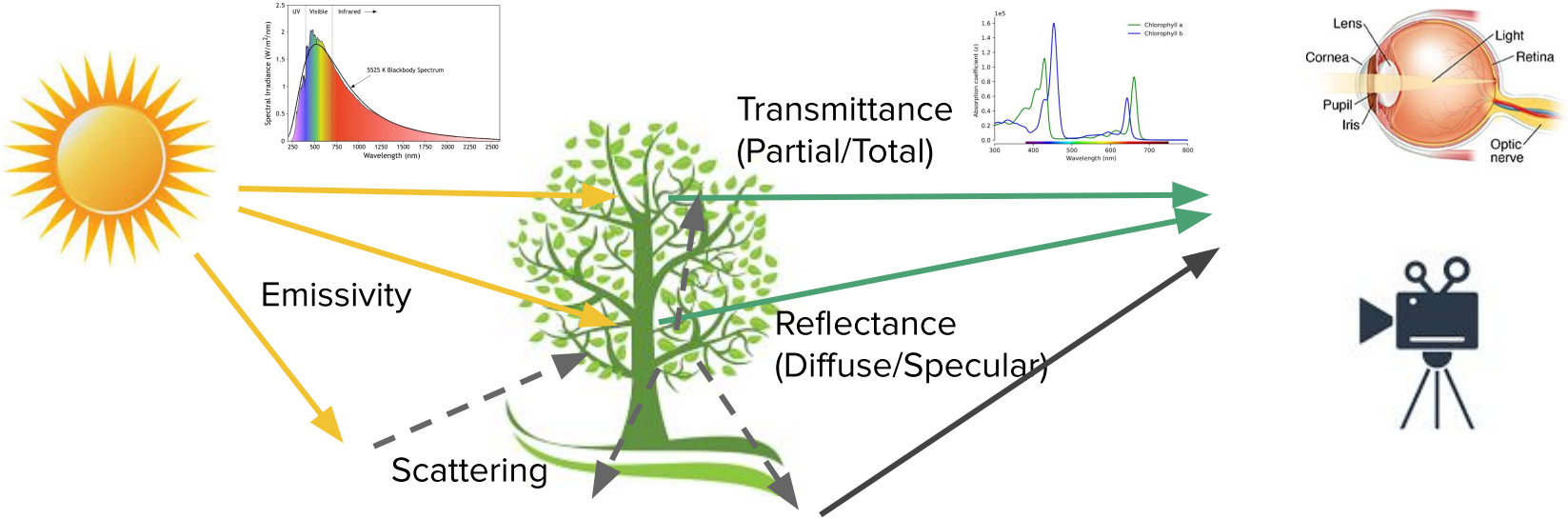
IFT6758 Lab

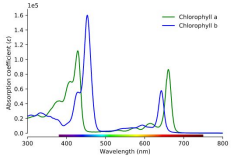
Practical Computer Vision (with OpenCV)

