REAL TIME FACE RECOGNITION

Real time face recognition is a two part problem.

1. Face detection
   Detecting Faces in an image with multiple objects.

2. Face Recognition
   Recognizing the detected faces

Face Detection

To Tackle the face detection problem we have three classifiers that will help us classify if the image has a face in it or not.

1. HAAR Cascade

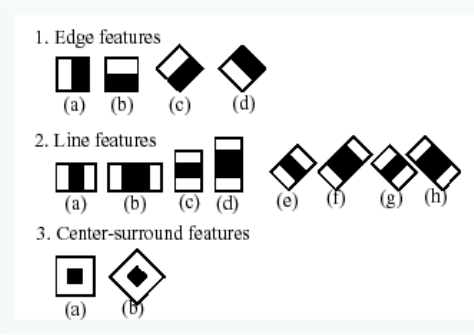
Image Processing

Make a dataset of cropped Faces of faces and Non-Faces
Convert all images to greyscale
Normalize the sub image vectors (0 mean / unit variance)

Algorithm

HAAR Features

HAAR Features are are certain patterns whose combinations when found in an image determines if it is a face or not.



These Features represent pixels in the image matrix. Smaller rectangles and squares windows in the image can show such patterns. These patterns are located using difference in the pixel densiies of these sub windows.

Integral Images

When naively implemented the HAAR feature calculation can be of the order $O(n^2)$; and this makes sense because in just a base image of size 24 x 24 there can be more than 180000 features present. Fortunately, we have a solution of this in constant time which allows applications in real time. This O(1) solution is based on Integral images.

The integral image of an image is formed by sum of all pixels in the rectangular area below and to the left of each pixel.

$$ii(x,y) = \sum_{x' \le x, y' \le y} i(x', y'),$$

ii(x',y') -> Integral image of pixel at x',y'

Integral image is acheived using the following set of recurrences

$$s(x,y) = s(x,y-1) + i(x,y) \quad (1)$$
$$ii(x,y) = ii(x-1,y) + s(x,y) \quad (2)$$

i(x,y) -> original image pixel value at x,y
s(x,y) -> cumulative row sum

Using this the features can be calculated with mere 4 references to the integral image.

ADABOOST

This is a boosting algorithm that is modiefied to implement the HAAR cascade.
Adaboost makes many weak models (stumps in case of decision trees) and trains them on a binary classification problem. The misclassified points are then given more weight such that they are classified correctly in the next boosting round.

The HAAR cascade slightly modifies this algorithm to reduce the number of features from a huge number of features.

For each feature a classifier is trained and then the error is calculated with respect to the weights assigned to each image and the absolute value of the | predicted Y – actual Y| .

$$h_j(x) = \begin{cases} 1 & \text{if } p_j f_j(x) < p_j \theta_j \\ 0 & \text{otherwise} \end{cases}$$

<--- Defining a weak learner with one feature , parity and a threshold

The feature with the least error is stored.

Like this in every pass of the adaboost an important new feature is filtered out.

This gives rise to the stronger model which can classify if the image is a face or not.

Attention Cascades

Using the trained ADABOOST on the subwindow gives compelling results but it also detects a lot of false positives. This can be solved using Cascades. The authors of the HAAR cascade paper (Viola and Jones) use Degenerate trees as cascades. A cascade is built up of increasingly complex classifiers.

When a classifier labels the subwindow as a face it goes to the next classifier ; Or else it is rejected completely.

This helps remove a lot of false positives and increases the precision.

This also increases classification speed as many subwindows are rejected in the beginning itself

Implementation

The **Open CV** library has the **cv2.cascadeclassifier** module that has the implementation of the HAAR cascade. It lies in the **'data/haarcascade_frontalface_alt.xml'** folder.

```
haar_face_cascade = cv2.CascadeClassifier('data/haarcascade_frontalface_alt.xml')
```

Use this code to load the classifier.

During the implementation a problem of false positives may arrive and the reason for that may be some of the faces being closer to the camera than others. A simple tweak to the scale factor of the size of subwindows can improve the result.

