FACE DETECTION

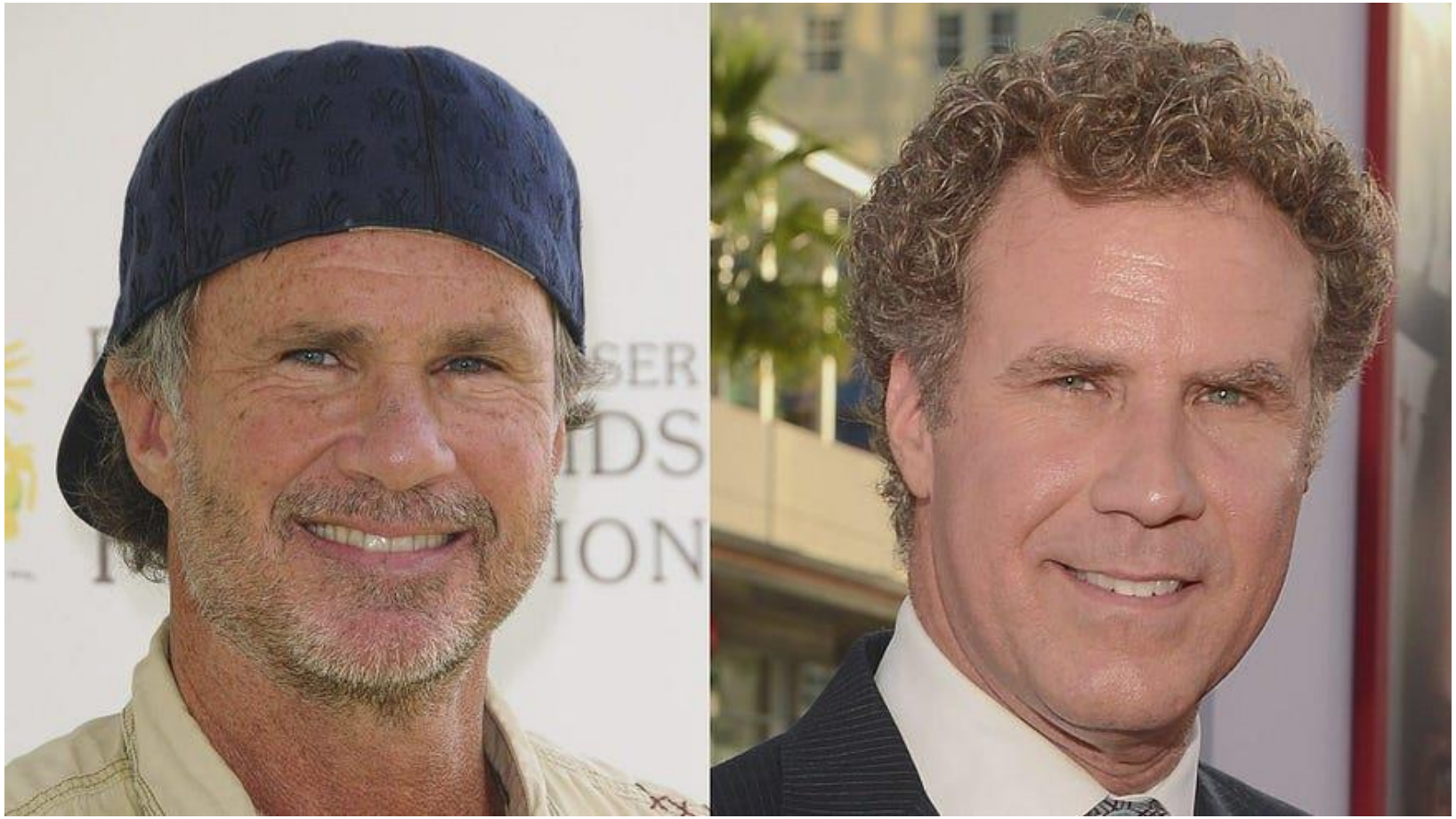
Sathyadharini S

2021115097

Modern Face Recognition with Deep Learning

- Have you noticed that Facebook has developed an uncanny ability to recognize your friends in your photographs? In the old days, Facebook used to make you to tag your friends in photos by clicking on them and typing in their name. Now as soon as you upload a photo, Facebook tags everyone for you *like magic*!
- This technology is called face recognition. Facebook's algorithms are able to recognize your friends' faces after they have been tagged only a few times. It's pretty amazing technology — Facebook can recognize faces with 98% accuracy which is pretty much as good as humans can do!

See this men



- Let’s learn how modern face recognition works! But just recognizing your friends would be too easy. We can push this tech to the limit to solve a more challenging problem — telling Will Ferrell (famous actor) apart from Chad Smith (famous rock musician)!

- we’ve used machine learning to solve isolated problems that have only one step — estimating the price of a house, generating new data based on existing data and telling if an image contains a certain object. All of those problems can be solved by choosing one machine learning algorithm, feeding in data, and getting the result.