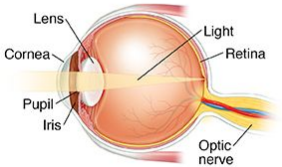
Low-Level Computer Vision

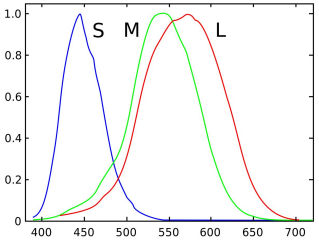
Vision is Complex



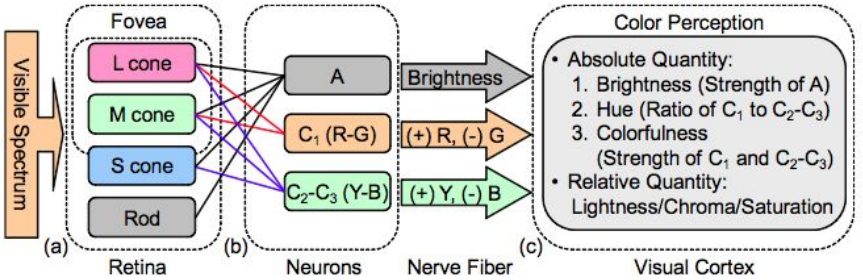
Vision is Complex

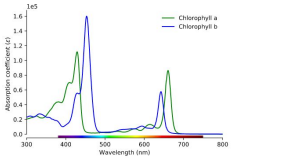


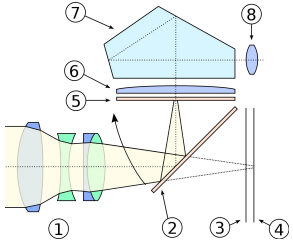


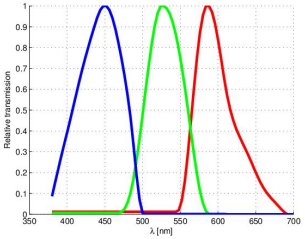


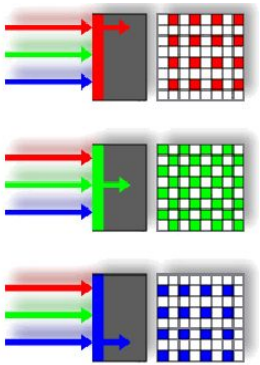
Opponent Process Theory of Vision



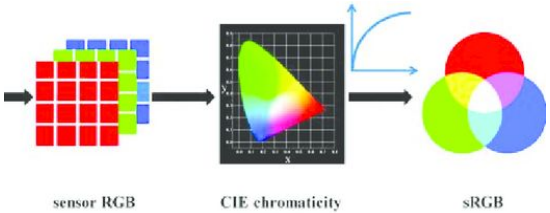






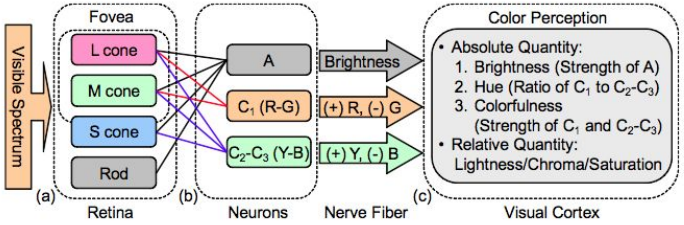


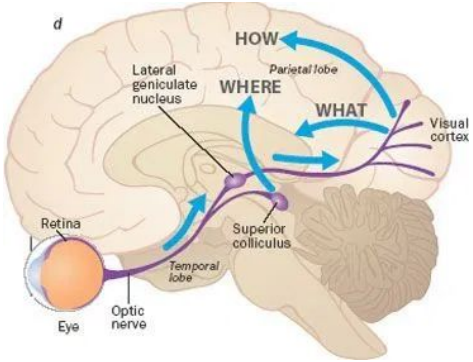
Bayer Filter



Color Correction

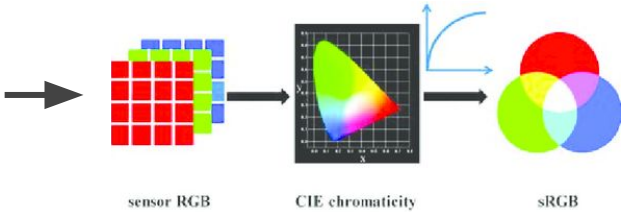
Vision is Complex

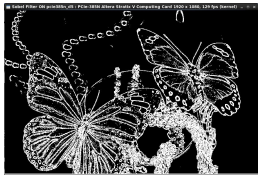


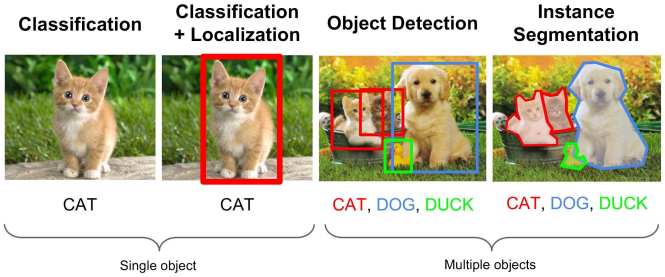


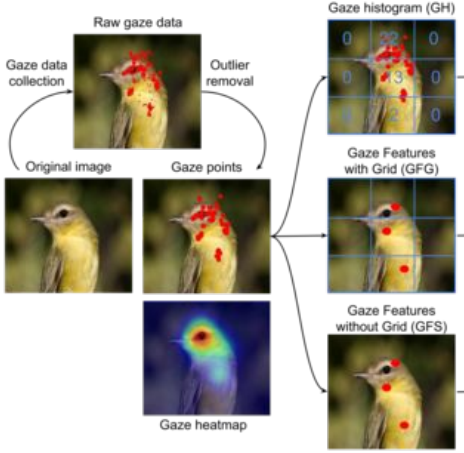
Human Vision

Computer Vision









Objectivity vs Subjectivity

Should the computer try to emulate human vision?

Or it is fine if the results are close enough?

How to handle illusions? Should computers be fooled the same way as humans?

Illusions, Good or Bad?

Maybe some illusions exist to increase our chances of survival in the wild. (Evolution argument)

Color Assimilation Illusion

Color Assimilation Illusion

Most of the image is in black and white!

Yet we see a colour image.

Color Assimilation Illusion

What we see

Color Assimilation Illusion

However, the computer is not fooled...

R

G

B

What the computer ‘sees’

Color Assimilation Illusion

From evolution, our visual system learned that details in colors conveys less information than variations in luminance, and we don’t need to pay attention to discontinuities in colors, as they might be distracting and detrimental to our survival.

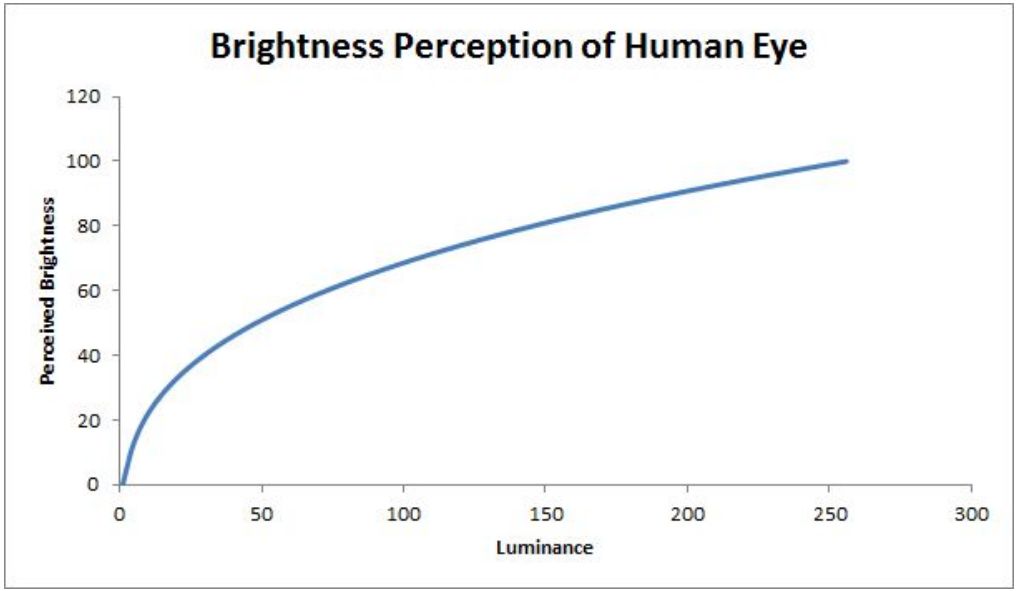
We will see in the next slides that this property of our visual system is used in photo/video compression techniques.

It is still up to debate whether computer vision systems should mimic the human visual system.

Gamma and Color Spaces

Perceived Luminance

The relationship between luminance (amount of light hitting the retina) and perceived brightness is not linear! We are less sensitive to lighter shades of grey.



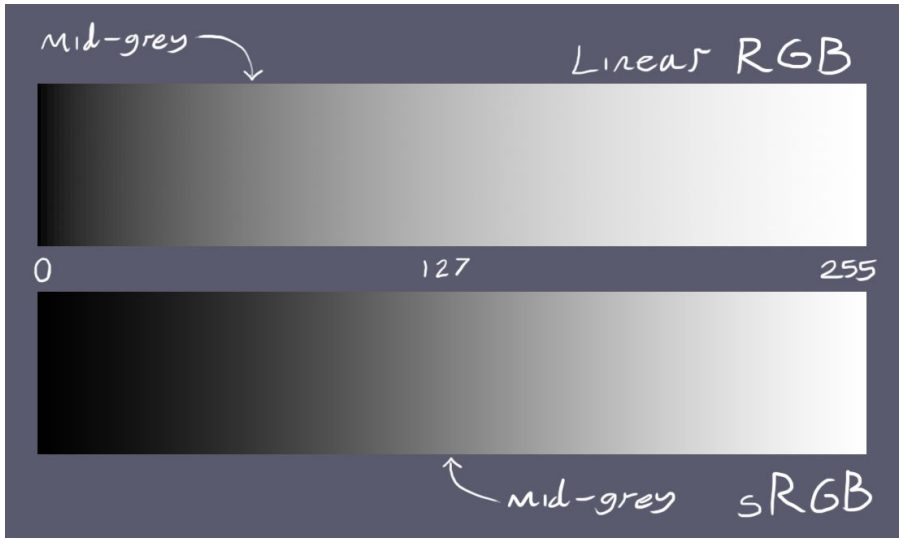
Gamma Compression

$$V_{\text{out}} = AV_{\text{in}}^{\gamma}$$

Since screens have a limited dynamic range (they have a limit to brightness) and we are more sensitive to darker shades, we can “compress” out the lighter shades, allowing the display to show more details in darker areas.

