CV2 Notes

Vision → To know what is where by looking.

{CV → Art of perception by M/cs.}

Applications:-

1. Computational photography (increasing exposure and facial recognition).
2. Biometrics (used as facial recognition for security purposes)
3. Optical character recognition (for recogn. chars on car numbers & hand-writing recognition).
4. Health (used in radiology and Clevr-II and III for doctors/pathologists).
5. Gaming (recognizing hand gestures and postures along with location).
6. Shopping (Amazon Go).
7. Security and Tracking (Recognizing images and use it to track miscreants / law breakers).
   - → where do you draw the line?
8. Special Effects → in movies, thru CV, images/videos are animated & they have to track fine details on the face to track it to animations.
9. Insight → "Track stock prices of walmart dep. on no. of cars in front of stores".
   CV could help us get data / insights into the data.
10. Self-driving cars → CV detection, human-figure & sign detection and make changes (RL).
11. Augmented Reality → "Lighting correct, location correct, shadows also considered".
    (into real-time, looks as real objects).
12. Space Exploration → "Mars Rover".

(X) Why is CV so hard?

1. firstly, Computers sees images as numbers (as a matrix, 3D matrix of numbers) → Rep. digital images.

(2.) Illumination → As lighting is different, the image changes based on illumination when a photo is taken. (a shadow could even change the image by a lot).

(3.) Occlusion → we live in a 3D world but try to look at the world through 2D. (having overlaps of background and foreground). → in a 3rd axis which can't be shown correctly / detected in a 2D setting.

(4.) Class variation → "Diff kind of chairs", it has so many variations, so to categorize an object, it's really hard problem.

(5.) Clutter and camouflage → we can see camouflage but comps. (evolution) can't see camouflage easily as the intensity graph formed is really hard to separate the object and background. (or find a needle in a haystack). map

(6.) Color → It depends on diff. kinds of intensities in an image. (colors seem different due to that).
⇒ [color of light * color of the object] → might seem different.

(7.) Motion → "Blur occurs in an image" → due to which processing becomes diff. for an object (motion detection).

(8.) Ill-posed Problem → "Necker cube". (diff. 3D shapes could project the same 2D image).
→ How to solve this inverse transf. problem from 2D to 3D. (Accidental view point).
(diff. angles used for perception).
↳ mostly used for art!

"Cambrian Explosion". → "sudden evolution of vision" (why?)
↳ sets off an evolutionary arms race where animals either evolved / died!

{Evolution of biological eye} → Aperture control → Exposure control → Intensity (Camera)

→ About 1/3rd of the brain is devoted to visual processing. (2)

Loops are important! → (for complex processing).

Lecture 2.

Image Denoising ⇒

film grain ⇝ {black & pepper noise on an image.}

Ways →

(1) Average many photos (to remove noise) → Problems:-

(a) What if you have only one?

(b) Frame shouldn't be moved!

Image as functions ⇒ $F[x,y]$ → x,y coord → {returns the pixel values in the matrix of the image.}

Type of filters that could be used →

(1.) Moving Avg. (Take an avg. of the neighborhood.)

(filter sizes shouldn't be very big) → otherwise you lose a lot of data!

→ Kernel

(This will blur the image).

Linear filter props:

(1) $filter(im, f1+f2) = filter(im, f1) + filter(im, f2)$

(2) $c * filter(im, f1) = filter(im, c * f1)$.

(Avg. kernelization)

$\Rightarrow F[x,y] * \frac{1}{9} \times G_1[x,y] = $ Output Image.

(* → convolution)

$F[x,y]$ ↑ image

$G_1[x,y]$ ↑ filter

1	1	1
1	1	1
1	1	1

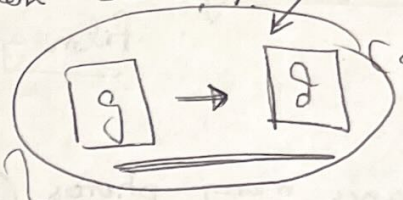
Defn ⇒

$$(f * g)[x,y] = \sum_{i,j} f[x-i, y-j]\, g[i,j]$$

Code flip the Kernel and then apply it to ^mathematics easy.

flip LR, UD
(left-right, up-down)

(for symmetric filters, this doesn't matter much!)
(but for asymmetric filters → this makes much more sense!)



But we generally do!

{ dec. indices in f and inc. in filter
eq. to flipping the kernel }

Translation filter

because we flip the filter and move in the image from bottom right, the image moves accordingly & an extra white/black would be added to the opp. edge.

0	0	0
0	0	1
0	0	0

→ move img. to the right & a white col is added to LHS.
(bcoz. of the flip)

$\frac{1}{9}$

1	1	1
1	1	1
1	1	1

→ "Blur filter".

(As you increase size of the filter, you lose more data!)

→ "No linear filter" can be used to rotate/transpose the image.

If we put a 0 in the averaging filter, we see it as an intensity loss (info. loss) and because of which the blur looks a little intensified.

Sharpening filter

$$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 0 \end{bmatrix} - \frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

Ans.

Sharpening :-

Image − Blurred = Details lost (edges)

Image + Details lost ⇒ Sharpened Image

Essentially, edges are intensified & other details remain the same and noise will also be intensified. (in case there is noise).

Explanation →

Image = D_2 + Noise

Blurred = D_1

Image − Blurred = $D_2 - D_1$ + Noise

↓

Details lost

Image + Details lost =

= D_2 + Noise + ($D_2 - D_1$ + Noise)

= $2D_2 + 2\text{Noise} - D_1$

↓

"noise intensified"

you get the edges intensified. (due to this)

* As noise is intensified, the edges look sharper to us (another way).

Ans

Convolution Properties :-

Commutative :-

$$F * H = H * F$$

Distributive :-

$$(F * G) + (H * G) = (F + H) * G$$

Associative :-

$$(F * H) * G = F * (H * G)$$

Scale →

$$\text{filter}(A * f) = A * \text{filter}(f)$$

Ans

Shift invariance :-

$$\text{filter}(\text{shift}(f)) = \text{shift}(\text{filter}(f))$$

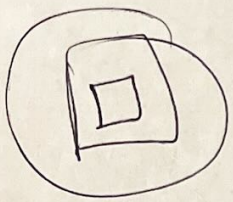
Cross-correlation)

but not nice props

$$(f * g)[x,y] = \sum_{i,j} f[x+i, y+j]\, g[i,j]$$

Ans.

Box filter :-



1	1	1
1	0	1
1	1	1

→ middle pixel not being considered.

(losing the info. in the middle)

& blurring the image.

+ you will see multiple details. Ans.

(multiple glasses, multiple left eye, etc).

{ Kind of shifted images are averaged together. }

(Instead we use gaussian filter)

(symmetric filter)

(this solves the problem with box filters)

$$G_\sigma = \frac{1}{2\pi\sigma^2} e^{\frac{-(x^2+y^2)}{2\sigma^2}}$$

→ mean = 0

(Constant factor at front makes volume sum to unity).

{ σ → determines the extent of smoothing }

Complexity?

↳ of filtering an n×n image with an m×m kernel?

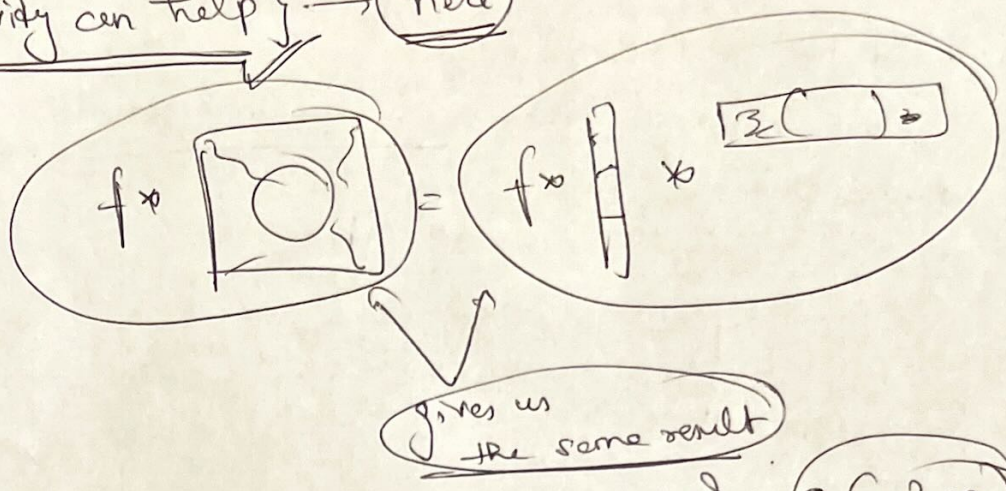
$$O(n^2 m^2)$$

$$G(x,y) = \frac{1}{2\pi\cdot\sigma^2} \exp\left(\frac{-x^2+y^2}{2\sigma^2}\right)$$

$$= \left(\frac{1}{\sqrt{2\pi}\,\sigma} \exp\left(\frac{-x^2}{2\sigma^2}\right)\right)\left(\frac{1}{\sqrt{2\pi}\,\sigma} \exp\left(\frac{-y^2}{2\sigma^2}\right)\right)$$

Ans

{Associativity can help} → here



gives us the same result

(*) Kernel separable than complexity ∂ $O(n^2 m)$ Ans.

Gabor ↓

(Cosine * gaussian)

$$\psi(x,y) = e^{-\frac{x^2+y^2}{2\sigma^2}} \cos(2\pi\mu x)$$ Ans

biologically related to cat's receptive field.

How do you take derivative of an image?

{gradient tells → how does intensity change along an axis. (wrt its derived)}

First derivative

$\frac{\partial I}{\partial x}$ → vertical edges are shown → change in horizontal dir^n

$\frac{\partial I}{\partial y}$ → horizontal edges are shown → change in vertical dir^n

↳ How can we do this using convolution.?

If Image * $[-1, 1] \Rightarrow \frac{\partial I}{\partial x}$ ①

Image * $[-1, 1]^T = \frac{\partial I}{\partial y}$

for rate of change, take ~~second~~ derivative

$\frac{\partial^2 I}{\partial x}, \frac{\partial^2 I}{\partial y}$

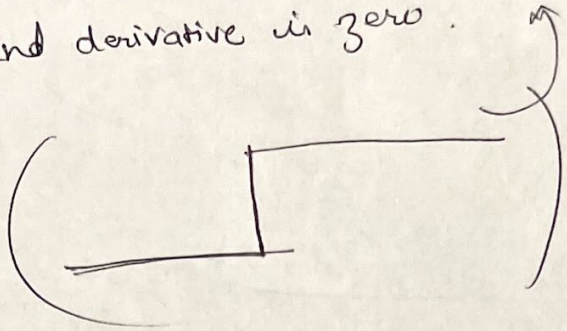
① * $[-1, 1]$ ✓ Ans

Edge?

⇒ Change is measured by derivative in 1D.

— Biggest change, derivative has maximum magnitude. (large magnitude)

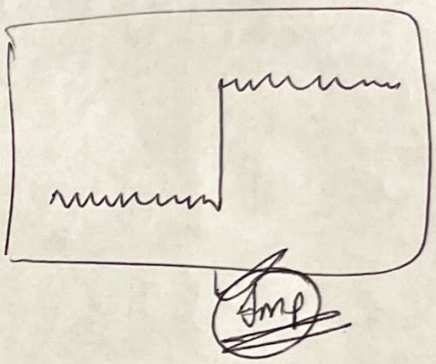
— Or 2nd derivative is zero. (this shows!)



(*) What about noise?

→ Derivative is high everywhere.

→ Must smooth before taking gradient!

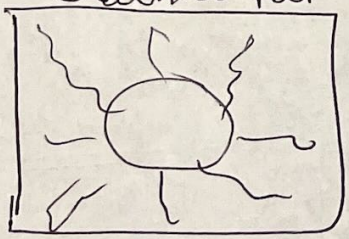


Handling Noise ⇒

→ filter with gaussian to smooth, then take gradients.

→ But, convolution is linear.

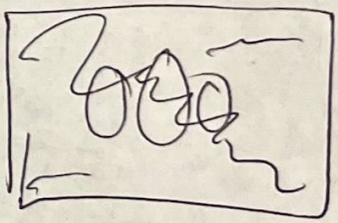
Gaussian filt



$* [-1, 1] * [-1, 1] =$

for vertical edges (change in hor. dir^n)

Laplacian filter



Transpose these to find horizontal edges (change in vertical dir^n)

(rest of change will be more prominent!)

(that's why!)

Doing first $[-1, 1]$ & then $[1, -1]$ will give blank (nothing) → might only see noise.

Laplacian filter defined as sum of second order partial derivatives

$$\nabla I^2 \quad \frac{\partial^2 I}{\partial x^2} + \frac{\partial^2 I}{\partial y^2}$$

Ans.

finding boundaries:)

$f * g$ (laplacian) $\Rightarrow$ $\frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2} > \lambda$ → why use a threshold?

2 reasons:-

1. To take coarse lines/edges rather than taking into consideration all the fine lines/edges.
2. For visualization purposes, we binarize the output image to get either black/white pixel values. (giving edges on a background of black).

finding things:)

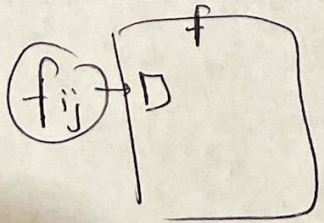
If we take a filter with an image of an object, then:—

$\Rightarrow f *$ (eye image in the filter) g $\Rightarrow$ $f * g$ (weird blurred image).

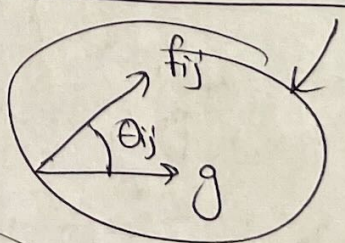
(cannot be used for image detection!).

(a difficult problem!) → need neural nets for this.

Response for one window:)



$\Rightarrow$ $f_{ij}^T g = \|f_{ij}\| \|g\| \cos\theta_{ij}$



(norm of f and g gets multiplied which won't make sense!)

makes norm of g smaller!

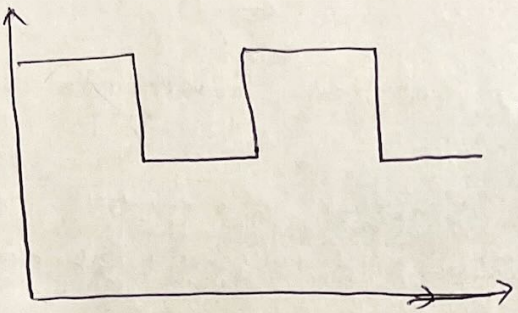
$f * (g - \bar{g})$ → to find the filter!

after thresholding → get false +ve's and true +ve's

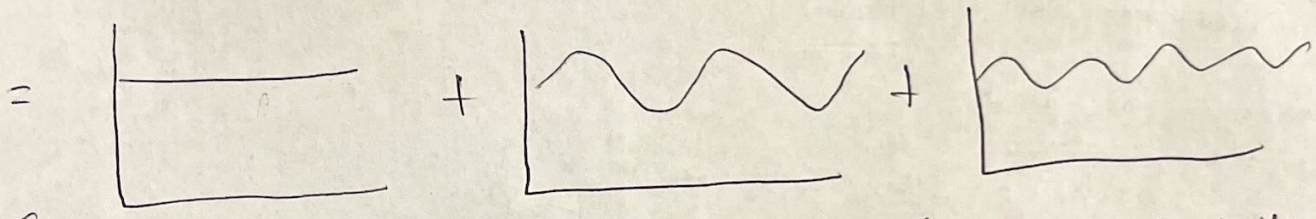
Lecture-3

Fourier Transform

Any univariate func^n can be rewritten as a weighted sum of sines & cosines of different frequencies.



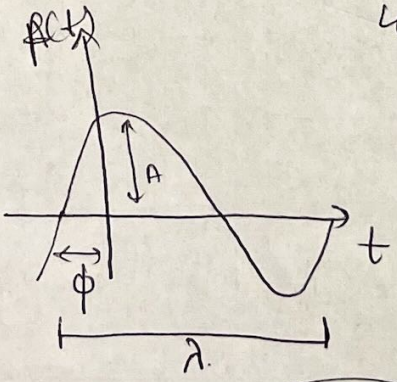
Q. How to build this 1D signal using sin waves?



+ ... when till ∞, freq. increases of the waves.

(can be applied to images)

Sinusoids



$$f(t) = A\sin(2\pi ft + \phi) = A\sin(\omega t + \phi)$$

A = amplitude
ϕ = phase
f = frequency

Signal = Amplitude (vs. freq) + phase (vs. freq) → (also applicable to images)

(freq. is implicit)

treated as signal.

black → 0
white → 1

black → -ve
white → n large (+ve)
grey → zero

* Both are same to represent image and its fourier components.

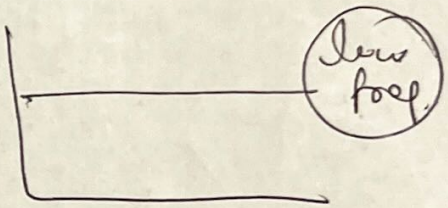
Imp

(Most of the info. is in the phase!)

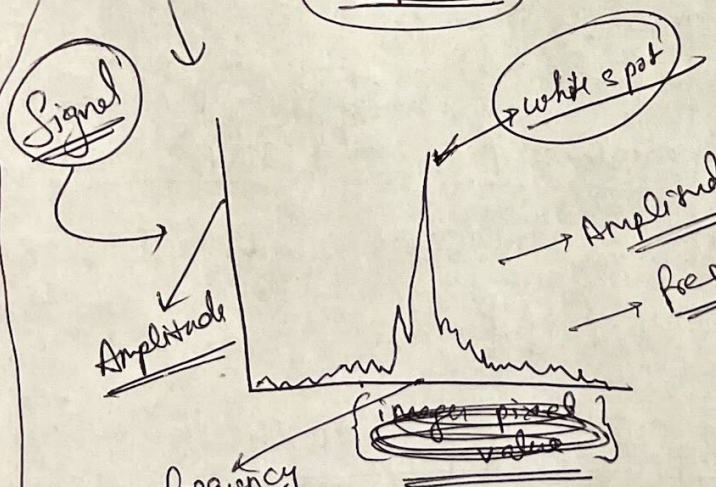
(Amplitude has less info. about the image.)

(high freq. data is mostly stored in the phase → which gives the details of the image).

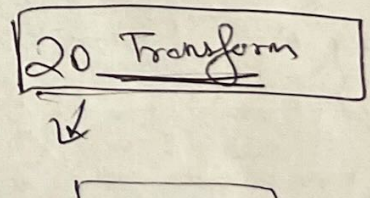
{Amplitude → basically mostly gives you the low freq. part of the data.}

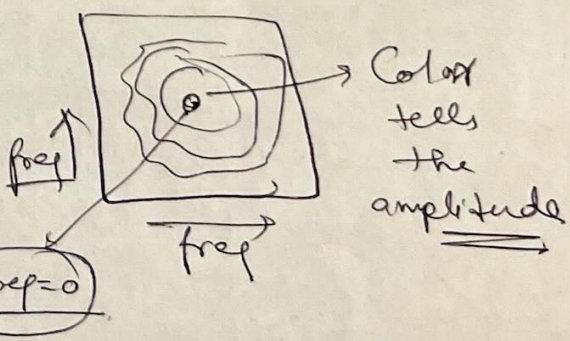


{Low pass filter → lower freq pass + higher blocked}



{freq ∝ 1/Amp}





* Can we reduce the size of the image using this?

As high freq. waves are mostly black (0) in terms of amplitude which doesn't provide us with a lot of data about the image. So, maybe we can do so!

Imp. ⇒ All the signals (components of fourier transform) can be mapped to amplitude & phase, which when combined would give us amplitude & phase for the whole image.

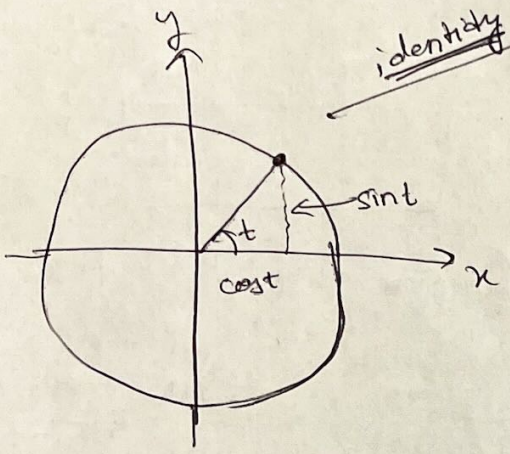
⇒ Convolution basically does this transformation for us.

⇒ (FT is also a linear transformation).

(*) How fourier transform works?

Sine / Cosine and circles

identity ⇒ $e^{ift} = \cos ft + i \sin ft$



Amplitude :- Radius of circle.

Frequency :- How fast you change t. (fastly to move around the circle).

→ when you change the center of the circle then, giving it a phase and adding them up, we generate

(*) Each circle is mapped to a particulars freq. and amplitude.

$g(t)\, e^{-2\pi ift}$ → maps g(t) on to the unit circle with freq. f.

{this is essentially the amplitude}

→ You wrap g(t) around the circle with freq f, then calc avg. pos^{n} of g(t).

$$G_r(f) = \int_{-\infty}^{\infty} g(t)\, e^{-2\pi ift}\, dt$$

→ when f move from $-\infty$ to ∞.

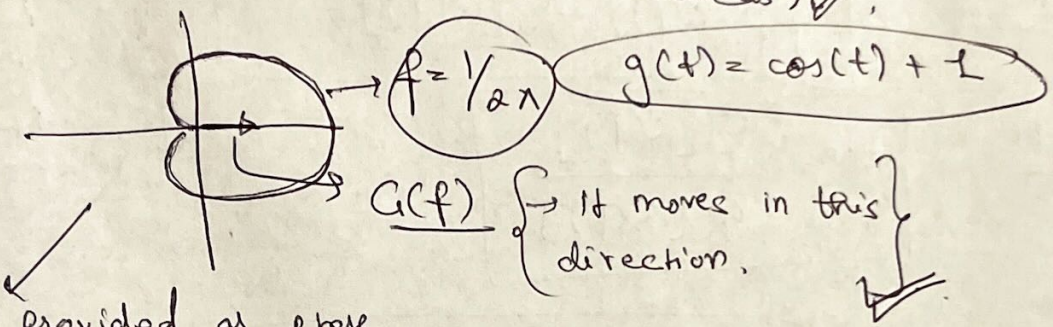
(wait this?) → we move indefinitely.

{g(t) transformed to the freq. domain.}

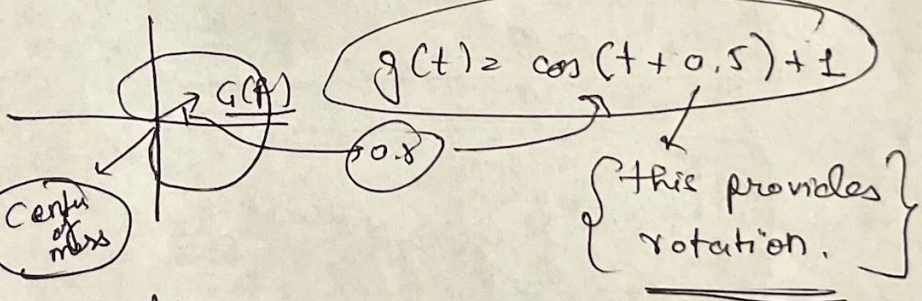
→ Very low freq goes around the circle very slowly.
(some signals can be mapped around.)
→ when freq. doubled, you would go faster.

→ If we keep inc. f, then move till ∞, you will generate diff. kind of patterns
↳ this can be mapped to image data then (to generate diff. pixels).

$f = 1/\pi, 1/2\pi$,
↳ it translates to lines (rather than covering areas) ✓.

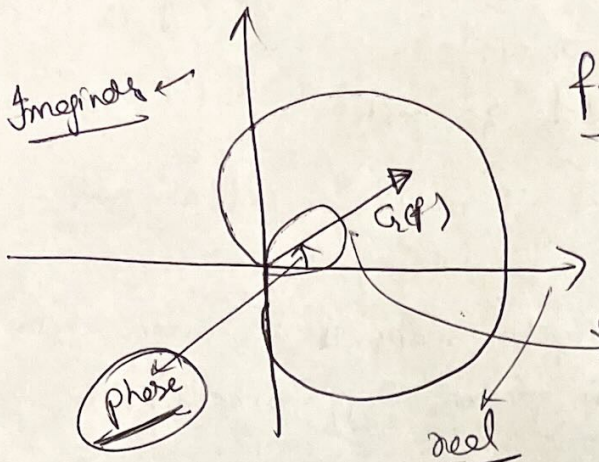


when provided as phase,
G(f) would move (shifts).



Amp
→ Amplitude → radius of the pattern.
→ frequency → how fast you move along the pattern.
(this would change the avg. position in the pattern)
→ Phase → this rotates the pattern. (rotation of the whole pattern)

(*) COM is only affected by Amplitude.



$f = 1/2\pi$, $g(t) = 2\cos(t+0.5)+1$

$g(t)\, e^{-2\pi i f t}$, $t = 0 \ldots 100\pi$

{vector becomes longer, thus, COM changes.

Can we charged the amplitude).

Amplitude:- $\sqrt{R[G(f)]^2 + I[G(f)]^2}$

Phase:- $\tan^{-1} \dfrac{I[G(f)]}{R[G(f)]}$

(Real & Imaginary components).

