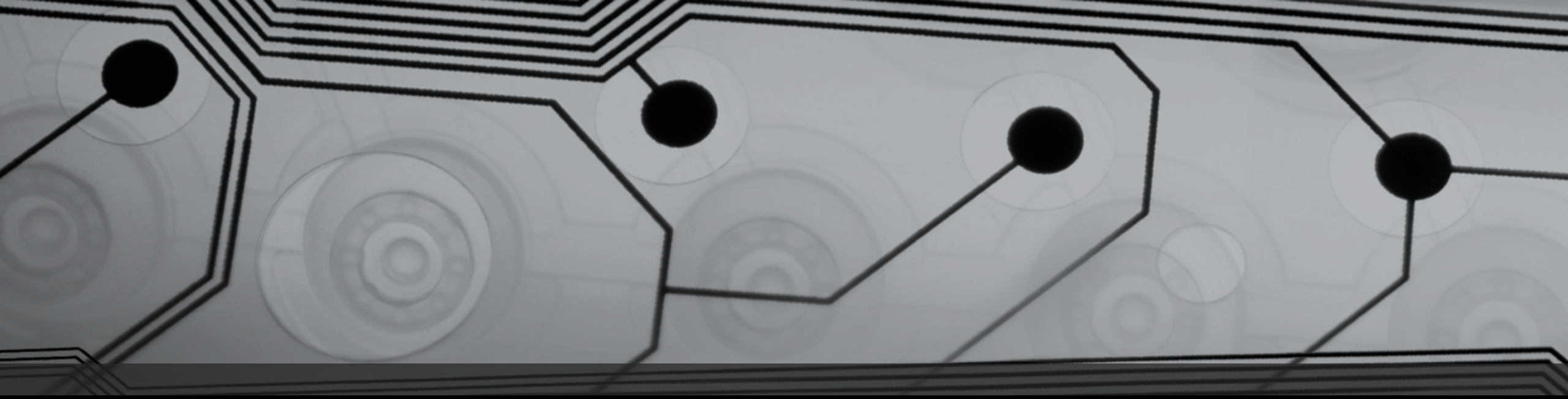
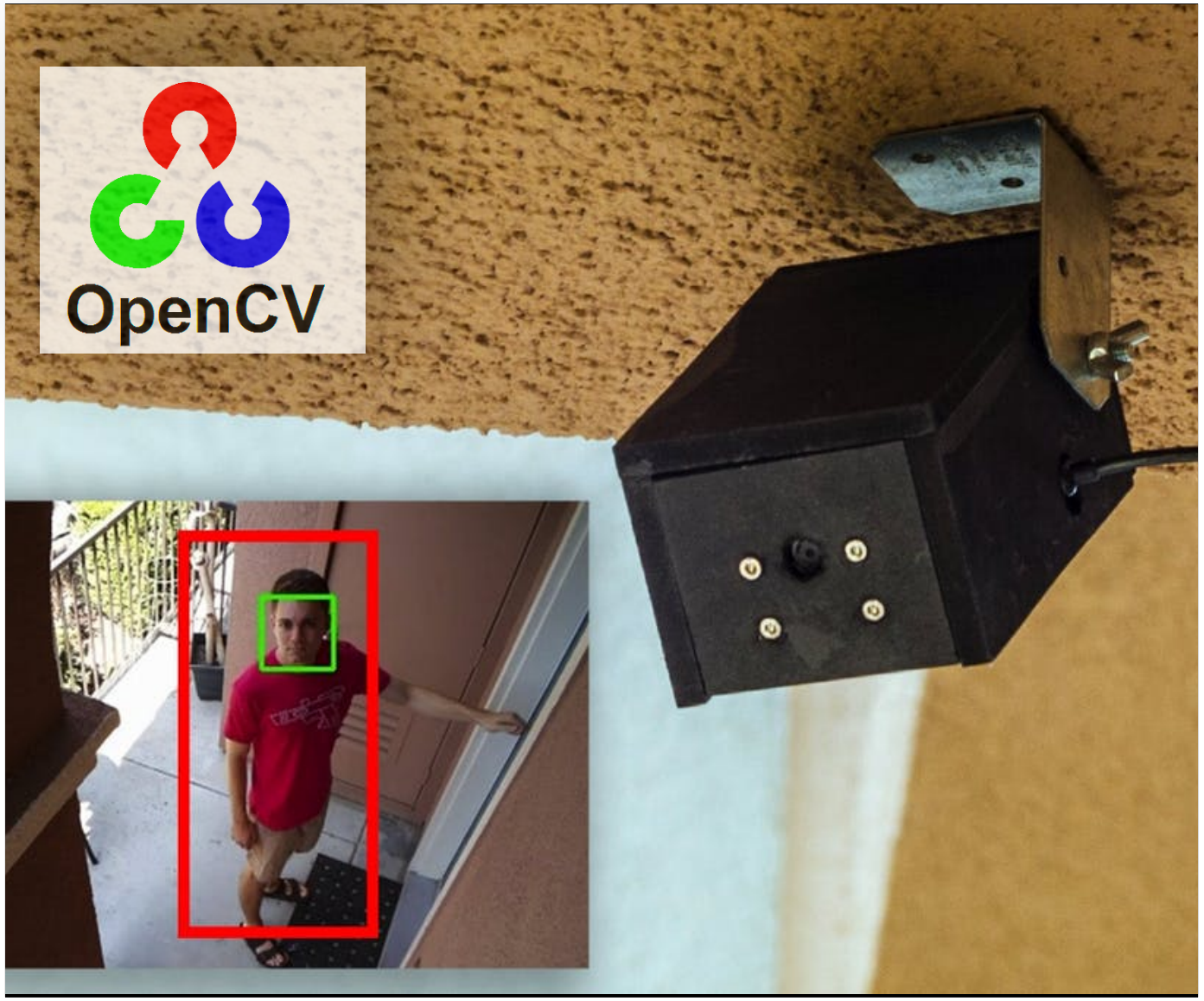