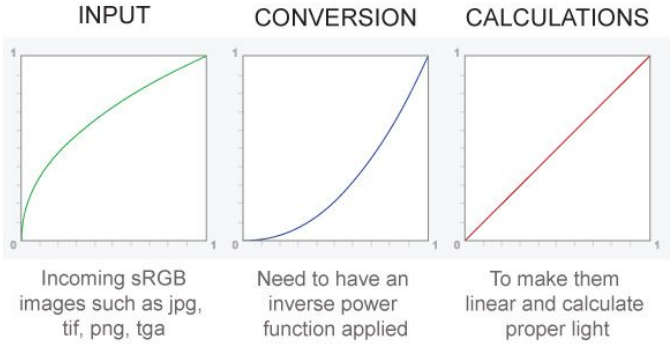
Gamma Compression

Furthermore, since images have a limited bandwidth (8 bits, 0-255), we can use gamma compression to encode more details in the darker shades (where our eyes are most sensitive).



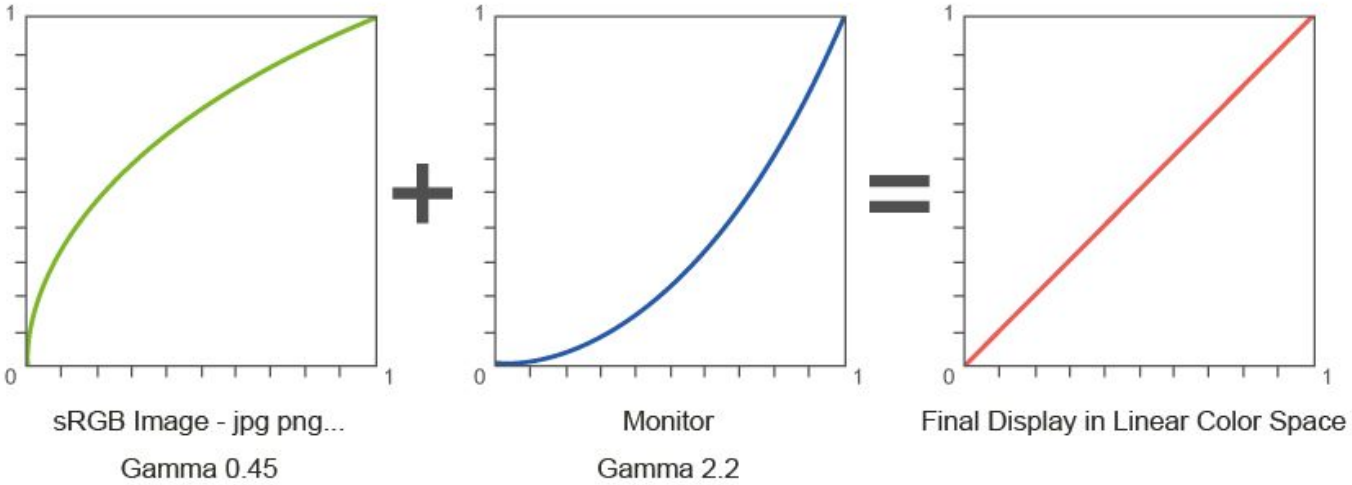
Gamma Correction

Thus, before applying mathematical operations on images, it is necessary to undo the gamma compression, or you will be doing operations on a non-linear color space!



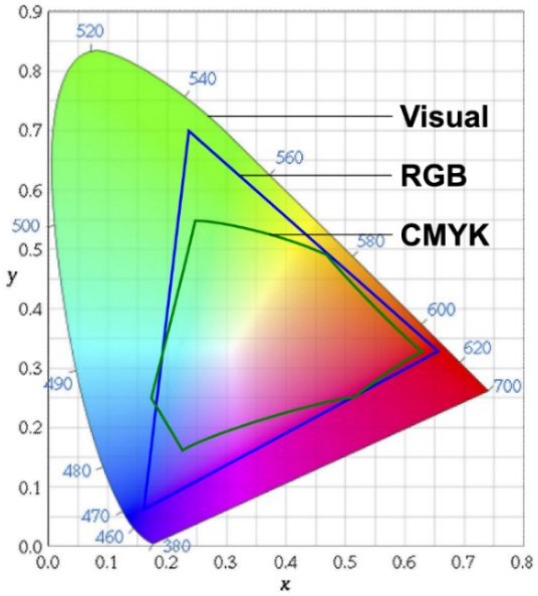
Gamma Correction

Computer monitors and TVs automatically do gamma correction on images and video. The default gamma for pictures is 0.45 and for monitors, 2.2.



Color Spaces

A color space describes the relationship between a number (or a tuple of numbers) and its corresponding color in real life.



Linear RGB

The most mathematically “pure” of all color spaces.

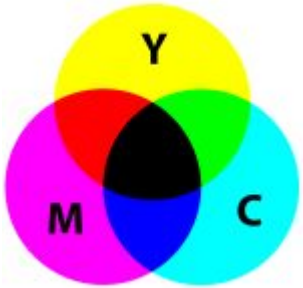
A 3-tuple (R, G, B) describes the relative intensity of Red, Green and Blue light. This is a linear space, thus an 10-times increase in the value equals a 10-fold increase in real luminance.

Useful for mathematical operations and physical simulations, but is very different from how humans perceive light and color.

It is also **not** how digital pictures are usually encoded,
as seen previously from the gamma compression slides.

CMYK

Used by printers. This is a subtractive color model, as adding more ink causes more light to be absorbed. Not very important for computer vision.



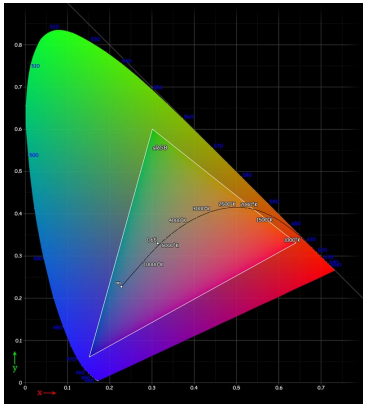
sRGB

Similar to Linear RGB, but has a gamma transform applied to it.
This is how most images are encoded!
You can consider sRGB to be Linear RGB, but with a gamma compression of 1/2.2 applied to it as an approximation.
You can convert sRGB to Linear RGB by using a gamma correction of 2.2.

In reality, the correct specification (IEC 61966-2-1:1999) is more complicated.

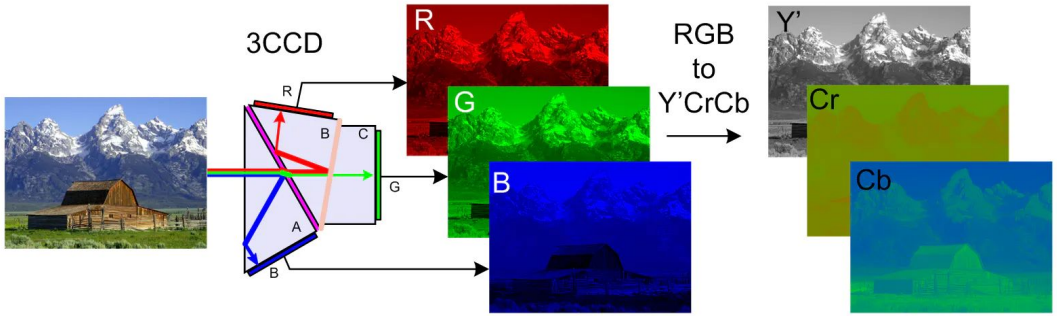
$$\begin{bmatrix} R_{\text{linear}} \\ G_{\text{linear}} \\ B_{\text{linear}} \end{bmatrix} = \begin{bmatrix} 3.2406 & -1.5372 & -0.4986 \\ -0.9689 & 1.8758 & 0.0415 \\ 0.0557 & -0.2040 & 1.0570 \end{bmatrix} \begin{bmatrix} X_{D65} \\ Y_{D65} \\ Z_{D65} \end{bmatrix}$$

$$\gamma(u) = \begin{cases} 12.92u & u \leq 0.0031308 \\ 1.055u^{1/2.4} - 0.055 & \text{otherwise} \end{cases}$$



YUV, YCbCr

This color space separates the Y (Luminance) from the UV or CbCr (Color) components. It is mostly used for compression purposes.
Usually UV/CbCr components are compressed more aggressively than the Y component, since our vision is less sensitive to color compared to luminance.
This color space is used to encode JPEG and most videos. Note that they are also first encoded in sRGB space, then converted to YCbCr.