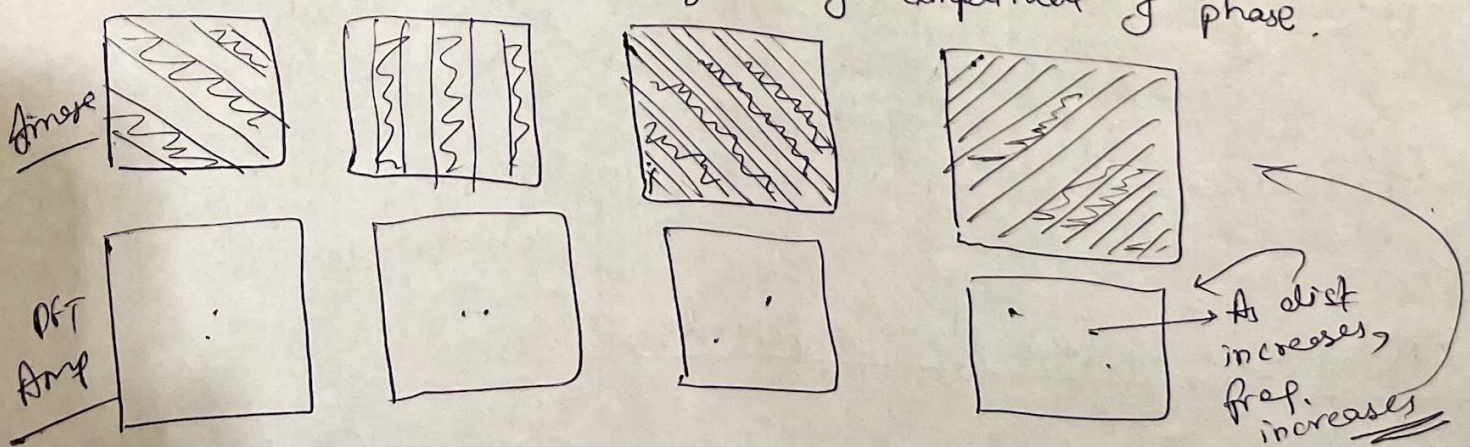
Inverse FT:-

$$g(t) = \int_{-\infty}^{\infty} G(f)\, e^{2\pi i f t}\, dt$$

FT:-

$$G(f) = \int_{-\infty}^{\infty} g(t)\, e^{-2\pi i f t}\, dt$$

→ You can reconstruct an image using amplitude & phase.



⇒ FT has peaks at spatial frequencies of repeated structures (for repeated info. in the image, spatial freq. data changes in a repeated manner).

⇒ Acc. to grad changes, you get similar patterns in the amplitude part in accordance with the frequencies (freq. of change).

⇒ Might be both in the x- & y- directions. (depending on sharp changes in the corresponding directions).

{Ex:- Bricks wall image example.}

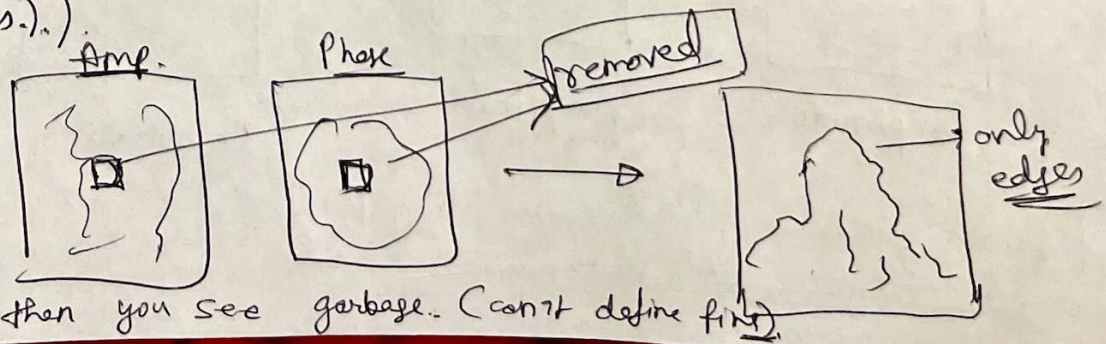
we can even throw away some of the values that are very close to zero or zero and only stores the frequencies that contributes a lot. (using a low-pass filter).

Could hold a threshold that could work as a filter, thus, giving us compressed image. ↑ (How JPEG works?)

→ when we convolve and avg. out some of the pixels data, that removes the high freq. data (mirrors a low-pass filter). → this is actually convolution mapped to fourier transform.

Imp (as more convolution map (larger), gets more blurred).

→ for edge detection, we basically use high-pass filter. (that let go high freq. details (only the fine ones) → which are basically edges.).



→ Convolution in time space is multiplication in freq. space:

$$g(x) * h(x) = \mathcal{F}^{-1}[\mathcal{F}[g(x)] \cdot \mathcal{F}[h(x)]]$$

→ Conv. in freq. space is multiplication in time space:-

$$\mathcal{F}[g(x)] * \mathcal{F}[h(x)] = \mathcal{F}[g(x) \cdot h(x)]$$

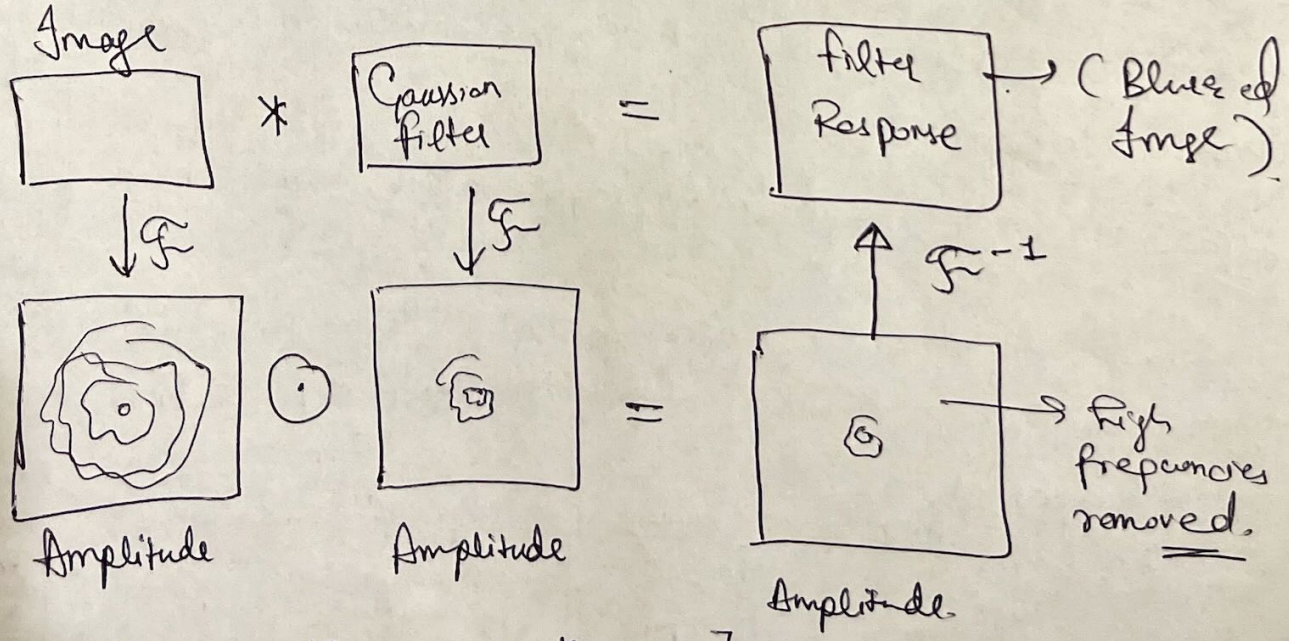
Which is more computationally efficient?

⟹ Depends on how fast, Fourier transform can be done.
[from time → freq. or freq → time]

⟹ [If filter is larger, then taking Fourier transform and then taking inverse will be faster].

⟹ [If filter is small, then direct convolution will be faster].

⟹ Steps:-



[Process gives us the same thing!]

Mean filter ⇒ Low-pass filter → less blurred image than gaussian.

Gaussian ⇒ Low-pass filter → blurred image. ← {Fourier Transform filter} (9)

Laplacian ⇒ High-pass filter → edges (edge detection)

Sobel-x ⇒ High-pass filter → only vertical edges detected (some edges might be lost)

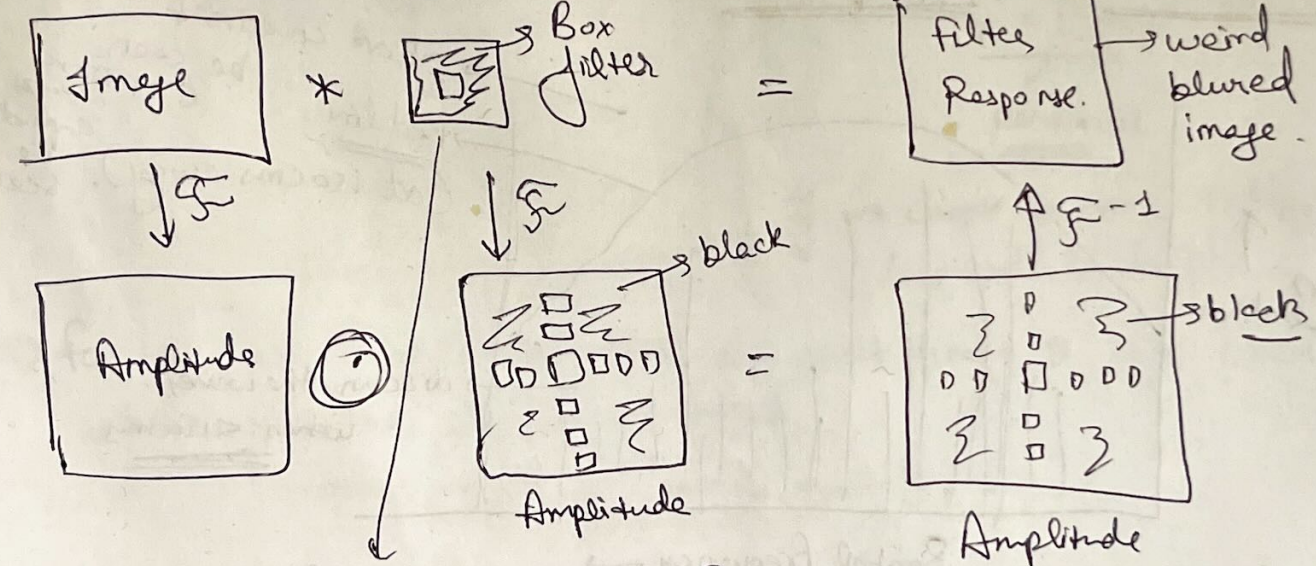
Sobel-y ⇒ High-pass filter → only horizontal edges detected.

~~Box filter ⇒ high pass filter → edge detection.~~

but for Scharr_x → we will mostly all vertical edges

$[-1, 2, 1]$

→ FT of gaussian is gaussian.

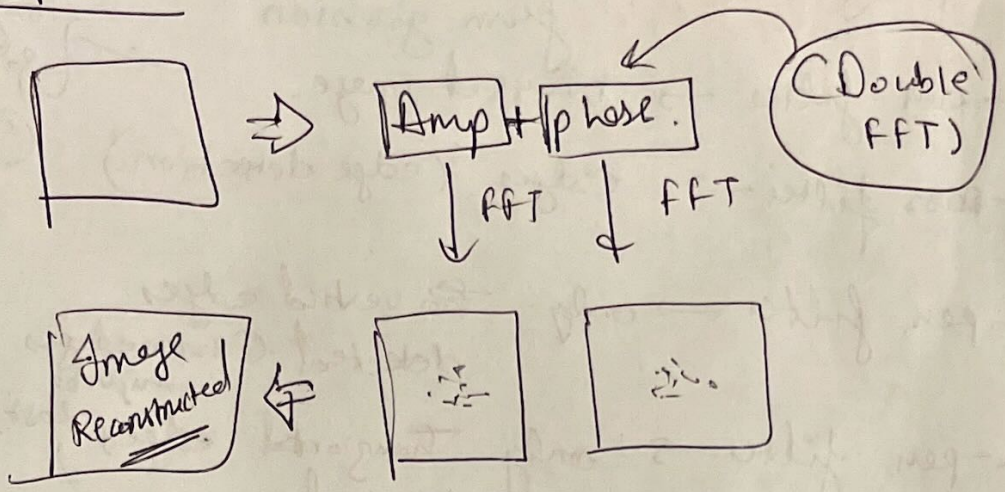


{because of the sharp boundaries}

⇒ $f \rightarrow \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2}$ (Laplacian filters)

⇒ In FT space, this is high-pass

Compression ⇒



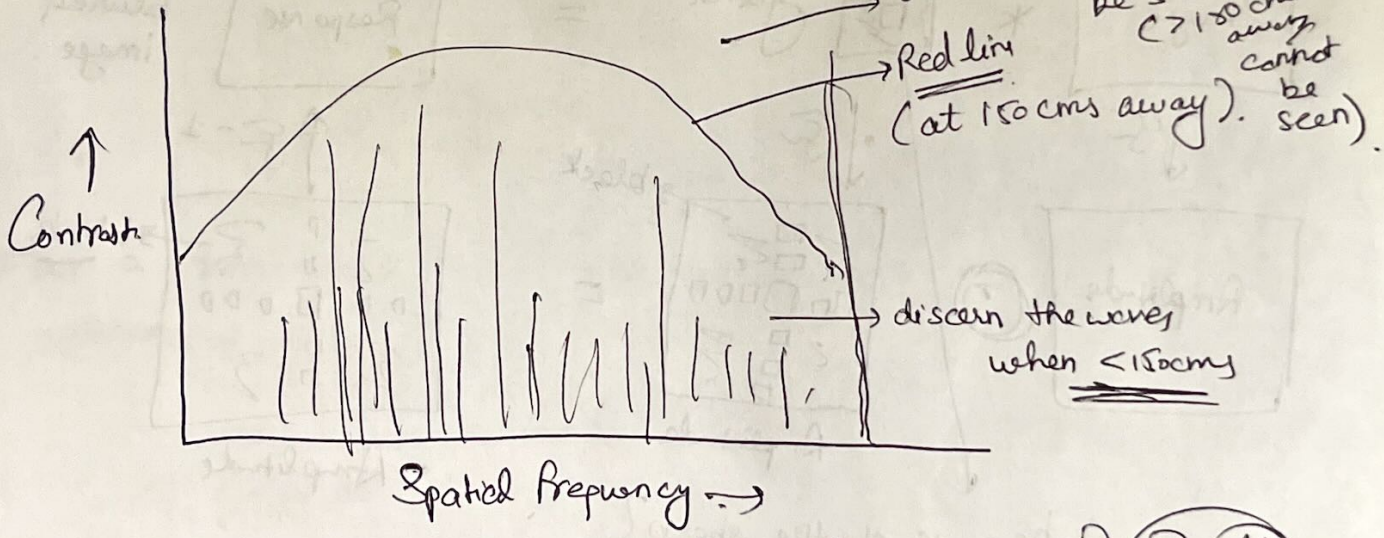
(compressed by a lot!)

Hybrid Images ⇒ (for creating illusions)

FT of image 1 — FT of image 2

Low frequencies + High frequencies ⇒ Hybrid Image

→ >150 cms away (Hide image within another)



(A) When far away, can see the low frequencies.

(B) When close, can see the high frequencies.

Result

Ans.

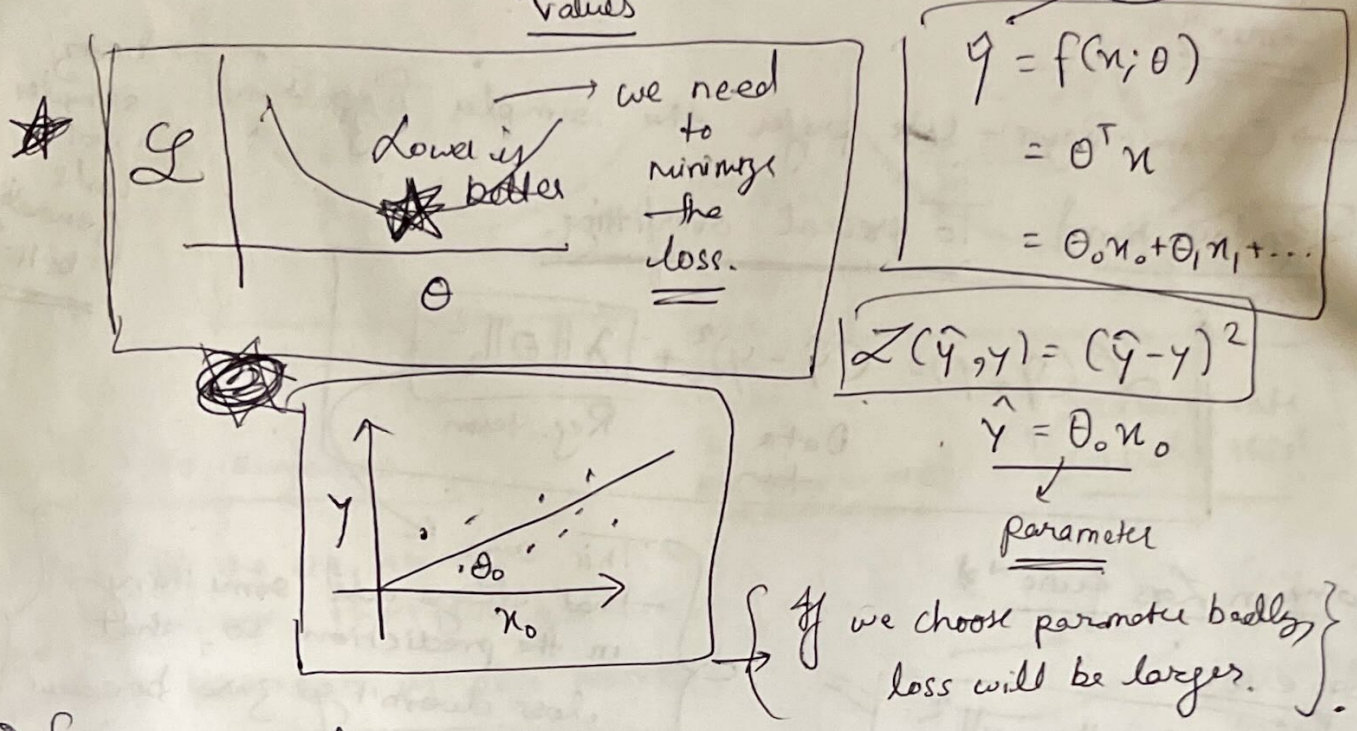
Machine Learning

→ we need translation invariance. (Same operation everywhere).
& scale invariance.

Regression basics:-

Model:- $\hat{y} = f(x;\theta)$; Input image: x, Parameters θ.

Loss function:- $\mathcal{L}(\hat{y}, y)$ ← pred. values; ↳ target labels



⇒ for a non-linear relationship, you can't find a θ with linear relationship where loss is minimized,

Now, we use polynomial models.

2nd order polynomial:-

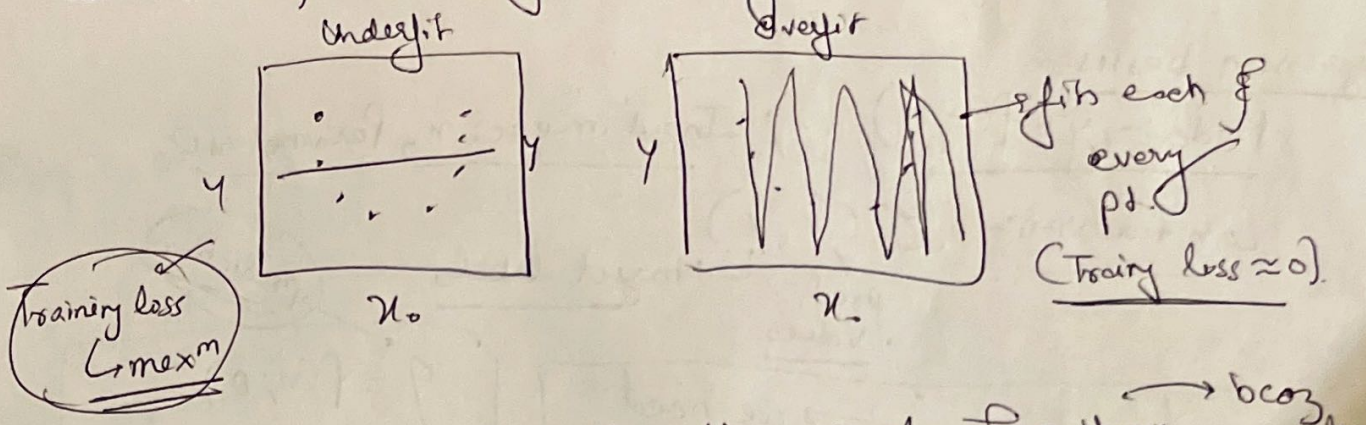
$$f(x;\theta) = \theta_0 x_0 + \theta_1 x_1 + \boxed{\theta_2 x_0^2 + \theta_3 x_1^2} + \theta_4 x_1 x_2 + \theta_5$$

$$= \theta^T \phi(x)$$ (we change the feature space)

ϕ → Transformation function (transforms the space) { S.t we can do linear relationship in that feature space. }

(*) Nth order polynomial possible.

If we go beyond a point where ~~1000~~ Nth order polynomial overfits the data (all predictions = labels). When tested, would perform poorly.



→ Occam's Razor:- We prefer the simpler hypothesis. → bcoz simpler set will generalize better.

Regularization → To prevent overfitting.

$$\mathcal{L}(\hat{y}, y) = (\hat{y} - y)^2 + \lambda \|\theta\|_2^2$$

Data term | Reg. term.

This says that always add something in the prediction so, that loss doesn't go zero because of the data term.

Common Loss func^{n} →

Sq. error :-

$$\mathcal{L}(x, y) = \|x - y\|_2^2$$

Hinge loss :-

$$\mathcal{L}(x, y) = \max(0, 1 - x \cdot y)$$

→ minimize this

Cosine Similarity :-

$$\mathcal{L}(x, y) = \frac{x^T \cdot y}{\|x\| \|y\|}$$

→ maximize this

Learning →

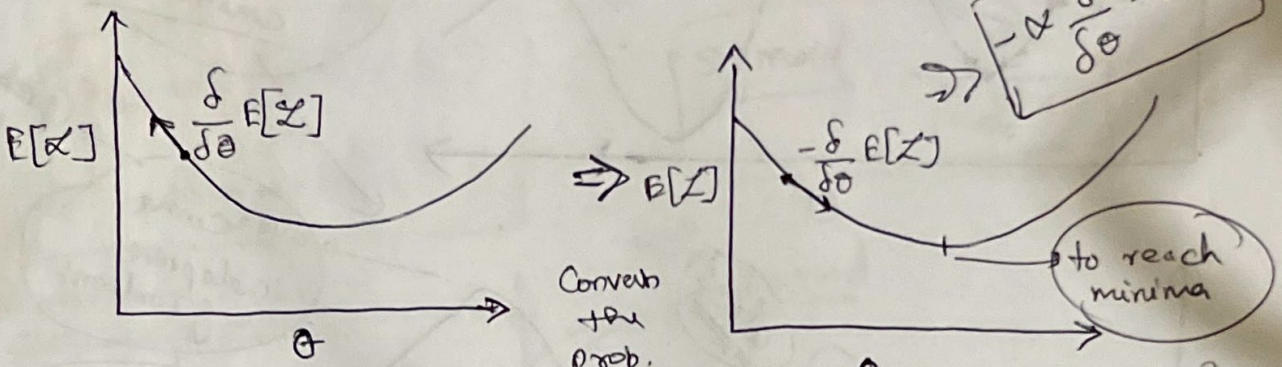
$$\min_{\theta} E_{(x,y) \in D} \left[\mathcal{L}(f(x_i; \theta), y_i) \right]$$

{What are the params that minimize the value of the loss func^{n} in expectation over the entire dataset?}

We want to minimize the diff. b/w prediction and the actual label, as we want to minimize the expectation of loss.

Gradient Descent:-

Steps:
1. Make an initial guess for params.
2. Calc. gradient of loss w.r.t to the params
3. Step parallel to the gradient.



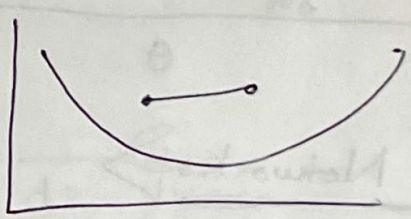
Converts the prob. into a minimization one!

{learning rate controls this!}

Convex loss surfaces:-

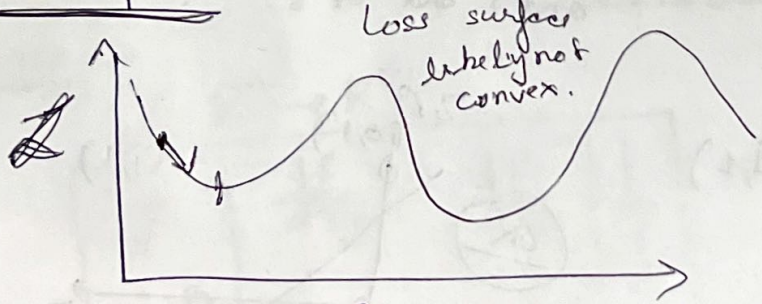
local = global maxima/minima

{If the 2 points are above, then everything in between us also above.}



(we solve optimization problem)

Local vs. Global:-



sometimes, millions of dimensions

(we need to see that we don't get stuck on local maxima/minima).