EL213 ANALOG CIRCUITS - PROJECT

Smart Security Camera with Face Detection & Recognition using OpenCV and Raspberry Pi

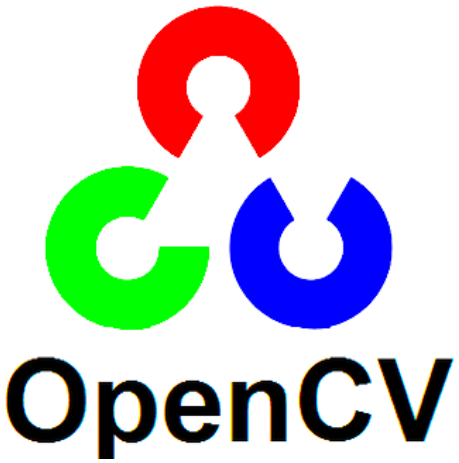
GROUP – 21

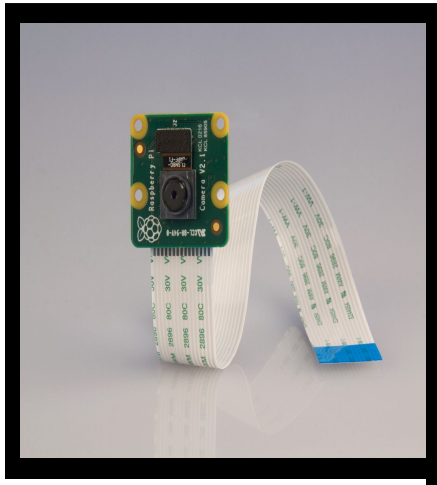
- Rutvik Shah - 201601416
- Rutvik Kothari - 201601417
- Ayush Jain - 201601404
- Himil Patel - 201601409
- Vidhin Parmar - 201601003
- Deep Thanki - 201601068
- Ravi Sawlani - 201601120
- Ujjval Patel - 201601234
- Hiren Vaghela - 201601203
- Dhruvesh Asnani - 201601423