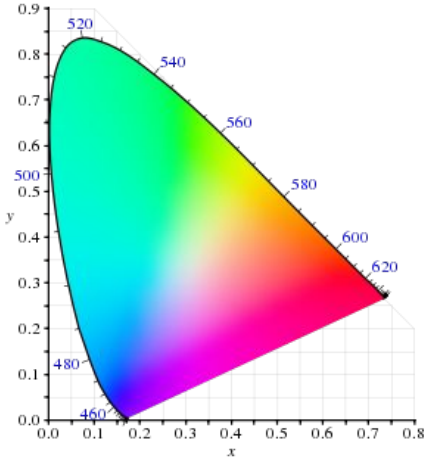
CIE 1931

The foundation of most perceptual color spaces. This color space tries to calibrate for how humans see colors. A 2x increase in value in this color spaces corresponds to a 2x increase in perceived brightness, not the actual luminance.

Its dimensions are Y (Luminance), and X, Z (Color Components).

$$\begin{bmatrix} X \\ Y \\ Z \end{bmatrix} = \frac{1}{b_{21}} \begin{bmatrix} b_{11} & b_{12} & b_{13} \\ b_{21} & b_{22} & b_{23} \\ b_{31} & b_{32} & b_{33} \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix} = \frac{1}{0.176\,97} \begin{bmatrix} 0.490\,00 & 0.310\,00 & 0.200\,00 \\ 0.176\,97 & 0.812\,40 & 0.010\,63 \\ 0.000\,00 & 0.010\,00 & 0.990\,00 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix}$$

$$\begin{bmatrix} R \\ G \\ B \end{bmatrix} = \begin{bmatrix} 0.418\,47 & -0.158\,66 & -0.082\,835 \\ -0.091\,169 & 0.252\,43 & 0.015\,708 \\ 0.000\,920\,90 & -0.002\,549\,8 & 0.178\,60 \end{bmatrix} \cdot \begin{bmatrix} X \\ Y \\ Z \end{bmatrix}$$



CIELAB

Based on CIE 1931, this is a more modern perceptual color space model.

Its dimensions are L (Lightness) and a*, b* (Color Components).

$$L^\star = 116\, f\left(\frac{Y}{Y_\mathrm{n}}\right) - 16$$

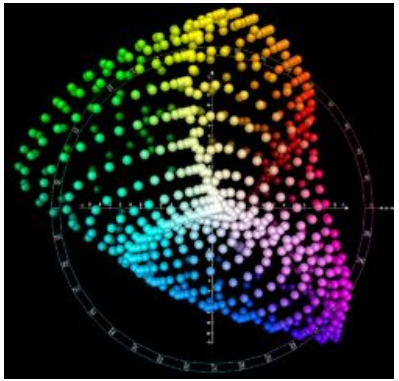
$$a^\star = 500\left(f\left(\frac{X}{X_\mathrm{n}}\right) - f\left(\frac{Y}{Y_\mathrm{n}}\right)\right)$$

$$b^\star = 200\left(f\left(\frac{Y}{Y_\mathrm{n}}\right) - f\left(\frac{Z}{Z_\mathrm{n}}\right)\right)$$

$$f(t) = \begin{cases} \sqrt[3]{t} & \text{if } t > \delta^3 \\ \frac{t}{3\delta^2} + \frac{4}{29} & \text{otherwise} \end{cases}$$

$$\delta = \frac{6}{29}$$

$$X_\mathrm{n} = 95.0489,$$
$$Y_\mathrm{n} = 100,$$
$$Z_\mathrm{n} = 108.8840$$



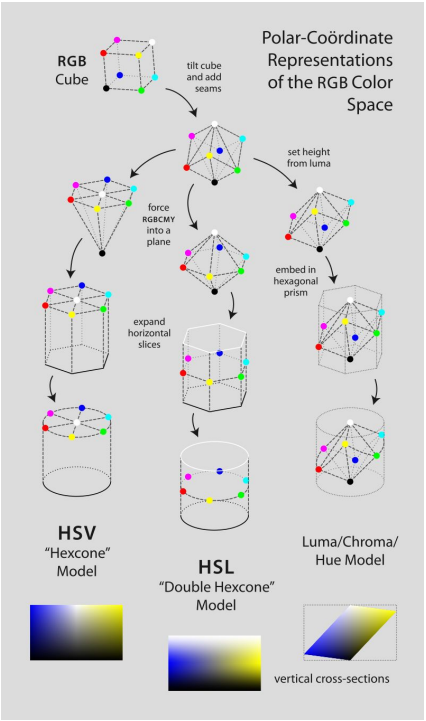
Cylindrical Transforms (HSL, HSV)

HSL and HSV are cylindrical transforms of trichromatic color spaces. (Usually sRGB, but can be any other, such as CIELAB.)

They map the color space cube into a cylinder or hexcone.

H: Hue (What color)
S: Saturation (How intense the color is)
L: Lightness (How white the color is)
V: Value (How bright the color is)

This is useful to describe the color more intuitively. Most color pickers use this color space.



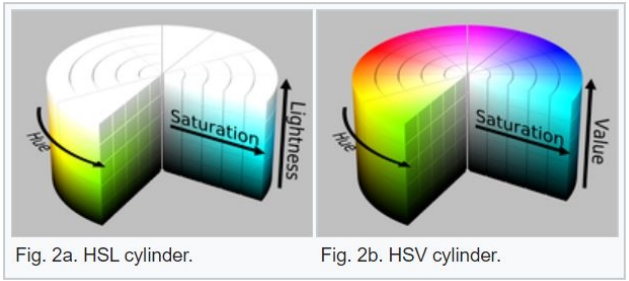


Fig. 2a. HSL cylinder. Fig. 2b. HSV cylinder.

Basic Operations on Images

Basic Operations on Images

As seen in the theory class, we can apply many operations on images to extract basic features or do preprocessing.

OpenCV implements most of them.

