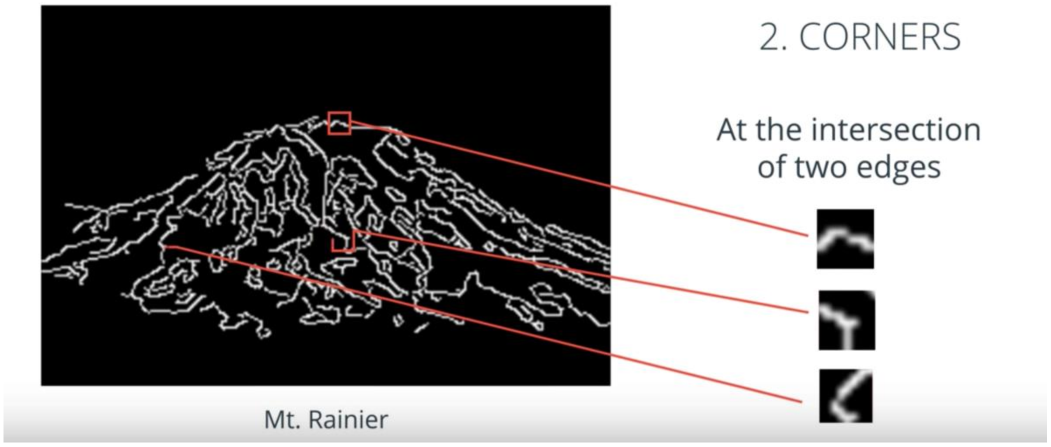
Features alone and together

Now, you've seen examples of shape-based features, like corners, that can be extracted from images, but how can we actually uses the features to detect whole objects?

Well, let's think about an example we've seen of corner detection for an image of a mountain.



Corner detection on an image of Mt. Rainier

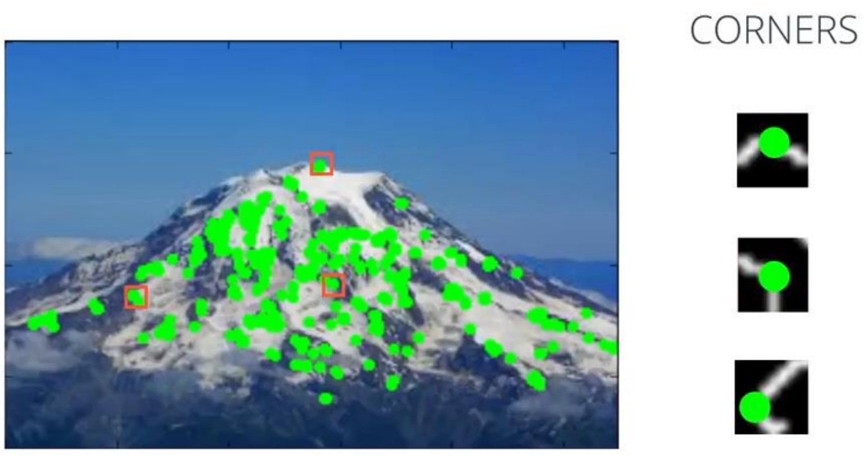
Say we want a way to detect this mountain in other images, too. A single corner will not be enough to identify this mountain in any other images, but, we can take a **set of features that define the shape of this mountain, group them together into an array or vector, and then use that set of features to create a mountain detector**!

In this lesson, you'll learn how to create feature vectors that can then be used to recognize different objects

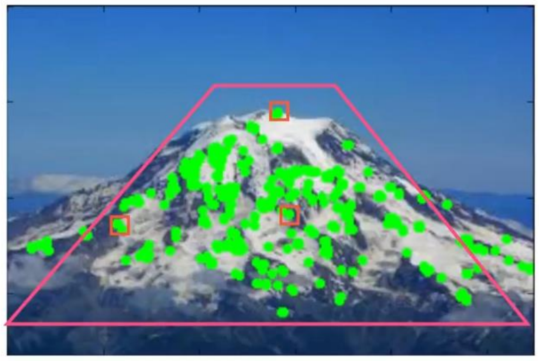
Feature Vectors

Now u have seen the corner are good unique features that can help identify points on an object and this is a great example of how object structure and image gradient can be useful in feature recognition

FEATURE_VECTORS

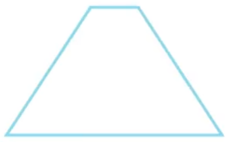


But if we`re interested not only in feature points like corners , but in whole object detection on recognize relies on recognizing distinct sets of feature often called ‘feature vector ‘ and in fact if you look at the direction of multiple gradients around the center point in an image you can get a feature vector a robust representation of the shape of an object



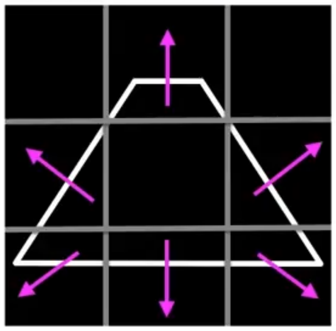
Let`s look at a few simple example to better understand this idea

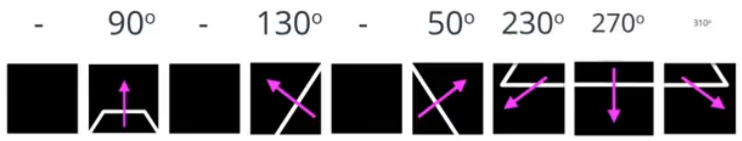
Take this trapezoid and its corresponding gradient which is just an edge detected image .