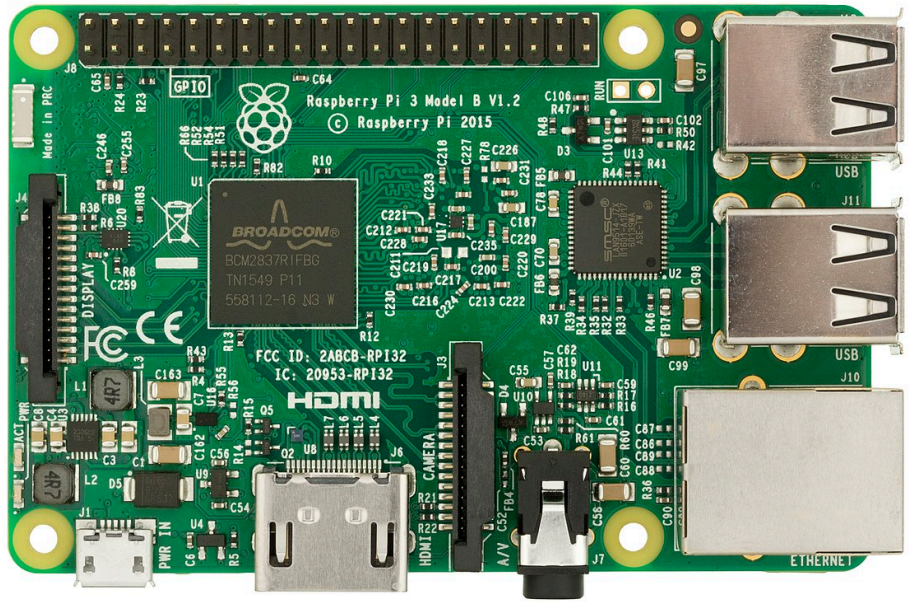


INTRODUCTION

We've tried to apply Machine Learning and Computer Vision techniques to build a Smart Security Camera which identifies any movement at your door step, identify the object/ person and immediately respond to the owner via email.







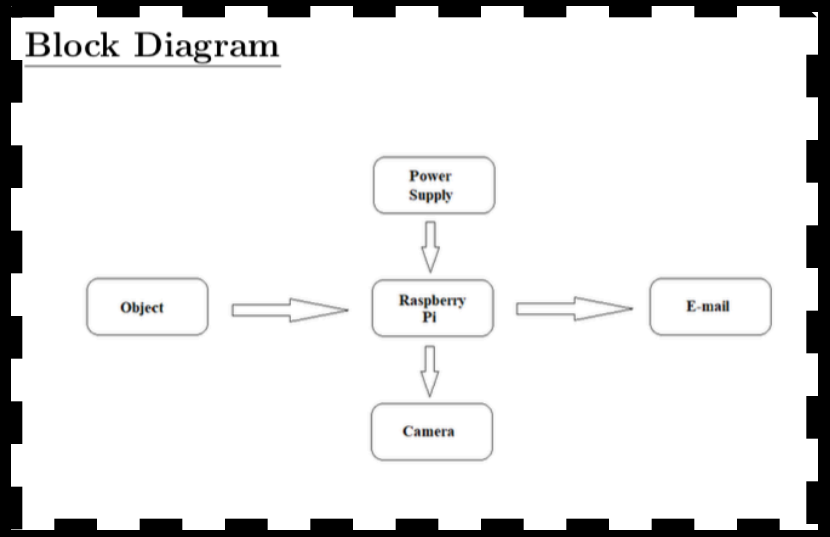
COMPONENTS

HARDWARE

- *Raspberry Pi Zero*
- *Raspberry Pi Camera Module*
- *Power Supply*

SOFTWARE

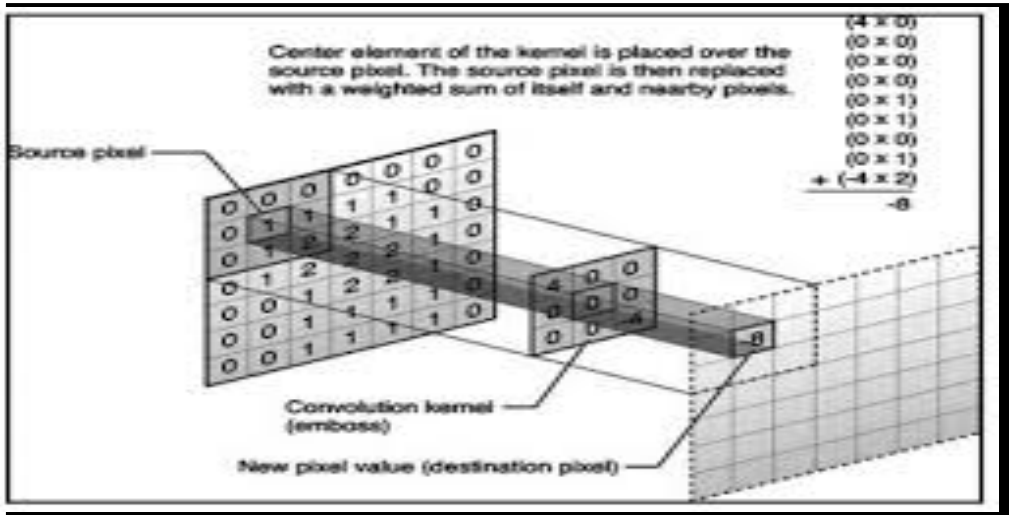
- *Python 3*
- *Open Computer Vision Library*