(*) We need momentum here to see if we aren't stuck at a local maxima/minima (optima)

Gradient descent :-

$$\theta_{t+1} = \theta_t + \alpha \frac{\delta \mathcal{L}}{\delta \theta_t}$$

Grad. descent with momentum :-

momentum

$$z_{t+1} = \beta z_t + \frac{\delta \mathcal{L}}{\delta \theta_t}$$

change

$$\theta_{t+1} = \theta_t + \alpha z_{t+1}$$

momentum = High

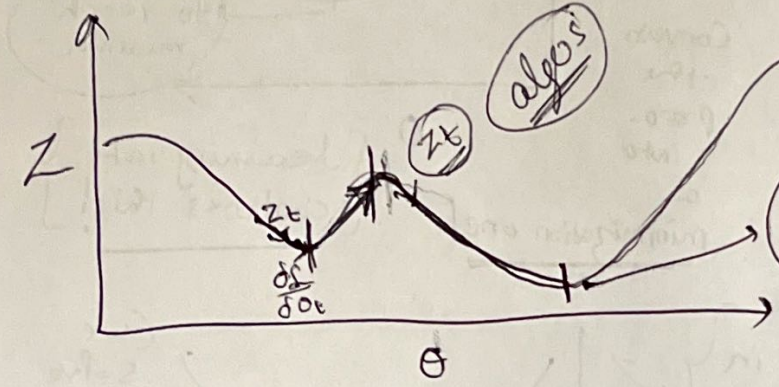
search speed constraint

(problem with gradient)

overcome adaptive gradient

+ momentum

Adam

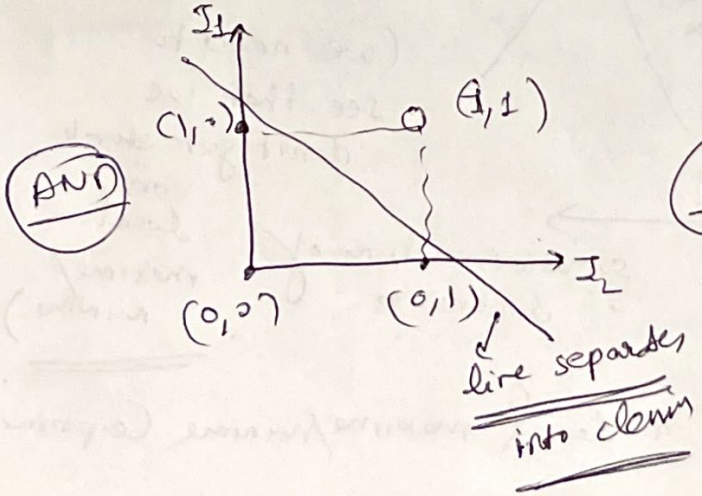


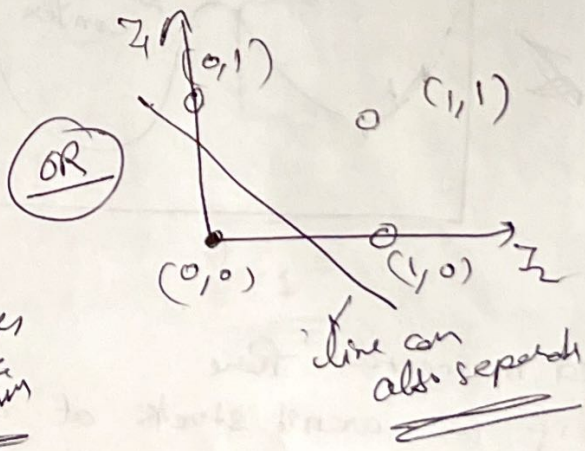
algos

global optima

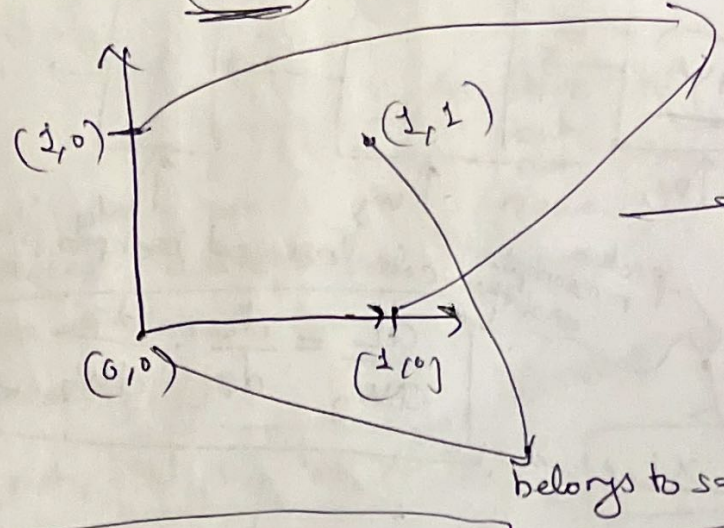
Neural Networks :-

Perceptron :- If I/P features are binary, can model logical "and"s and "or"s.





Q → What about (XORs) ?



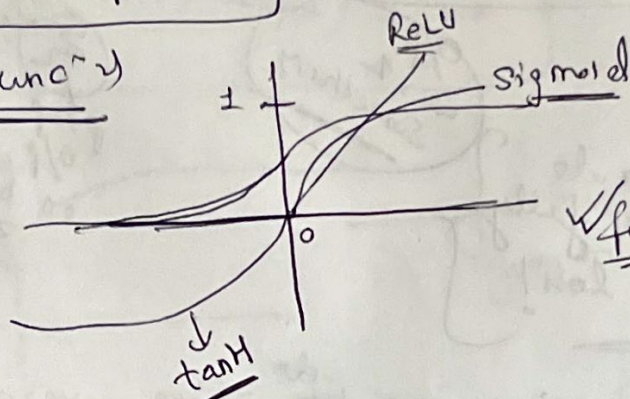
→ {not a linear problem!}

$x_{i+1} = \phi(W_i x_i + b_i)$

↓ Activation func^n

Activation func^n

Sigmoid :- $\frac{e^{-x}}{1+e^{-x}} = \frac{1}{1+e^{x}}$



func^{ns}

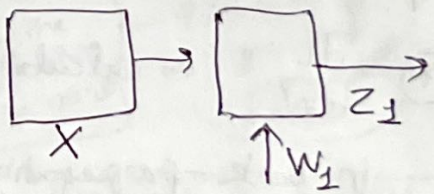
Chain Rule:-

$y = f(x),\ z = g(y) \Rightarrow \frac{dz}{dx} = \frac{dz}{dy} \cdot \frac{dy}{dx}$

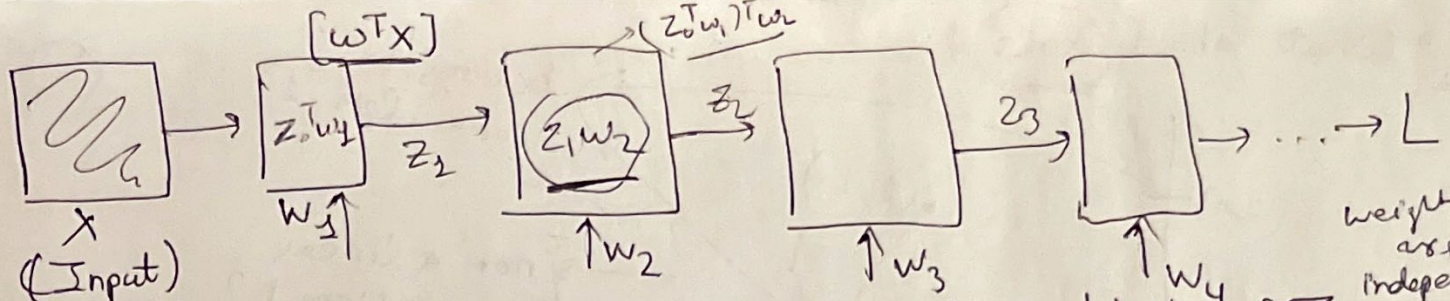
Jacobian:-

$\in : R^n \rightarrow R^m$

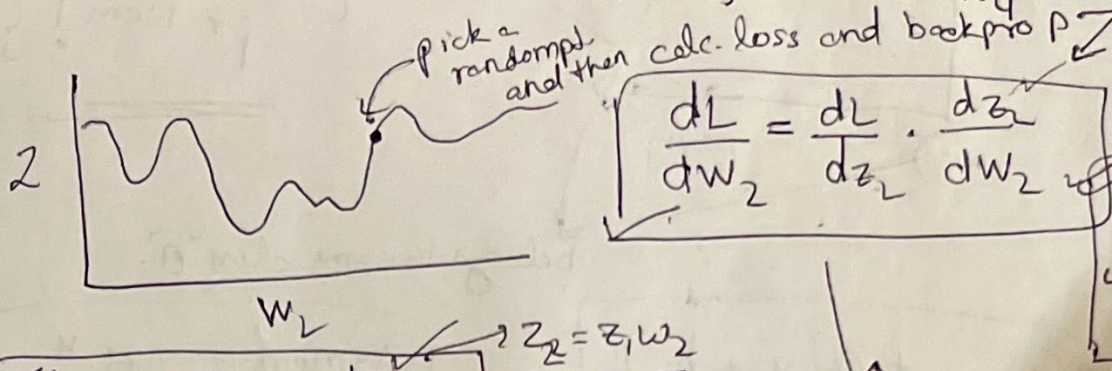
$$J = \begin{bmatrix} \frac{df}{dx_1} & \cdots & \frac{df}{dx_n} \end{bmatrix} \Big| m \quad (\text{rows}),\ n \text{ (columns)}$$

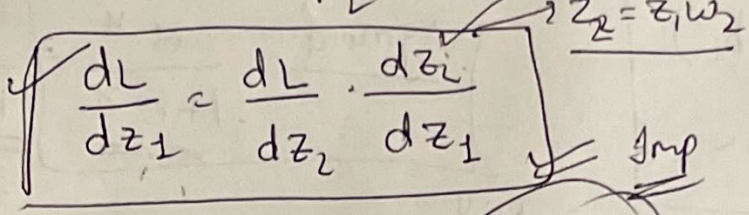


$z_1 = w_1 x$



weights are independent





this was bcoz. z_1 & w_2 aren't dependent on each other.

z_2 prop o/p on both z_1 & w_2 thus

o/p w.r.t something

Imp ~~Imp~~ can apply chain rule now to get the fully backpropagated loss!

→ Most activation functions ~~are~~ ~~does~~ do have -ve as well as +ves as input and when the inputs are normalized, it would mirror those activation functions, that's why standardization seems so important and also because of exploding gradient problem.

(*) $\frac{dL}{dW_{n-1}}$ is calculated before $\frac{dL}{dW_{n-2}}$ so, that it can be reused in back-propagation. (reuse the computations already done from the end to the start).

→ Vanishing gradient → if $\frac{dL}{W_{n-1}}$ goes to zero, all $\frac{dL}{dW_{n-2}} \dots \frac{dL}{dW_1}$ goes to zero. (backprop problem).

$\Rightarrow$ If $\frac{dL}{dw_n}$ shouldn't go to zero, then $\frac{dL}{dz_7}\frac{dz_3}{dz_2}\frac{dz_2}{dw_n}$ —

gradients ≈ 0 but not zero.

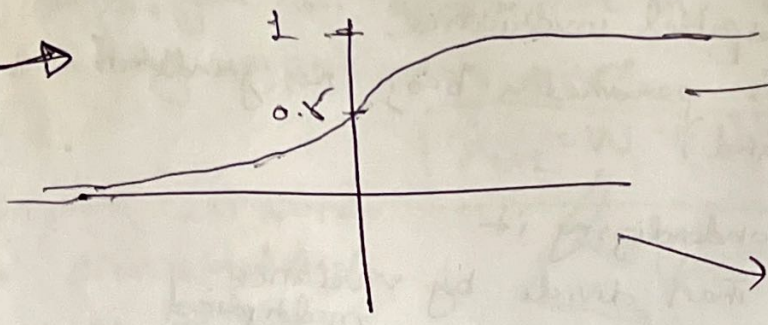
Should not be zero

That's why we introduced activation function to introduce non-linearity.

↳ If not, $W_4 W_3 W_2 W_1 X \Rightarrow WX$ (linear op^{n}s; matrix multiplication)

this has the potential to give lots of zeros.

$\ldots act(W_3 act(W_2 act(W_1 X)))\ldots$ → non-linearity introduced.



for this gradient never goes to exactly zero.

$\frac{d(\text{sigmoid})}{dx} \neq 0$

act(input)

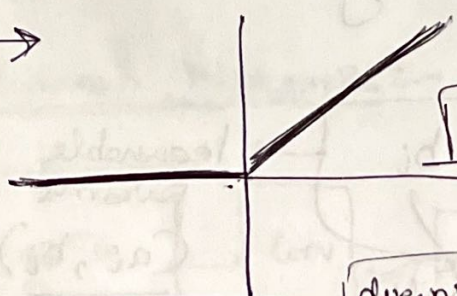
$0 \Leftarrow \frac{d(act(input))}{dx}$

if this goes really small

effectively zero

ReLU → $\max(0, x)$

half of the time this can be zero → vanishing gradient prob



expected value $\neq$ doesn't come eq to zero → greater than zero

(n large) → high number of training data

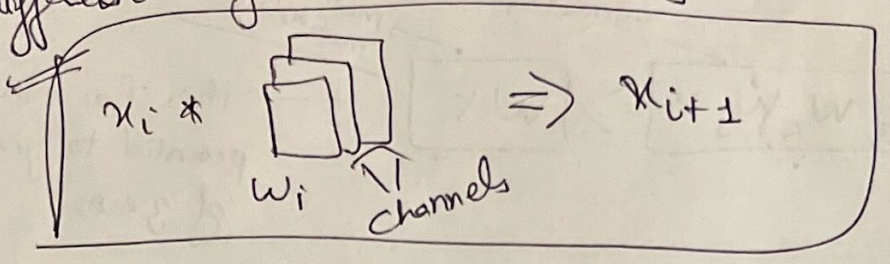
	n small	n large
N/W size small	ReLU ✗ L-ReLU	ReLU ✓
N/W size large	ReLU ✗ L-ReLU	ReLU & L-ReLU ✓ → experiment

$x_i * W_i = x_{i+1}$ → for Convolutional layer.

Conv. filter

$x_{i+1} = w_i * x_i + b_i$

⇒ for each layer, there might be some n filters. that detect different features in the image.



(*) Max Pooling ⇒ [Non-linear convolution] → only outputs the max value in the window spanned by the filter in a point in time.

⇒ Pooling effectively helps in spatial invariance.
(no learnable parameters bcoz no gradients can be calculated).

Batch Normalization ⇒ (Prevents overfitting)

→ Standardizing it
→ Then divide by variance.
→ Scalar coeff → ~~shift multiplied by added~~ multiplied.
the value.
→ some linear value for a batch multiplied.

Intuition ⇒ build some second-order information into the N/w by normalizing the variables before the activation function.

$$x_{i+1} = a_i \frac{x_i - E[x_i]}{Var[x_i]} + b_i$$

→ learnable params (a_i, b_i)

a_i → ~~shift~~ scalar coefficient
b_i → shift

Ans

Activation functions ⇒

Softmax

$$x_{i+1}^{j} = \frac{\exp(x_i^j)}{\sum_k \exp(x_i^k)}$$

(multi-class)

→ probability for an o/p over probabilities for all the o/ps. (the full vector).

Dropout ⇒

j ↳ dimension

$$x_{i+1}^{j} = \begin{cases} x_{i+1}^{j} & \text{with prob. } p \\ 0 & \text{otherwise.} \end{cases}$$

(to prevent overfitting).

(To provide robustness to the N/w nodes).

Width vs. Depth ⇒

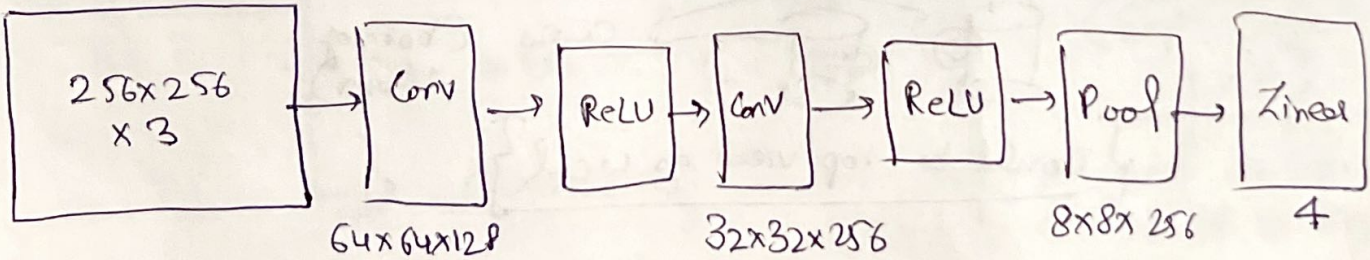
With sufficiently wide network and just one layer, you can approximate any continuous function with arbitrary approximation error.

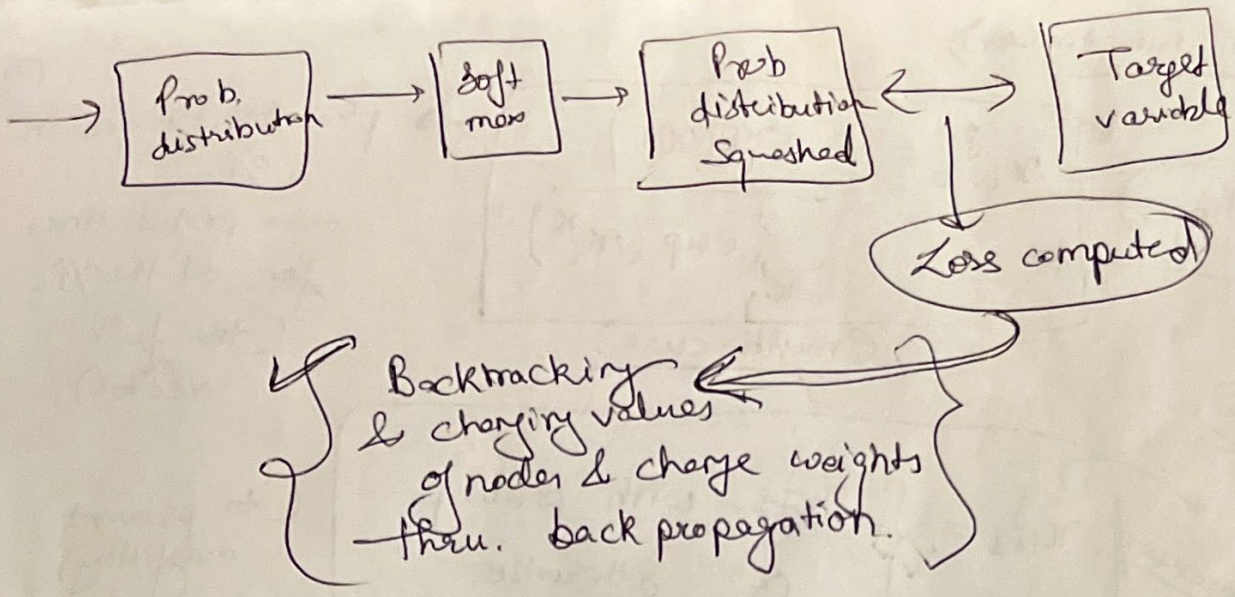
$$x_{i+1} = W_i x_i + b_i$$

- (wide) → (corresponds to size of W).
- (deep) ↳ corresponds to how many matrices do you have.

(*) (Deepness corresponds to dimensionality/complexity of the features while width corresponds to type of features that need to be extracted).

Convolutional Networks :-





Cross Entropy Loss:-

$$\mathcal{L}(y, \hat{y}) = H(y, \hat{y}) = -\sum_i y_i \log \hat{y}_i$$

(entropy; $\hat{y}$: prediction)

Entropy ⇒ Average number of bits needed to encode y if the coding scheme from $\hat{y}$ is used instead.

(the lower number of bits you need, the closer should be the distributions for y and $\hat{y}$).

→ so, the loss would be less when distributions are similar.

→ {easy to calculate gradients.}

Object Detection ⇒

→ Canonical Perspective:- "The best and most easily recognized view of an object".

↳ This hinders learning if dataset is biased.

cups (both of them)

{Could be top view as well}

→ Object Recognition is so hard because of different viewpoints.

→ Changes in illumination.

→ Change in scale.

→ Background clutter.

→ within-class:- lots of variations of an object.

(18) Problems for object recognition.

Visual object categories ranges from 10,000 – 30,000.

huge variations + views + other env. changes

So many possibilities!

Crowdsourcing → paying people to do mini-tasks.

View of categories →

→ A category is formed by some defining properties.

→ Anything that matches all/enough of the properties is part of the category.

→ Any thing else doesn't belong to the category.

Color categories →

2 categories → white & black (Stage 1)

3rd added → + Red (Stage 2)

→ + Yellow/Green (Stage 3)

4th added

5th added → + Yellow/Green (Stage 4)

6th → + Blue (Stage 5)

7th → + Brown (Stage 6)

8th → Other colors → so on and so forth.

(Category to desc. could be diff)

(Perception can be done without any language)

(Sep^n from NLP)

→ Affordance ⇒ An object is defined by what action it affords!

↳ (dog in front of computer) → ? (object could be perceived differently by the machine).

↳ {could help/not with training a model w.r.t perception}

→ Entry-level categories ⇒

Recog. prediction

grampus griseus → Dolphin ← we will say this! Could have called this an animal!

What should I call it?

Levels of abstraction ⇄ Hard to model this

{Level of category is picked by us.}

※ Perception is always related/grounded in language!

Exemplar SVMs → Learn a separate linear SVM for each instance (exemplar) in the dataset.

→ Each exemplar-SVM is trained with a single +ve instance.

→ Each exemplar-SVM is more defined by "what it is not" vs. "what it is similar to"!

Large-scale training :-

→ Each exemplar performs its own hard -ve mining.

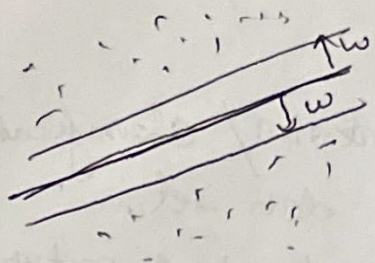
→ Solve many convex learning problems.

→ Parallel training on cluster.

(to parallelize workflow)

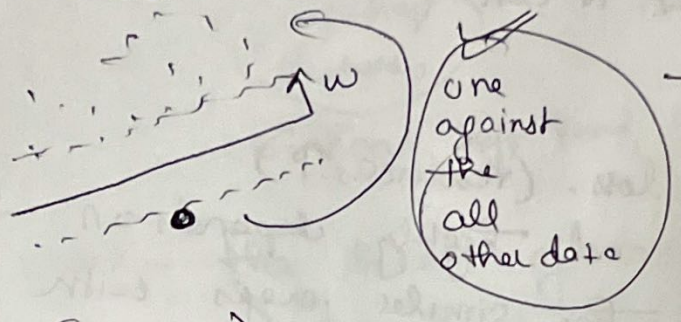
Support Vector Machine (Linear SVM):-

$$\min_{w} \lambda \|w\|_2^2 + \sum_i \max(0, 1 - y_i w^T x_i)$$



→ when a data pt. is perturbed, the classification shouldn't change! (that's what SVM takes care of).

for exemplar:-



→ used to classify that single data point.

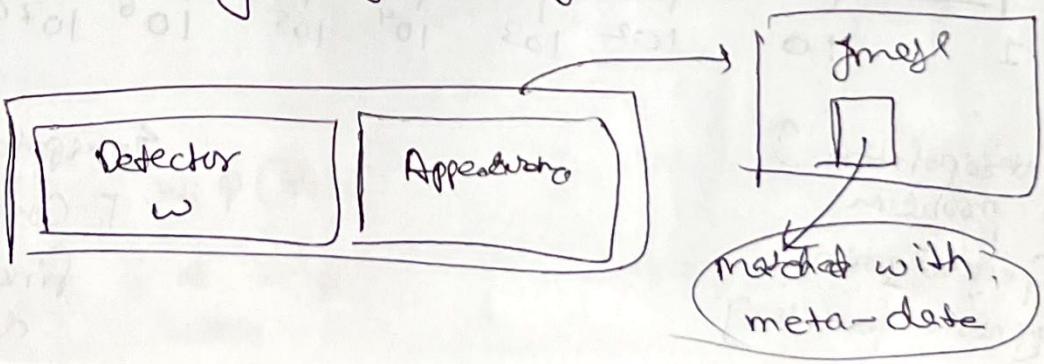
Exemplar:- (in images)

The detector makes little windows to go over the whole image and detects different features in the image, considers diff features and/or geometry inside that. Then, that is translated to features in one of the images in the data set to match an appearance.

But this doesn't necessarily map really well and depends on the data set.

→ should have very large training data set.

Exemplar



This acts like a look-up table. It might also find correlations b/w diff. objects. (similarity b/w things).

for ex:- If examples detect a bike with some features of a person, it can say that a person is sitting on top of it.

⇒ This can also be used for segmentation/ classification of objects from/in the images in the data set.

→ This look up table won't be exact but perturb it a little bit. (so, that it can match to images with little changes).

Problems:-

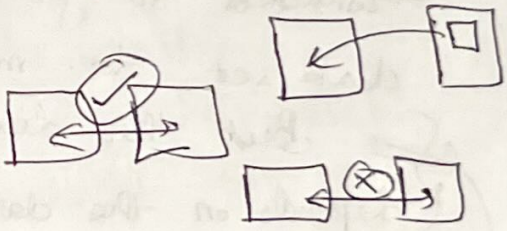
1. Computation power loss. (resources ↑)
2. It's too specific and highly depends on the training dataset, if it has similar images with similar geometry and view of the object (depends a lot on view of perception).

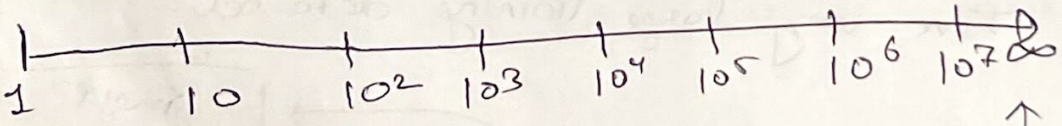
↑ (This was an interpolation problem). → that's why we need a lot of data!

⊗ Now, we will go to the Extrapolation problem.

Two extremes:-

(finding features) → features



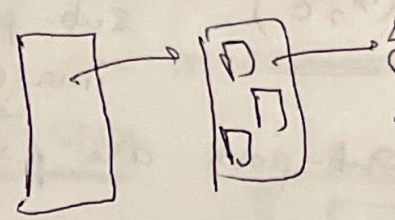


Extrapolation problem [Generalization, Diagnostic features]

Interpolation problem [Correspondence, finding the differences].

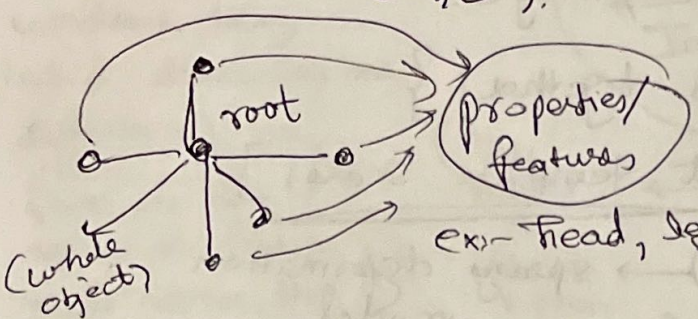
→ Deformable part models:-

(Model encodes local appearance + pairwise geometry).



finds features and encodes them to gether st if it is found together in any form, acc. to those properties, it will find that object (also considers relationships b/w objects).