Face capture

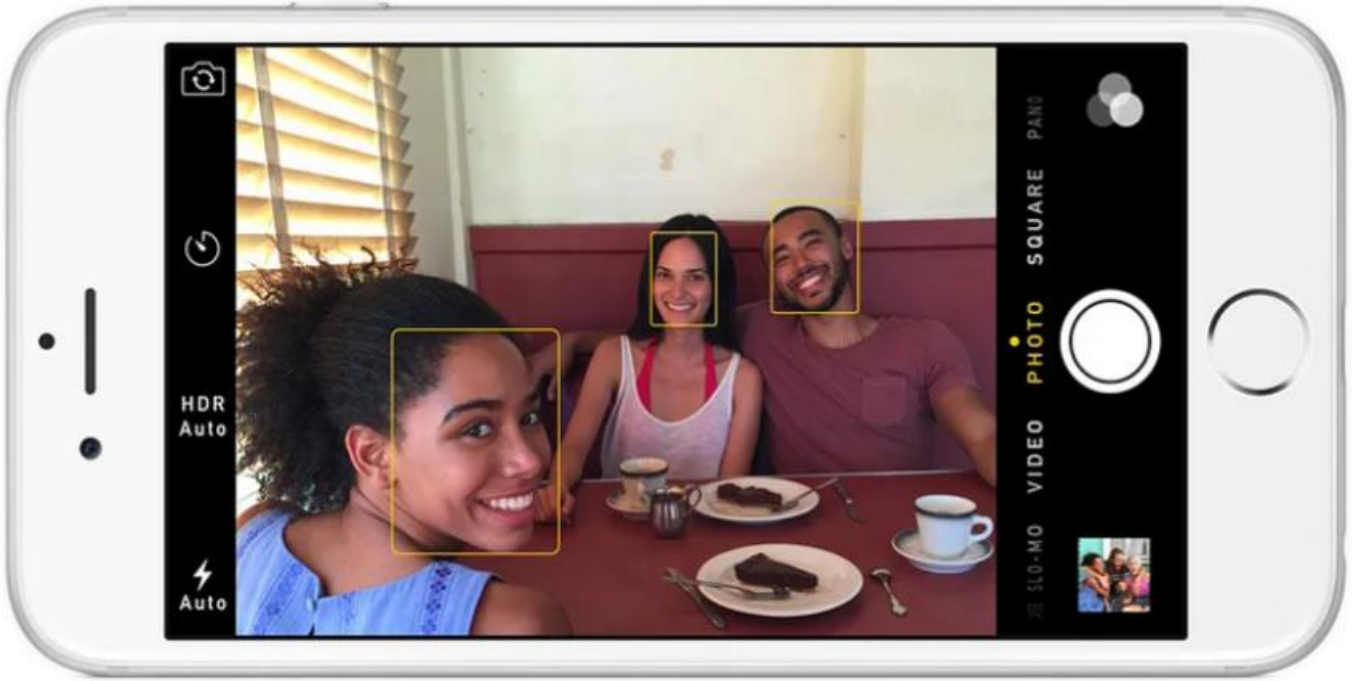
- But face recognition is really a series of several related problems:

1. First, look at a picture and find all the faces in it
2. Second, focus on each face and be able to understand that even if a face is turned in a weird direction or in bad lighting, it is still the same person.
3. Third, be able to pick out unique features of the face that you can use to tell it apart from other people— like how big the eyes are, how long the face is, etc.
4. Finally, compare the unique features of that face to all the people you already know to determine the person's name.

Actual Methodology!

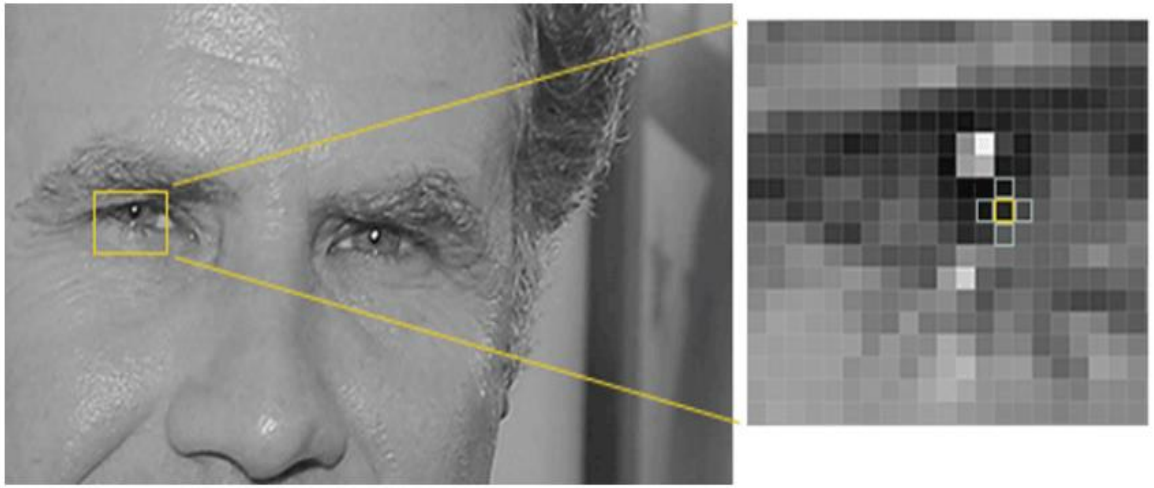
Face Detection

- you'll learn the main ideas behind each one and you'll learn how you can build your own facial recognition system in Python using OpenFace and dlib.
- The first step in our pipeline is face detection. Obviously we need to locate the faces in a photograph before we can try to tell them apart!
- If you've used any camera in the last 10 years, you've probably seen face detection in action:



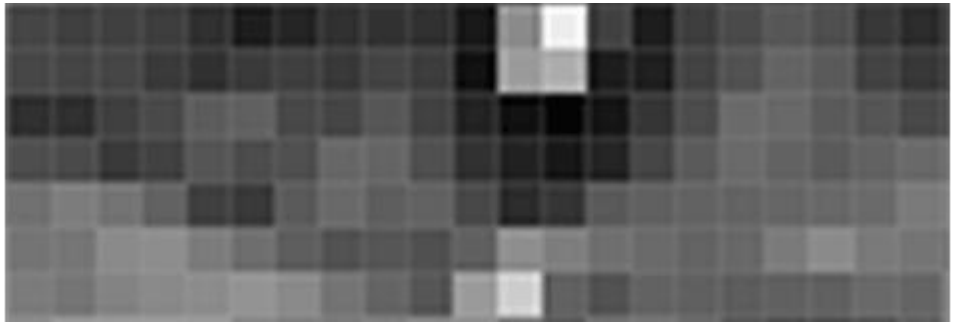
- Face detection went mainstream in the early 2000's when Paul Viola and Michael Jones invented a way to detect faces that was fast enough to run on cheap cameras. However, much more reliable solutions exist now. We're going to use a method invented in 2005 called Histogram of Oriented Gradients — or just ***HOG*** for short.
- To find faces in an image, we'll start by making our image black and white because we don't need color data to find faces:

FD!