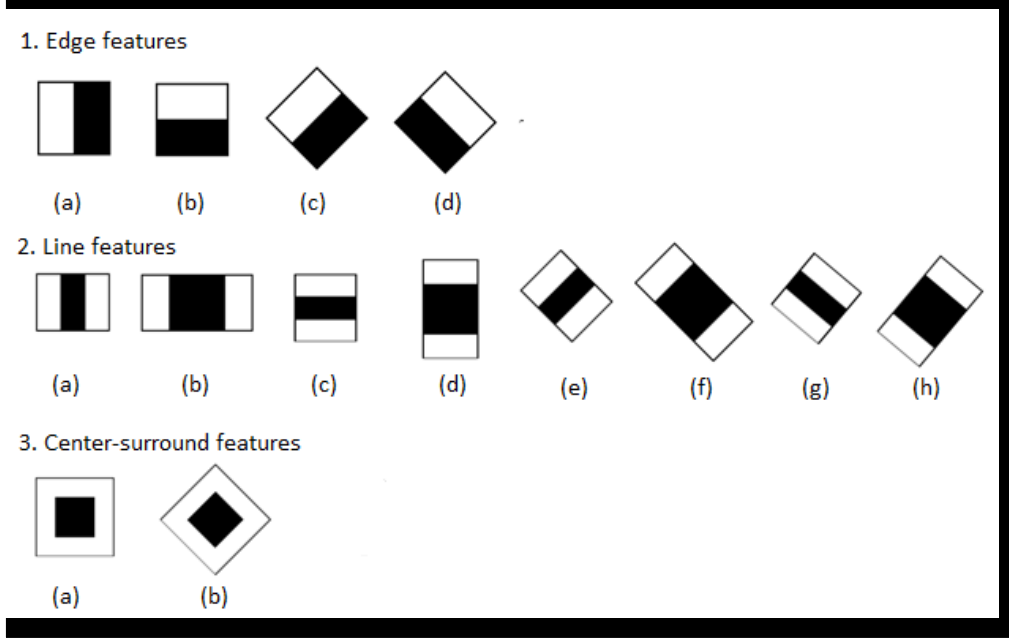


BASIC WORKING

The camera module captures live video and it sends this live video to the user to an html page which can be accessed on localhost using IP Address of Raspberry Pi. Now, whenever the camera detects an object, the data is sent to an algorithm which runs in the background which detects this objects, puts a frame over it and sends the owner e-mail the image of the object which it has detected.

If possible, we could also workout some sort of detection mechanism which triggers a payload like a lock to do whenever known visitors are detected.





ALGORITHM

We can use the “Haar Cascade Classified” which can do feature extraction by convolution with models described on the left and thereby performing boundary detection on the object by manipulating the pixel values.

For detection purposes, we need to train the model on the required face and tune classifier for those features.

Local Binary Patterns Histogram

- This is the algorithm which we've applied for image recognition
- Local Binary Patterns, or LBPs for short, are a texture descriptor, which compute a local representation of texture
- This local representation is constructed by comparing each pixel with its surrounding neighbourhood of pixels
- The first step in constructing the LBP texture descriptor is to convert the image to grayscale. For each pixel in the grayscale image, we select a neighbourhood of size *r* surrounding the centre pixel. A LBP value is then calculated for this centre pixel and stored in the output 2D array with the same width and height as the input image

Local Binary Patterns Histogram

- In the algorithm we consider,

  A. The number of points *p* in a circularly symmetric neighbourhood to consider (thus removing relying on a square neighbourhood)

  B. The radius of the circle *r*, which allows us to account for different scales

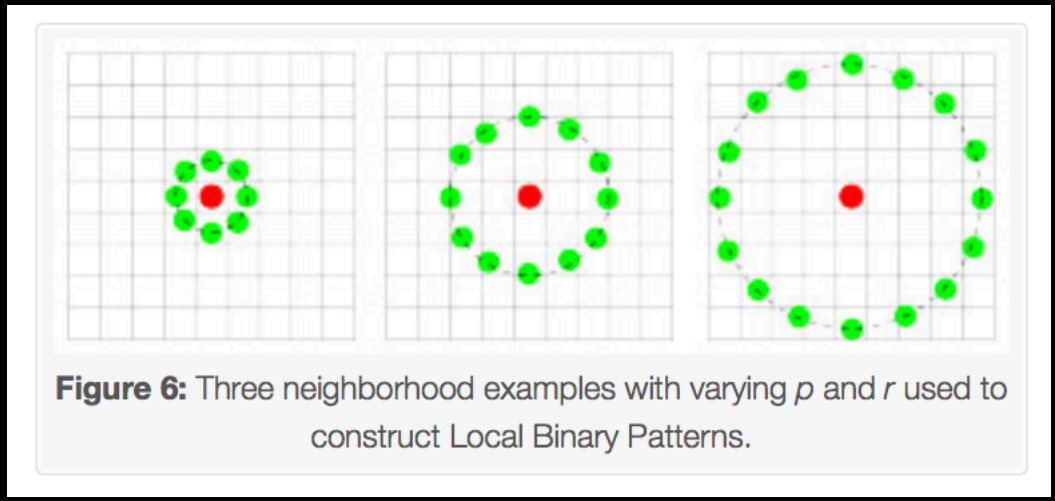


Figure 6: Three neighborhood examples with varying *p* and *r* used to construct Local Binary Patterns.