Geometric Transformations

https://docs.opencv.org/master/da/d6e/tutorial_py_geometric_transformations.html

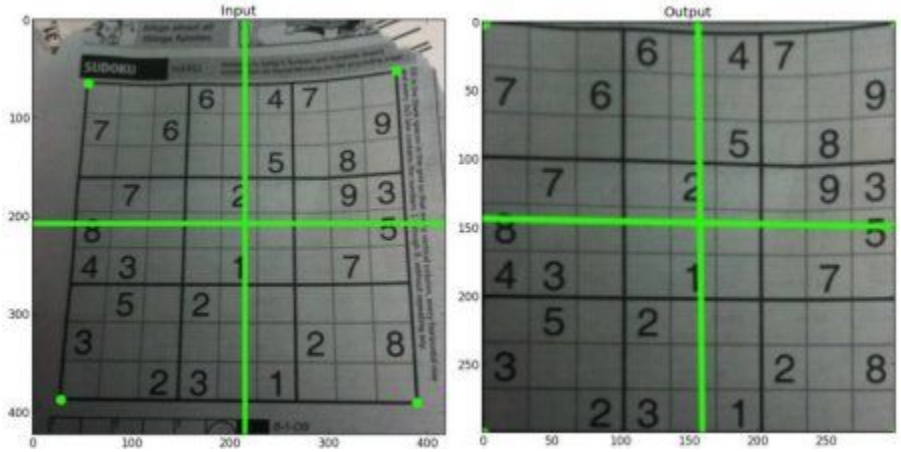


Image Thresholding

https://docs.opencv.org/master/d7/d4d/tutorial_py_thresholding.html

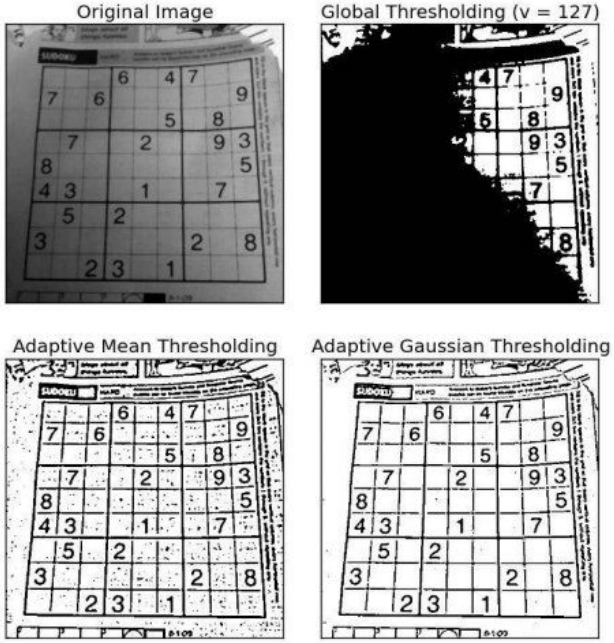
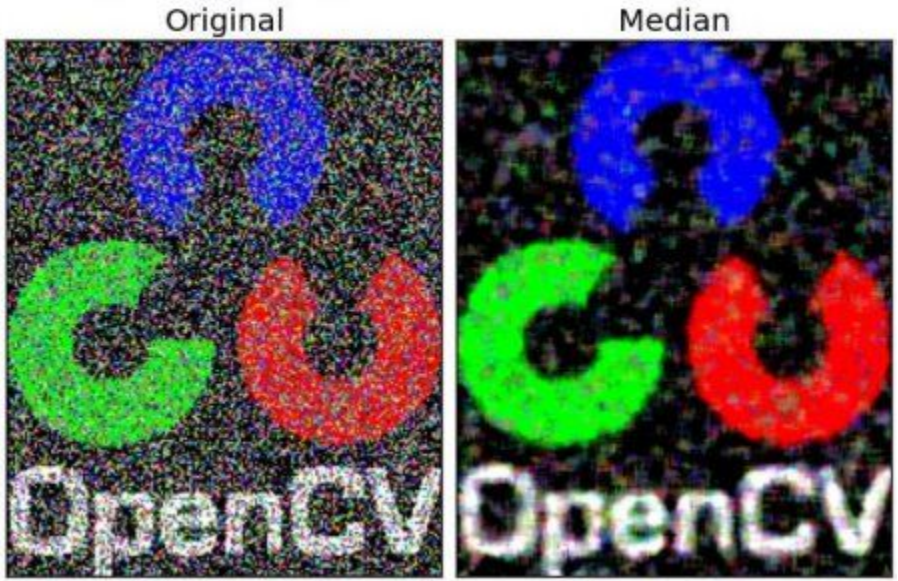


Image Filtering

https://docs.opencv.org/master/d4/d13/tutorial_py_filtering.html



Morphological Transformations

https://docs.opencv.org/master/d9/d61/tutorial_py_morphological_ops.html

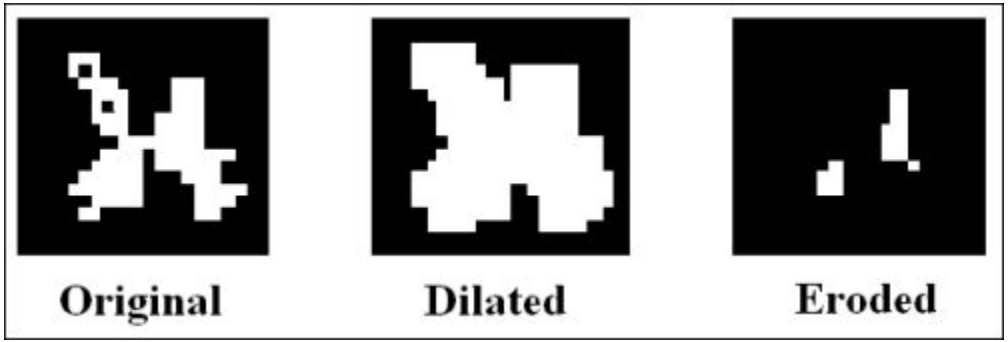
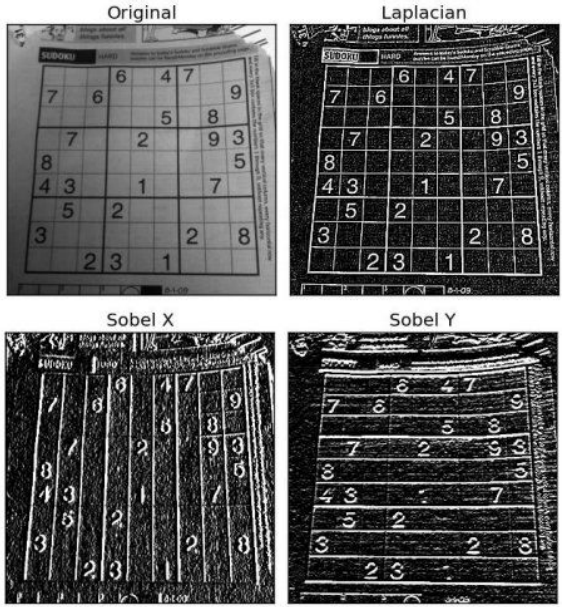