Inference :- max score (x, z).



ex:- head, left hand, right hand, torso, etc.

Star model:- the location of the root filter is in the anchor point.

given the root location, all part locations are independent.

(That's how properties can be mapped).

↳ thus, help in classification of the whole object.

(we are trying to find relative relationships b/w sub-parts of an object).

→ This can work on different view-points as well! (even different configuration & might apply to illumination settings as well).

Scoring function:-

$$score(x, z) = \sum_i w_i \cdot \phi(x, z_i) + \sum_{i,j} w_{ij} \cdot \psi(z_i, z_j)$$

$x \rightarrow$ image
$z_i \rightarrow \{x_i, y_i\}$
$z = \{z_1, z_2, \dots\}$

part template scores

spring deformation model.

⇒ Score is linear in local template w_i and spring parameters w_{ij}.

$$Score(x, z) = w \cdot \phi(x, z)$$

{z → position of the sub-parts in the image}.

1st part ⇒ where / what all sub-parts are present in the image

2nd part ⇒ relative relationships between the sub-parts of the whole object / image.

↳ {considers both the things together!}

⇒ CNN has the part ⇒ [part template scores]

↳ (doesn't model) → spring deformation model.

why not do this?

* Side note:- we can transform image / object into its FFTs and then store those beforehand, basically FFTs convert space & time so, that might cover both part template scores & spring deformation part. Then, training a DNN on these waves data with corresponding labels can help in classification, [or might even help in segmentation]. }

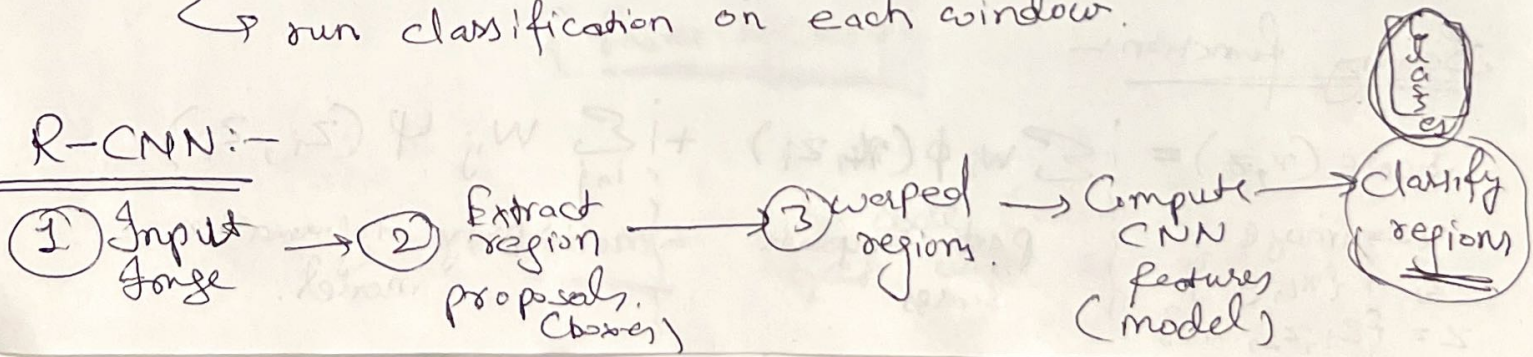
Localizing objects:-

→ Scanning window approach & image pyramids

→ select many windows over the complete image.
(candidate bounding boxes)

↳ run classification on each window.

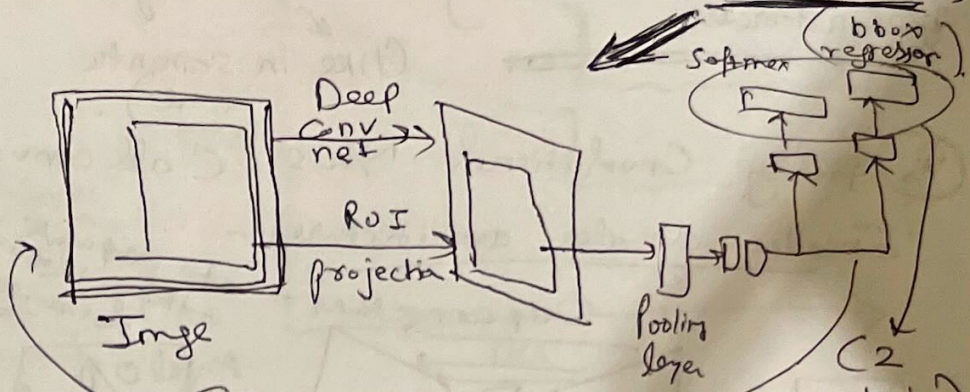
R-CNN:-



→ Learn for each of the windows and combine results.

(How do we know that learning doesn't directly mirror the training data).

⇒ If we make the structure end-to-end like in fast R-CNN, and faster R-CNN.



(Max pooling is done on these windows b/c initial dimensions are different)

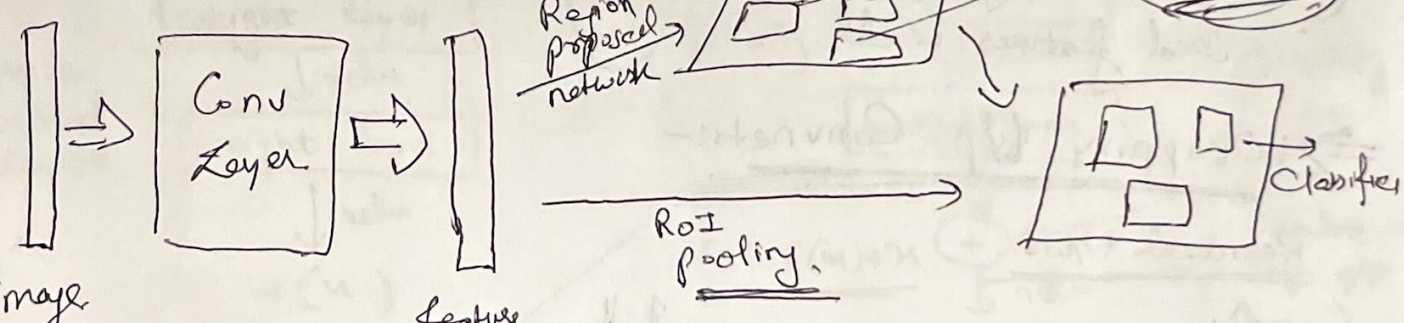
{we are able to resize the feature vector rather than the image, that's why it's faster, so, that we can reuse the computation}.

[This makes it faster than R-CNN).

(we combine finding the boxes with classification of objects in the images).

(*) this also saves resources.

{to make it follow similar dimension, so, that we see features of similar size/dimensions}.



{faster R-CNN}

Segmentation :-

(1) Semantic segmentation :- Color code (one-hot) an object with same code and background as another.

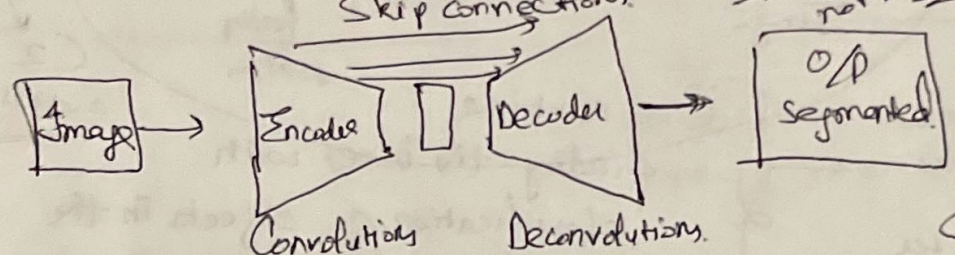
Instance segmentation:- color code different instances of an object with diff. grades of the similar color and similarly whole background with one color.

Challenge:-
{unbounded number of o/p instances.}

(like in semantic one).

(*) fully Convolutional N/ws. (all convs + pooling layers).

Encoder - decoder architectures:-



(downsampling → easier to get more features data and then upsampling → to maintain resolution).

{U-Net}

{sees the bottleneck}.

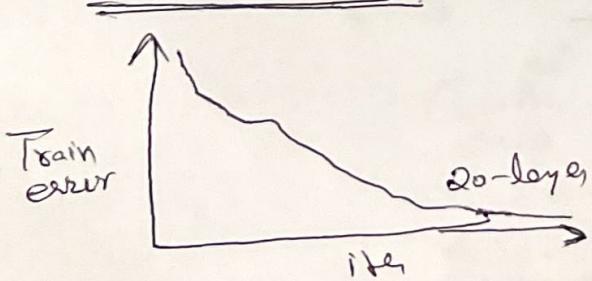
trade-off between:- finding local vs. global features in the image.

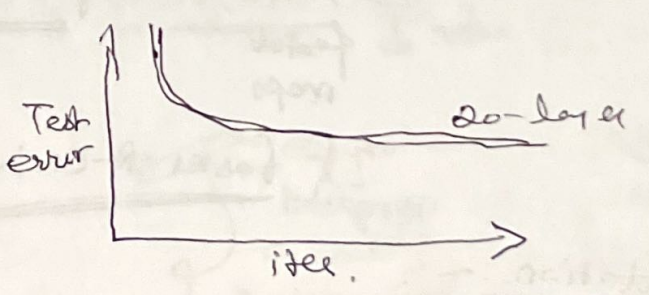
(*) If you never resize, then there is no way to get long-range connections. (global features)

(*) If you go higher in resolution, you are going to consider local features well.

⇒ Wrapping up Convnets:-

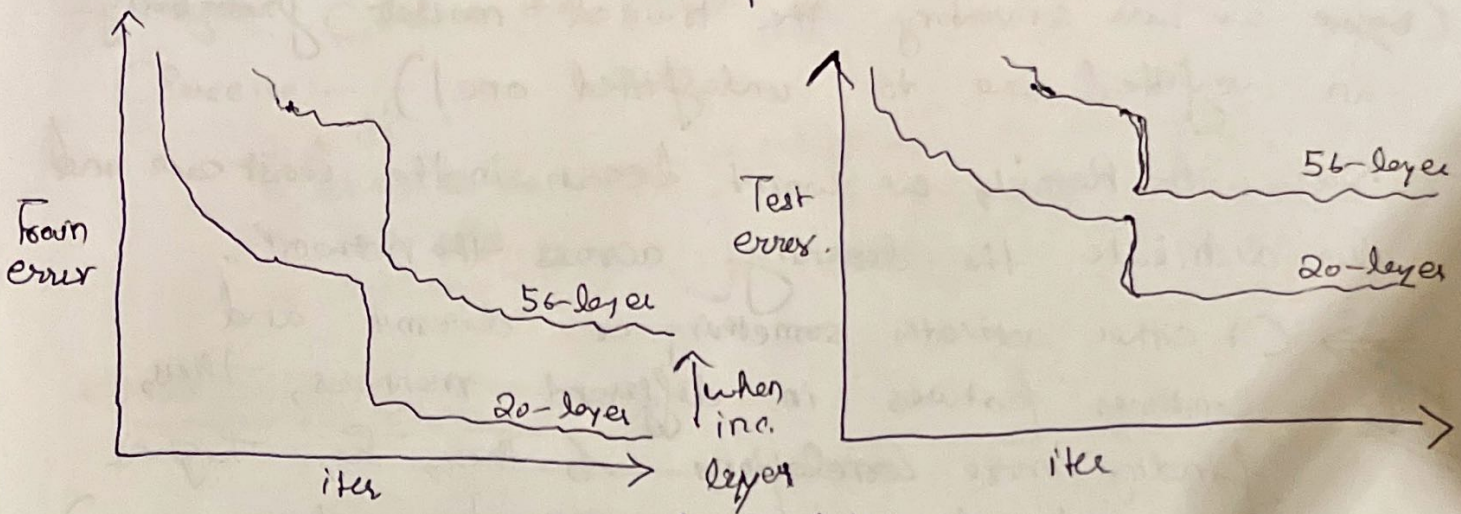
Residual N/ws:-





(*) What should happen if I train a deeper N/w?

→ **Train error** :- As this is much lower than test error, this model overfits. We can also see that test error plateaus out.

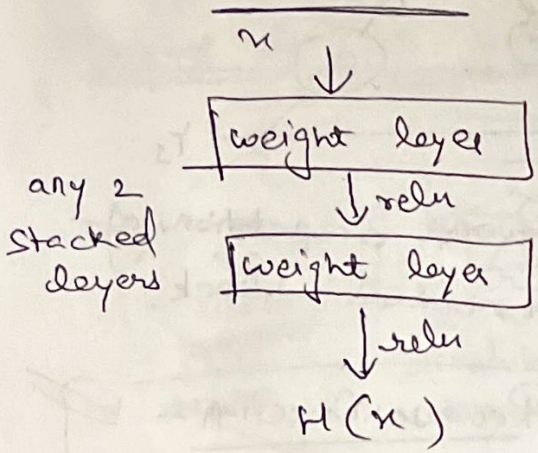


↑ when inc. layer training & test error both goes up (performs worse!).

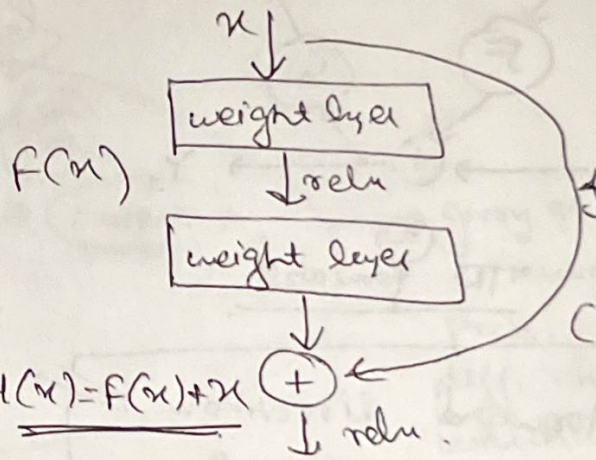
(*) **Why it goes up?**

(you should learn to do nothing in case it would go up because this shouldn't happen). (we need to inc. data along with layers)

Plain N/Ws.



Residual N/Ws



(it's easier to learn to o/p zero.)

(with skip connections basically)

easy to backprop as well, independent of other N/w structure.

(At + sign, gradient splits).

→ Now, with ResNets, when we increase the number of layers, the train error would atleast go down now.

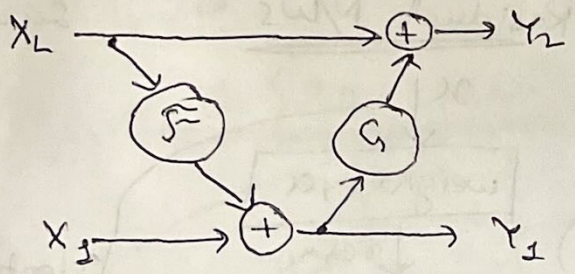
(Before we were converting the trained model from being an overfitted one to underfitted one!).

→ Now, with Resnets, we won't learn in the worst case and also distribute the learning across the network..

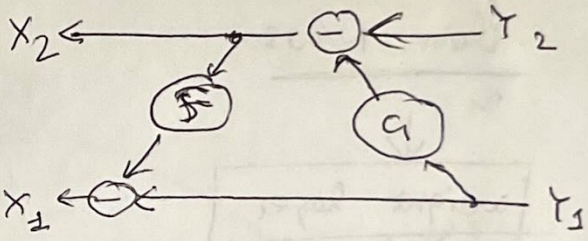
Imp ⇒ (it either activates something or doesn't and combines features in different manners, thus, finding more correlations & thus, has higher potential for the train error to go down now).

⇒ **Reversible ResNets:-**

The backprop goes from $o(n)$ to $o(1)$ if n were nodes in the path from start to end.



(a) The forward



(b) reverse computation of a residual block

forward direction:-

$$y_1 = x_1 + F(x_2)$$

$$y_2 = x_2 + G(y_1)$$

eqn for above flows.

Reverse Direction:-

$$x_2 = y_2 - G(y_1)$$

$$x_1 = y_1 - F(x_2)$$

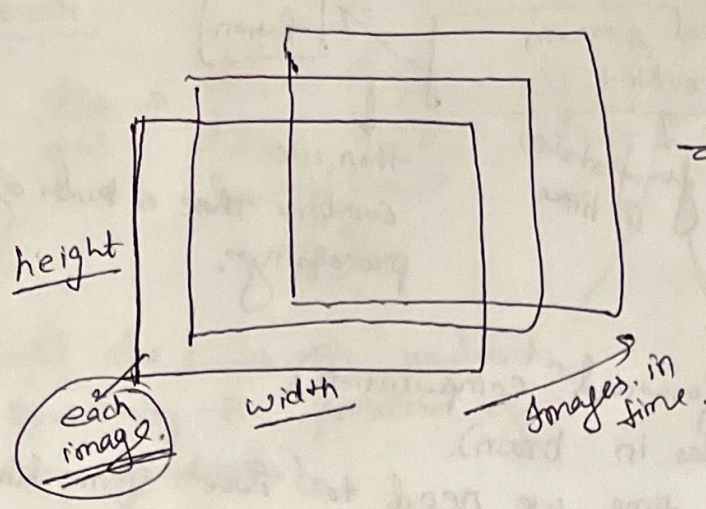
Video Recognition :-

→ How to derive insights from just the video or sequence of images? (That's the problem for the machines!)

(Perceive :-
1. goals
2. objects in the video.
3. sub-tasks.
4. emotions)

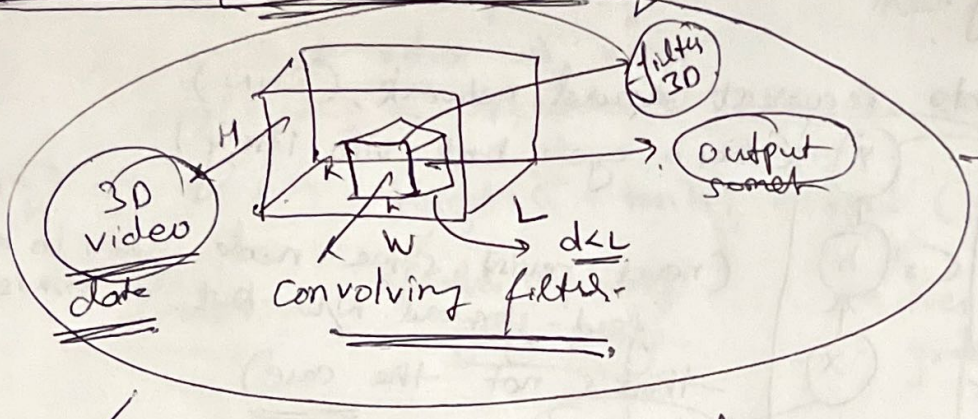
→ difficult to recognize this in temporal form from a video.

⇒ Representing video :-



⇒ frames played fast enough creates an illusion of motion!

Now, we do {3D convolutions}



→ giving o/Ps in terms of probs. of diff. classes which can change in time according to the frames under consideration.

↳ This learns spatio temporal features as 3D convolutions.

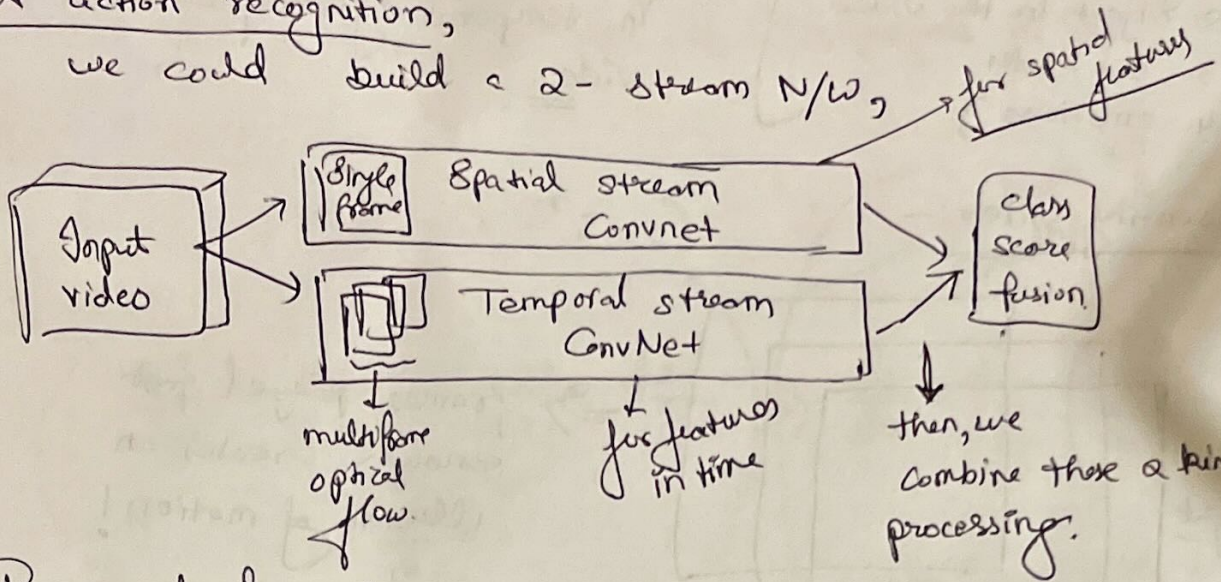
Motivation :- Separate visual pathways in nature.

→ Dorsal stream :- "where/how" → recognizes motion and located objects.

→ Ventral ("what") stream performs object recognition.
(Interconnection b/w these streams can help process these 2 together).

(*) CV is good at ventral stream part but not for dorsal stream part.

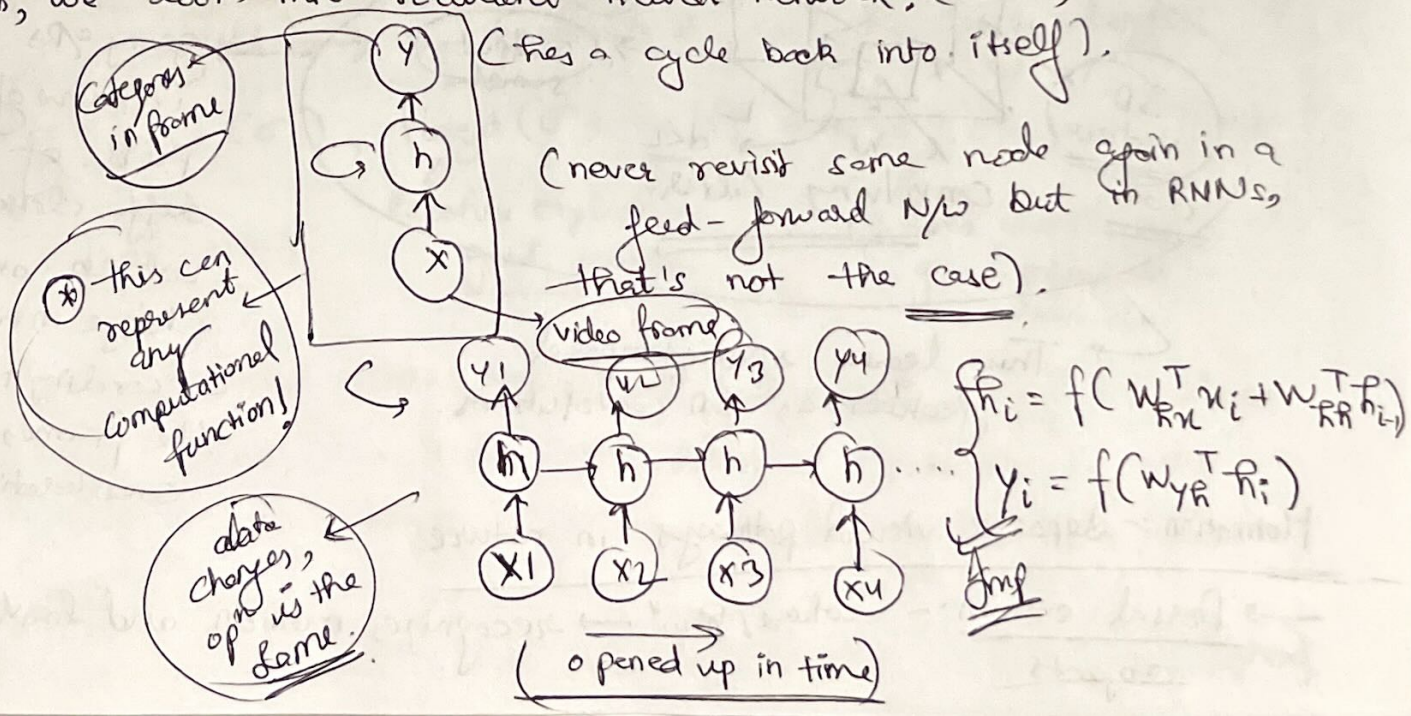
(*) For action recognition,
we could build a 2-stream N/W,



Recurrent Processing:-

(This is to be feed-forward computation).
(can't have cycles in brain).
but to incorporate time, we need to have some kind of looping.

Thus, we look into recurrent neural network. (RNN)



(has a cycle back into itself).

(never revisit same node again in a feed-forward N/W but in RNNs, that's not the case).

$$h_i = f(W_{hx}^T x_i + W_{hh}^T h_{i-1})$$
$$y_i = f(W_{yh}^T h_i)$$

fmf

→ h is basically a memory being propagated through time!

→ With RNNs, sequences are of variable length; they do some computation for ∞ no. of steps (possibly). If for videos, we need to model the sequence of frames, (temporal).

Backpropagation through time:-

we also do backprop for the hidden layers in time.

forward pass:-

$$z_i = f(w_x^T x_i + w^T z_{i-1})$$

Gradients:-

$$\frac{dL}{dw} = \underbrace{\frac{dL}{dz_{i+1}} \cdot \frac{dz_{i+1}}{dz_i}}_{\frac{dL}{dz_i}} \cdot \frac{dz_i}{dw}$$

→ gradients are multiplied (multiplied with itself) (for all h_i's).

$$\Rightarrow \frac{dL}{dw} = \frac{dL}{dz_T} \left(\prod_{j=i}^{T-1} \frac{dz_{j+1}}{dz_j} \right) \cdot \frac{dz_i}{dw}$$

→ very prominent in nature, if gradient get very small/very big.