FD!

- Our goal is to figure out how dark the current pixel is compared to the pixels directly surrounding it. Then we want to draw an arrow showing in which direction the image is getting darker:



Looking at just this one pixel and the pixels touching it, the image is getting darker towards the upper right.

Continue applying

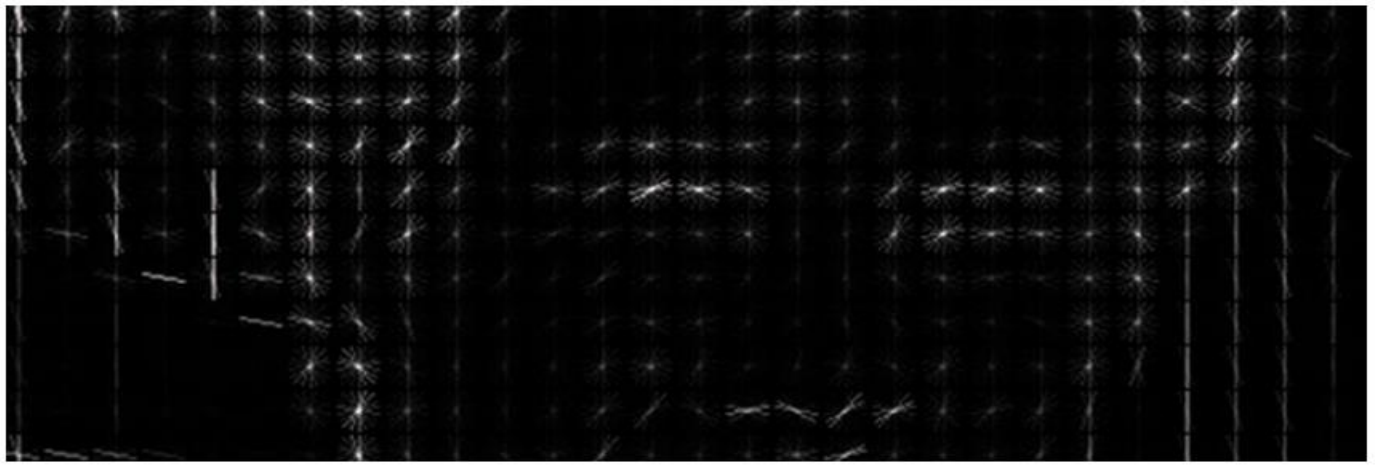
Why? gradients

- This might seem like a random thing to do, but there's a really good reason for replacing the pixels with gradients. If we analyze pixels directly, really dark images and really light images of the same person will have totally different pixel values. But by only considering the *direction* that brightness changes, both really dark images and really bright images will end up with the same exact representation. That makes the problem a lot easier to solve!

Next step!

- To do this, we'll break up the image into small squares of 16x16 pixels each. In each square, we'll count up how many gradients point in each major direction (how many point up, point up-right, point right, etc…). Then we'll replace that square in the image with the arrow directions that were the strongest.

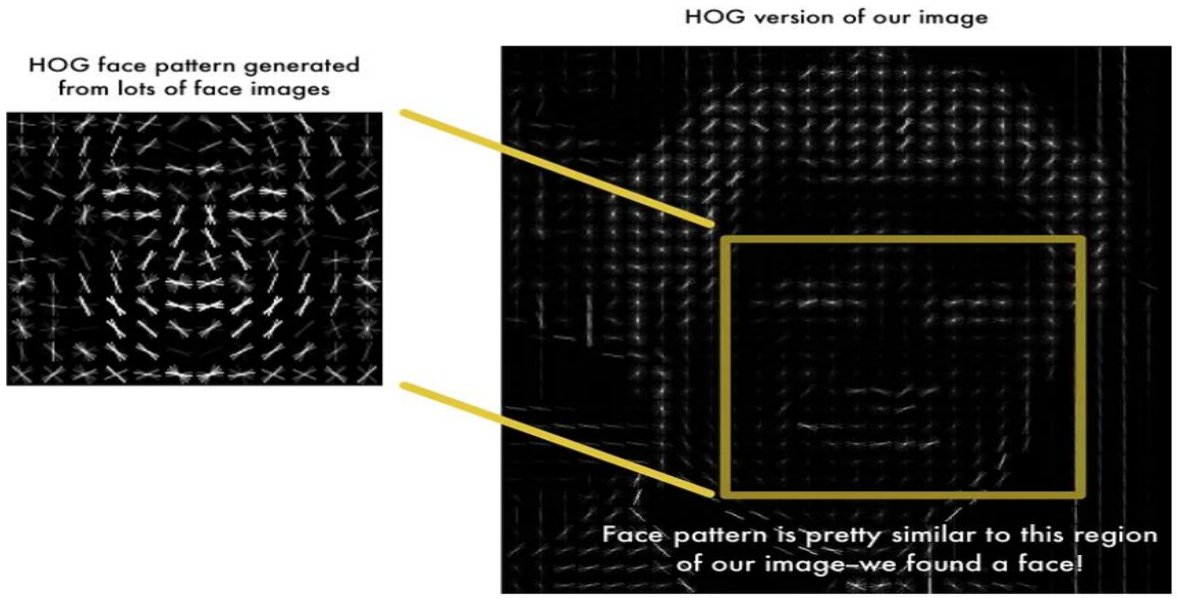
The end result is we turn the original image into a very simple representation that captures the basic structure of a face in a simple way:



The original image is turned into a HOG representation that captures the major features of the image regardless of image brightnesss.

Using this technique, we can now easily find faces in any image:

To find faces in this HOG image, all we have to do is find the part of our image that looks the most similar to a known HOG pattern that was extracted from a bunch of other training faces:



After detecting a face structure now to identify?!