If we break this image up into a grid of cells and look at the direction of the gradient in each with respect to the center of the trapezoid

FEATURE_VECTORS





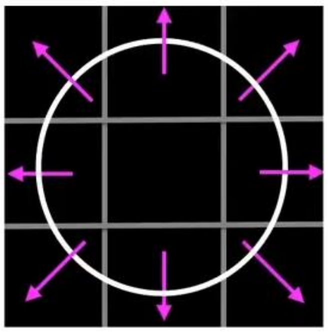
We can flatten this data and create a 1D array this is feature vector

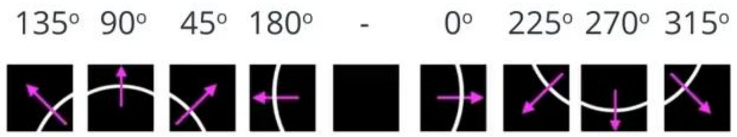
And in this case it`s a vector of gradient directions

We can do the same thing for other shapes like circle taking taking a gradient and calculating the direction of the gradient in each grid cell and we get a different feature vector for each shape

These two feature vectors exactly represent these two shapes , but ideally to accuracy identify any circle or trapezoid at different sizes or from different perspectives , these vectors should allow for enough flexibility to detect some variation in these shape while remaining distinct enough to be able to distinguish different shapes

Next we will talk about some common feature methods that allow for this flexibility and use pattern in image gradient to recognize different objects





Realtime Feature detection

Having a fast object recognition algorithm is essential for many computer vision application augmented reality , robotics and self-driving car

These kind of applications need to respond to input video stream in real time and so the time is critical

A Self-driving car for example need to constantly check whether another car in front of it so that it can decided whether to slow down brake or accelerate

Similarly augmented reality applications rely on applications live view of the real world to decided where to place virtual elements in a given scene For these real-time applications to work they need to be identify the object to identify the objects in the field of view in a fast and efficient manner This is where OBR come in The OBR algorithm is used quickly to create feature vectors of key points in an image These feature vectors can then be used to identify objects in image

How algorithms work ?

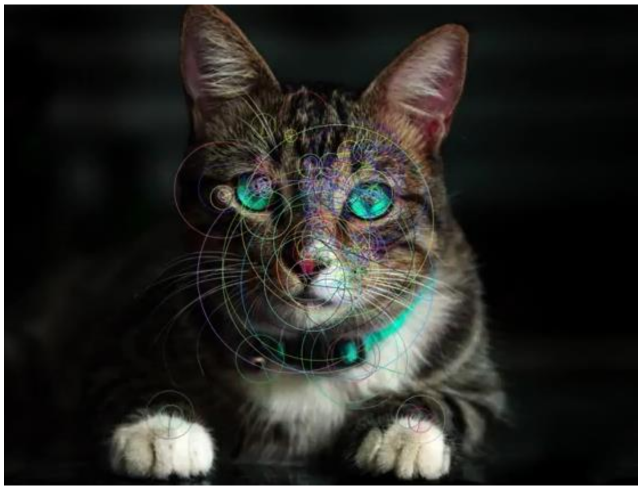
Oriented fast and Rotated Brief are feature detection a vector creation algorithm respectively Orb ,start by finding special region in an image called key points

In general you can think of a key point as small region in an image that is particularity distinctive.

An example is corners , where is pixel value sharply change from light to dark

Here we can see example of ORB generated key points in the image of a cat

We see the points around the cat`s eyes and at and the edges of its facial features



FEATURE_VECTORS

Once the key points in the image had been located orb then calculating the corresponding feature vector for each key point

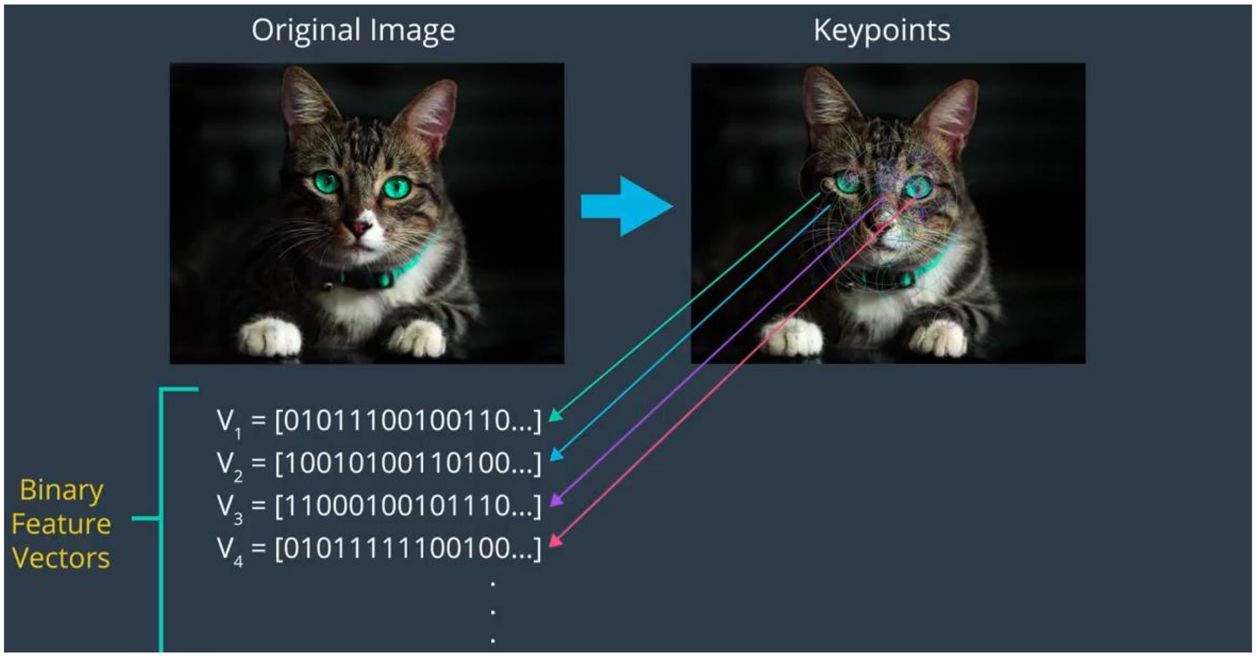
The orb algorithm creates feature vectors that only contain ones and zeros for this reason they `re called a binary feature vector