↳ problem of vanishing / exploding gradient comes into picture.

→ we could also scale the gradients, (for countering the problem of exploding gradients), but still problem of vanishing gradients could come into picture!

(if gradient value is very small) ⟹ as recurrence can be very large.

LSTM ⇒ Input (σ), forget (σ + tanh), output (σ × tanh) gates.

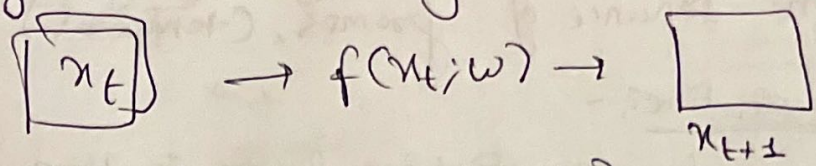
- Input: (simple activation)
- Forget: Resolves exploding grad. problem
- Output: Resolves vanishing grad. problem

→ Speed of action alters perceptual judgement, so, if we slow down frames in a video and train a CNN on it, the perception can change. (we could also model this slow motion).

→ We always see the video according to the action (perception).

⇒ we don't pay attention to other features.

future generation ⇒ Try to predict what's gonna happen next depending on the actions before.



Why can't we do this?

Minimize Euclidean distance ⇒

$$\min_w \sum_i \| f(x_t^i, w) - x_{t+1}^i \|_2^2$$

⇒ we might know what's gonna happen but how? (it's very hard problem!).

(in case of multiple modes (actions with some probs), it's hard to predict, averaging out actions doesn't help).

What a Machine could anticipate?

for example:

— A child stand outside a candy store, reaching for the door. What does the child predict will happen once they go inside?

→ The child does not predict the color of lollipops inside!

→ Rather, the child predicts their own sensations of what they will see, hear and taste.

this is what a M/C would predict → this is what we had want to predict — Ans

(we might need multiple predictions based on perception).

↳ (multiple futures possible).

(But one of them should match!)

If we did know this, then we could incorporate that in loss func^n

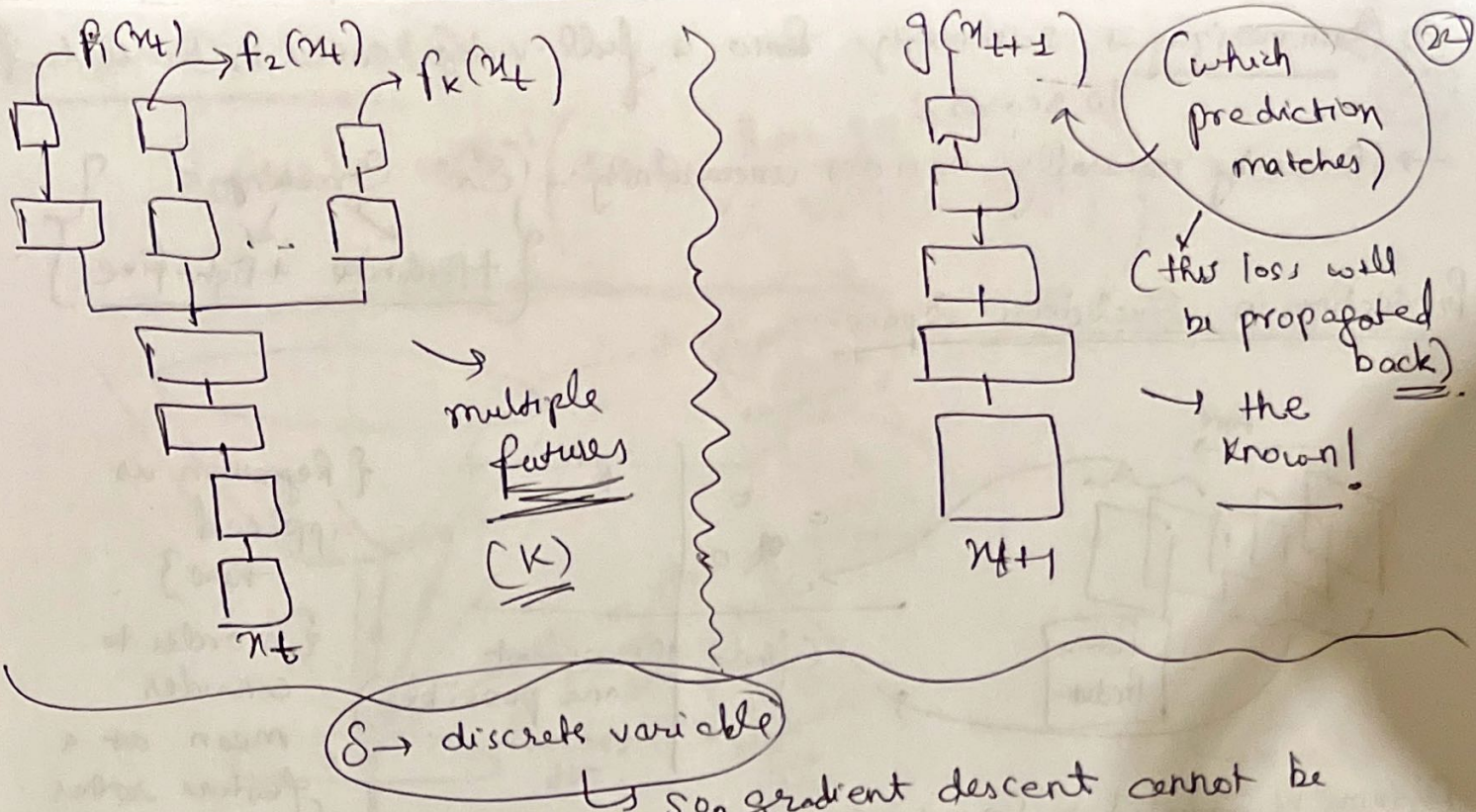
latent variable → latent variable

$$\min_{f, \delta} \sum_i \sum_k^K \delta_k^i \| f_k(x_t^i) - g(x_{t+1}^i) \|_2^2$$

K futures

$$\text{s.t. } \delta^i \in \{0,1\}^K \text{ and } \|\delta^i\|_1 = 1 \ \forall i$$

↑ allow only one residual term (latent)



$\delta \rightarrow$ discrete variable

↳ so, gradient descent cannot be directly applied here!

we use something new!

↳ Expectation-Maximization ⇒ (basically clustering neural nets).

—Guess the latent variable.

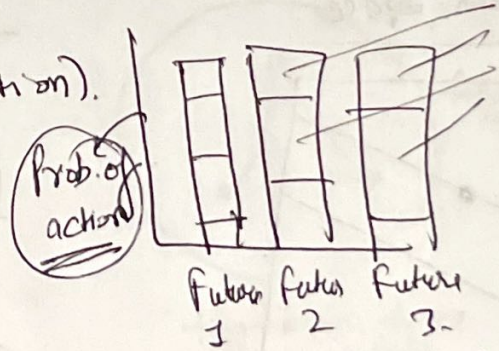
→ Try to solve the loss and back-propagate. (holding ϕ/δ at a time)

→ Repeat the above steps.

1. E-Step :- fill in missing variables and estimate latent variables
2. M-Step :- fit the model (with back-propagation in this case).
3. Repeat

(we compute probability of action).

(we need to decide time frame too).

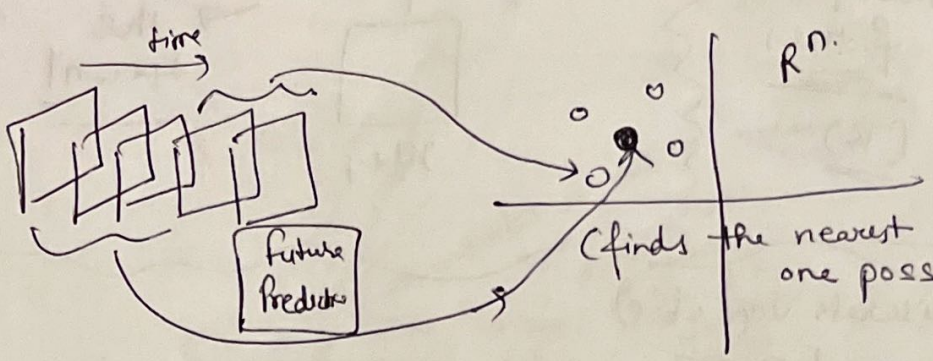


→ we could also turn this into a feedback solution-specs Ans

⇒ Summarizing → summarize how a full-video could be translated to scores.

→ Hierarchy naturally encodes uncertainty :- {Ex:- Greeting → Handshake, High-five}

Predicting in Euclidean Space :-



{Regression is applied here}

{in order to consider mean as a feature rather than data}

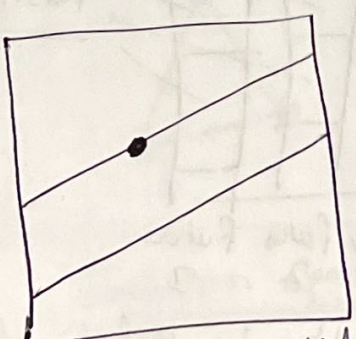
(we need to do something.)

Axioms of Euclidean Geometry :-

1. There is one and only one line segment between any two given points.
2. Any line segment can be extended continuously to a line.
3. There is one & only 1 circle with any given center and radius.
4. All right $\angle$s are congruent to one another.
5. Given any straight line and a point not on it, there exists one and only one straight line which passes through that point & never intersect the 1st line. (parallel lines).

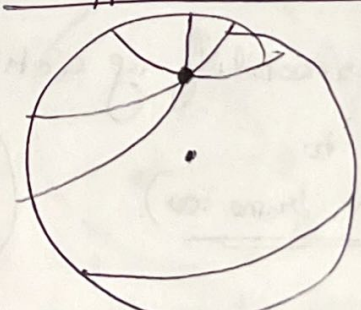
→ {infinitely many} → for Hyperbolic geometry

Euclidean Space



{Only one parallel line}

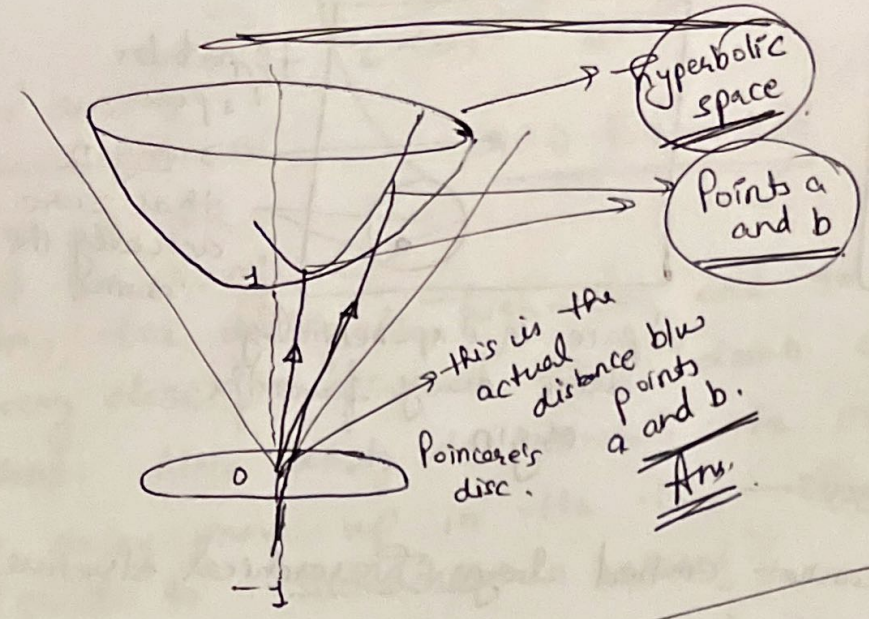
Hyperbolic Space



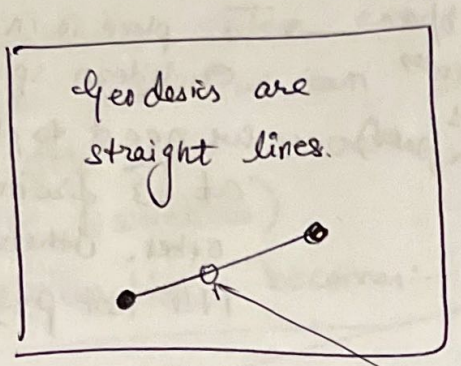
{Infinite parallel lines.}

Poincare Disk Model ⇒

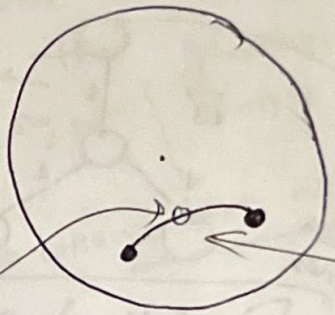
$$d_D(a,b) = \cosh^{-1}\left(1 + 2\cdot\frac{\|a-b\|^2}{(1-\|a\|^2)(1-\|b\|^2)}\right)$$



"Geodesics are basically shortest path lines b/w any 2 points in the space".



Euclidean space



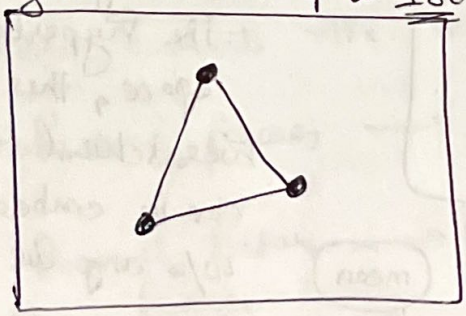
Hyperbolic space

Geodesics don't "look" straight.

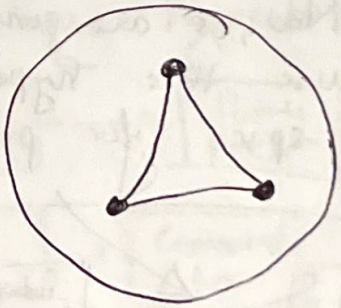
Mean points. (Half-way)

Carry mean pt in this space is actually closest point to the origin)

Angles must add up to $\pm 180°$.



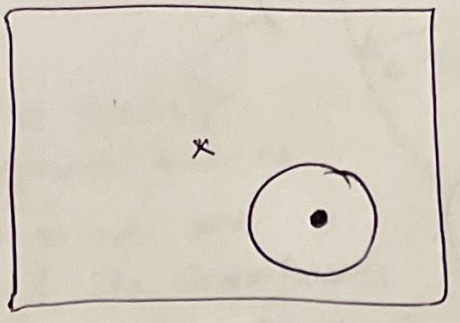
Euclidean space



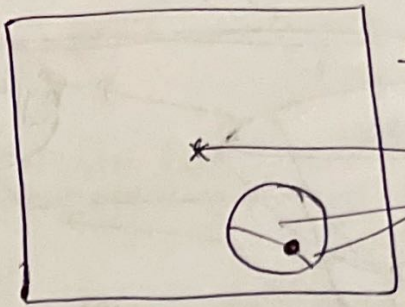
Angles add upto $< 180°$.

Hyperbolic space

→ All angles are $< 90°$ in a hyperbolic space - the shapes such as squares & rectangles aren't formed in the hyperbolic space.



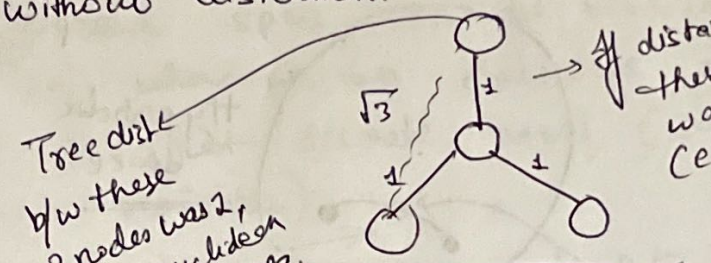
Euclidean space
Standard circle.



Space is exponentially dense away from the origin.

⇒ Distortion of the space:—

→ Euclidean space cannot embed large hierarchical structures without distortion.



Tree dist b/w these 2 nodes was 2, but in euclidean space, it becomes $\sqrt{3}$.

(If distance b/w their nodes was 1, (each part)

→ To place it in euclidean space, we need to place it at $\sqrt{3}$ from each other. Otherwise, it's not possible.

Distortion → $2/\sqrt{3}$.

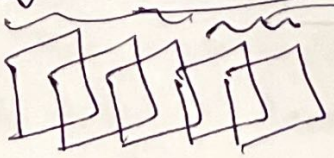
⇒ Hyperbolic space naturally embeds trees (because of the inherent density / distance in the hyperbolic space, this hierarchical structure can be embedded w/o any distortion).

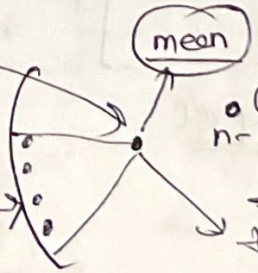
Imp

Why to use this?

Now, we are gonna use this hyperbolic space for predictions

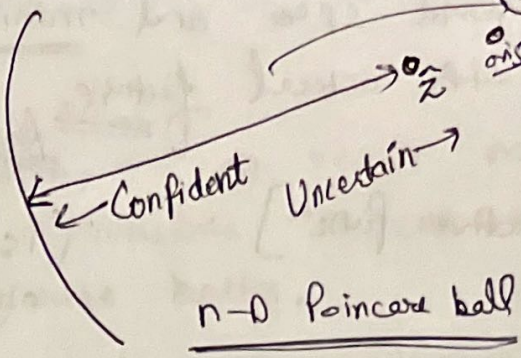
Hedging the bet →





the parent of all pts.

24



(*) As in hyperbolic space, as we move away from the origin, the density varies from point on disk and so, the tree hierarchy is ~~embedding~~ embedded in this space!

→ this density difference tells us that as we are around the circumference boundary, the density is high there and to predict a point out of very close pts, we need to be damn sure, thus, more confident. Since, density dec. towards the origin, pred^n becomes easy as we move up in the tree hierarchy. (similar to hierarchies in nature!).

→ hyperbolic space can be viewed as a continuous form of the tree where we can calculate the geodesics easily now. It's not discrete now! (So, we won't need the latent variables).

So, loss function becomes:- → func^n to convert that distance.

$$\min \sum_i \left[d_D^2(\hat{z}_i, z_i) + \log \sum_j \exp\left(-d_D^2(\hat{z}_i, z_j)\right) \right]$$

Obj func^n →

$d_D^2(\hat{z}_i, z_i)$: $\hat{z}_i$ → future prediction; z_i → actual future.

Neural nets converts the images in the hyperbolic space

(Both the models are trained on the same objective func^n).

feature frames → NNet → Future Pred^n → Pred^n

Actual future frames → NNet → Converted to hyperbolic space → Compared to compute the loss

{ The hierarchical structure of the tree is learned over time }

→ Maximize the distance of the prediction from any random point on the hyperbolic space and minimize the distance of the pred^n from the actual future. Ans.

[2nd part of the objective func^n]

[1st part of the objective func^n].

(x) Features are encoded in the hyperbolic space as well. & the scenes in the imgs are stored in the hyperbolic tree structure as well (and thus, as tree hierarchies which are learnt by the NNet).

The z's are actually vector representations of the frames/images and can be treated as an embedding for the training. (We sum over all the samples in our tree to find the loss from the objective function).

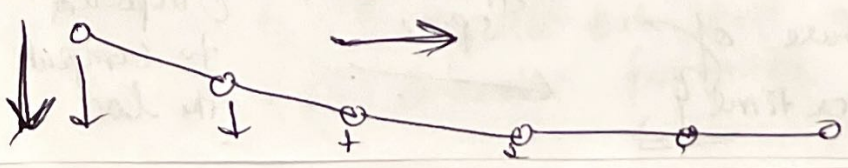
(basically we predict the mean of all the points corresponding to the frames in the video).

→ Here, we design an objective function!

→ We need to create a hyperbolic classifier to train the N/w for the use-case.

{Hierarchy provides us the notion of uncertainty:}

↳ because based on the input frames, we can forecast a particular category at a particular level in the hierarchical tree structure formed which is in turn based on the Poincare ball radius. As this radius goes low, that means that our prediction is coming closer to the ground truth in the hierarchical tree structure.



Poincare ball radius

Note:- As we would never reach ∞ where the leaf nodes in the hyperbolic space should be, so, the loss might never become zero.

→ In hyperbolic space, we can reach much higher accuracy in less number of dimensions as we can model the hierarchical structure better.

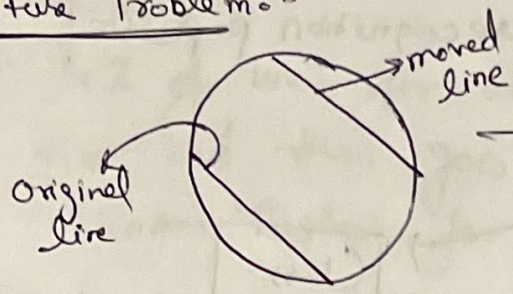
*How to replicate the motion stream?

Optical flow:-

flow field → assign a flow vector to each pixel. → to suggest the flow in a scene (set of frames).

Visualize:- flow magnitude as saturation, orientation as hue.

Aperture Problem:-



→ which way did the line move? (It's ambiguous as it can move these in many ways).

To find the optic flow, motion magnification can be used. Then, the optic flow map of the image/video can be used and trained with to make predictions of the flow. (Or learn to extract such frames).

optic flow + original frame ⇒ next frame

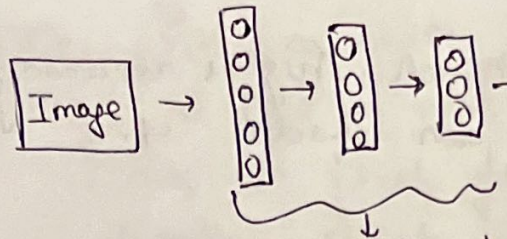
(This kind of prediction can be done quite easily)

Possible Qns:-

1. filter Convolutions mapping
2. hyperbolic space.
3. a little bit about backprop.
4. Activation functions.

Interpretability :-

→ what do deep nets internally learn?



what's learnt through these?

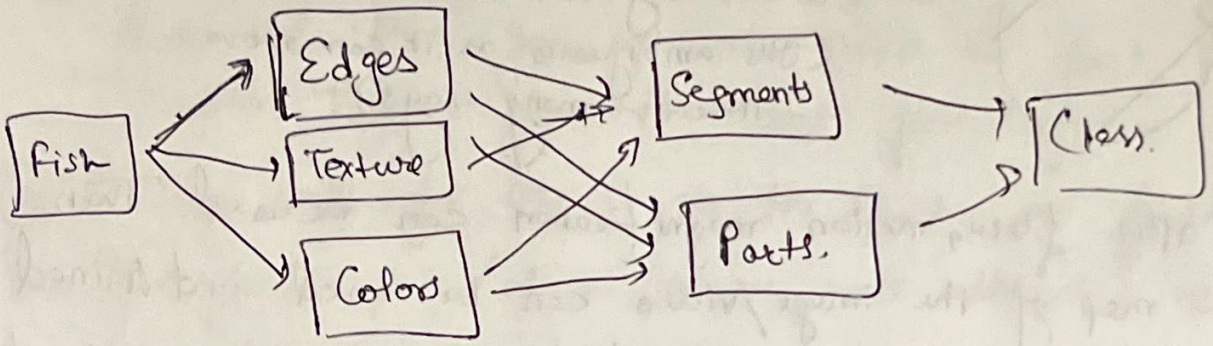
⇒ Generally, some set of neurons are associated with a class rather than just one, this makes memory more resilient! Such might be done in neural networks as well.

"Grandmother" neurons :-

⇒ Always one neuron is activated. (even one neuron might be activated for a particular class).

this shows that certain neurons might have some info. about particular label/type of activity!

⇒ CNNs learned the classical visual recognition pipeline!



→ Different neurons are activated differently, to detect particular features of an image. → "grandmother" neurons.

Distributed Representation learning :-

→ In this we distribute the learning for each of the neurons on different systems as each neuron is activated for some particular type of feature.

{Image patches activate some particular set of neurons most strongly}.

Object detectors Emergence in Deep Scene CNNs

→ As we go deeper in the model:-

- The percent of units for simple elements & colors is the most at first and it goes down constantly. (Bcoz at first, first few layers are responsible for detecting simple features).
- The % of units for texture materials is high at first and then goes down slowly.
- The % of units for regions/surfaces is the least at first and then goes up by a lot (at a higher rate).
- The % of units for objects & segments is very less at first and then goes up at quite a fast pace and becomes highest for the last few layers.
- The % of units for scenes is the least at first and then goes up (at a slower rate though) in the last few layers.

Thus, model becomes smarter at detecting coarse details while moving to the end of the model (going deeper).

Investigating a representation via similarity analysis:-

For all the image/subparts modeled in the model, we form a representational dissimilarity matrix to find how dissimilar these images are! Activation vectors are formed from the

images, that is used to find the dissimilarity b/w those 2 vectors & thus the images. (L1 norm).

$\|h_i - h_j\|$ → Neural activation vector

Imp

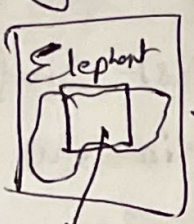
We can form the matrix for all the images/patterns learnt by a model and generate some kind of heat map to see what all features are being extracted & how many!

Saliency by Occlusion:-

→ CNN model → p(elephant) = 0.95

we mask some part of the image

When we mask some part of the image & find o/p prob. for that class drops, then that marked part was significant!



→ CNN model → p(elephant) = 0.75

masking some part of the image, this time head of elephant is marked.

By occluding some parts of the image, we can find which part of the image helps the classification!

[After doing the above and moving mask over the full image, we can generate a heat map and see how the change in score happens; higher the change in score, more imp. that subpart of the image and higher the heat, thus, that heat map gives us the imp parts of the image for that class.]

Saliency maps Reveal Bias ⇒

By occluding parts of the image by increasing the mask size, we could find the point till when we mask, the image is classified correctly. For example:- husky is classified as one till the whole husky is marked with some minor part left and snow. This happens because mostly with snow, huskies are associated. This reveals the bias in the model on the surrounding of the obj. under consideration. (imp)

Guided backpropagation ⇒

→ we compute the gradient of hidden unit w.r.t the input pixels, this gives us the changes w.r.t the input. Basically, if an input pixel is significant to classify the image correctly it's gonna have a +ve gradient while backprop. If not significant, then going to have -ve gradient.