Whew, we isolated the faces in our image. But now we have to deal with the problem that faces turned different directions look totally different to a computer:

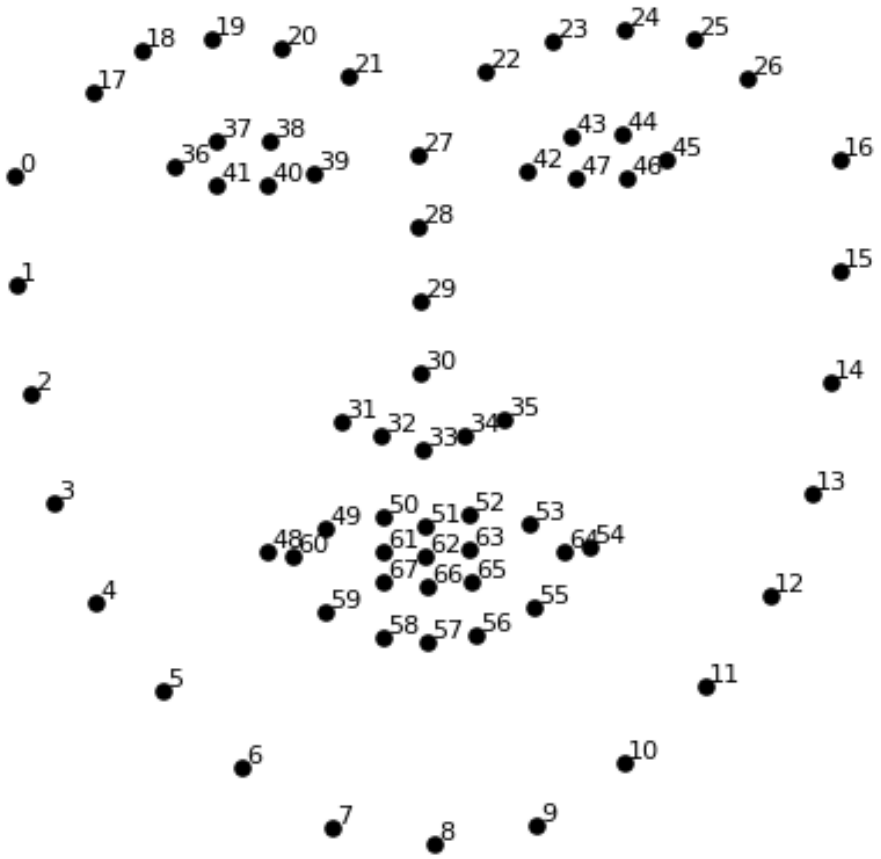
Humans can easily recognize that both images are of Will Ferrell, but computers would see these pictures as two completely different people.

To account for this, we will try to warp each picture so that the eyes and lips are always in the sample place in the image. This will make it a lot easier for us to compare faces in the next steps.

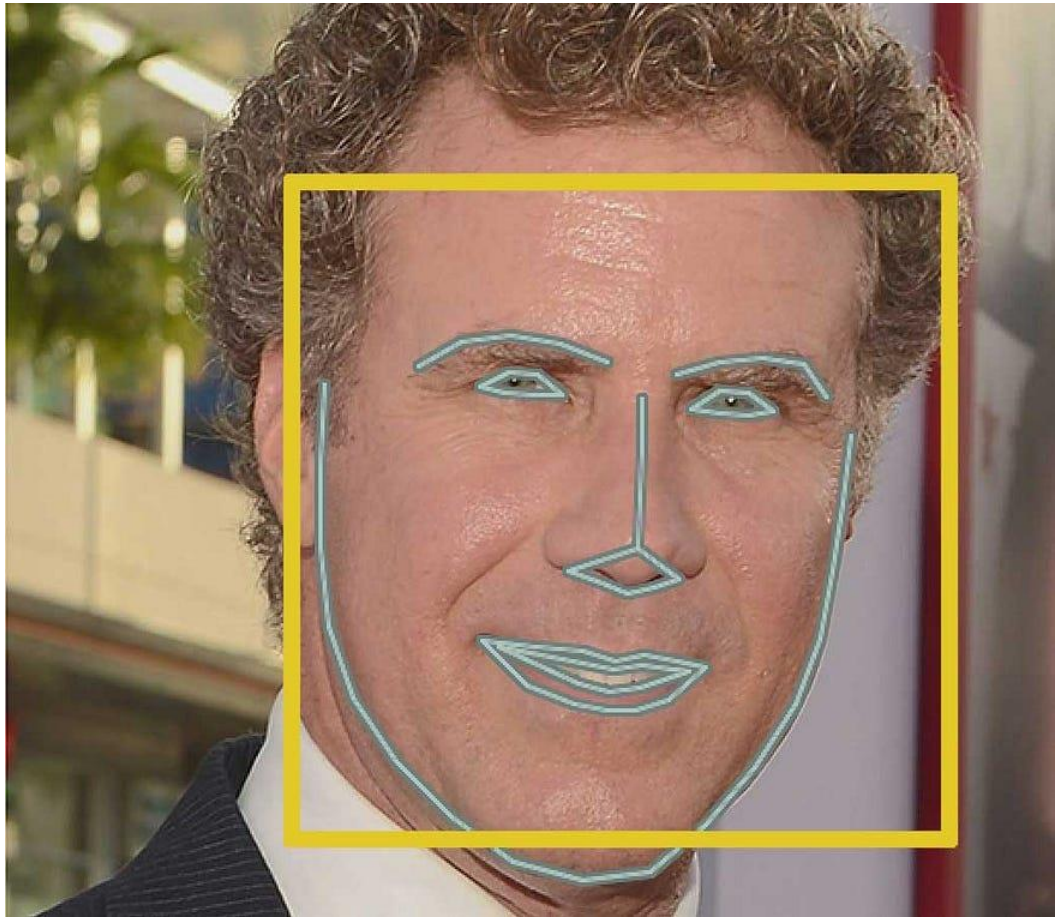
Let’s detect who the person is?