The sequence of ones and zeros varies according to what a specific key point and its

Surrounding area look like

The vector represent the pattern of intensity around a key point

So multiple feature vectors can be used to identify larger area and even a specific object in image


Original Image
Keypoints
Binary Feature Vectors
V_1 = [01011100100110...]
V_2 = [10010100110100...]
V_3 = [11000100101110...]
V_4 = [01011111100100...]

ORB is not only incredible fast but it`s also impervious to noise illumination and image transformation such as rotations I mentioned that ORB , stands for Oriented fast and Rotated Brief

ORB is FAST algorithm which is done is done by FAST algorithm

FAST stands for Features from Accelerated Segments Test and it quickly select the key points by Comparing the brightness levels in a given pixel area

Example

Given a pixel , which I will call p in an image

FAST compares the brightness of p to a set of surrounding pixels that are in a small circle around p . Each pixel in this circle is then sorted into three classes

1- Brighter than p
2- Darker than p
3- Similar to p

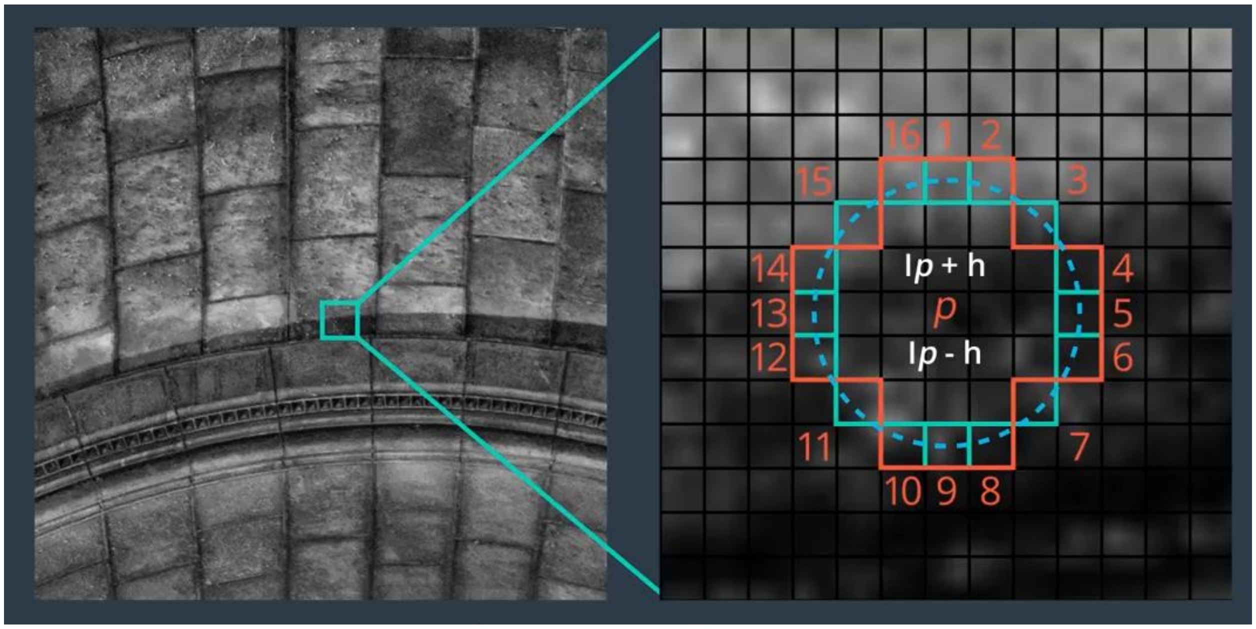
I refer to brightness ip => intensity of pixels p

So if the brightness of a pixel is ip then for a given threshold h

1- brighter pixels will be those Whose exceeds ip [plus + h]
2- Darker pixels will be those whose brightness is below [ip - h]
3- Similar pixel will be those brightness lie between those values