Image Gradients and Edge Detection

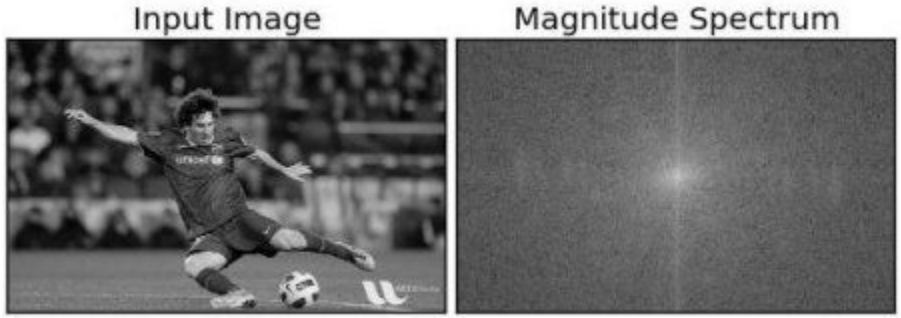
https://docs.opencv.org/master/d5/d0f/tutorial_py_gradients.html

https://docs.opencv.org/master/da/d22/tutorial_py_canny.html



Fourier Transform

https://docs.opencv.org/master/de/dbc/tutorial_py_fourier_transform.html



Common Pitfalls

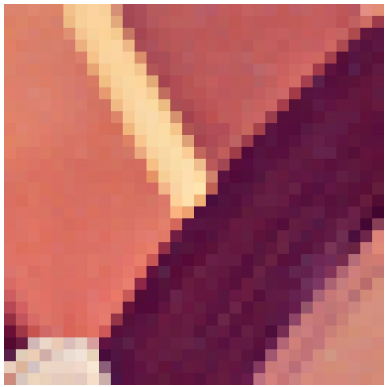
Resampling in sRGB

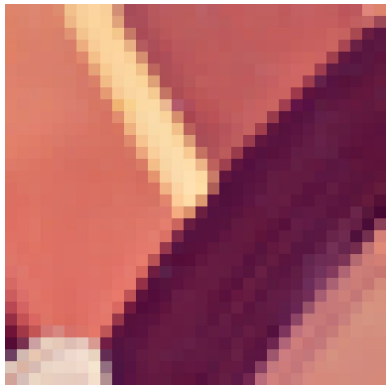
Image upscaling or downscaling should be done in linear RGB space, and not sRGB. Unfortunately this mistake is extremely common and practically in every photo editing software and scientific package. (more recent software tend to have the option to rescale in linear space)

Bicubic Downscaling

Downsampling using bicubic should be discouraged, as it adds ringing and changes the image in a way that it is no longer similar to natural images. In fact you can build a classifier that distinguishes images downscaled using bicubic from real images with almost perfect accuracy.
The correct operation is supersampling in linear RGB.

Bicubic

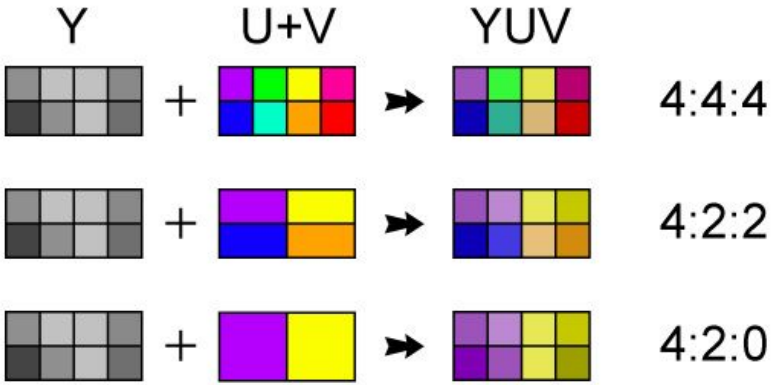




Supersampling

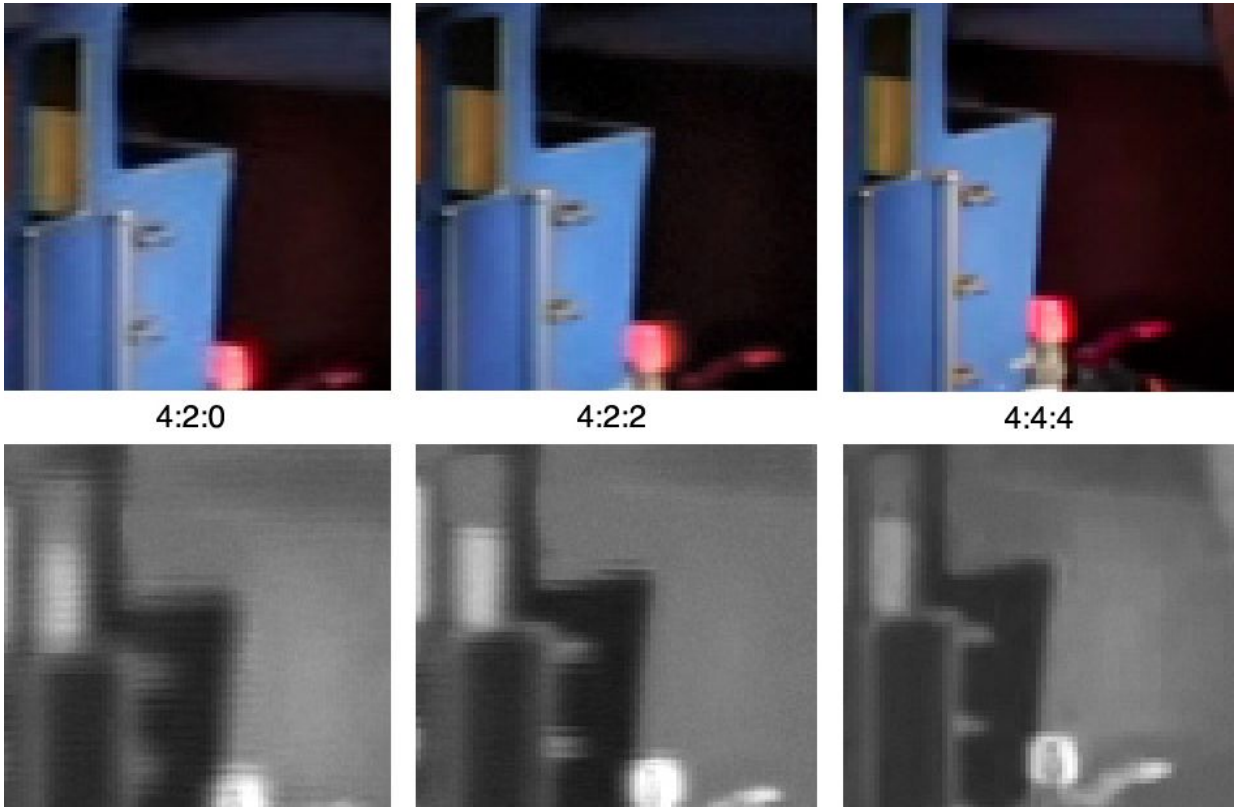
Chroma Subsampling

Most video and JPEG images are encoded with chroma subsampling. The U+V channels in YUV color space are downsampled and compressed to reduce file size. Humans don’t perceive this change but the quality degradation exists, and you should take it into account when doing computer vision.
The most common chroma subsampling is 4:2:0.