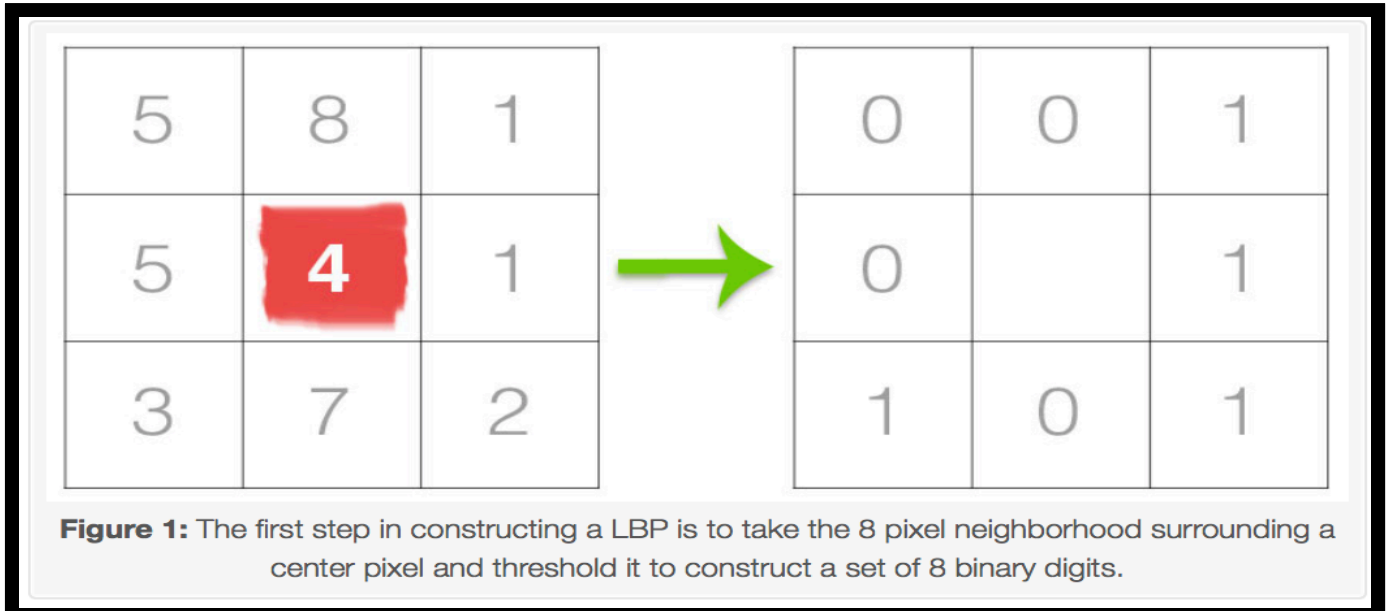


Figure 1: The first step in constructing a LBP is to take the 8 pixel neighborhood surrounding a center pixel and threshold it to construct a set of 8 binary digits.