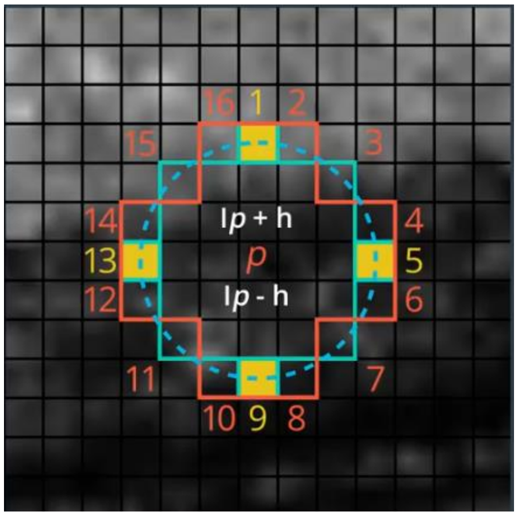
Once the pixels are classified , pixel p is selected as key point if more than eight connected pixels on the circle are either darker or brighter than p .

The reason FAST is so efficient is a take advantage of the fact that the same result can be achieved by comparing p to **only four equidistant pixels in the circle instead of all 16 surrounding pixels**



For example

We only have to compare p to pixels 1 , 5 , 9 and 13 in this case p , is selected as a key point if there are at least a pair of consecutive that are either brighter or darker than

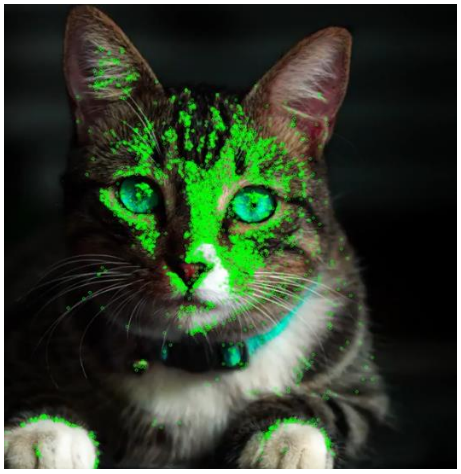


The optimization reduces the time required to search an entire image for key by a factor of four

what type of information are these key point providing us ?

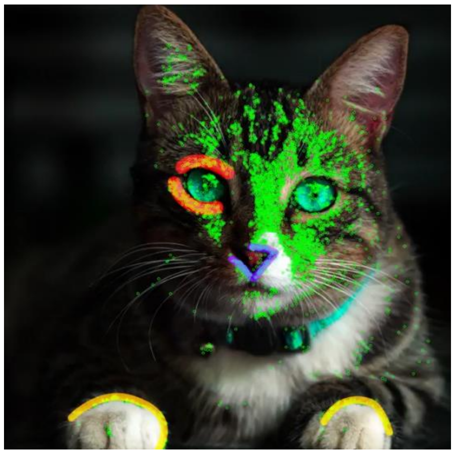
what`s so meaningful about comparing the brightness of neighboring pixels ?

well . Let`s look at some of the key points found by FAST on the image of a cat



There are key point at the edge of the eyes there`s another group of key points at the edge of the nose as we can see the key points are located in regions where there is a change in intensity such regions , usually determined an edge of some kind like in the cat`s paws

Edges define the boundaries of the cat and the boundaries of the facial components and so these key points give us a way to identify this a cat as opposed to any other object or background in the image so the key points found FAST GIVE US information's about the location of object defining edges in an image



However , one thing to note that these key points only give us the location of an edge and don`t include any info about the direction of the change of intensity so we can now distinguish horizontal and vertical edge and we `ll see later this directionally can be useful in some cases Now that we know how ORB uses FACT to locate the key points in an image let`s take a look at how uses the brief algorithm to convert these key points into a feature vectors

Brief algorithm convert these key points into feature vectors

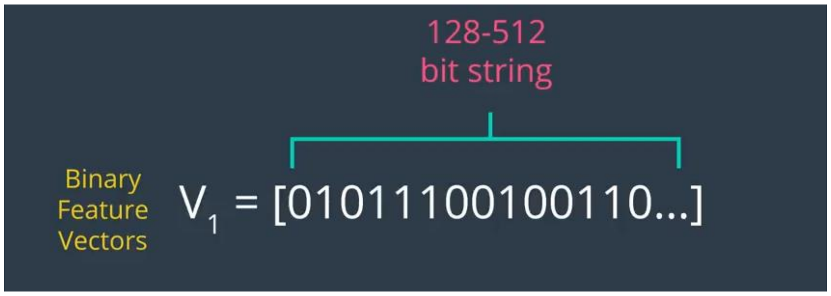
Second part of algorithm to convert to vector feature can represented an object to create a feature vector orb uses the BRIEF algorithm

BRIEF stands for binary robust independent elementary features and its purpose is to create binary feature vectors from set a key points

Binary feature vector also known as a binary descriptor is just a feature vector that contains only ones and zeros