- To do this, we are going to use an algorithm called **face landmark estimation**. There are lots of ways to do this, but we are going to use the approach invented in 2014 by Vahid Kazemi and Josephine Sullivan.
- The basic idea is we will come up with 68 specific points (called *landmarks*) that exist on every face — the top of the chin, the outside edge of each eye, the inner edge of each eyebrow, etc. Then we will train a machine learning algorithm to be able to find these 68 specific points on any face:

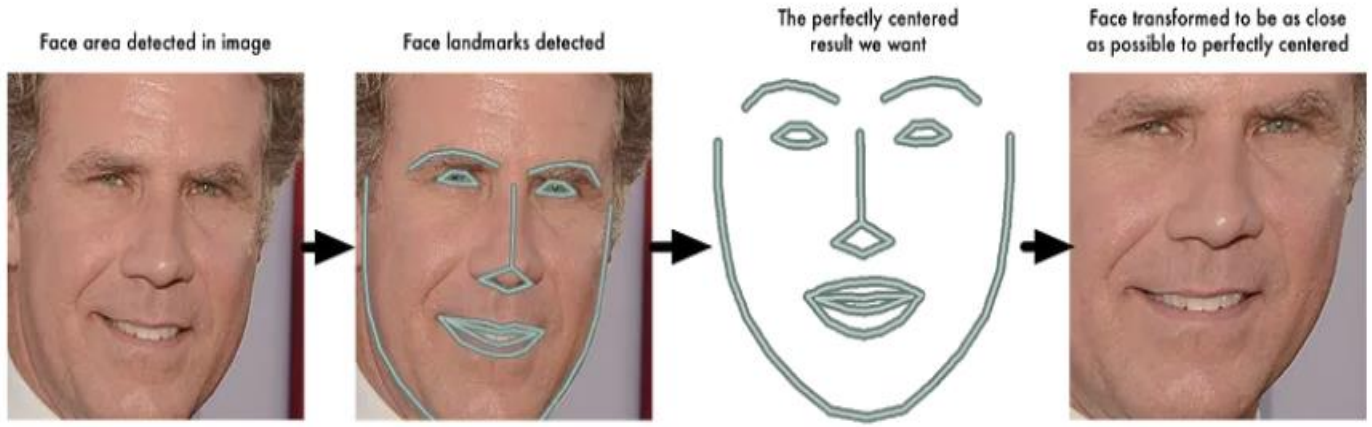
After Applying the face landmark estimation!



After applying it on the actual image (68 land mark features) plotted.



Basic affine transformations applied!

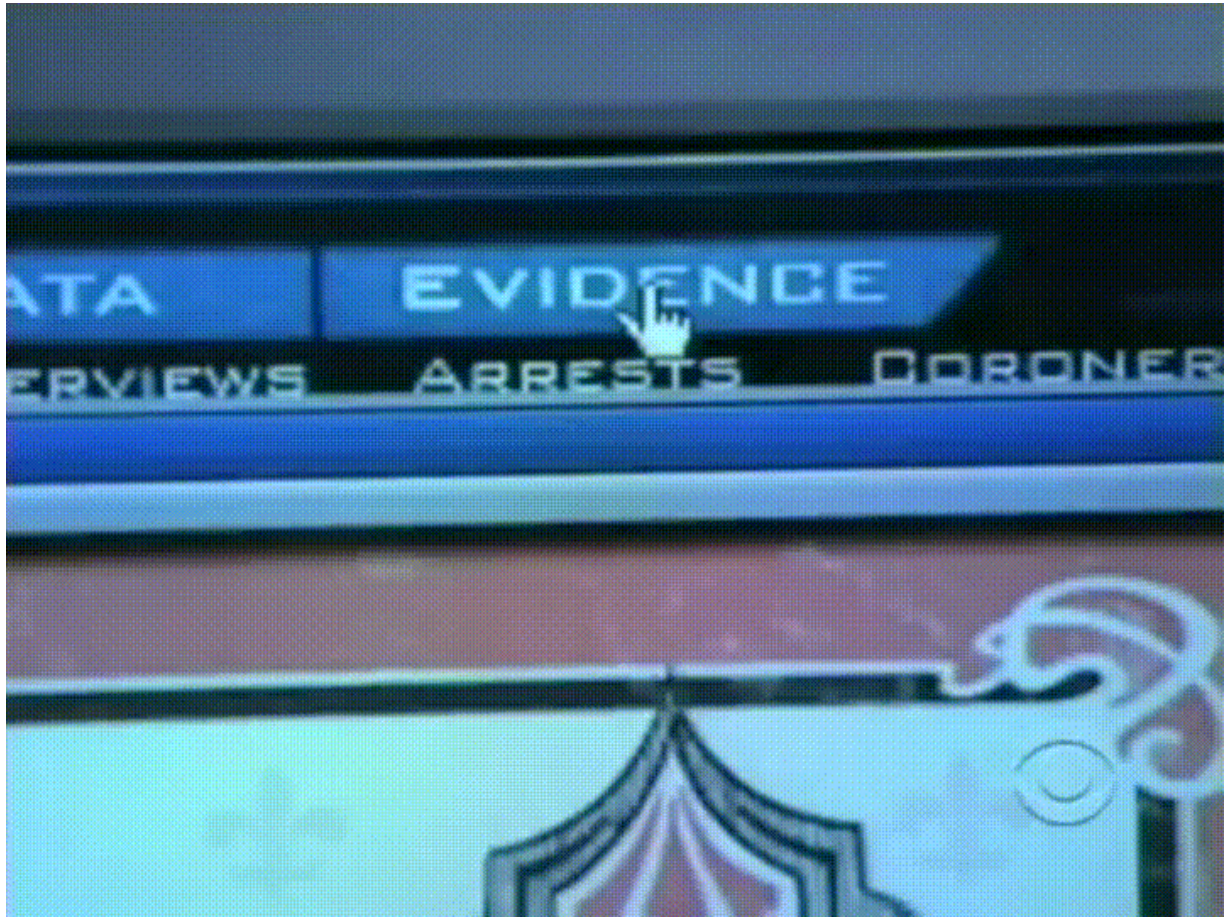


- **Step 3: Encoding Faces**
- Now we are to the meat of the problem — actually telling faces apart. This is where things get really interesting!
- The simplest approach to face recognition is to directly compare the unknown face we found in Step 2 with all the pictures we have of people that have already been tagged. When we find a previously tagged face that looks very similar to our unknown face, it must be the same person. Seems like a pretty good idea, right?
- There’s actually a huge problem with that approach. A site like Facebook with billions of users and a trillion photos can’t possibly loop through every previous-tagged face to compare it to every newly uploaded picture. That would take way too long. They need to be able to recognize faces in milliseconds, not hours.