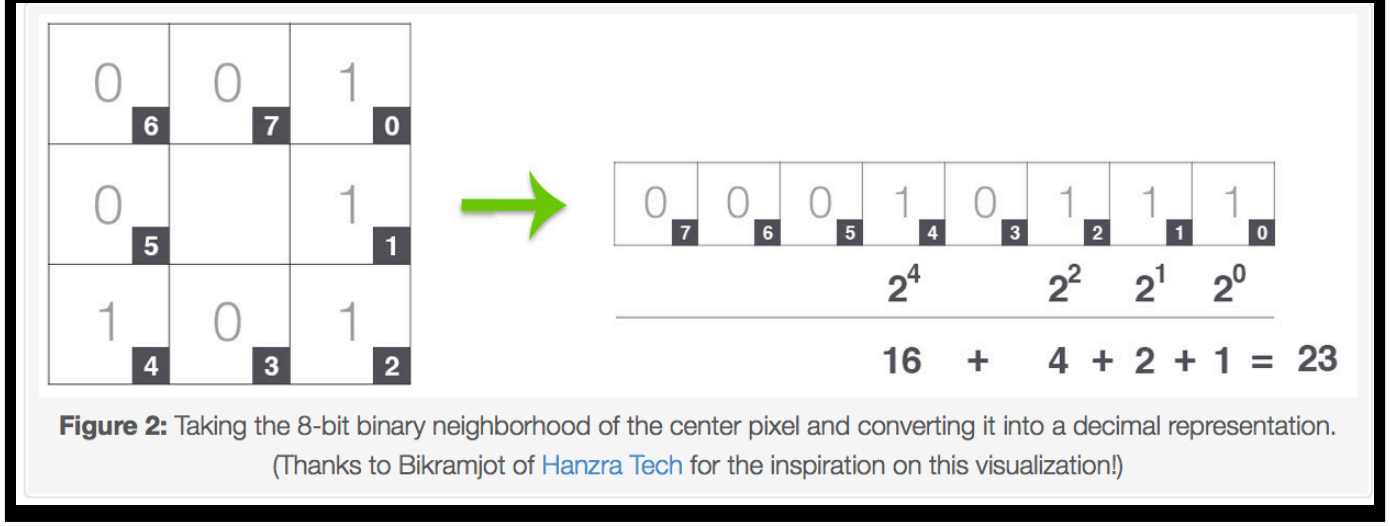


Figure 2: Taking the 8-bit binary neighborhood of the center pixel and converting it into a decimal representation. (Thanks to Bikramjot of Hanzra Tech for the inspiration on this visualization!)

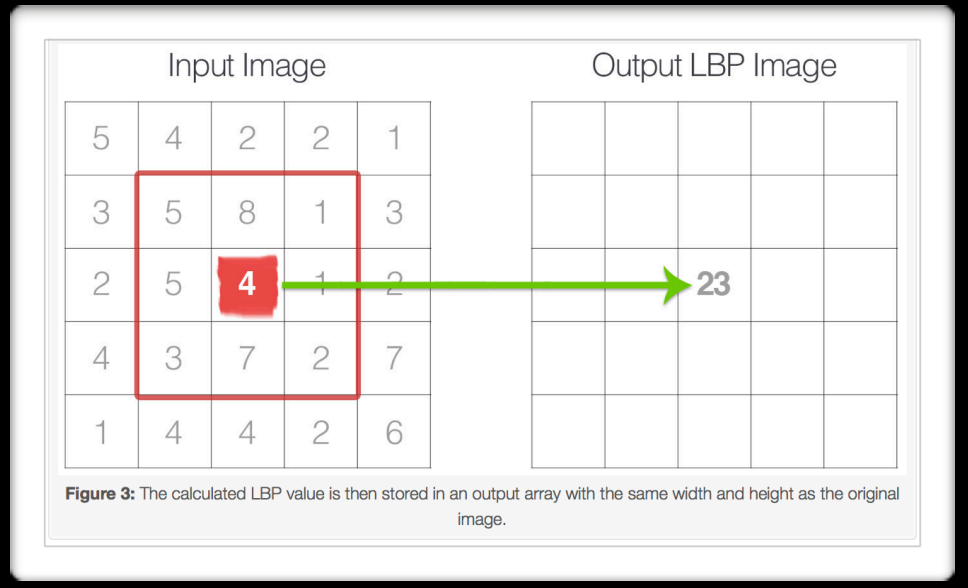
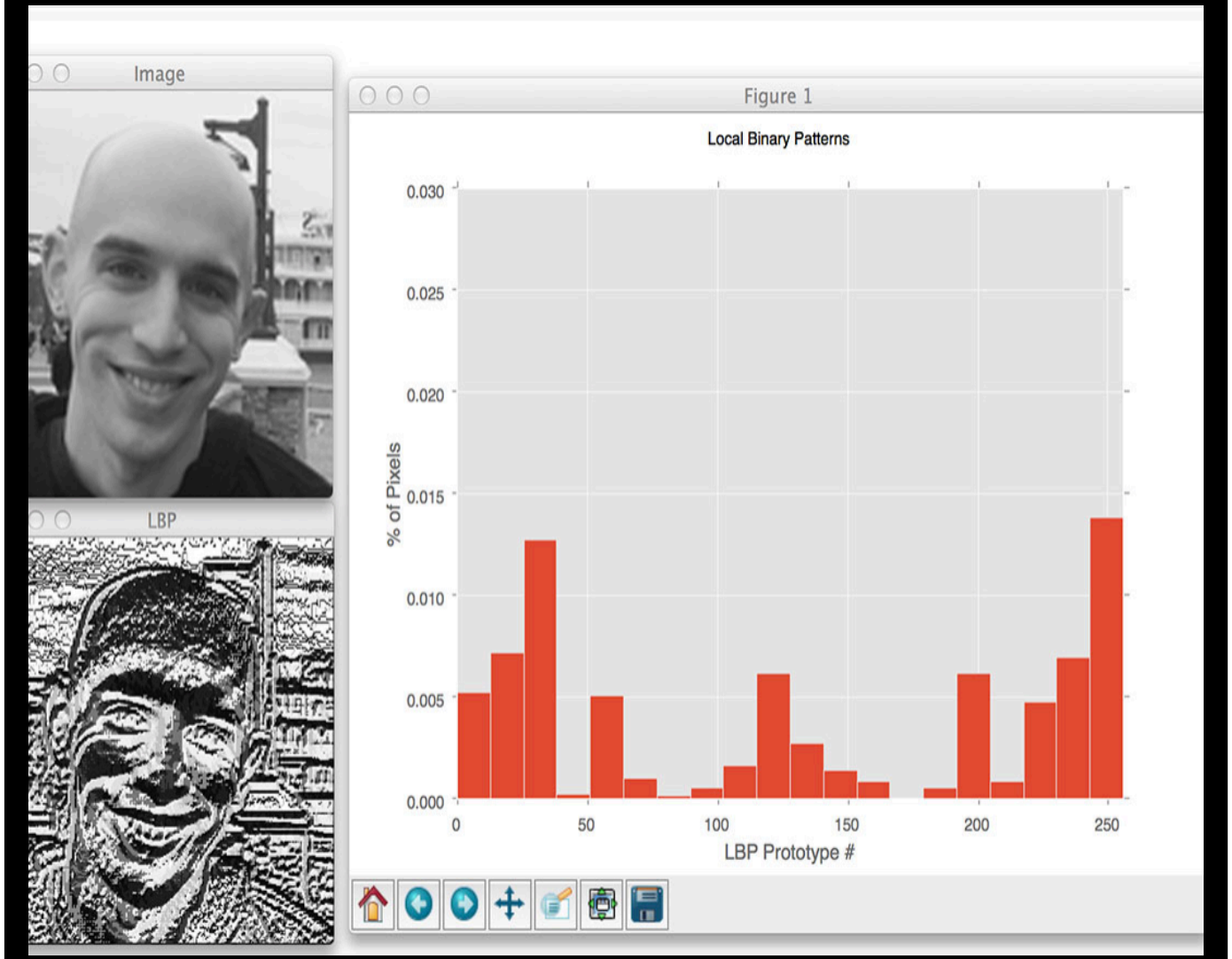