2. LBP Cascade

Stands for Local Binary Pattern

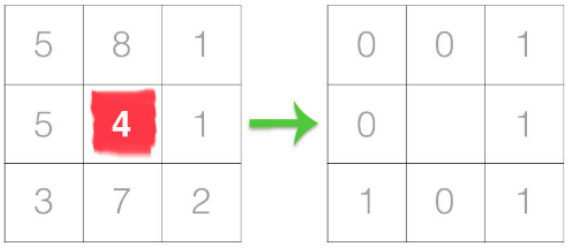
Image Preprocessing

Make a dataset of cropped faces and label them as faces
Have an equal number of Non-Face images and label them as not face
Convert all images to greyscale

Algorithm

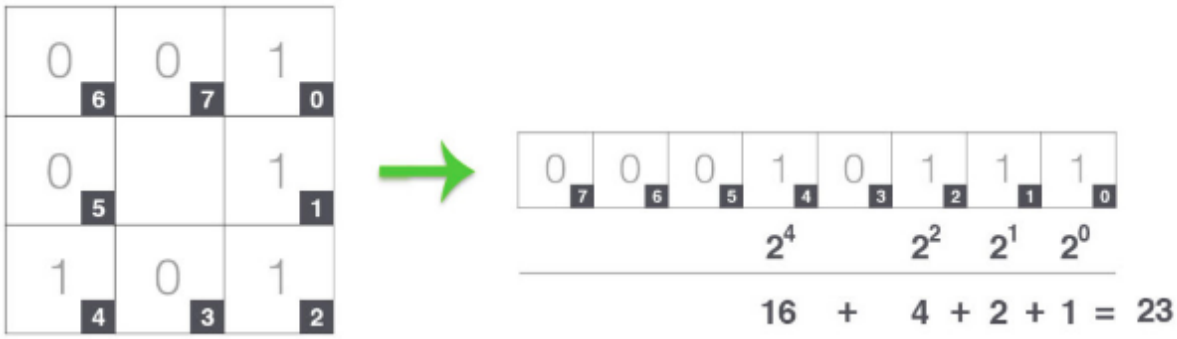
LBP determines local features of faces using values of neighbouring pixels as new pixel values. This helps it compute the local representation of texture

The features are calculated by choosing 3x3 pixel matrixes as sub windows in the image matrix. The middle pixel is then use to set the neighbouring pixels to either 1 or 0 depending on the value of the neighbouring pixel being greater or lesser than the middle pixel.



Then this series of 1s and 0s is aligned and the binary result is converted to decimal. This decimal value is the new value of the middle pixel.

Hence mathematically describing minute textures formed from these 9 pixels.



Since there are 8 bits 2^8 combinations of bits are possible.
Therefor 256 different patterns can rise from this method.

So finally we plot a 256 bin histogram and use it as a feature vector.

Then the job remains for the machine learning model to learn what set of features are for Face images and what set of features are for Non-Face images.

I couldn't find the original implementation used on the internet but since the name has cascades I assume the implementation is pretty similar to that of HAAR cascades and hence I believe that using Cascades of Adaboost should give the desired results.

Instead of 100000s of features like in HAAR Cascade we just have 256 features in the LBP Cascade.
Hence Training is much faster and is the program is much lighter.

Implementation

XML training files for LBP cascade stored in the **openCV/data/lbpcascades** folder

lbp_face_cascade = cv2.CascadeClassifier('data/lbpcascade_frontalface.xml')
Use this to load the classifier

During the implementation a problem of false positives may arrive and the reason for that may be some of the faces being closer to the camera than others. A simple tweak to the scale factor of the size of subwindows can improve the result.

4. YOLO

You Only Look Once

Unlike the HAAR Cascade and the LBP Cascade this object detection algorithm is based on Convolutional Neural Networks and is much more accurate and object detection

PreModelling

Ths model is expected to output a 5 + C dimensional vector where C is the number of classes that the model is expected to detect. The five fixed outputs are Pc , bx , by , bh , bw. [Pc -> if 1 determines existence of atleast one of the classes ; bx,by --> midpoint of bounding box around detected object ; bh,bw --> height and width of the bounding box]
Hence for the model to learn we are expected to give it data labelled with these vectors. Therefore we need to manually give the box cordinates ,etc.
Normalizing the pixel matrix of the image

Algorithm

Initally sliding window algorithm was used where a sub window is strided throughout the image in search of the object it has to detect. A convolutional implementation to do this was also proposed and was used for quite some time. But the problem with sliding boxes was that the bounding boxes were never perfect which limited its real world application in for example self driving cars.