So what’s the approach!

- What we need is a way to extract a few basic measurements from each face. Then we could measure our unknown face the same way and find the known face with the closest measurements. For example, we might measure the size of each ear, the spacing between the eyes, the length of the nose, etc.

Measuring leangths



Calculating lengths

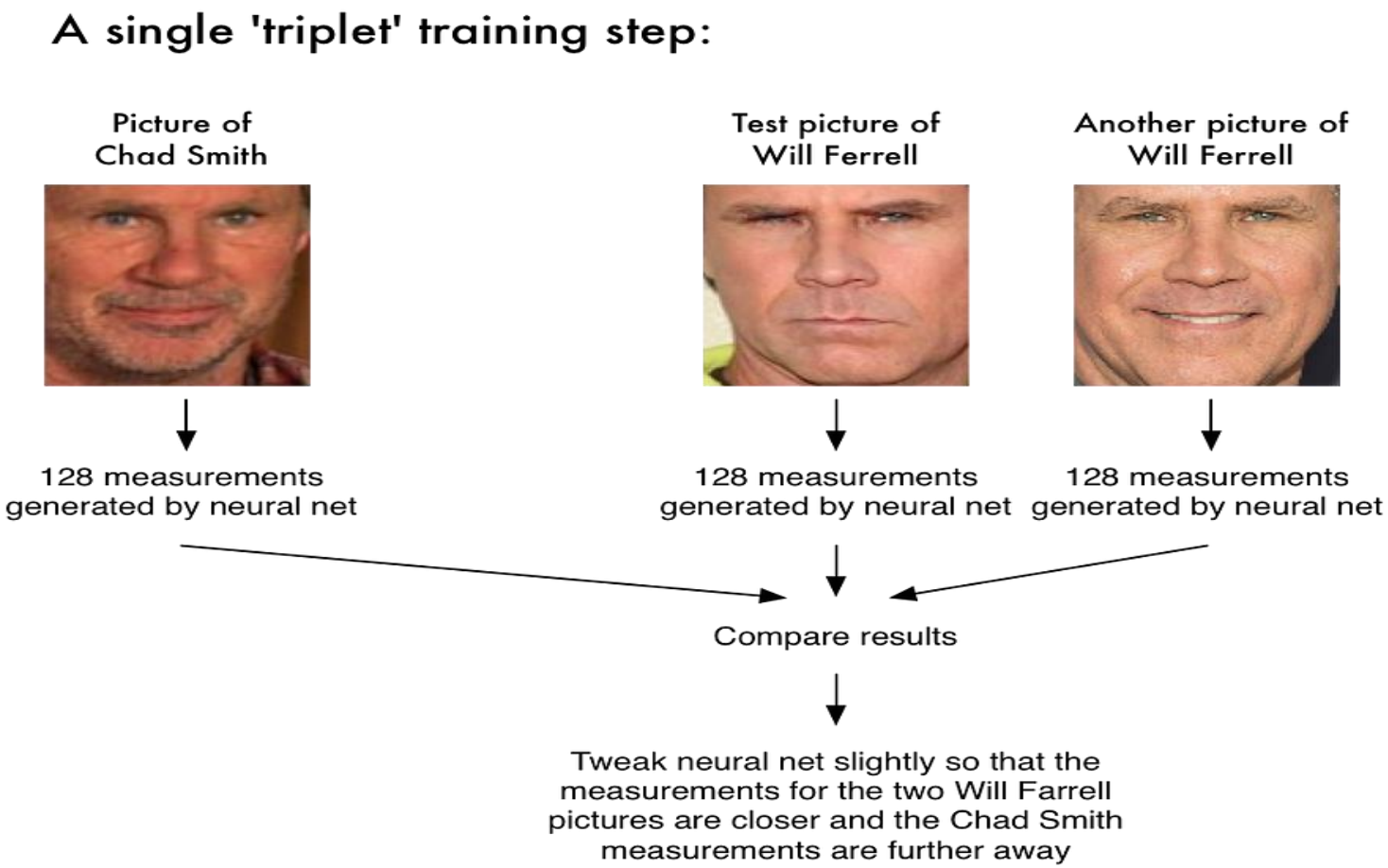
- It turns out that the measurements that seem obvious to us humans (like eye color) don’t really make sense to a computer looking at individual pixels in an image.

- Researchers have discovered that the most accurate approach is to let the computer figure out the measurements to collect itself. Deep learning does a better job than humans at figuring out which parts of a face are important to measure.

CNN training

- The solution is to train a Deep Convolutional Neural Network (just like we did in Part 3). But instead of training the network to recognize pictures objects like we did last time, we are going to train it to generate 128 measurements for each face.

Triplet training at a time

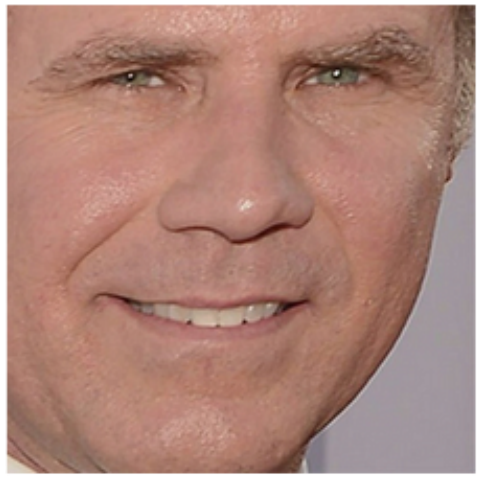


Results

- After repeating this step millions of times for millions of images of thousands of different people, the neural network learns to reliably generate 128 measurements for each person.
- Any ten different pictures of the same person should give roughly the same measurements.
- Machine learning people call the 128 measurements of each face an **embedding**. The idea of reducing complicated raw data like a picture into a list of computer-generated numbers comes up a lot in machine learning (especially in language translation). The exact approach for faces we are using was invented in 2015 by researchers at Google but many similar approaches exist.