→ Also, we only propagate the +ve gradient backwards & truncate -ve gradients to zero.

→ Intuition ⇒ find which regions cause the object, not the regions that do not cause it!

⇒ for maximally activated patches, we find o/p of the neurons i.e. only the most significant parts are highlighted after using guided backprop.

This helps us in interpretability by a lot!

For Gradient Ascent ⇒

This optimization problem is non-convex. and not guaranteed

to respect natural image priors.

This way we tell the model to give us the best representation of a class and it provides us with something from which we can see the biases associated and also the imp. features for that class (or atleast what the model thinks is!).

$$\max_{x} \; \mathcal{L}(x) - \lambda \|x\|_2^2$$ → w.r.t input

⇒ Pick a unit, perturb the activation to go up rather than go down in the obj. function.

ex:- we saturate a particular type of neuron for the images to find most imp. features for that class.

(we want to turn on rather than turn off some of the neurons that we need for a class).

(*) This is done on a trained model. In fact, all the interpretability is done on trained models.

Flows:- (Related to motion flow)

How you track an object in time?

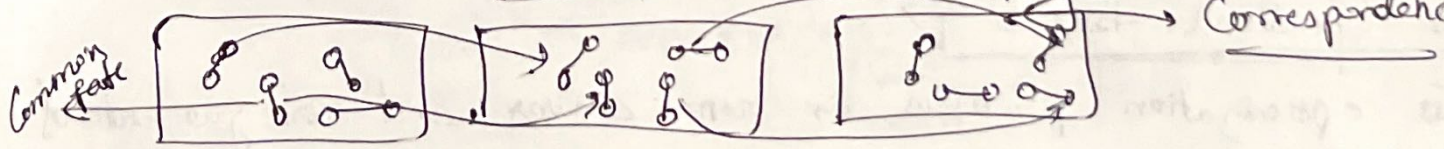
{Dynamics in video} → how to harness w/o supervision?

Common fate ⇒ what connected pieces are going to move together in a video?

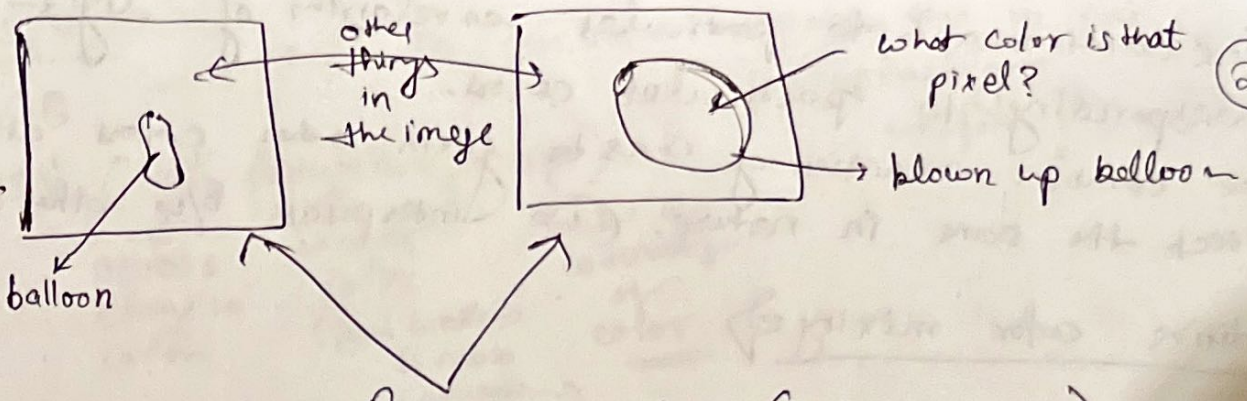
Correspondence ⇒ how an object/person is the same one across diff frames in the video.

We need to find these!

(Object tracking) → problem of recognition and consider above 2 things.



Ex:-



2 frames in a video (in time →)

How does m/c tell this? (about color)

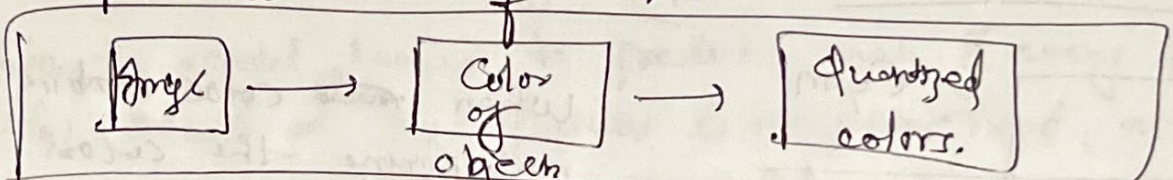
(1) It needs the track that the objects are the same in time (correspondence)

(2) How does the object change (move) across time (in terms of frames) (common fate).

→ We can learn to track an object in a frame across the full video and maybe color that object in the grey-scale video (moving racing car).

With some exceptions, we can find some coherence in colors for diff. objects in the frames of the video.

For achieving the objective of tracking, we need to find the temporal coherence of color.



{ Flow to find temporal coherence. }

About human eye:-

Retina has:-

(1) Rods → responsible for determining intensity of light, doesn't care about colors.

(2) Cones → sensitive to colors.

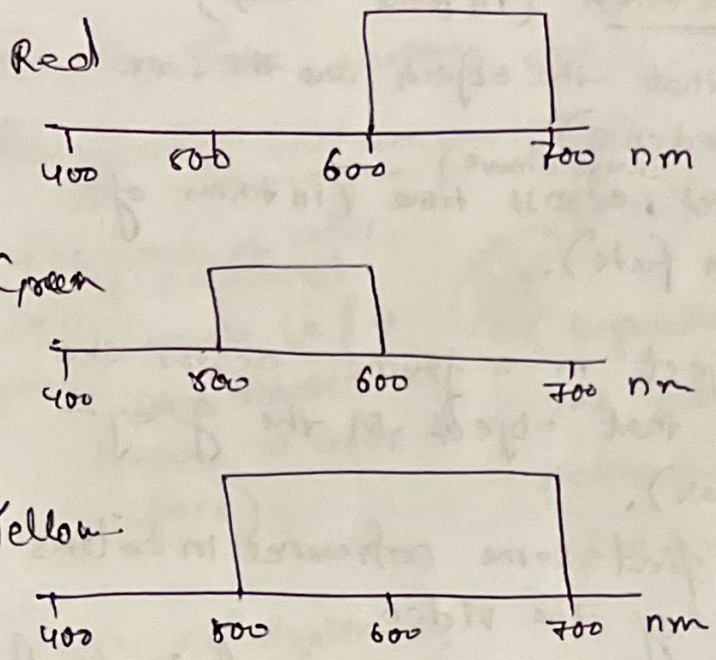
(3 diff. types of cones)

{ sensitive to :-
(1) Blue
(2) Green
(3) Red }

These cones react to particular wavelengths of light corresponding to particular colors.

→ we combine reaction of cons to form other colors and detect the same in nature. (we interpolate b/w other colors).

Additive color mixing ⇒

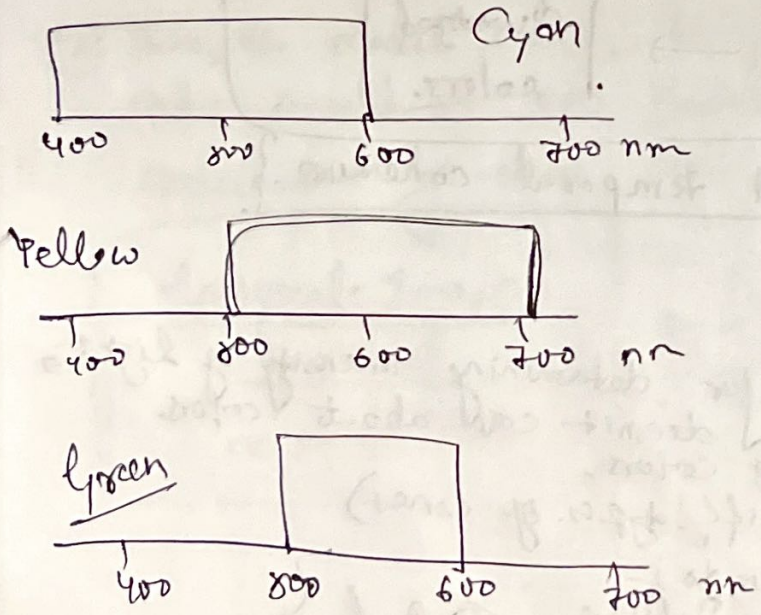


When colors combine by adding the color spectra.

Ex:- color displays that follow this mixing rule:- tiny display dots on a monitor screen, multiple projectors aimed at a screen.

R + G make Yellow!

Subtractive color mixing ⇒



When ~~reds~~ colors combine by multiplying the colors spectra. Examples that follow this mixing rule:- most photographic films, paints, cascaded optical filters, crayons & light reflecting off a diffuse surface.

Cyan & Yellow → Green!

Color spaces:-

(1) RGB → (in euclidean space, this is not perceptually uniform!)

(2) HSV (Hue, Saturation, Value)
- Hue ↓ across change in colors
- Saturation ↓ within color (from white → to max saturation)
- Value ↓ value of intensity of color.

So, we have another one:- **Lab color space**

Perceptually uniform:- Euclidean distance matches human perception of color similarity.

L → brightness of color (intensity)
a → } channels (x & y co-ordinates covering a color space)
b → }

(Imp) separates the color dimensions from the intensity dimension making it perceptually uniform, this is used a lot now!)

Image colorization:-

Training data → $\{x, y\}$ where $\{x \rightarrow$ grayscale image & $y \rightarrow$ colored image$\}$.

(*) Then, the model learns to predict each & every pixel color based on some features & neighbourhood and the correlation of those with the colors.

Grayscale Image
L channel.
$x \in R^{H \times W \times 1}$

→ f →

Color info:-
ab channels.
$y \in R^{H \times W \times 2}$

{This is what happens!}

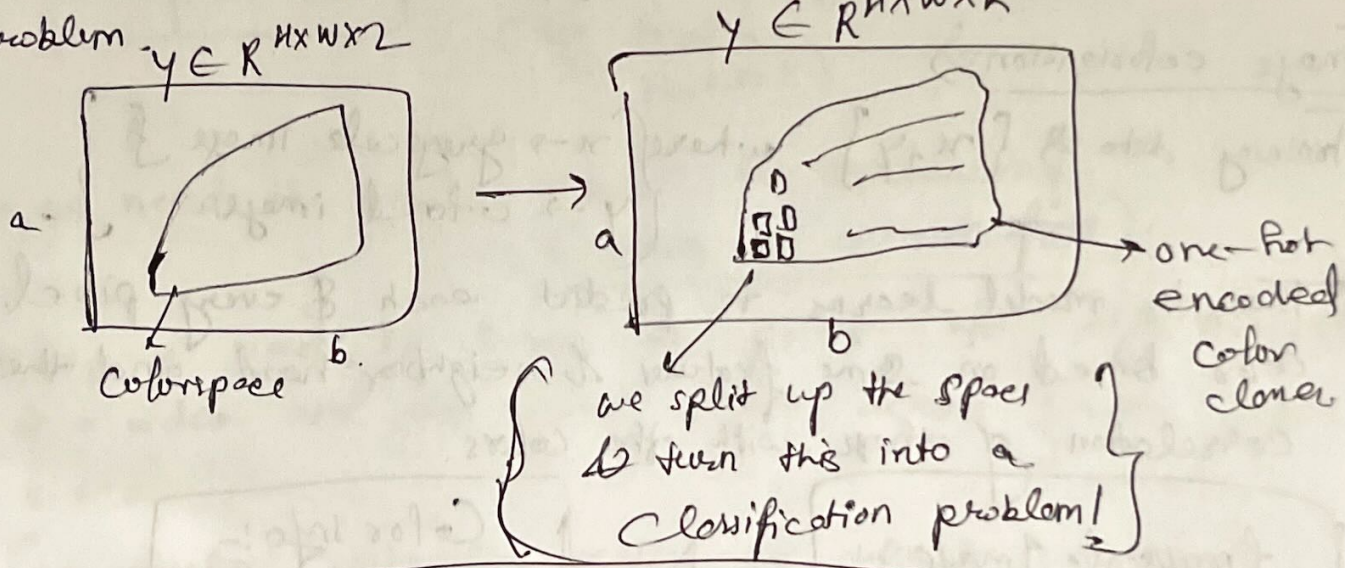
Choosing loss & representation

If we choose a loss fn such as:- $\mathcal{L}(f(x), y) = \|f(x) - y\|_2^2$

This is going to train the N/w s.t. it will predict the averaged color of an object/pixel based on the learnt associated color with an object or object + surroundings.

(taking avg doesn't help us here much!)

One-way to convert this regression problem would be to one-hot encode k discrete classes representing k diff colors in the lab color space. (ab channels). This converts the problem to a classification one and thus, moves away from calc. averages which is involved in a regression problem.



Loss func^n $\mathcal{L}(y, f_\theta(x)) = H(y, \text{softmax}(f_\theta(x)))$

This gives us more vivid colors sometimes!

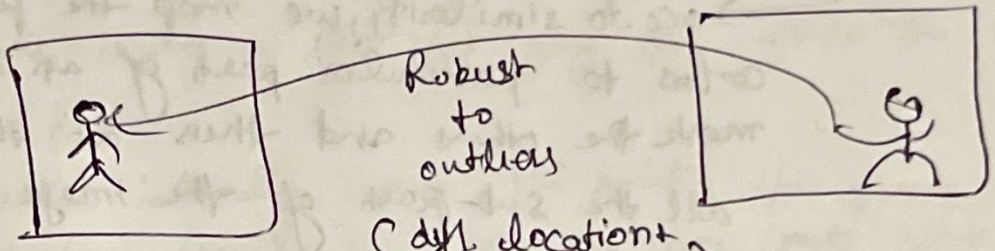
It might still give diff. colors but learns to detect object correctly in the images.

Instructive failures:-

Could lead to wrong coloring as well for ex: if we had lots of images in our training data with dogs with tongues out, then it could color a dog with no tongue out as still the tongue color. (wrong, right?)

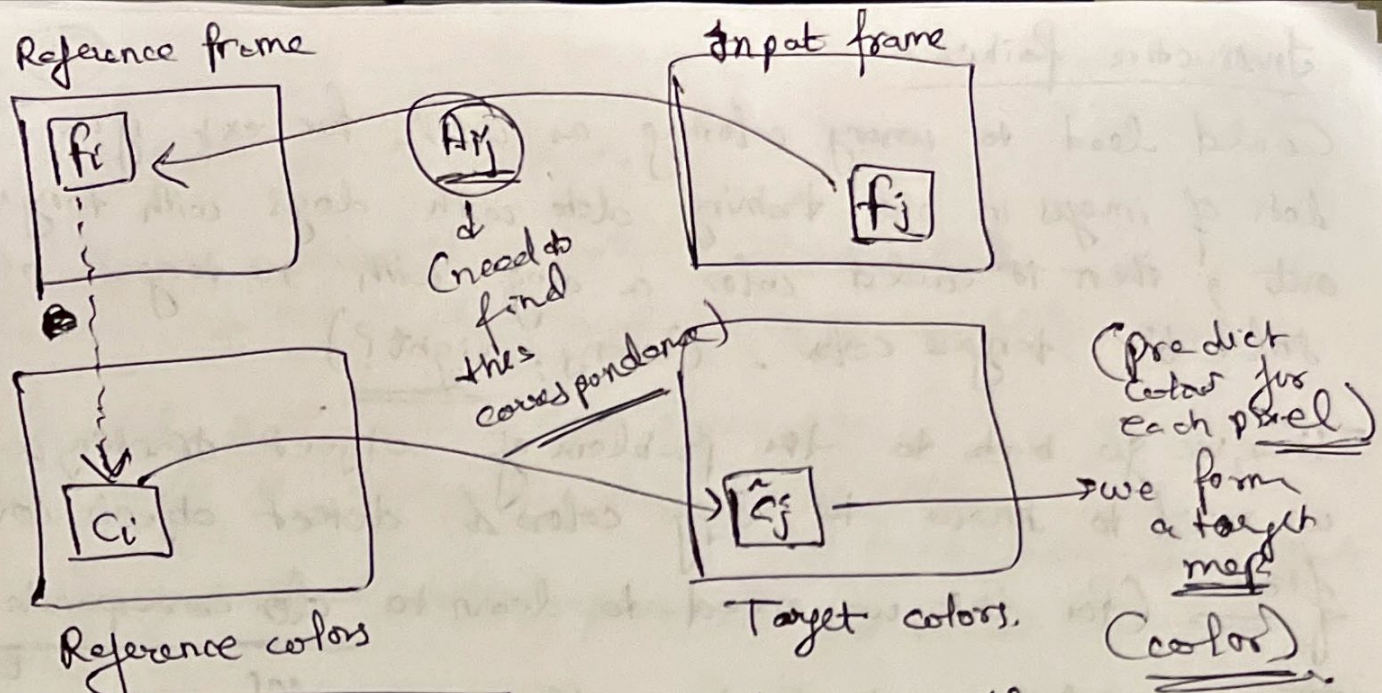
Thus, we go back to the problem of object tracking where we need to know to copy colors & detect objects correctly first. (for this, we need to learn to do correspondence).

→ This should be robust to outliers as well!
(Learning a policy how we should copy/paste colors).



Also, we need it to be robust to occlusions!
(driven by common fate)

→ So, we need to learn to point to diff. things in a video and for same object across frames in the video.
(we construct embedding vectors for different features in different frames and construct a similarity matrix (similarity A_{ij}) b/w patches of images.
(Also, we would convert these to color space and then mapping the same heat map for these).



$$\hat{C}_j = \sum_i A_{ij} C_i$$

(weighted sum of all colors in the input (ref) image and weighted by similarity matrix).

→ acc. to similarity, we map the particular colors to particular parts of an image & mask the others and then do this for all the sub-parts of the image acc to similarity.

where,

$$A_{ij} = \frac{\exp(f_i^T f_j)}{\sum_k \exp(f_k^T f_j)}$$

→ similar to softmax! probs.

Ans

objective func^n ⇒

$$\min_f \mathcal{L}\left(C_j, \sum_i A_{ij} C_i\right)$$

Ans

(Distance func^n should be able to copy paste colors, which is modeled by minimizing the loss func^n above) → (Applied to video colorization)

Ans

CGI : Computer generated Imagery

Lumière brothers : Inventors of motion picture, 1895.

When they invented it, they thought it to be completely useless.

↳ also invented first practical color camera, 1903.

Then sold it to a magician called Georges Méliès.

↳ discovered special effects

He accidently discovered double exposure and used it in his magic tricks.

In the colorization problem, rather than us trying to explicitly train the machine, we're tricking the machine to solve it.

We are creating a task that going to require the machine to solve a problem.

The model internally learns this pointer b/w frames to learn correspondence because it needs to copy / paste colors.

Now, we can reuse (repurpose the model) this pointer for different tasks.

Instead of propagating colors, we are now propagating the segmentation mask of an object. ⟶ to track objects!

We can also reuse the same equation used before :-

$$\hat{c}_j = \sum_i A_{ij} c_i \quad \text{where} \quad A_{ij} = \frac{\exp(f_i^T f_j)}{\sum_k \exp(f_k^T f_j)}$$

The meaning of c changes now ; it no longer denotes color, but now denotes segmentation mask.

It can also work on complex issues such as occlusion since the model is trained on massive amounts of data.

Neural networks, in general, specialize in one particular task but this can generalize to the different types of tracking tasks that are not related to colorization.

We can also do pose tracking.

---

* Visualizing Embeddings (PCA/dimensionality reduction)

Project embedding to 3 dimension and visualize as RGB.

Objects that are the same color are close in the embedding space. (calculate euclidean distance) The neural nets are not told about the objects but they are able to color an object with a different color — learns to distinguish / do grouping on its own. It makes mistakes.

Embeddings are very ↑ in dimension & might be confusing most of the times.

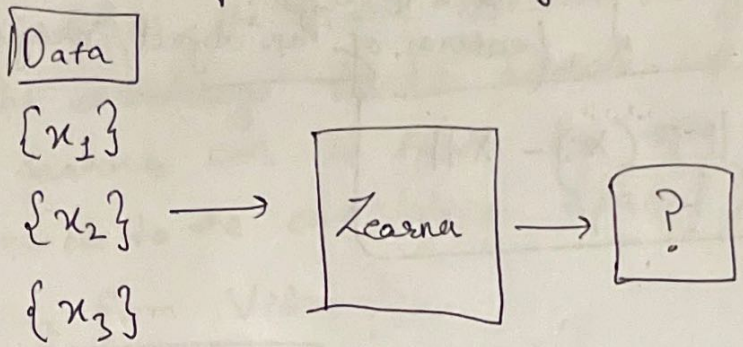
Self-supervised learning ⇒

How do you solve all diff. problems with a model?
(It's a really difficult task).

→ Rather than supervising the M/Cs, we want them to supervise themselves & learn recursively through the process.

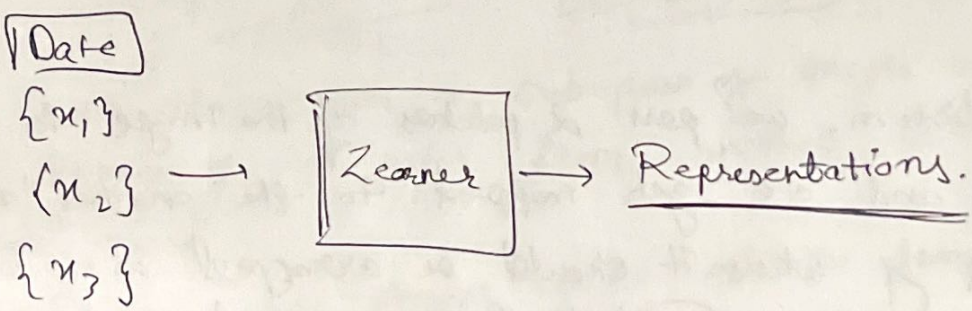
Prediction hypothesis:- Our mind when sees something, when redrawn again, we would extrapolate some information not always seen in reality.

Learning without examples:- This introduces noise such that we can learn without giving is what everything is!
(includes unsupervised learning and reinforcement learning).

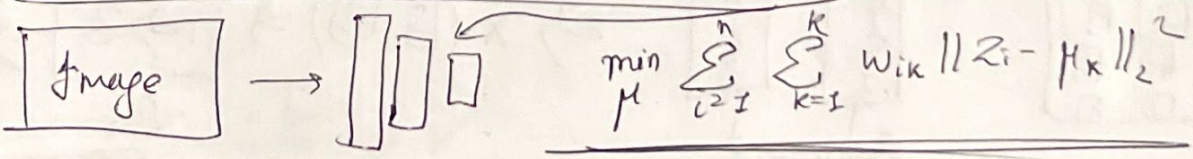


Unsupervised Representation learning:-

Learn features/representations of the world that help us learn patterns that can be applied elsewhere.

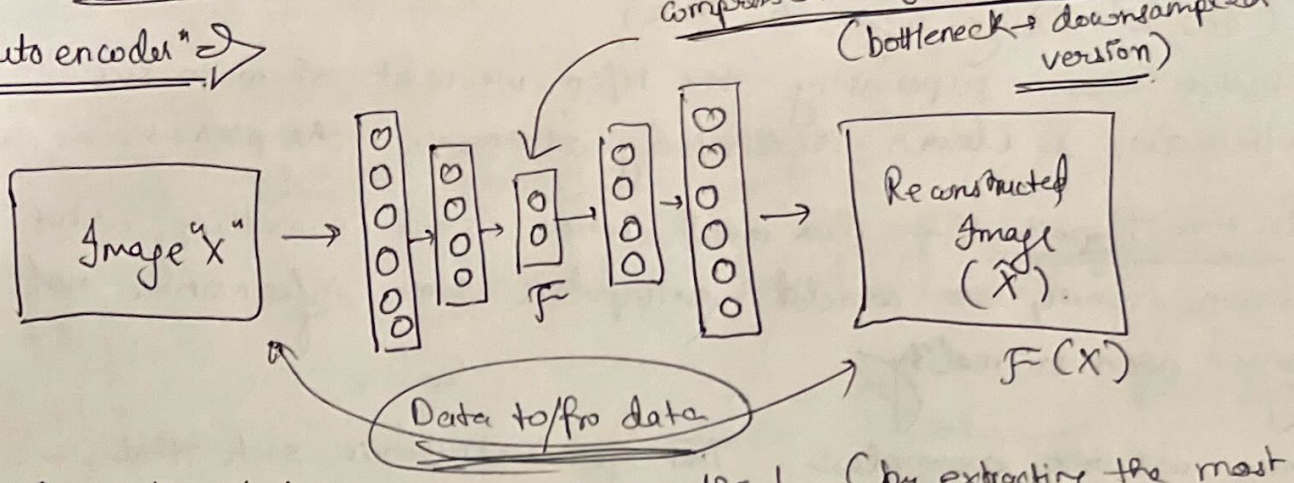


Unsupervised Learning by Clustering:- compressed image code (vector ≡)



s.t. $\sum_{k=1}^{k} w_{ik} = 1$, $w_{ik} \geq 0$ K-Means Objective

"Auto encoder" ⇒



⇒ Essentially, it learns to memorize this! (by extracting the most important features, reducing the dimensionality by removing the correlated features).

{ related to diff. features of an object/image. }

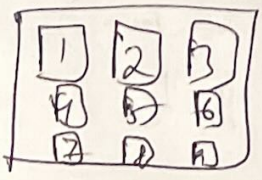
$$\arg\min_{F} E_x \| F(x) - x \|$$

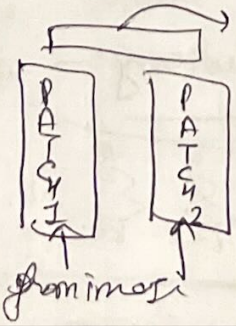
for which F the expected loss can be minimized.

for self-supervised, we extrapolate :-

⇒ Split the data in 2 parts and predict one from another! (this makes the model learns correlations in the image data).

for context prediction, we pass 2 patches of the image to the neural network and one gets mapped to the another as a classification task of where it should be arranged w.r.t one another.