YOLO instead divides the image into grids (around 19-20) and each grid is a sub window.

The Convolutional Net is made such that it outputs a 19 x 19 x (5 + C) matrix

Here we have 19 outputs , 1 for each grid.

The grid that has the midpoint of the object is considered as the grid containing the object and bounding box cordinates for it are set according to what the model learns.

While predicting , the model may predict the objects being in a lot of grids giving rise to false positives or inaccurate bounding boxes.

This problem can be tackled by using the Pc value that each grid outputs as a part of the vector. If Pc value which is the probablity of a class object being there, is lower than a threshold (let's say 0.5) we reject these bounding boxes.

Next part is removing inaccurate bounding boxes.

This can be done using Intersection over Union (IOU). IOU of 2 bounding box is the ratio of their intersection Area over their union Area. The Bounding box with highest probablity is chosen and if the IOU of the other boxes around it is greater than a threshold (let's say 0.5) then we reject them. This way we end up having detected what we wanted.

This above process is called Non Max Suppression

What if two objects that we wish to detect have the same midpoint?

We user anchor boxes for this. Anchor boxes are boxes of different sizes ; So at every midpoint detected, all anchor boxes are considered and the anchor box which has the maximum IOU with the predicted bounding box is chosen as the correct anchor box.

Note: The model learns all this using deep convolution networks.

Implementation

Darknet's github has the souorce code for all YOLO versions written in C.
It is challenging to implement the YOLO algorithm especially from scratch.

A third party implementation is available in tenserflow v2 – Keras in the keras-yolo3 library.

Chosing a Model

Comparing three factors:
1. Time taken
2. Acuracy
3. Ease of implementation (Since I am a beginner)

I choose the HAAR Cascade for object detection

#HAAR
1. Faster than YOLO ; Slower than LBP Cascade
2. Accurate than the LBP Cascade ; Less Accurate than YOLO
3. Easier to implement than YOLO ; LBP is the easiest to implement

Face Recognition

After the detection of faces the task is to recognize the faces.

One Shot Learning

Face recognition is a tough problem to solve as the solution to it demands a one shot learning approach.
A trivial approach to face recognition would be feeding labeled faces to convolution networks and expecting the neural net to learn the faces in order to recognize them.
But this is a pretty bad solution when trying recognize faces. This is merely due to the fact that one would need to retrain the model every time a new person's face has to be recognized.

This is exactly why we need to make the model learn only once, hence the term One Shot Learning.

The solution to this problem is to make a model in such a way that it learns a similarity or a difference function. This allows us to store the encodings of a person in a database so that now we can recognize them by the similarity function.

Difference Function

In face detection we saw that each face can be detected using certain features that nothing but faces have. Similarly each human face has slightly different features that differentiate them from others. These features can be extracted from digital images in the form of encodings and we can train our model to learn the differece between the encodings to identify faces from a given database of encodings.

First I will get into 2 feature extraction methods and then get into the difference function.

Feature Extraction

1. LBPH

LBPH stands for local binary pattern histograms.
It uses an extended version of the LBP algorithm for face detection

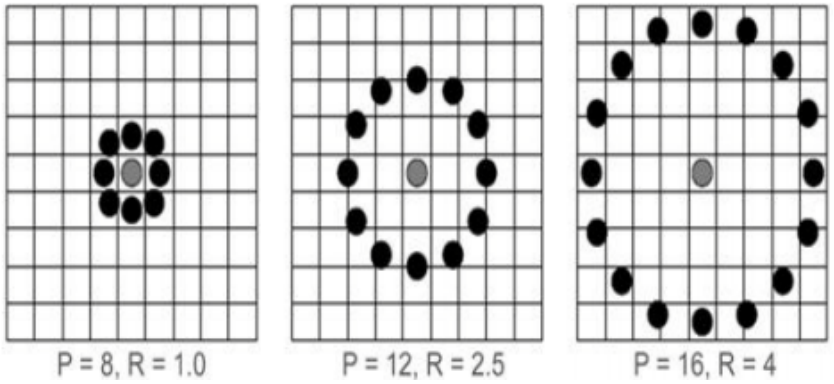
In the LBP algorithm we saw that in a 3 x 3 pixel subwindow a central pixel is chosen and the surrouding pixels are converted into 1s and 0s depending on their value either being greater or smaller than the central pixel. Using this a 256 bin histogram was formed which described a feature vector of the entire image.