Embedding of an image

Input Image



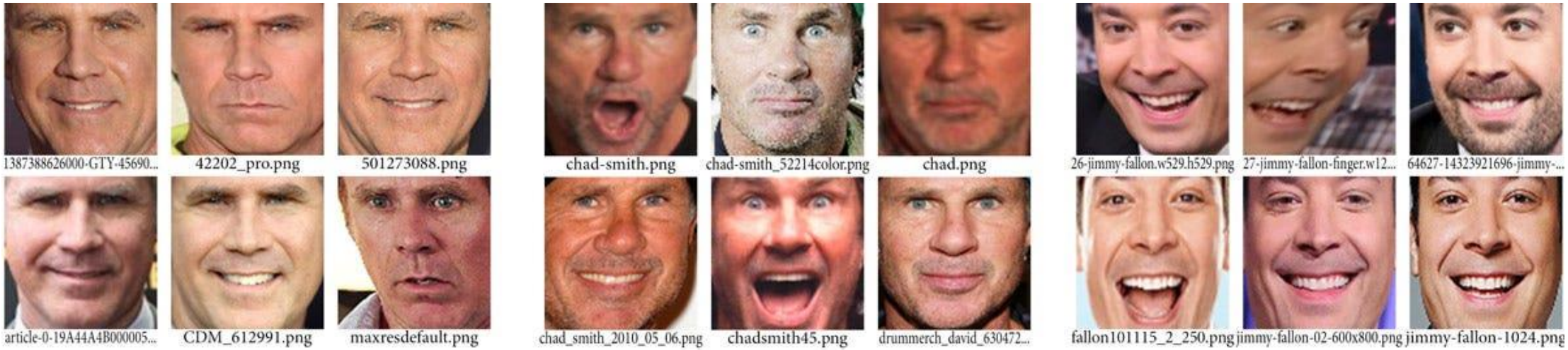
128 Measurements Generated from Image

0.097496084868908	0.045223236083984	-0.1281466782093	0.032084941864014
0.12529824674129	0.060309179127216	0.17521631717682	0.020976085215807
0.030809439718723	-0.01981477253139	0.10801389068365	-0.00052163278451189
0.036050599068403	0.065554238855839	0.0731306001544	-0.1318951100111
-0.097486883401871	0.1226262897253	-0.029626874253154	-0.0059557510539889
-0.0066401711665094	0.036750309169292	-0.15958009660244	0.043374512344599
-0.14131525158882	0.14114324748516	-0.031351584941149	-0.053343612700701
-0.048540540039539	-0.061901587992907	-0.15042643249035	0.078198105096817
-0.12567175924778	-0.10568545013666	-0.12728653848171	-0.076289616525173
-0.061418771743774	-0.074287034571171	-0.065365232527256	0.12369467318058
0.046741496771574	0.0061761881224811	0.14746543765068	0.056418422609568
-0.12113650143147	-0.21055991947651	0.0041091227903962	0.089727647602558
0.061606746166945	0.11345765739679	0.021352224051952	-0.0085843298584223
0.061989940702915	0.19372203946114	-0.086726233363152	-0.022388197481632
0.10904195904732	0.084853030741215	0.09463594853878	0.020696049556136
-0.019414527341723	0.0064811296761036	0.21180312335491	-0.050584398210049
0.15245945751667	-0.16582328081131	-0.035577941685915	-0.072376452386379
-0.12216668576002	-0.0072777755558491	-0.036901291459799	-0.034365277737379
0.083934605121613	-0.059730969369411	-0.070026844739914	-0.045013956725597
0.087945111095905	0.11478432267904	-0.089621491730213	-0.013955107890069
-0.021407851949334	0.14841195940971	0.078333757817745	-0.17898085713387
-0.018298890441656	0.049525424838066	0.13227833807468	-0.072600327432156
-0.011014151386917	-0.051016297191381	-0.14132921397686	0.0050511928275228
0.0093679334968328	-0.062812767922878	-0.13407498598099	-0.014829395338893
0.058139257133007	0.0048638740554452	-0.039491076022387	-0.043765489012003
-0.024210374802351	-0.11443792283535	0.071997955441475	-0.012062266469002
-0.057223934680223	0.014683869667351	0.05228154733777	0.012774495407939
0.023535015061498	-0.081752359867096	-0.031709920614958	0.069833360612392
-0.0098039731383324	0.037022035568953	0.11009479314089	0.11638788878918
0.020220354199409	0.12788131833076	0.18632389605045	-0.015336792916059
0.0040337680839002	-0.094398014247417	-0.11768248677254	0.10281457751989
0.051597066223621	-0.10034311562777	-0.040977258235216	-0.082041338086128

Tagging is easier coz ML

- **Step 4: Finding the person's name from the encoding**
- This last step is actually the easiest step in the whole process. All we have to do is find the person in our database of known people who has the closest measurements to our test image.
- You can do that by using any basic machine learning classification algorithm. No fancy deep learning tricks are needed. We'll use a simple linear SVM classifier, but lots of classification algorithms could work.
- All we need to do is train a classifier that can take in the measurements from a new test image and tells which known person is the closest match. Running this classifier takes milliseconds. The result of the classifier is the name of the person!

Actual training images can be like



Computers now can more accurately than humans!

Instance-level Recognition

- **I**nstance **L**evel **R**ecognition (ILR), is a visual recognition task to recognize a specific instance of an object not just object class.
- For example, as shown in the above image, painting is an object class, and “Mona Lisa” by Leonardo Da Vinci is an instance of that painting. Similarly, the Taj Mahal, India is an instance of the object class building.
- **Use cases**
- Landmark Recognition: Recognize landmarks in images.
- Landmark Retrieval: Retrieve relevant landmark images from a large-scale database.
- Artwork Recognition: Recognize artworks in images.
- Product Retrieval: Retrieve relevant product images from a large-scale database.