FEATURE_VECTORS

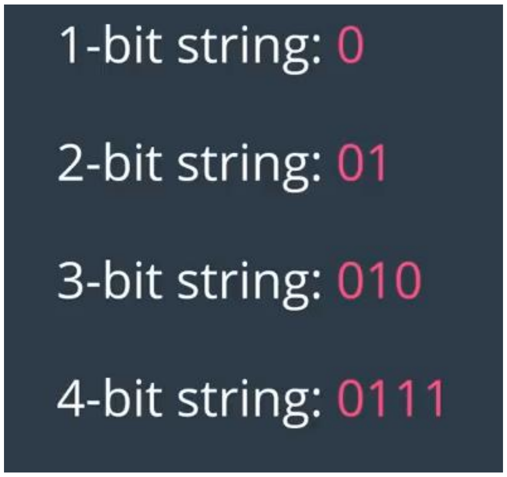
In BRIEF each key point is described by a binary feature vector that has a 128 – 512 bits string that only contains ones and zeros



Just as a reminder bits is short for a binary digit and one bit can hold only a single binary value either one or zero and string a group of bits

Here we see some example of bits string

1- the first is one bit string therefore it only holds 1 bit
2- The second is a two bit string therefore it can hold 2 binary digits (hold zero or one)
3- The third is a three bit string and so it can hold 3 bits and so on

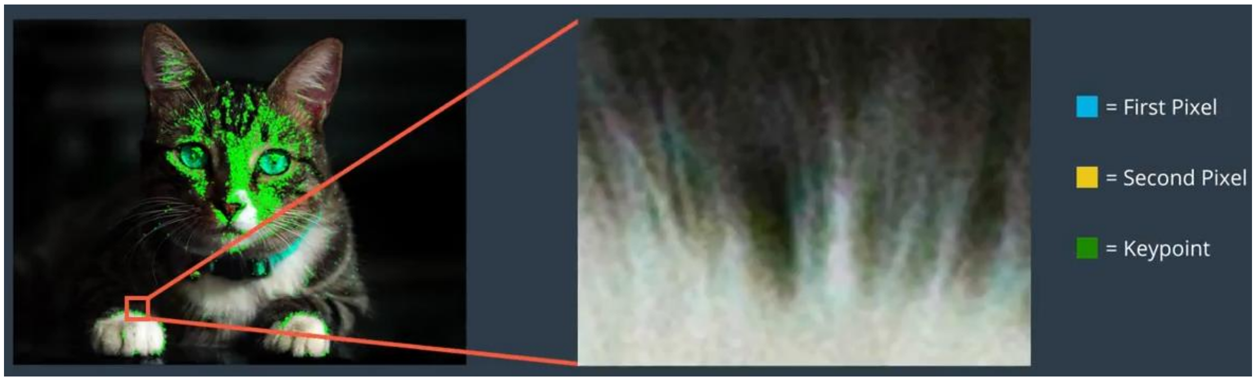


Computer run a binary or machine code and so great advantage of using **binary feature vectors** is that they can be stored very efficient in memory and they can be computed quickly These properties not only **make BRIEF very fast which is crucial for real time applications** but they also allow BRIEF to run on devices with very limited computational resources such as smartphone

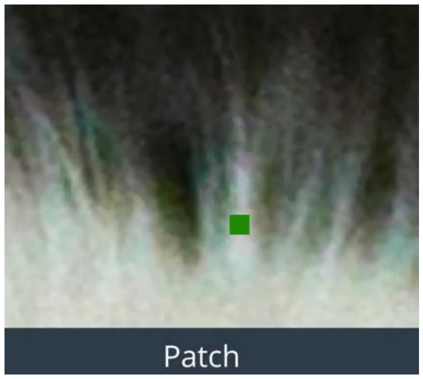
How does BRIEF create these binary descriptions for each point

The BRIEF algorithm starts by smoothing a given image with Gaussian kernel in order tp prevent the descriptor from being too sensitive to high frequency noise

Next given a key point like the one here and the cat`s paw BRIEF select random pair of pixel inside a defined neighborhood around that key point

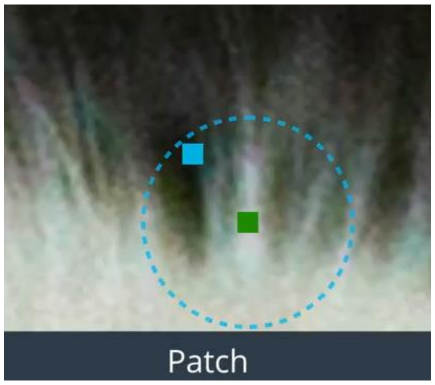


The neighborhood around the key point is know as patcH which is the square with some pixel width and height

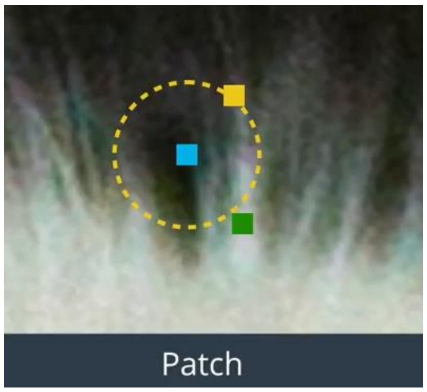


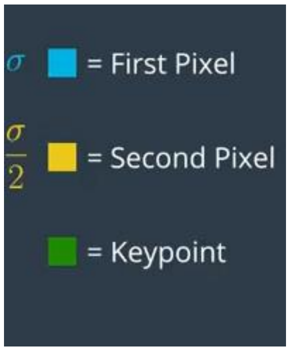
1- The first pixel in the random pair , show is a blue square is drawn the a Gaussian distribution centered the key point and with the standard deviation or spread of sigma
2- The second pixel in random pair show here as yellow square , is drawn from a Gaussian distribution centered around that first pixel and with a standard deviation or Sigma over 2 it`s show by empirically that this choice of Gaussian improvers the feature matching rates