There are two problems to this technique:-

1. The 256 bin histogram is not enough for having a feature vector that can specify details of a face needed for a recognition problem.

2. Choosing a 3x3 grid gives huge insight into miniscule local feattures but the larger features are not considered then.

There is a common solution for both these problems.
Instead of constraining the grids to be 3x3 we can consider different radii and extend the features for the task.



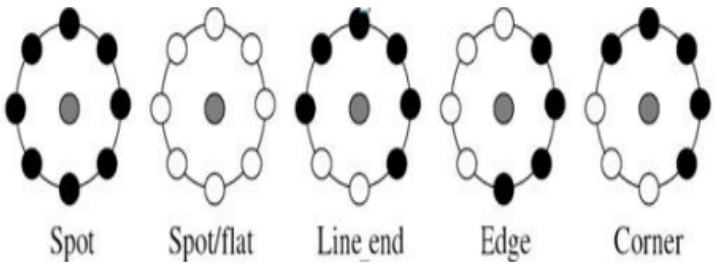
P -> Number of pixels in the circumference

R -> Radius

UNIFORM LBP

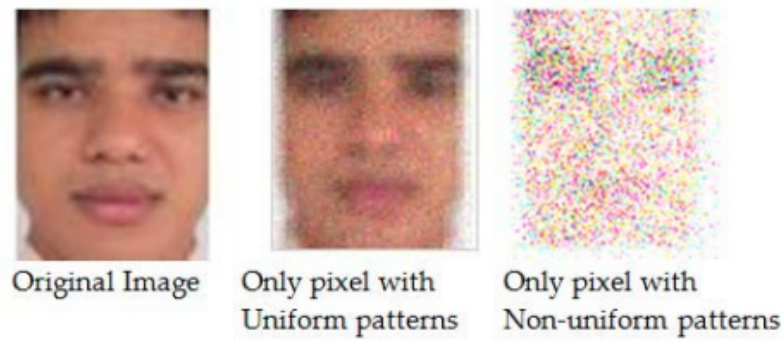
A local binary pattern is said to be uniform if the number of transitions the 1s and 0s show is 2.
for eg. In an 8 bit binary 11000011 , the number of transitions are two while in for eg 11111111 or 11001100 The number of transitions are 0 and 3 respectively.

It is seen that when considering only Uniform LBPs 99% of the important features are extracted. Along with these 2 Non Uniform LBPs (11111111 and 00000000) are considered.



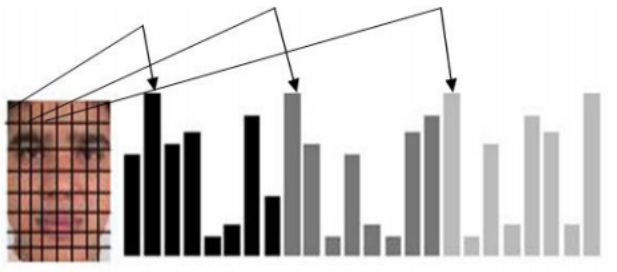
The figure on the left shows the important features Uniform LBPs along with 11111111 and 00000000 can extract.

The below image also clearly shows that how considering only Uniform LBPs can detect around 99% of the features along with keeping the background colour intact!



FEATURE VECTORS

Finally the feature vectors are encoded by dividing the image into k x k grids and finding the LBP of the subimage in each grid with respect to different R (radii). Considering only Uniform LBPs we have P(P-1) bins. Upon adding 11111111 and 00000000 we have P(P-1) + 2 bins in the histogram. Finally putting all the Non-Uniform LBPs in a single bin each grid has **P(P-1) + 3 bins**.