Classify and label is the goal!

Training and dataset

- **Large scale:** Most of the current state of the art results of recognition tasks are measured on very limited categories e.g. ~1000 image classes in ImageNet, ~80 categories in COCO. But use-cases like landmark retrieval and recognition, has 200K+ classes e.g. in Google Landmark Dataset V2 (GLDv2), 100K+ classes of the product on Amazon.
- **Long-tailed:** Few popular places have more than 1000+ images but many less know places have less than 5 images in **GLDv2.**

MOST USE CASES!

- Instance-level recognition aims to distinguish individual instances of objects within a category, significantly enhancing the capabilities of deep learning technologies for semantic image classification and retrieval across various domains such as eCommerce, travel, media & entertainment, and agriculture.

Popular DL Models used!

- The backbone network is a crucial component in deep learning architectures as it extracts essential features from input images. Some popular backbone networks include:
- **Residual Networks (ResNet)**: ResNet employs residual learning to ease the training of deep networks by introducing shortcut connections that bypass one or more layers. This helps in mitigating the vanishing gradient problem and enables the training of much deeper networks.
- **Squeeze and Excitation Networks (SENet)**: SENet enhances the representational power of a network by explicitly modeling the interdependencies between channels. It does so by applying a "squeeze" operation to aggregate global spatial information and an "excitation" operation to recalibrate channel-wise feature responses.
- **EfficientNet**: EfficientNet uses a compound scaling method that uniformly scales all dimensions of depth, width, and resolution using a set of fixed scaling coefficients. This results in a more balanced network that achieves better performance with fewer parameters.

Image Labeling Tool

An image labeling or annotation tool is used to label the images for bounding box object detection and segmentation.

Open-source image labeling tool like :

- VGG Image
- Annotator
- LabelImg
- OpenLabeler
- ImgLab

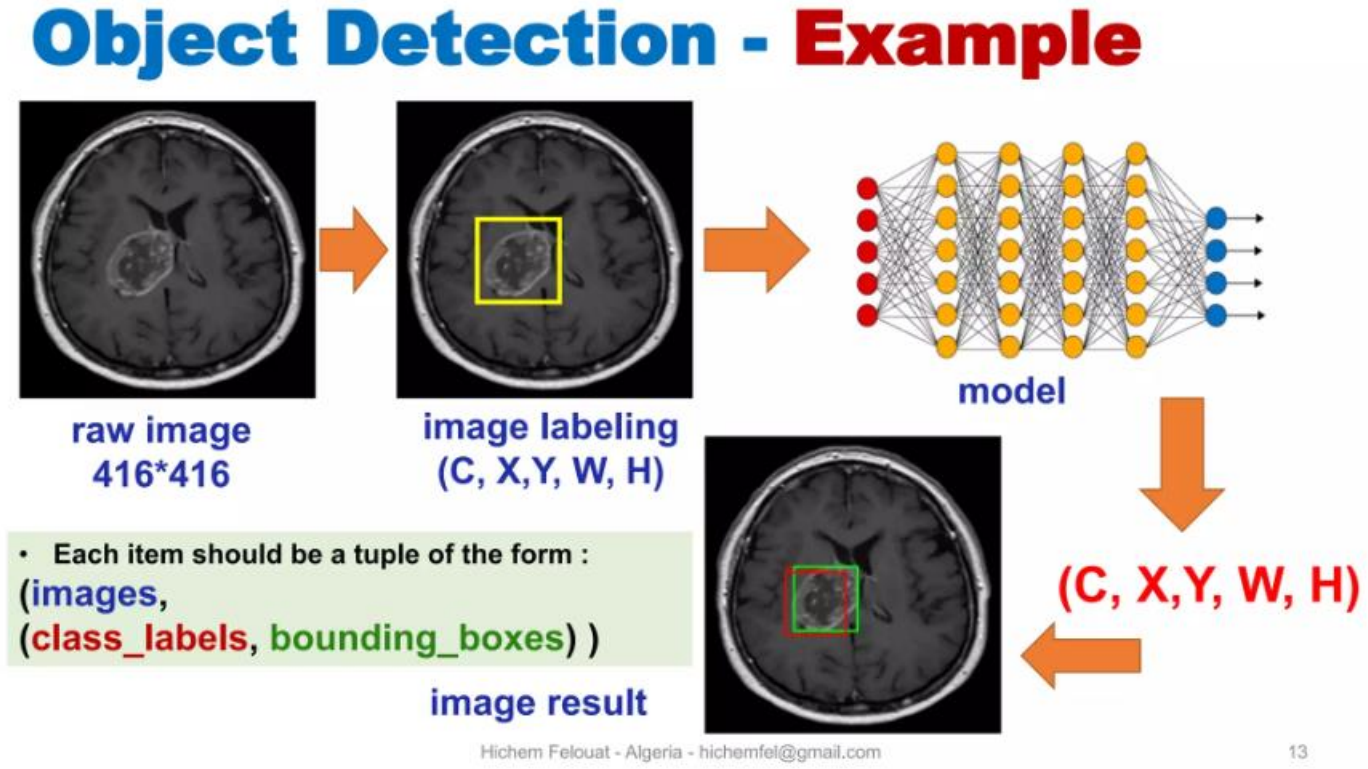
Commercial tool like :

- LabelBox
- Supervisely

Crowdsourcing Platform like :

- Amazon Mechanical Turk

Object identification



Yolo V5,V7

What is Yolo?

- **You Only Look Once (YOLO)** is an algorithm that uses convolutional neural networks for object detection.
- It is one of **the faster** object detection algorithms out there.
- It is a very good choice when we need **real-time detection**, without loss of too much accuracy.

Anything from tie, tumour to Palace can identified ,labelled and classified!