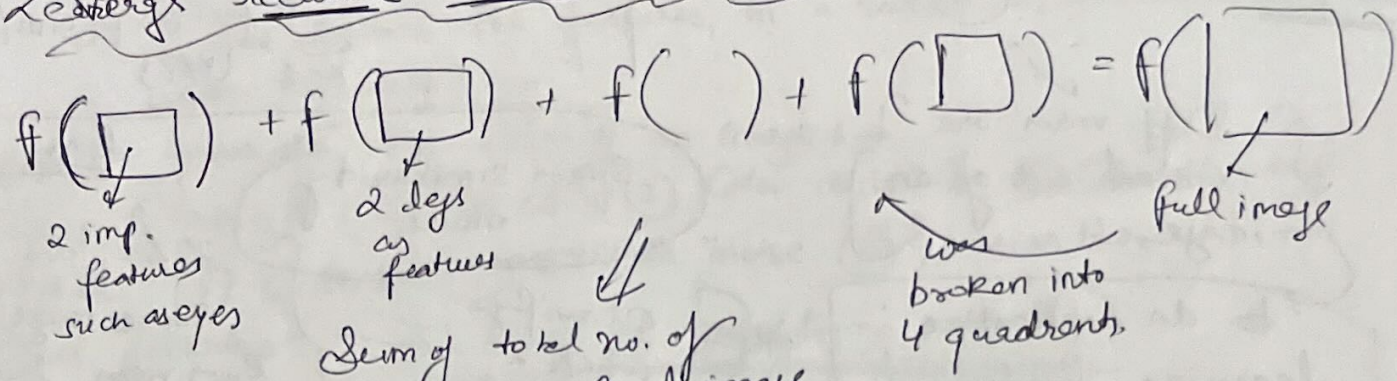
⇒ X ([5, 3]) ⇒ 4 ⇒ 3 → spatial arrangement

Ans.

* Even with random initializations of the patches, the self-supervised model learns the correlations b/w diff. patches/objects in different images on its own.

Learning by counting

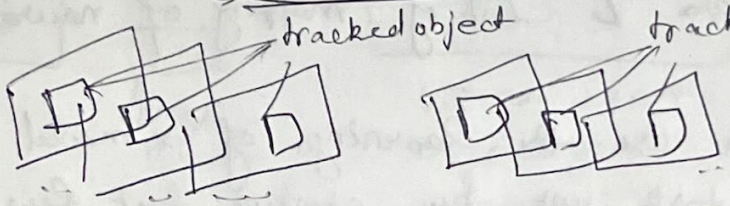
Learns recursive structure to images



Sum of total no. of eyes in the final image should be the same as the no. of eyes present in all the embeddings that's added up. This implicit count of features can be learnt by the model. (as close as possible to the embedding space!)

Learning from Video

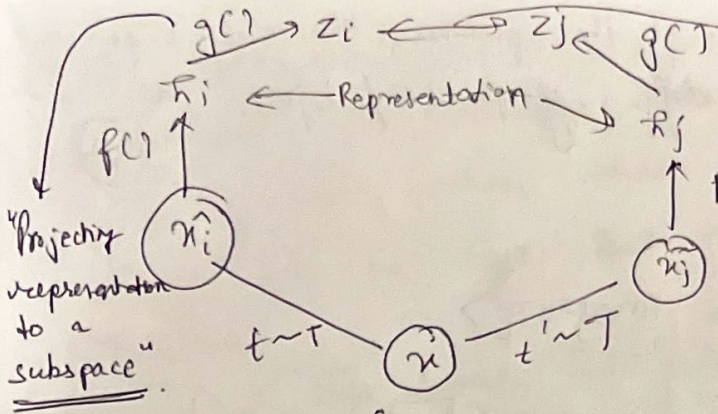
{Unsupervised tracking in video}



⇒ We rank the images or sub-part of images according to the distance of one patch from another in the embedding feature space.

{ Distance b/w similar images could be less, when compared with a random image, distance will be large. }

Contrastive Learning



$$l_{i,j} = -\log \frac{\exp(\text{sim}(z_i, z_j)/T)}{\sum_{k=1}^{2N} \mathbb{1}_{[k \neq i]} \exp(\text{sim}(z_i, z_k)/T)}$$

Ans

Sim → similarity metric

z_i & z_j are from same image

z_k → view from a random image

Diff. views are created out of an image

to do contrastive learning, finding contrasts & similarity b/w diff. views.

classify which 2 images belong to same image, while making that image as far as possible from a random image

Contrastive learning needs a lot more parameters to reach the accuracy for a supervised model solving the same task.
(A need lot of data & lot of mapping of neurons in NNet).

finetuning ⇒ (we can use the learnings of a model to use it for a different task just by changing last few layers by retraining the last few layers (might have to change the loss func^n) on the images for the specific application for which finetuning is being done).

Pretraining with lots of data → finetuning with "little" data → Test it for new application

(*) for contrastive learning, z vector stores the minimum required information for minimizing the loss func^n. Representation layer has a lot of data, when that is projected into a subspace, a lot of info. is lost but the most significant ones are preserved! (34)

(This learns the features in a much better manner than auto-encoders).

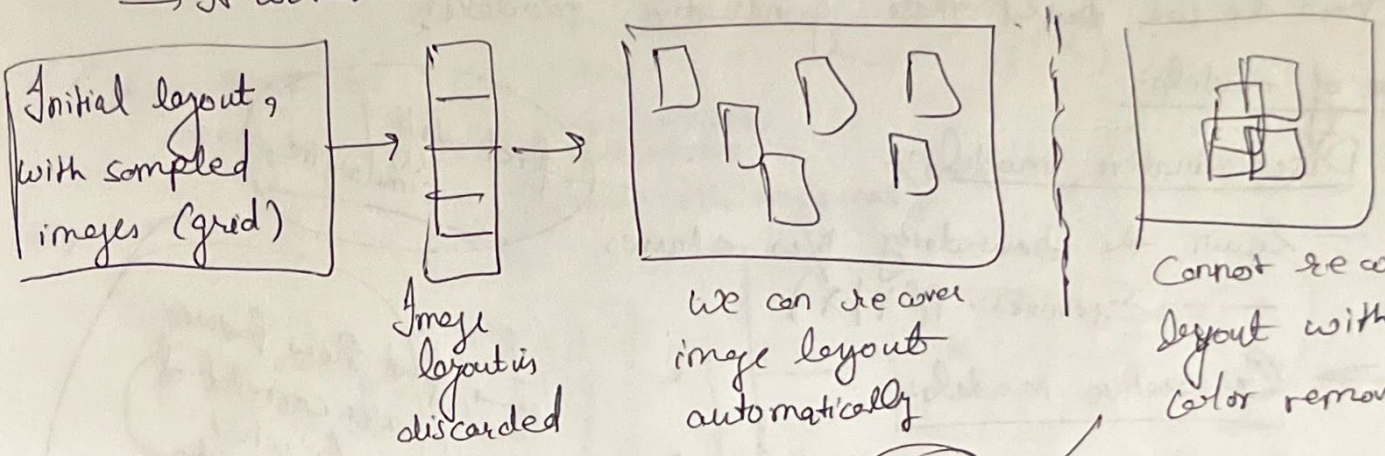
(*) Ways to generate the views of an image:—

(1) Crop & resize (2) flipped (3) Color distorted (4) Color distorted (jitter).
(5) Rotate (6) Cutout (7) Gaussian noise (8) Gaussian blur
(9) Sobel filtering. → (way of generalizing as well!)

Shortcuts:— {NNets are exceptional at finding patterns!}

Trained a model to predict the absolute (instead of relative) posn of a patch.

→ It works until color is removed!



Why?

→ "There is some extra info. encoded in the color channels that's very imp for this task".

Chromatic Aberration:—

This is inherent because of the optical lenses used, there are some inherent patterns in the image and also vignettes.

(8) These are implicitly encoded in the color channels of an image which might be helping in the use case before.

Ans → If the chromatic aberration is not centered in the image, then the image might be cropped, we can find whether an image was cropped.

In a photo taken with a camera. ⇒

Vignette

⇒ Due to camera lenses. Ans.

Synthesis:- (Also a supervised learning approach!)

→ Now we can change images by synthesizing diff. features/info. from diff. frames/images to form a counterfeit image.

(Basis)

So, how do we build these generative models!

Type of models:-

→ Discriminative models:-

— Learn the boundary b/w classes.

— Estimates:- $P(Y|X)$.

→ find diff./similarities

→ Generative models:-

— Learn the distribution of the data.

— Often you can sample from it.

— Estimates:- $P(Y, X)$.

→ find how features are correlated & what distribution they follow to generate new images.

BigGAN → example of the above model.

We were learning discriminative models till now.
Now comes models through which we do not just ~~learning~~ learn boundary but the distribution.
A lot of models of synthesis are generative models!

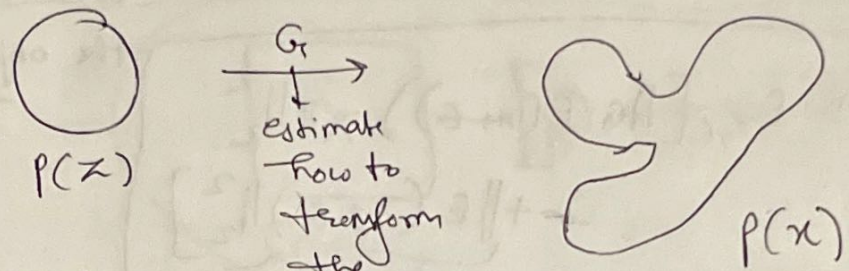
→ we can generate images from natural language (sentences entered).
(even learn reflection of objects).

→ DALE-2

→ but this can be used to create fake images now, it becomes an easy task.

⇒ Deep generative models are transformations of distributions ⇒

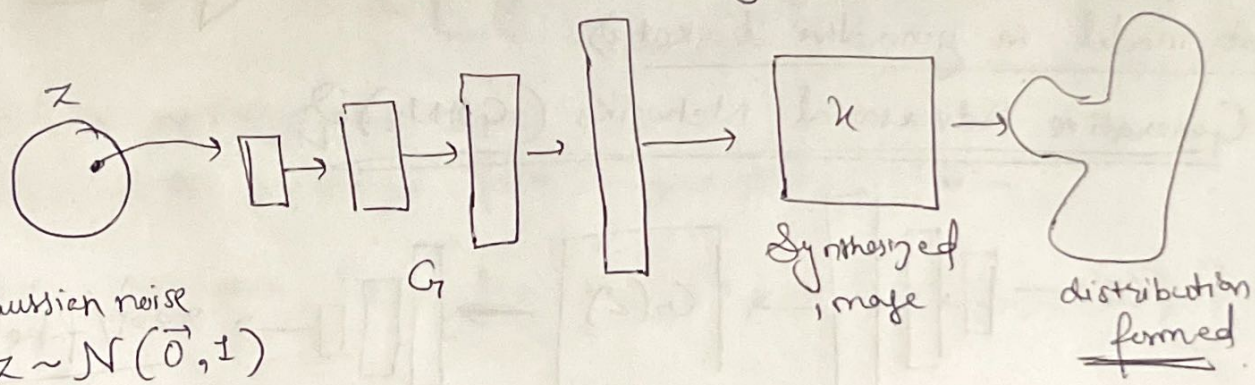
Prior distribution — Target distribution (can be very complex)



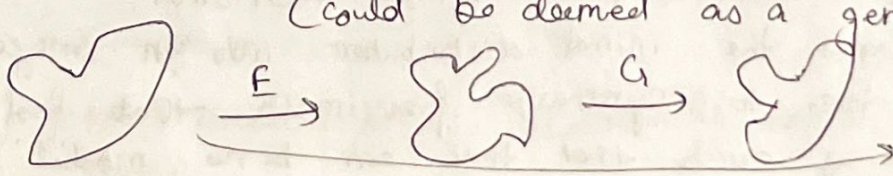
(done thru. a nnet)

(learn the mappings)

⇒



Classical auto encoder ⇒ we transform it in between from which the original distribution can be regenerated.
(could be deemed as a generative model)!

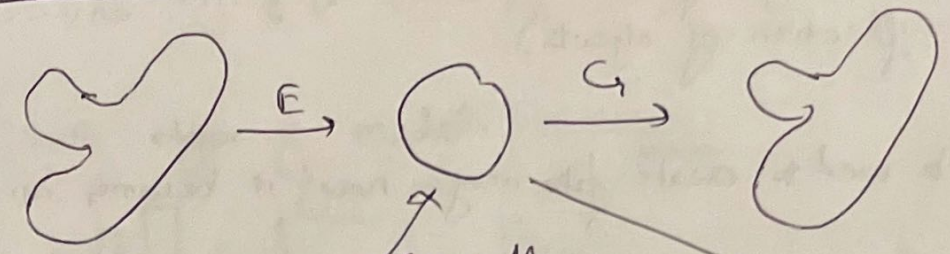


no way to parametrize this and this is intractable!

That's why not exactly a generative one!

$$\arg\min_{G,E} E_x\left[\|G(E(x)) - x\|_2^2\right]$$

Variational Autoencoders:



imposing a prior on the internal representation (it samples accordingly!)

(generally a gaussian)

Representation is generated acc to a prior distribution which is part of the objective function!

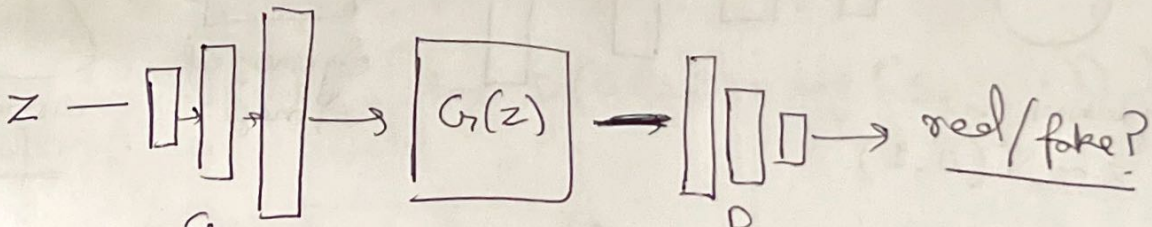
$$\arg\min_{G,E} E_{x,\epsilon}\left[\|G(E(x+\epsilon)) - x\|_2^2 + \|E(x+\epsilon)\|_2^2\right]$$

this is the prior information.

(kind of a generative model!)

Best model in generative basket:

Generative Adversarial Networks (GANs):



G (Generator)

D (Discriminator)

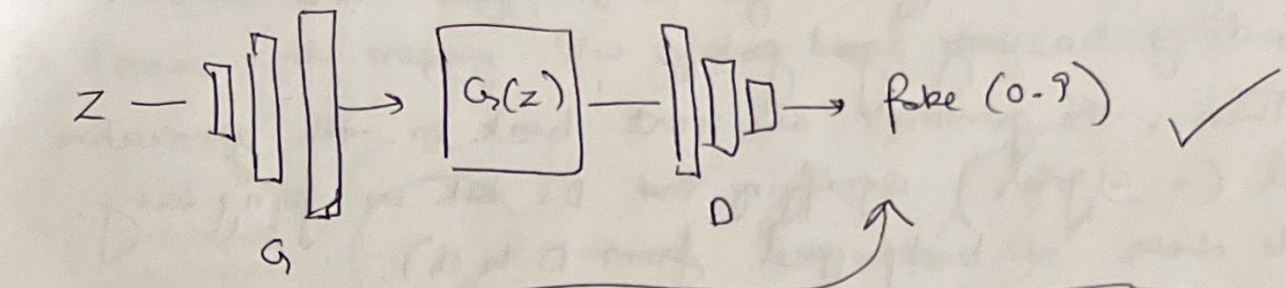
we have 2 diff. models competing against each other.
first model transforms the initial distribution into an image distribution (G tries to synthesize fake images that fool D)
(it tries to generate as such that there can be no model that

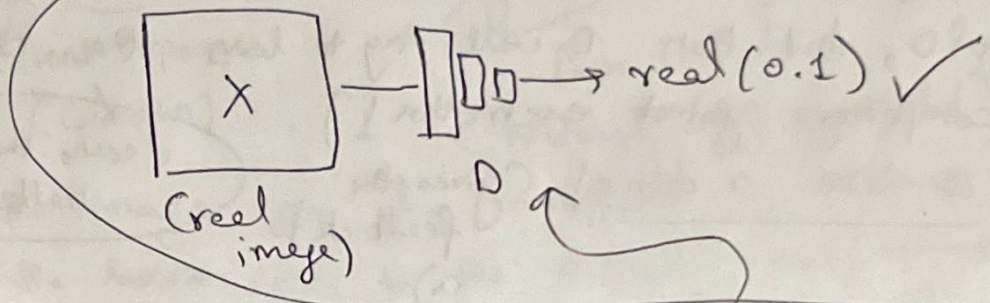
can tell us that G(z) came from a generator or is an actual image) → (generative prob.)

Then, D tries to identify the fakes! (classification prob)

G is trying to maximize the objective of generating real fakes while D is trying to identify those. G is adversary of D.

We map both type of models in one.



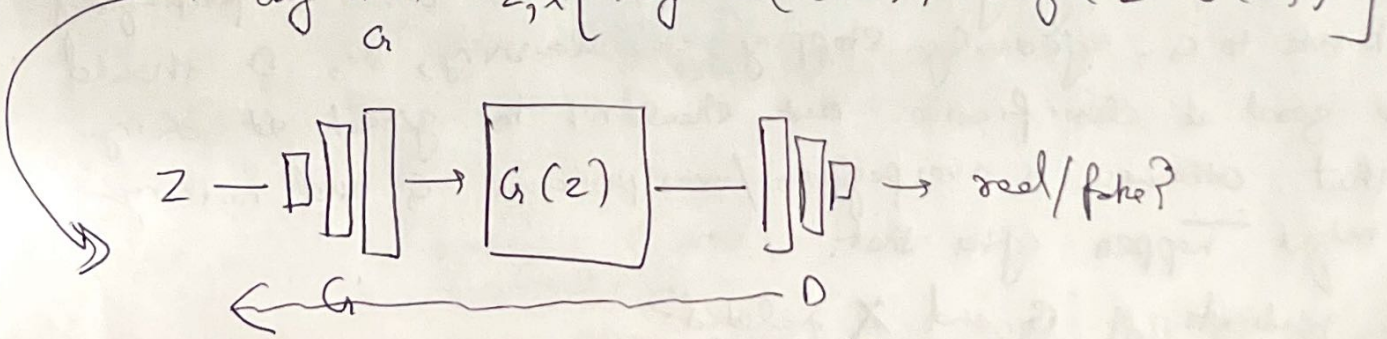


$$\arg\max_D E_{z,x}\left[\log D(G(z)) + \log(1 - D(x))\right]$$

G tries to minimize the same!

G tries to synthesize fake images that fool D:-

$$\arg\min_G E_{z,x}\left[\log D(G(z)) + \log(1 - D(x))\right]$$



Now, when used together. Essentially we are :-

G tries to synthesize fake images that fool the best D :-

$$\arg \min_G \max_D E_{z,x}\left[\log D(G(z)) + \log(1 - D(x)) \right]$$

Objective func^n

Initially, noise is entered to G to generate something, D has an easy job to compare / discriminate real / fake. D learns the boundary to classify image as real/fake and once it starts doing a very good job,

Then, the gradients are sent back to the generative model (a signal) signifying that D's task is going easy. (whole chain is backpropped from D to G).

Then, G tries harder to generate more real images so, that it can fool D, but then D will try to learn the boundary again. (competing against each other!)

D is like a critic! (gives the feedback to G).

learns the features better

When this is done for a long time, G will start generating more real images while D also gets better at classifying. It might go around forever as it could never converge. we need to balance G and D such that its descent enough. (That's why hard to train)

(X) If D does really good then no gradients will be propagated back to G, effectively stopping the learning, so, D should be good at classification but shouldn't be great at doing that. otherwise, it overperforms / overpowers G and nothing might happen after that.

Are gradients of G and X related?

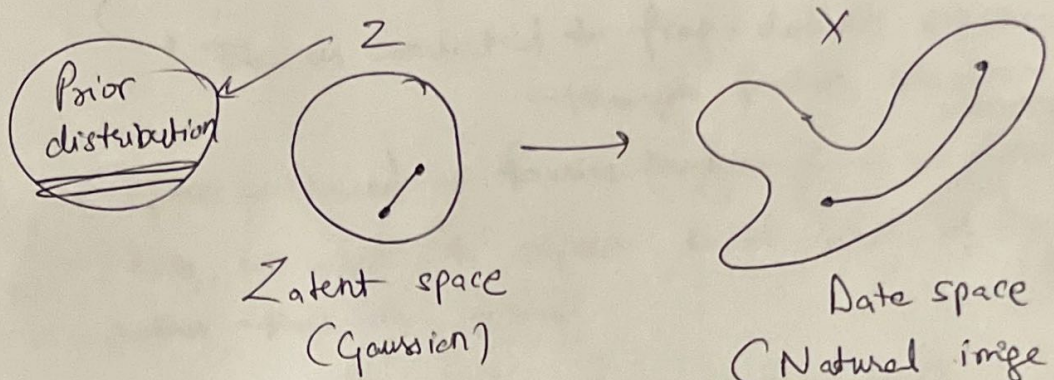
→ They are but not directly! This is primarily because G

never gets to see what it is actually generating, it just knows that what it generated was real/fake (probs. of it being real/fake) and the loss is propagated back from D to G. (37)

(**) G only learns through the gradients of D!
(indirect signal).

Works so well ~~becomes~~ because D tells what boundaries for classes are and that gets propagated to G. G then considers that learning. (x → Training image)

It learns the mapping b/w gaussian and image distribution!



The views of the image are also manifested in the mapping learnt from the latent space to the actual image representation.

(Angle of the camera → this correlation is also learnt)
(Disentangles the camera viewpoint from the features in an image)
(In space, one dimension is direction of view point) ⇐ correction found!
(finding new basis in gaussian space)

Different things can also be learnt:-

→ size of the object
→ color of the object
→ viewpts. of the object

(Imp) StyleGANs

BigGAN → is a massive GAN!

→ How do we know that GAN hasn't memorized all the images?
→ Its replicated in diff. scenarios to suggest that it's not all

memorization but it is also able to extrapolate.

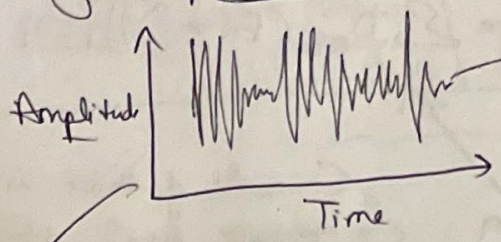
Mode collapse ⇒ From obj. func^n for GAN, we know that we sample from z, its not bound to cover the whole space and natural image x is very indirectly put into G. (through gradients) G just focuses on generating something that helps it trick D. Sometimes, some clones might collapse altogether. Thus, bias can come into GANs. This is a problem with GANs.

Vision and Sound

→ Interaction b/w modalities is quite important! (How do we integrate these together).

for humans → We believe our eyes more than other senses. (bcoz. visual system is faster).

⇒ Waveform:- pressure variation over time.



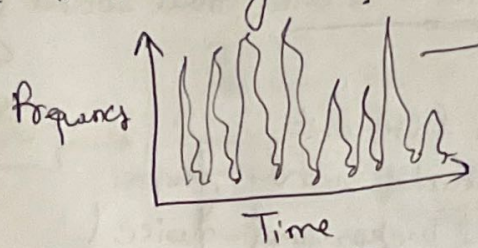
→ silence:- no variation in air pressure.

(in CV, we work on spectrogram images).

↳ This is converted to freq-domain representation through fourier transform.

(often windowed → fourier transform of a window).

↳ bcoz. we want to capture local details in the frequencies rather than the global!



→ colored graph.

→ our hearing ability

[20Hz – 20kHz]

⇒ for different noises, we have diff. kind of spectrograms.

{sometimes images can be hidden in fourier transforms of sound}

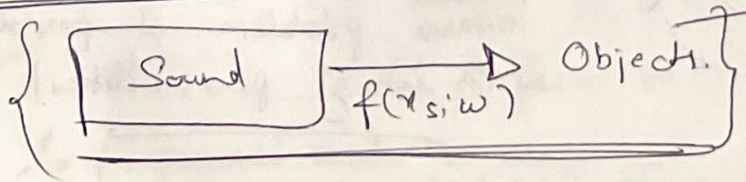
① Diff. b/w sound & light ⇒

[wavelength of light < wavelength of sound]

↳ concept of diffraction, bcoz. of this sound diffracts a lot more that light (vision)

(*) Light doesn't diffract much. (very little)

Problem with sound ⇒

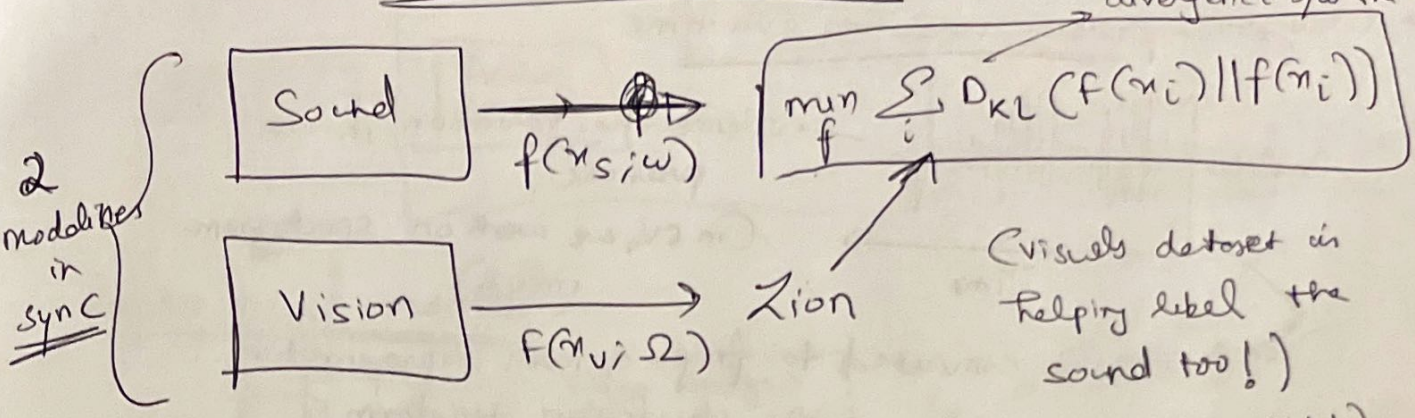


→ can detect them through waveforms.