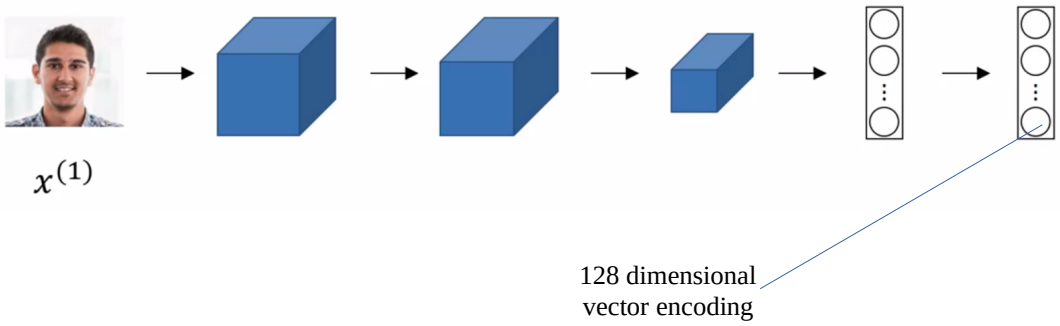


2. Convolutional Approach

Compared to the LBPH feature extraction the convolutional approach is straight forward.

It includes feeding the image in a convolution but not classifying it using a softmax function at the end. Instead the deep conv net is ended at a flattened 128 dimensional vector that can be seen as the encoding for the image.

Thsis is called a Siamese Network.



Difference Function (continued...)

Now that we have got the encodings we have to train our model in such a way that it learns to find the difference between these encodings. A higher difference should result in a No- Match and a lower difference should result in a Match. Hence for this we reduce a Triplet Loss Function.

Triplet Loss Function

First we choose three images Anchor , positive and negative.

The anchor image is an image of a face. The positive image is the image of the same person and the negative image is an image of a different person's face.

Assuming we can represent the histogram we received as an N dimensional flattened vector, let the encoding of an image be f(x).

Now we want to train the neural network such that $||f(x_i) - f(x_j)||^2$ is small for same faces and high for diferent faces.

If anchor denoted by A , Positive denoted by P and Negative denoted by N , we finally want
$d(A,P) - d(A,N) <= 0$

NOTE : d() deonotes the diference function

Now since we don't want a trivial solution of d(A,P) = 0 and d(A,N) = 0 is to be learnt by the neural network we introduce a hyperparameter alpha.

Where $d(A,P) - d(A,N) + alpha <= 0$

Adding alpha not only prevents the model from learning these trivial solutions but also pushes the d(A,P) down and d(A,N) up increasing the difference between the two and making the classification more prominent.

Therefore using this we define our loss function to be :-
$L = max((d(A,P) - d(A,N) + alpha) , 0)$

This way if the condition of the difference + alpha not being lesser than 0 is not fulfilled the function gives 0 as output. Hence the loss function tries to minimize itself towards a negaive number.

False Positives

Face recognition is used as a security strengthning solution in a lot of places and if the model recognizes an unauthorized face, it can lead to trouble. Hence it is important to handle false positives.

This can be indirectly done while training the triplet loss function. Instead of choosing random anchor, positive and negative images we must choose images that are hard to train on. This will make the model much more robust towards similar looking faces which are actuallyl different.

And by doing this we will ensure that the model actually learns something because if the positives and negative are very different then the loss function will not learn anything as the loss function will always input negative values.

Implementation

For implementing the LBPH recognizer we can use open cv's **cv2.face** library.

lbphrecognizer = **cv2.face.createLBPHFaceRecognizer()**

For implementing the deepface's convolutional approach we can easily use keras from tensorflow 2 and ceate our own model for the reducing the triplet loss function. This would be comparatively trivial as we would already have detected the faces. Getting the encodings is equally trivial as it involves making a deep convolutional network that can be done using keras as well.
One can always refer to the original research paper for more intricate implementation details.

CONCLUSION

Face Detection -> HAAR Cascade to be used
Face Recognition -> Deepface's convolution approach

According to the results from the first implementation we can change few things too

1. If there are more false positives during detection, we can play with the subwindow scale sizes.

2. If there are more false positives during recognition, we can maybe change our model to punish false positives by increasing the value of alpha.

3. If we have false negatives we can check our dataset to see what images lead to a false negative. For example in some images the difference in lighting in the pictures can lead to no detection or recognition of faces. Then the images can be augmented such that programatically lighting is brightened or reduced before retraining the model on the new dataset.

4. Underfitting can be solved by getting more data or by augmenting the existing data by for example playing with the lighting , flipping the images along a central vertical axis or by adding noise to the data. Increading number of iterations may also help.

5. Overfitting can be solved by getting test data and train data from the same distribution / using dropout , L1 or L2 regularization / early stopping.