Figure 3: The calculated LBP value is then stored in an output array with the same width and height as the original image.



LBPH

How LBPH Works Internally !

Local Binary Patterns Histogram

createLBPHFaceRecognizer

C++: Ptr<FaceRecognizer> **createLBPHFaceRecognizer**(int **radius**=1, int **neighbors**=8, int **grid_x**=8, int **grid_y**=8, double **threshold**=DBL_MAX)

Parameters:

- **radius** – The radius used for building the Circular Local Binary Pattern. The greater the radius, the
- **neighbors** – The number of sample points to build a Circular Local Binary Pattern from. An appropriate value is to use `` 8`` sample points. Keep in mind: the more sample points you include, the higher the computational cost.
- **grid_x** – The number of cells in the horizontal direction, 8 is a common value used in publications. The more cells, the finer the grid, the higher the dimensionality of the resulting feature vector.
- **grid_y** – The number of cells in the vertical direction, 8 is a common value used in publications. The more cells, the finer the grid, the higher the dimensionality of the resulting feature vector.
- **threshold** – The threshold applied in the prediction. If the distance to the nearest neighbor is larger than the threshold, this method returns -1.

CODE

```
email_update_interval = 60 # sends an email only once in this time interval
object_classifier = cv2.CascadeClassifier("haarcascade_frontalface_alt.xml") # an opencv classifier

Subjects = ["","Deep Thanki", "Rutvik Shah" ]

time.sleep(10)

def detect_face(img):

    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    face_cascade = cv2.CascadeClassifier('haarcascade_frontalface_alt.xml')
    faces = face_cascade.detectMultiScale(gray, scaleFactor=1.2, minNeighbors=5 , minSize=(30, 30), flags = cv2.CASCADE_SCALE_IMAGE)

    if (len(faces) == 0):
        return None, None

    (x, y, w, h) = faces[0]

    return gray[y:y+w, x:x+h], faces[0]
```

Image Detection

CODE

```
def prepare_training_data(data_folder_path):

    dirs = os.listdir(data_folder_path)

    faces = []
    labels = []

    for dir_name in dirs:

        if not dir_name.startswith("s"):
            continue

        label = int(dir_name.replace("s", ""))

        subject_dir_path = data_folder_path + "/" + dir_name

        subject_images_names = os.listdir(subject_dir_path)

        for image_name in subject_images_names:

            if image_name.startswith("."):
                continue

            image_path = subject_dir_path + "/" + image_name
            image = cv2.imread(image_path)
            cv2.imshow("Training on image...", cv2.resize(image, (400, 500)))
            cv2.waitKey(100)
            face, rect = detect_face(image)
            if face is not None:
                faces.append(face)
                labels.append(label)

    return faces, labels

cv2.destroyAllWindows()
faces, labels = prepare_training_data("training-data")
face_recognizer = cv2.face.createLBPHFaceRecognizer()
face_recognizer.train(faces, np.array(labels))
```