Chroma Subsampling

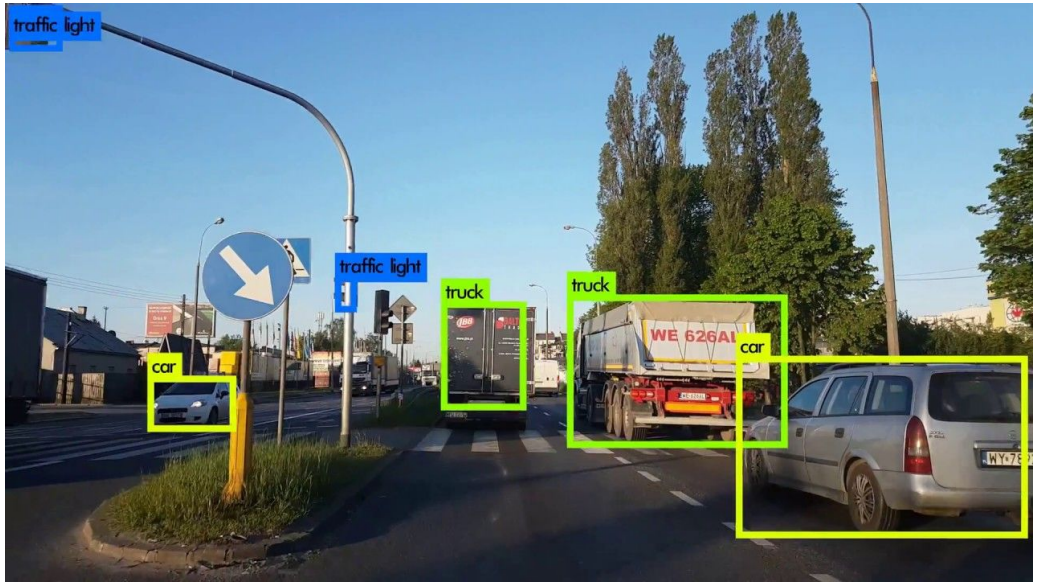
Chroma Subsampling



High(er) Level Computer Vision

Object Detection

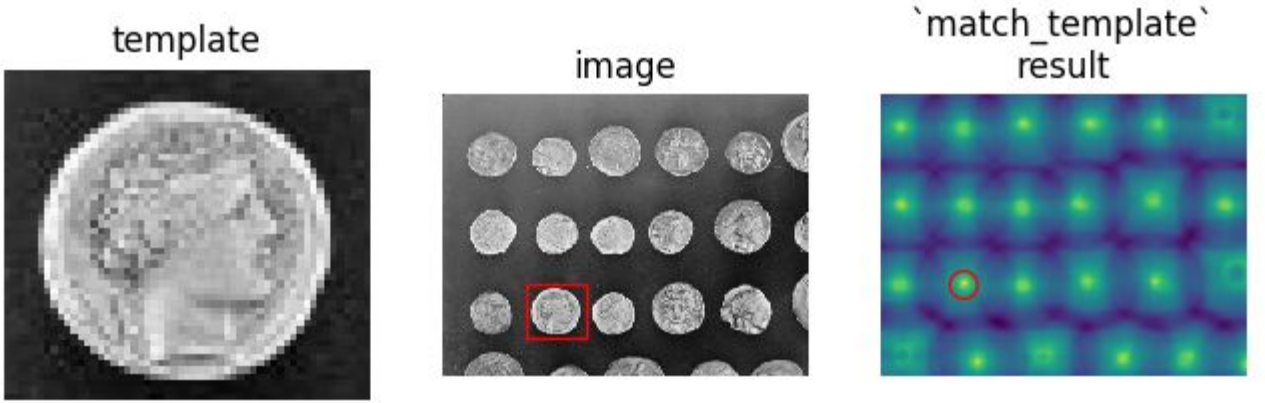
Now that we have preprocessed the image and extracted all the relevant features, how do we detect an object?



Template Matching

https://docs.opencv.org/master/d4/dc6/tutorial_py_template_matching.html

Simply use the template to convolve over the target image, and take the local maxima as possible detections.

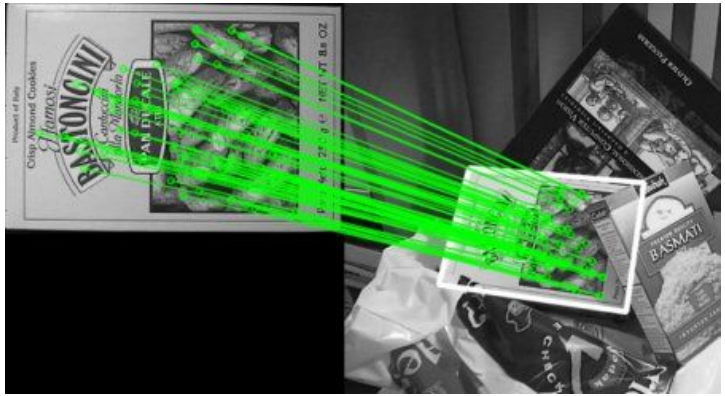


SIFT/ORB Feature Matching and Homograpy

Template Matching is too slow for scale/rotational invariance! You would have to try every combination of scale, rotations and perspectives.

https://docs.opencv.org/master/da/df5/tutorial_py_sift_intro.html
https://docs.opencv.org/master/d1/d89/tutorial_py_orb.html
https://docs.opencv.org/master/dc/dc3/tutorial_py_matcher.html
https://docs.opencv.org/master/d1/de0/tutorial_py_feature_homography.html

+ Live demo in class

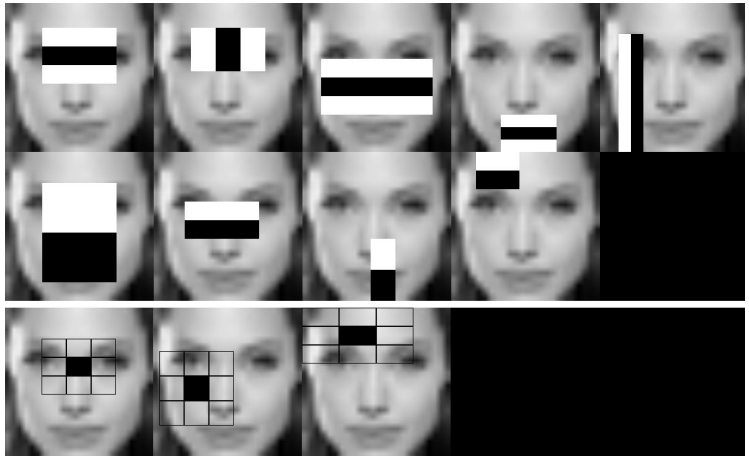


Haar Feature-based Cascade Classifiers

All human faces have similar features (eyes, nose, mouth, etc). We can generalize this into a cascade classifier.

https://docs.opencv.org/master/db/d28/tutorial_cascade_classifier.html

+ Live demo in class



Face Recognition and Verification

Now that we have a face detector, how do we recognize each individual using their faces?

Traditional Classification?