First Pixel



Second pixel



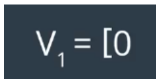


BRIEF then starts constructing the binary descriptor for the key point by comparing the brightness of the two pixels as follows

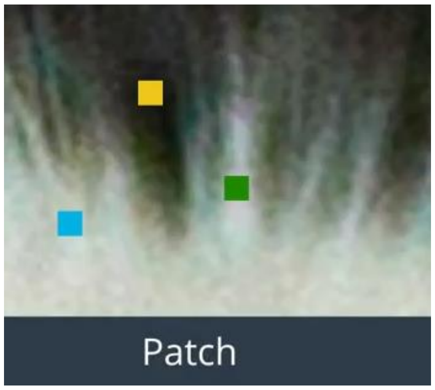
1- **If the first pixel is brighter than second (it assigns the value of one to the corresponding bit in the descriptor**)
2- **Otherwise it assigns the value of zero**

In the example

The second pixel is brighter than the first pixel so we assign a value zero to the first bit our feature vector



Now of the same key point , BRIEF selects a new random pair of pixels , compare their brightness and assigns one or zero to the next bit and the feature vector .



In our example we see that now the first pixel is brighter than is second therefore we assign a value of one to the second bit in our feature vector

For a 256 bit BRIEF then repeat this process for the same key point 256 times before moving onto the next key point 256 times before moving onto the next key point

The result of the 256 pixel brightness comparison and then put into the binary feature vector for that one key point

BRIEF creates a vector like this for each key point in an image

We will see how orb uses these to create vectors that are robust in the face or image rotation scale and noise

Feature Matching

How to use ORB descriptor to perform object Recognition ?

Example: Displays how ORB can detect ?

The same object at different scales and orientations

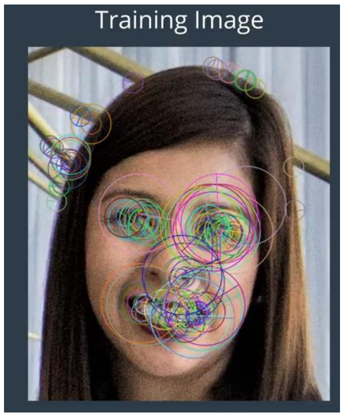
Suppose I want to be able to detect this person's face in other image , Say in this image of multiple people

FEATURE_VECTORS

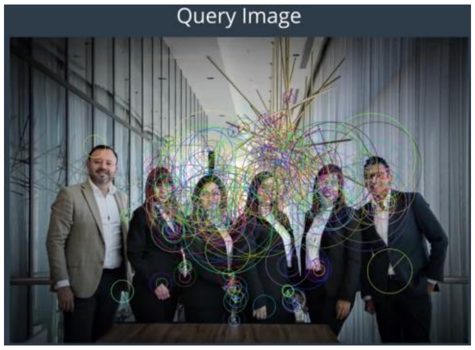
We call the first image in the left training image ,

This second image in the right in which I want to perform face detection will be called the **query image** . So given this training image , I want to find similar features in this query image

1- The First Step will be calculate the ORB descriptor for the training image and save in memory . The ORB descriptor will contain the binary feature vectors that describes each key point in this training image .



2- The second step will be compute and save he ORB descriptor for the query image .



Once we have the descriptor for both training and query images ,

3- The Finial step is perform **key point matching** between two image using their corresponding descriptors (**the matching is usually performed by matching function , whose aim is to match key points in two different images by** comparing their descriptor and seeing =>
   a. If they `re close enough together to make for a match)

When a matching function compares two key points it reaches the quality of the match according to some metric , something that represents the similarity of the key point features vectors .

You can think of this metric as being similar to the standard Euclidean distance between two key point some metric simply ask ,

1- **Do the feature contain a similar order of one and zeros ? (it important to keep in mind that different matching functions will have different metric for determining the quality of the match)**

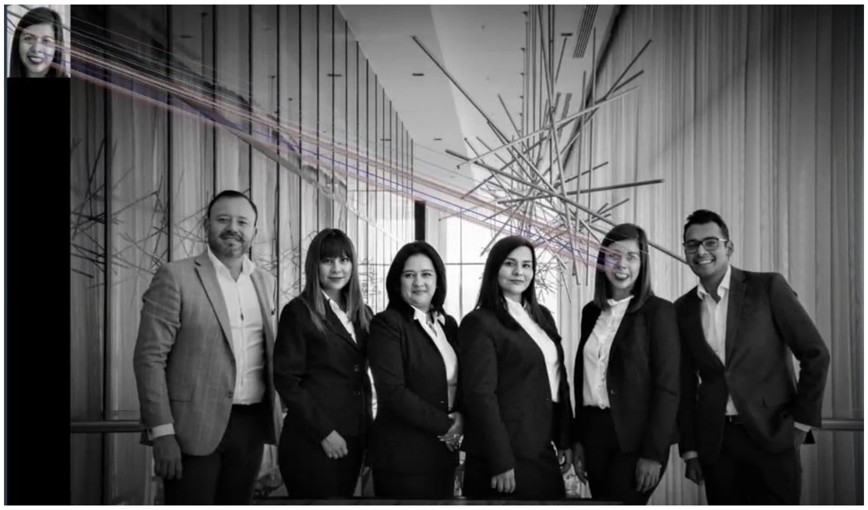
For binary descriptor like the ones used by ORB the hamming metric is usually used because it can be performed extremely fast .