Training Data Set

CODE

```
video_camera = VideoCamera(flip=True)

# App Globals (do not edit)
app = Flask(__name__)
last_epoch = 0

def check_for_objects():
    global last_epoch
    while True:
        frame = video_camera.get_frame()
        original = frame.copy()
        gray = cv2.cvtColor(frame , cv2.COLOR_BGR2GRAY)
        face_cascade = cv2.CascadeClassifier("haarcascade_frontalface_alt.xml")
        faces = face_cascade.detectMultiScale(gray, 1.2 , 5)


        for img in faces:
            x,y,w,h = img
            label= face_recognizer.predict(gray[x:x+w,y:y+h])
            cv2.rectangle(original, (x,y) , (x+w,y+h) , (0,255,0) , 1)
            name = Subjects[label]
            cv2.putText(original, name, (x, y), cv2.FONT_HERSHEY_PLAIN, 1.5, (0, 255, 0), 1)

        cv2.imshow("face_recognizer",original)
        ret, jpeg = cv2.imencode('.jpg', original)

        if cv2.waitKey(1) & 0xFF == ord('q'):
            break

        if len(faces) > 0  and (time.time() - last_epoch) > email_update_interval:
            last_epoch = time.time()
            print "Sending email..."
            sendEmail(jpeg.tobytes())
            print "done!"
```

Recognition

CODE

```
@app.route('/')
def index():
    return render_template('index.html')

def gen(camera):
    while True:
        frame = camera.get_frame_np()
        yield (b'--frame\r\n'
               b'Content-Type: image/jpeg\r\n\r\n' + frame + b'\r\n\r\n')

@app.route('/video_feed')
def video_feed():
    return Response(gen(video_camera),
                    mimetype='multipart/x-mixed-replace; boundary=frame')

if __name__ == '__main__':

    t = threading.Thread(target=check_for_objects, args=())
    t.daemon = True
    t.start()
    app.run(host='0.0.0.0', debug=False)
```


```
@app.route('/')
def index():
    return render_template('index.html')

def gen(camera):
    while True:
        frame = camera.get_frame_np()
        yield (b'--frame\r\n'
               b'Content-Type: image/jpeg\r\n\r\n' + frame + b'\r\n\r\n')

@app.route('/video_feed')
def video_feed():
    return Response(gen(video_camera),
                    mimetype='multipart/x-mixed-replace; boundary=frame')

if __name__ == '__main__':

    t = threading.Thread(target=check_for_objects, args=())
    t.daemon = True
    t.start()
    app.run(host='0.0.0.0', debug=False)
```

CODE

```
# Email you want to send the update from (only works with gmail)
fromEmail = 'projectgp21@gmail.com'
# You can generate an app password here to avoid storing your password in plain text
# https://support.google.com/accounts/answer/185833?hl=en
fromEmailPassword = 'elproject'

# Email you want to send the update to
toEmail = 'projectgp21@gmail.com'

def sendEmail(image):
    msgRoot = MIMEMultipart('related')
    msgRoot['Subject'] = 'Security Update'
    msgRoot['From'] = fromEmail
    msgRoot['To'] = toEmail
    msgRoot.preamble = 'Raspberry pi security camera update'

    msgAlternative = MIMEMultipart('alternative')
    msgRoot.attach(msgAlternative)
    msgText = MIMEText('Smart security cam found object')
    msgAlternative.attach(msgText)

    msgText = MIMEText('<img src="cid:image1">', 'html')
    msgAlternative.attach(msgText)

    msgImage = MIMEImage(image)
    msgImage.add_header('Content-ID', '<image1>')
    msgRoot.attach(msgImage)

    smtp = smtplib.SMTP('smtp.gmail.com', 587)
    smtp.starttls()
    smtp.login(fromEmail, fromEmailPassword)
    smtp.sendmail(fromEmail, toEmail, msgRoot.as_string())
    smtp.quit()
```

Email

APPLICATIONS

- Automated Attendance Systems
- Automated Door Locking - Unlocking Mechanisms
- Security Cameras
- Theft Detection Systems at ATMs
- Keeping Track of number of people at a place
- Keeping track of unusual activities at a public place
- Accident Detection on Roads

THANK YOU