Yes, will work, but...

- Need a large amount of data. Do we have a large amount of faces from each person we want to identify? (Hard to create a dataset!)
- Need to re-train the classifier each time we need to identify a new person! It would be a nightmare!
- We need something more flexible...

One-Shot/Few-Shot Learning

Learning from a single or a few examples is a difficult task, but it can be done.

Usually it is done by knowledge transfer. We train a model on a large dataset, then we can use transfer learning to learn from a single example.

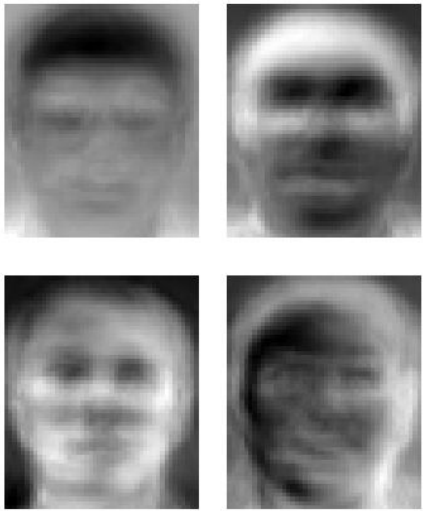
Eigenfaces (Turk, et al. 1991)

We can train PCA on a large number of faces to build a covariance matrix.

Then, we can reduce the dimensionality of a image of a known person, and use it as an identification vector.

For unseen images, we can simply apply PCA
and compare it against the identification vector.
If it is close enough, we assume it is the same person.

Any possible problems with this technique?



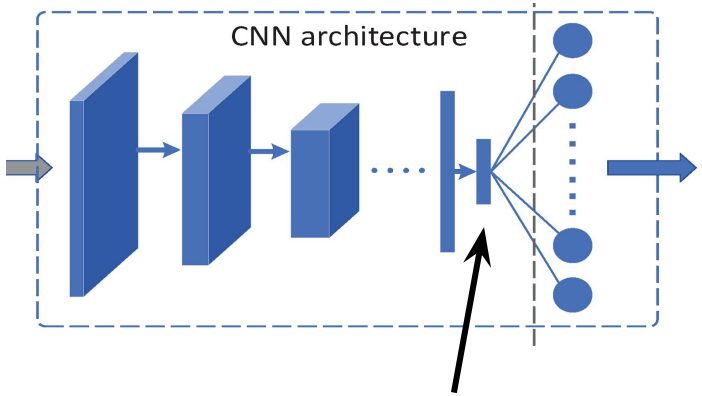
Autoencoders

Autoencoders could be used as an direct replacement for PCA, but it is not commonly used in practice.

Any possible problems with this technique?

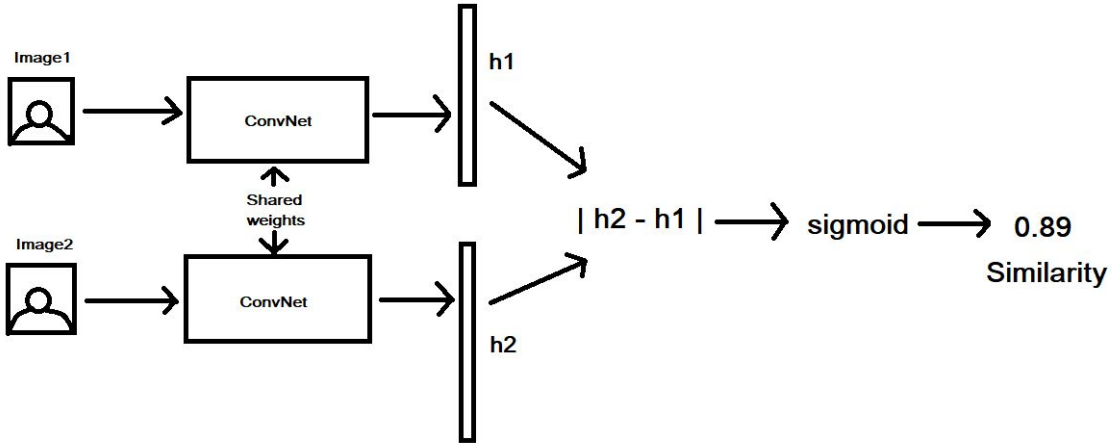
Bottleneck Classification (Taigman, et al. 2014)

We can train a classifier with a bottleneck on a large amount of faces. Then use vector space of the bottleneck as a replacement for the PCA eigenface technique seen before. This should produce a more meaningful embedding compared to PCA and autoencoders.



Siamese Networks and Similarity Learning (Koch, et al. 2015)

Instead of directly learning the class of the images, the goal of siamese networks is to learn the similarity/dissimilarity between two images.
After being successfully trained, it should be able to tell if two faces are similar or not.

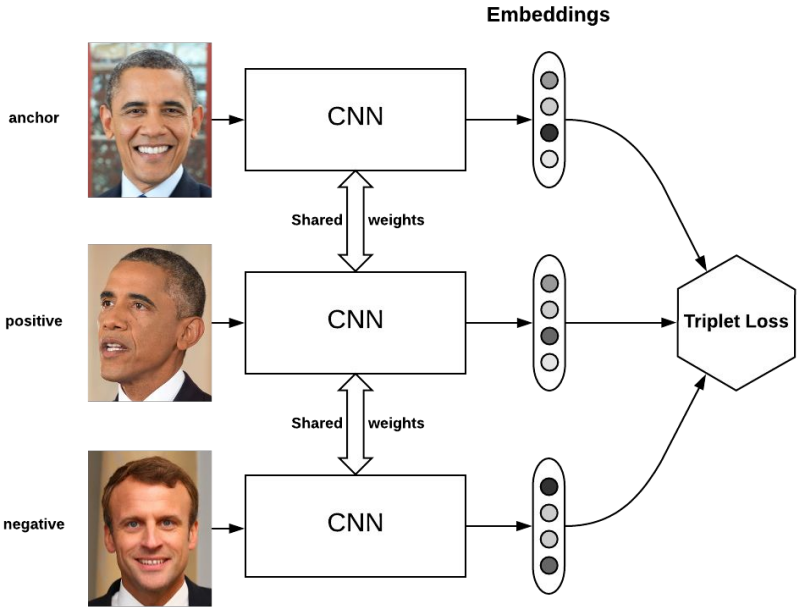


Triplet Loss (Hermans, et al. 2017)

Instead of training for similarity as a probability, we can directly learn an low dimensional embedding of similarity.

To define triplet loss, we need three inputs.

- The anchor, which is the image we wish to compare.
- A positive image, which belongs to the same class as the anchor.
- A negative image, which does not belong to the same class as the anchor.



Triplet Loss

Instead of training for similarity as a probability, we can directly learn an low dimensional embedding of similarity.

Triplet loss is defined as minimizing the distance between the anchor and a positive image, while maximizing the distance between the anchor and a negative image.

$$\sum_{i}^{N} \left[\|f(x_i^a) - f(x_i^p)\|_2^2 - \|f(x_i^a) - f(x_i^n)\|_2^2 + \alpha \right]$$