The hamming metric determines the quality of the match between two key points by counting the number of dissimilar bits between their binary descriptor.

When comparing the key points in the training image with the ones in the query image,

[the pair with the smallest number of differences is considered to be the best match]

Once the matching the function has finished comparing all the key point in the training and the query image [it return the best matching pairs of the key points]



The best matching points between our training image and our query image are displayed here

We can Clearly see that the best matching points between our training image and our query image mostly all correspond to the face in the training image .

There are one or more features that don`t quite match up , but may have been chosen because of similar patterns of intensity in the area of the image. Since most points correspond to the face in the training image , we can say that that matching function has been able to recognize this face in the query image correctly .

ORB in tracking video

In common ORB used tracking and identifying objects in the real time video streams .

In this case , we compute the ORB descriptors for any images or objects we want to detect , before seeing a video stream and save those descriptors .

In each frame in an incoming video stream we calculate the ORB descriptors and use a matching function to compare the key points in the current video frame with the saved descriptors .

For any object descriptor that we have saved.

1- If we find that the matching function returns a number of matches above some match threshold , we can conclude that the object in the frame .

The mesh thresholds a free parameter that you can set For example .

1- if the ORB descriptor for a particular object 100 key points , then you could set the threshold to be 35 percent , 50 percent or 90 percent of the number of key points for the particular object
2- If you set the threshold to 35 percent , then that means that at least 35 key points out of the 100 that describes that objects , must match in the order to say that object is in the frame .

All these step can be done in near real time because ORBs binary descriptors are extremely fast to compute and compare.

The algorithm works best when you want to detect objects that have a lot of consistent features that are not affected by the background of an image .

For example ORB works well for facial detection because faces have a lot of features such as

1- Corners of the eyes
2- Corner of the mouth

Don`t appear to change no matter where a person is .

These feature are consistent from image to image However ORB does not work so well when attempting to do more general object recognition

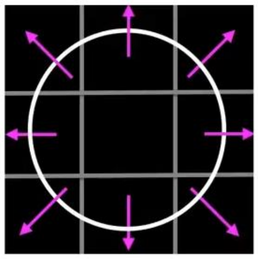
Say pedestrian detection in image in which the shape and feature of a person`s body vary depending on clothing and movement for this type of general object recognition other algorithm work much better

We will learn about methods that can be used to detect to do more general object recognition .

Pros and corns of ORB algorithm ?

Histogram of Oriented Gradient

In computer vision there are many algorithms that are designed to extract spatial features and identify object using information about image gradients



Direction of the gradient

One illustrative technique is called HOG or **Histogram of Oriented Gradient .**

Histogram of Oriented Gradient may sound a little intimidating , so let`s go through what these term actually means !!!

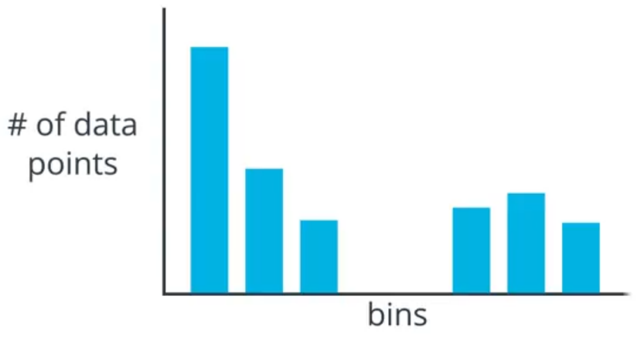
A histogram is a graphical representation of the distribution of data

It look a bit like a bar graph with bars of different heights.

Each bar represents a group of data that falls in a certain range of values.

Also called

1- **Bins**
2- **And taller bars indicate that more data falls into a certain bin**



For example

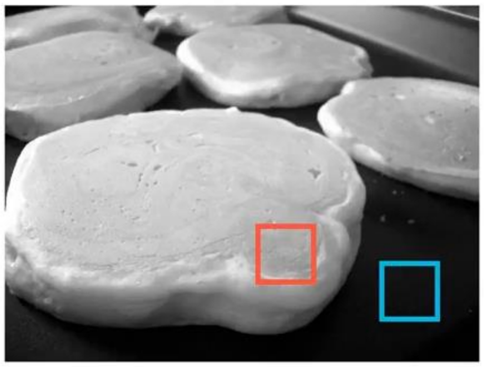
Say look at the a grayscale image . Say this image of pancakes and wanted to display a histogram of intensity data.

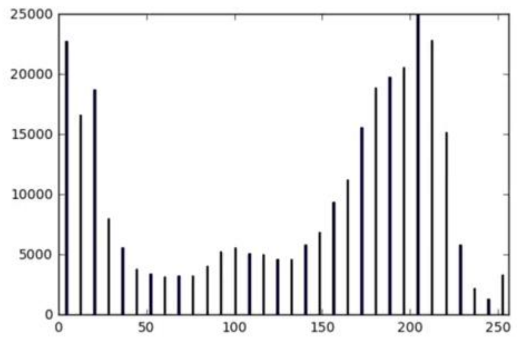
We know that all these pixel values range from 0 to 255 and we can create a bins to partition تقسيمthese values into ranges .

I will create 32 bins each holding a range of 8 pixel values 0 to 7 and so on up to 8 to 15 up to 248 to 255

Then to create a histogram we look at every pixel value in this range and put each one in its correct bin

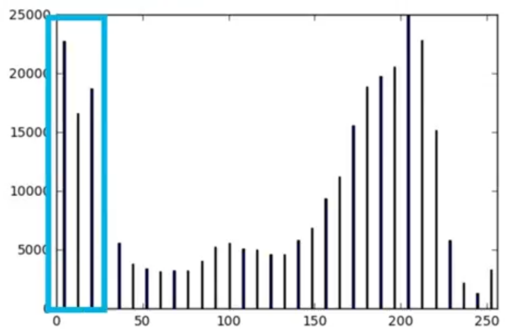
This image has a lot of bright values in the pancakes , but very dark background



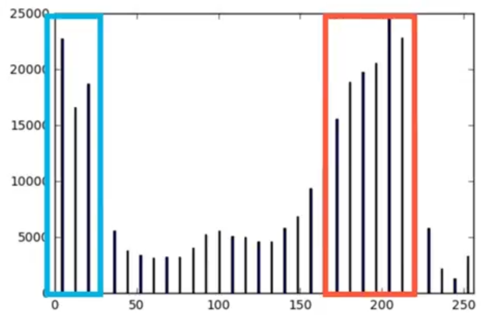


So we get a histogram that look like this .

This histogram shows that there`s a distinct dark grouping of pixels the background pixels, the fall in these low ranges .



And there`s another bright grouping of pixels that are often around a grayscale value of 200 which must be majority of the pancakes pixel each value



So now we know what a histogram of grayscale values look likes .

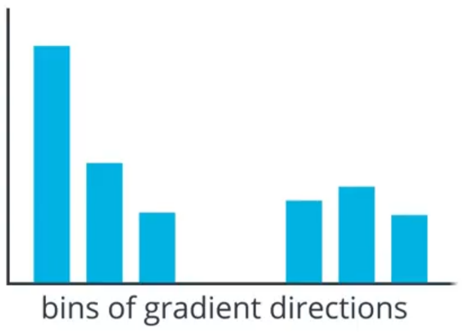
The next words we see are oriented gradients .

Oriented Gradients

Oriented just means the direction or orientation of an image gradient And we `re already discussed how both the magnitude and the direction of a gradient can be calculated using a **sobel operators** .

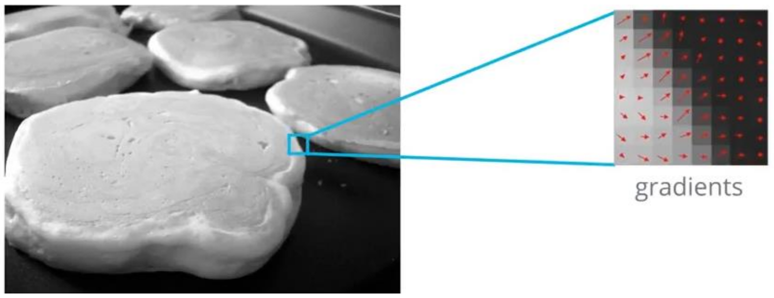
So let`s put to all together

HOG should produce a histogram of gradient directions in an image.

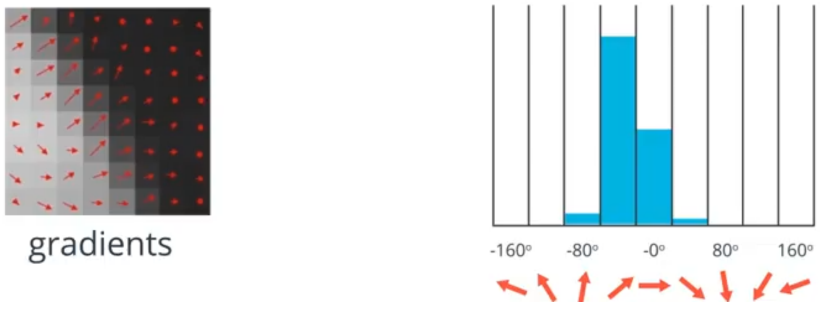


FEATURE_VECTORS

1- HOG take in an image like this pancakes image and calculates the magnitude and directions of the gradient at each pixel This is can be a lot of information's
2- Group these pixels into square cells (into a larger cells typically eight by eight or smaller grids for smaller pictures.) for eight by eight case , it will have 64 gradient values , then for each of these cells it counts up how many of these gradient are in a certain direction and sum the magnitude of the gradient so that the strength of the gradient are accounted for.



3- Then HOG places all the directional data into a histogram here i`m showing you nine bins or ranges of values but you can choose to use more bins to futher divide your data . and this is the histogram of oriented gradient that the pancakes edge into and HOG does this for every cell in the image.



This histogram of oriented gradients is actually a feature vector.

The next step will be to actually use these HOG features to train a classifier.

The idea is that among images of the same object at different scales and orientations , the same pattern of HOG feature can be used to detect object wherever and however it appears but first let`s see how to implement HOG in code This is will be a multi step process. So in this and the next for example