Neuroplasticity ⇒ When a person loses sight, their visual cortex adapts for sound, smell & touch! Takes about six months.

{2 weeks for ~~this~~}

There is some "Natural Synchronization" b/w these 2 modalities:-



→ we could label many unlabeled videos. (through SoundNet).

(helps in classification)

⇒ (finding correlations b/w vision & sound).

→ help devices ~~to~~ be environment-aware. from sound (as power consumption for sound is way less).

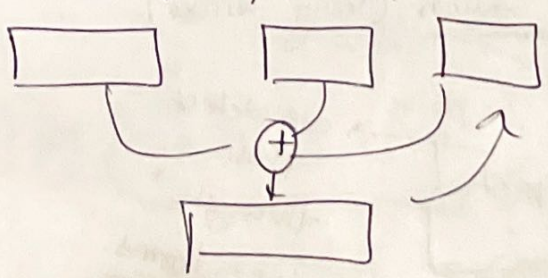
→ Cocktail-Party Problem ⇒ can filter out particular sounds from all background noise!

⊛ How can M/c do this?

→ Audiovisual Sprowdiy ⇒ Video + mixture of sounds

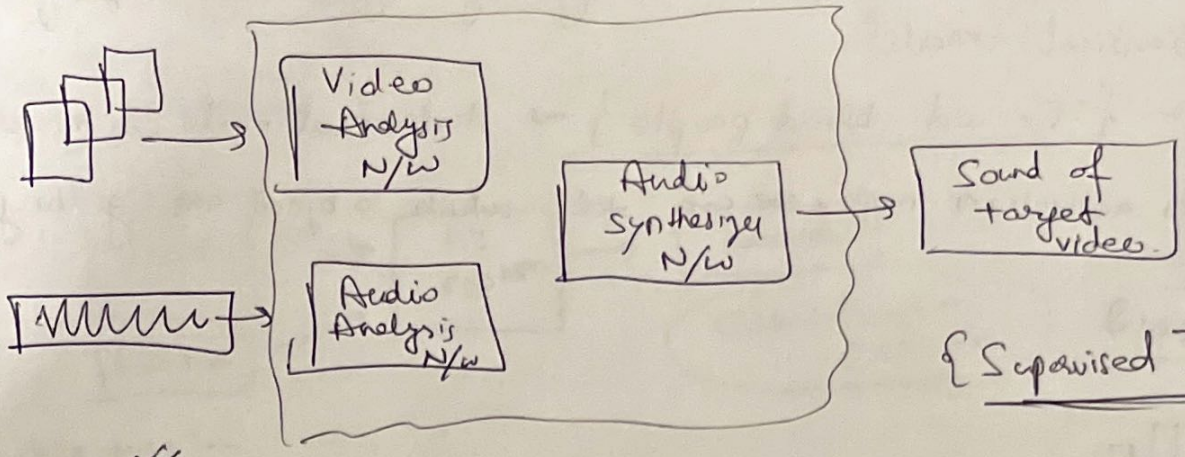
{find which part of sounds come from which objects/part in the video.

→ How do we recover originals in a mixed waveform of waveforms of various objects/parts?



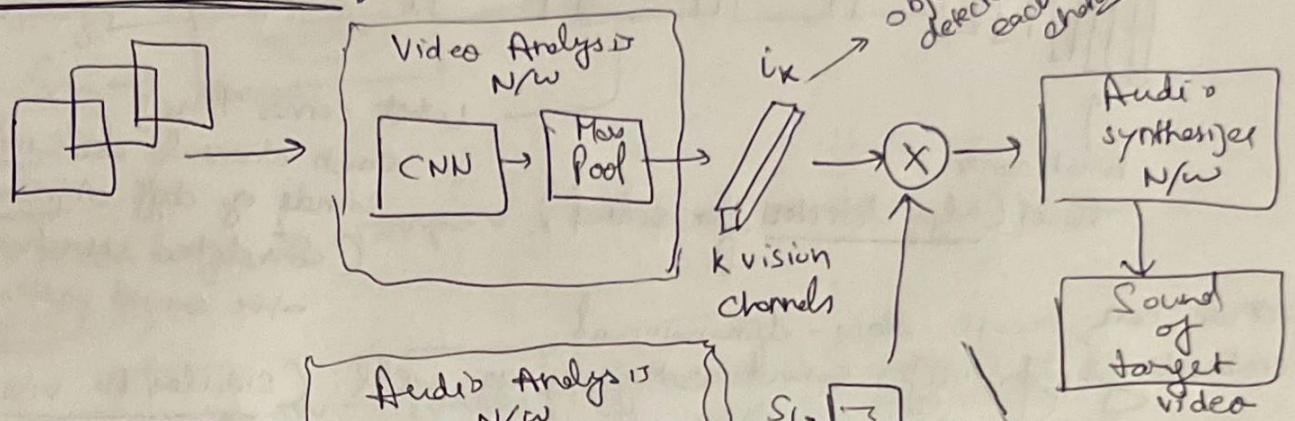
from just audio, it's ill-posed problem as basically it's an inverse problem of permutation with lots of possibilities!

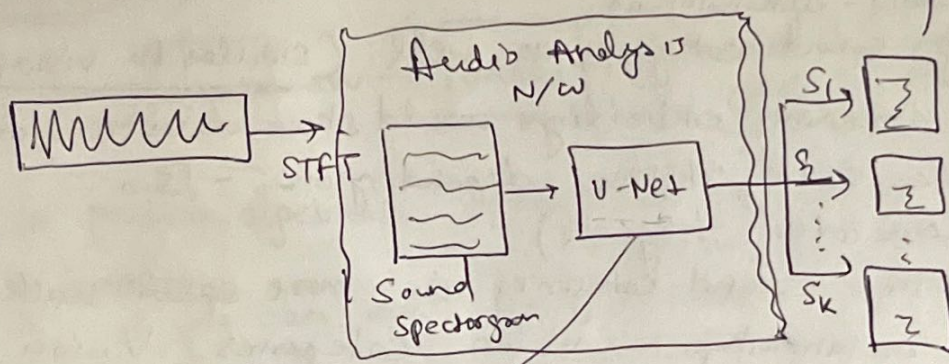
→ Vision can help!



{Supervised Task}

Audio-visual model





vision can select which sound channel to select out of K spectrograms

gives possibly sub-part of the spectrograms.

{k vision channels → (gives weights to k audio channels so, as to select some from them).}

X → $\sum_k i_k \cdot S_k$ = output

from this, loss is computed.

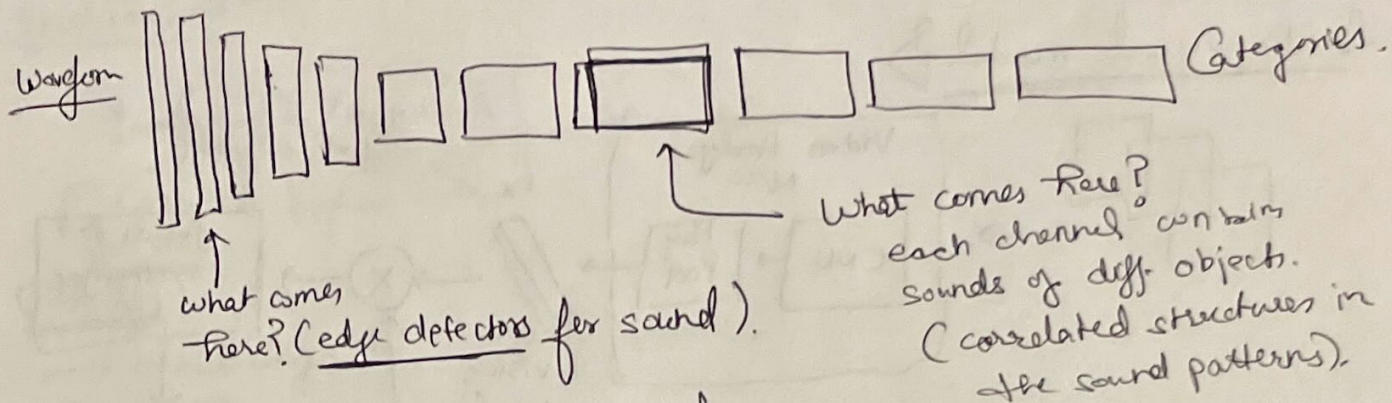
to propagated back to change the weights of all the models & weights applied to spectrograms.

(*) If we occlude an object from the video, then the associated sound shouldn't be mapped to anything after learning from the audiovisual model.

(*) Side note:- {To aid blind people} → Audiovisual mode can be used.}

→ Through activation maps, we can tell which objects are getting mapped.

SoundNet ⇒



⇒ we can create low-dimensional embeddings for diff. sound categories as well (similar to vision)
(2-dimensional embeddings would show distances b/w the sound categories depending on how correlated they are).

(*) These relationships in sound categories is more ~~spiatte~~ scattered in comparison to relationships in vision categories. Vision categories are more or less clustered together in comparison to sound categories (in the embedding space!).

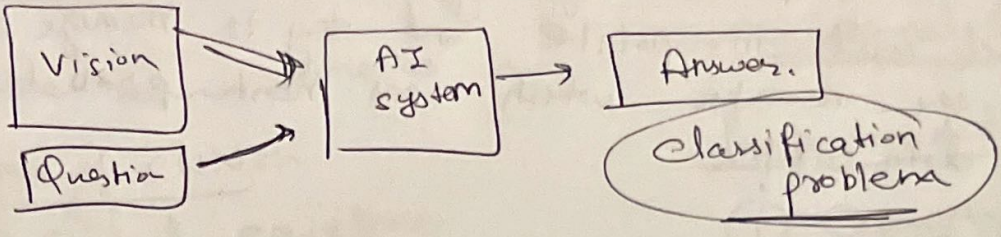
Denoising ⇒ How to suppress background noise?

Natural structure of speech ⇒

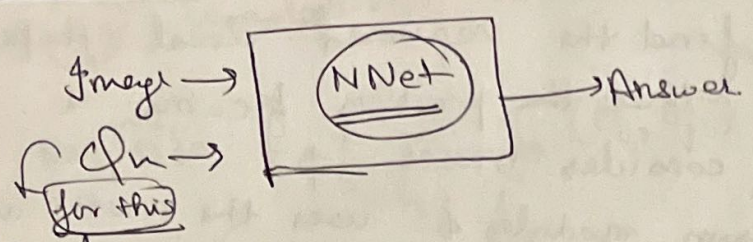
Waveforms have pauses (silent intervals), when no-one is talking. This can be used to estimate the distribution of noise.
If we can detect the silent intervals, we can estimate the noise distribution & suppress it.

→ It also works for multi-person inputs.
→ It even works for sounds of silence (only detect words/inputs).

Vision & Language ⇒



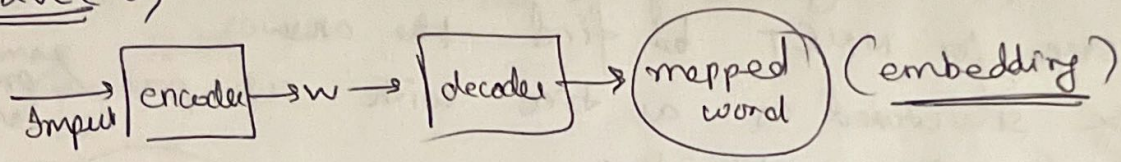
Architecture :-



Words :-

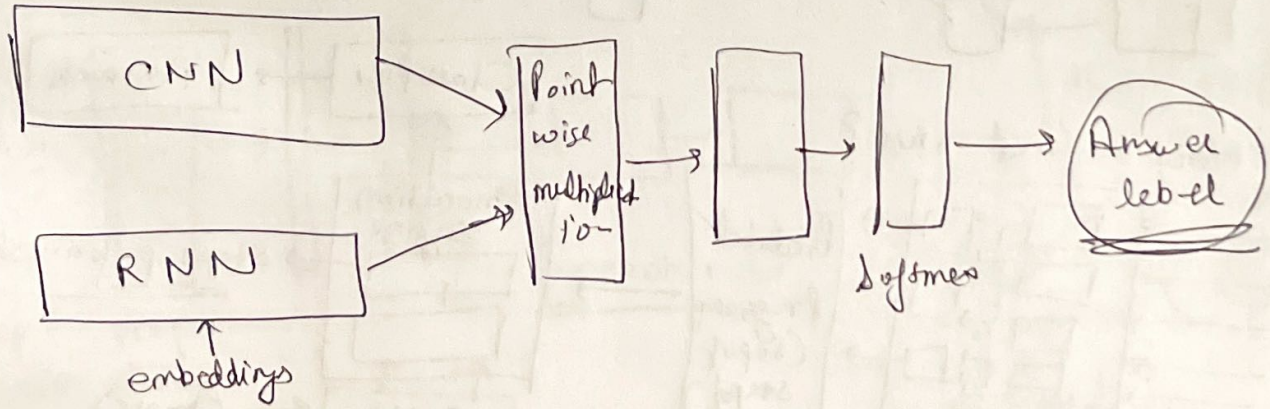
→ Need ways to compare words.

word2vec ⇒



→ Can perform algebraic operations on vector representation of words, (vector add^n & sub^n).
(you can even find closest neighbors in the embedding space).

So, now we form full N/w ⇒



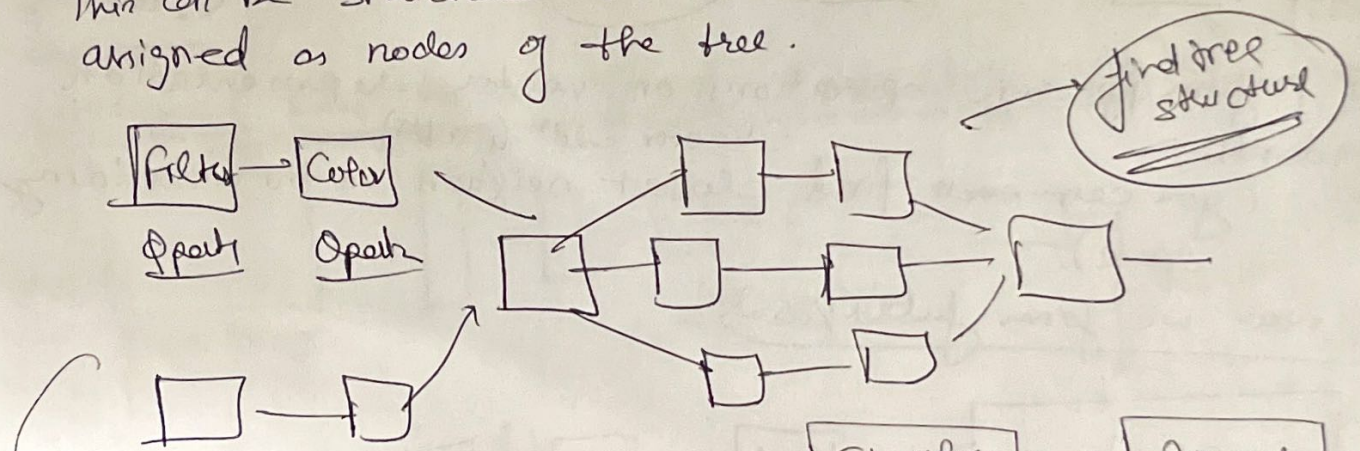
→ Sometimes the results aren't even consistent within itself. Generally, we extrapolate in nature from the knowledge that we have. But here we interpolate because the nature of this model is to find Qn-Answer pairs.

→ If there are no trains in the image given and asked for no. of trains, the model will interpolate and try to minize the loss and give the number which was most probable acc. to the training data set!

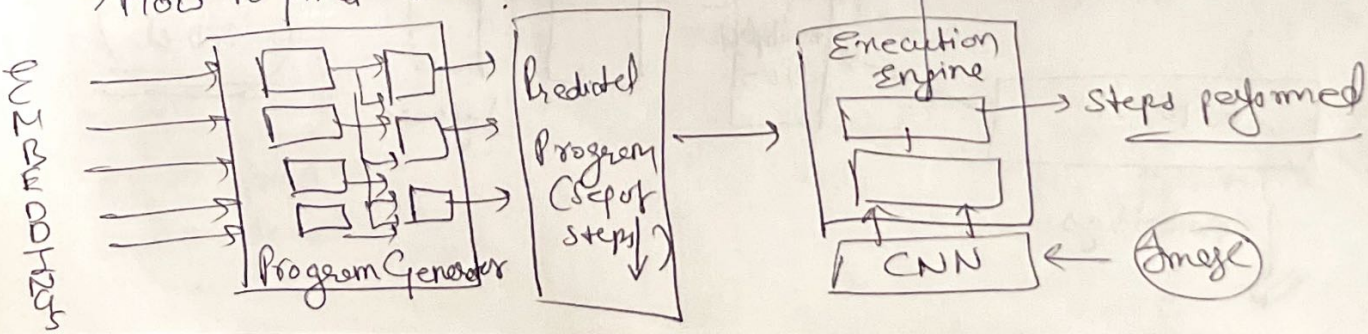
Compositional VQA ⇒ When you need to consider all the contextual knowledge in an image & find the required local significant features acc. to the question, then the problem becomes a hard one. Also, we need to consider biases.

→ we have to find the program modules & uses the NNET and assigns tags to each part of the question, each part of the question will be assigned to a program/question. This, basically becomes steps for NNET to find the answers.

This can be structured as a tree with sub-part & programs assigned as nodes of the tree.



How to find this?



Relational N/Ws:-

Based on a relational question, we need to find representations of the question to find the final answer. Because of so much structure before, one assumption (wrong) can impact a lot. Thus, generally, we won't follow a similar structure as before.

→ Is these ~~~ same as ~~~? (Relational Qn).

Properties of CNNs:-

1. Spatial equivalence.
2. Translational invariance

Properties of RNNs:-

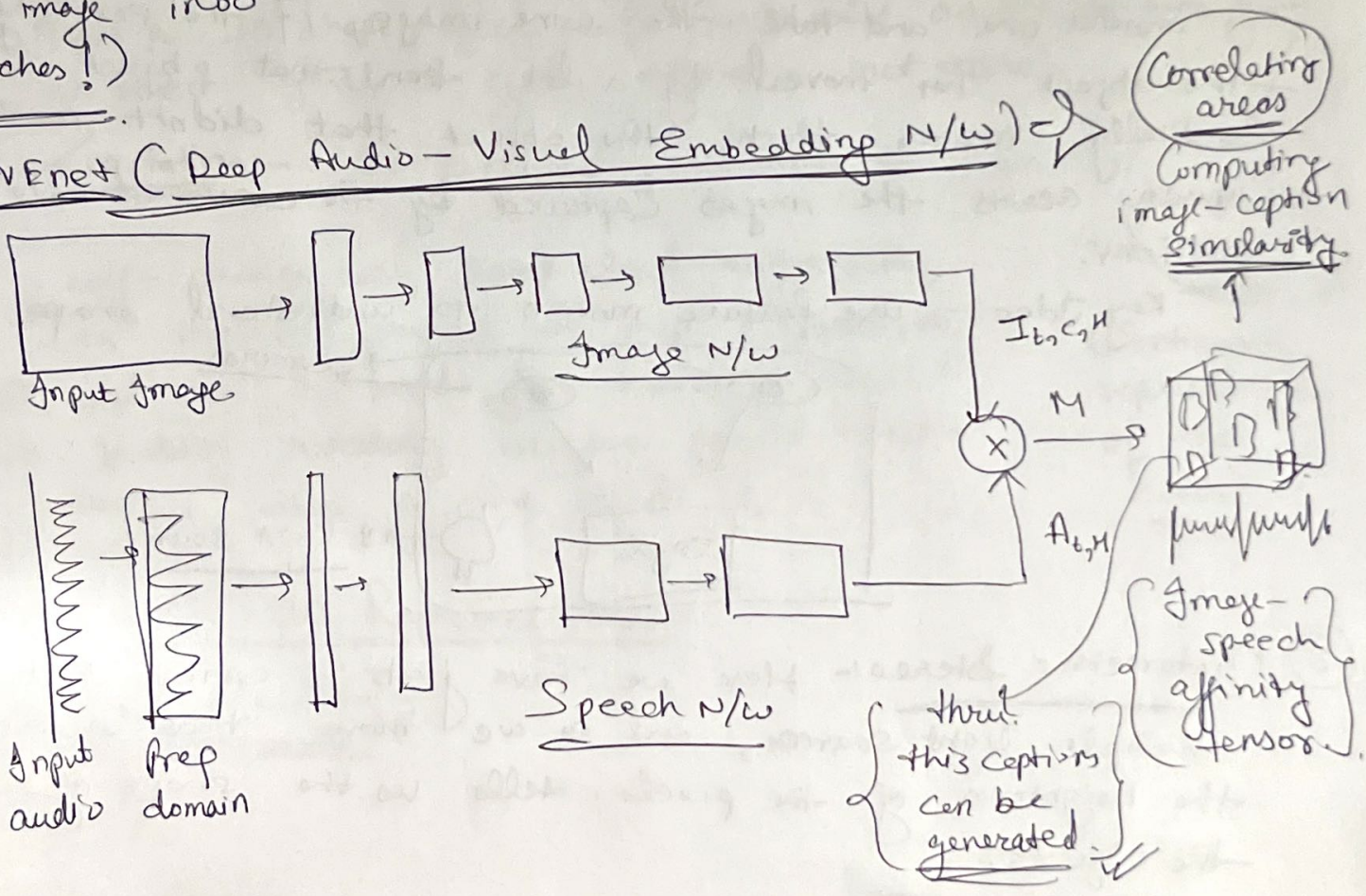
1. Capacity to compute relations.

$$RN(O) = f_\phi\left(\sum_{i,j} g_\theta(o_i, o_j)\right)$$

(relationships b/w pairs of objects in the image (finding that and adding up for all the possible cases.)).

(For this also, we need to distribute the image into patches!)

DAVEnet (Deep Audio-Visual Embedding N/w) ⇒



3D - Vision ⇒

→ Why put eyes together? → 3D Perception (depth sense)

⇒ Depth is so much more important.

Distance b/w eyes / cameras → baseliner.

(the far eyes are from each other, the better they are at depth perception).

Stereo vision ⇒ when eyes are seen together with some distance b/w the eyes.

Predators will have stereo vision with large distance
Preys have eyes as far as possible to cover 360°.

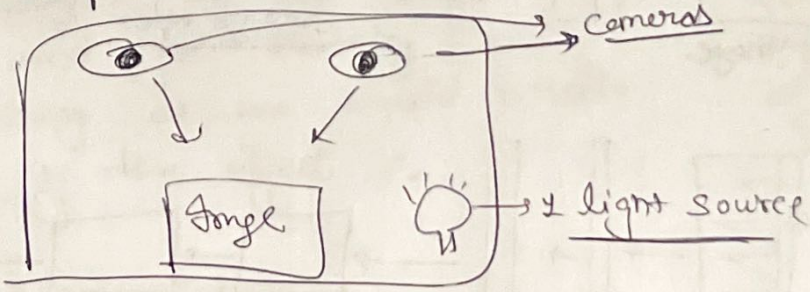
→ Depth without objects?

we do not perceive depth thru objects but due to some low underlying detection by 2 eyes, (in an integrated way).

Ways to perceive depth ⇒

① Binocular stereo :- when you move camera from one place to another one and take the same image, if the pixels for the object far moved by a lot, then that object is actually nearer than the object that didn't move much across the images captured by the camera at the two positions.

Key Idea :- use feature motion to understand shape.



② Photometric Stereo :- Here we have just 1 camera but multiple light sources, and as we move these around, the brightness of the pixels tells us the shape of the objects.

Key Idea:- use pixel brightness to understand shape.

Why learn 3D vision?

→ To calculate the depth of the pixel, we can move the view pt. around see the distance traveled by the particular pixel but this makes perception quite hard, that's why we learn 3D vision.

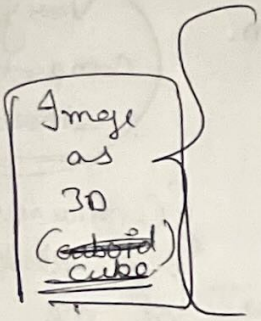
↳ Acc to the light source direction, the depth perception changes & it also does this bcoz. of vanishing pts.

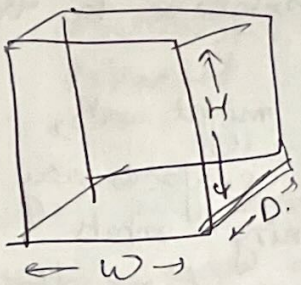
↳ We need to combine priors about the env. to correctly perceive 3D data.

→ The world is not ideal diffuse.

LiDAR Sensor → works like sonar to detect objects and can also be used to create 3D models of the environment (used in self driving cars).
(Rain is not good with LiDAR).

↳ We represent 3D images as Voxels. (VOXEL) (Volume element)
(similar to pixel).
(represented as a cube).





Limitations:-

(a) Very memory intensive as most of the space is empty.

(b) Trade-off with voxel resolution.
(depth can increase by a lot if we break the object into very small voxels).

To reduce memory, we might have to make a trade-off of the voxels to represent an object.

Another way of representation ⇒ Limitations :- Representation is sparse.

Point clouds → we generate these with only the activated voxels and is basically all the points that are needed to model an object.
(can scale large scenes).

↓
values are only on the surface of cloud with nothing inside

Mesh ⇒ Represent diff. parts of the object as made up of Δs.

Limitations :- Difficult to integrate with neural networks and takes expertise to label.
No camera/sensor directly senses in a mesh-form.

Implicit Surfaces ⇒ $f(x, y, z) = 0$ → only parameters of this func^n are stored → Very compactly stored.

(Model a 3D shape/scene with func^n value on the surface of an object equal to zero, otherwise not zero)

Limitations :- Computationally extensive to train.

(Continuous representation of the scenes)

(*) For a self-driving car, point cloud might work, but with implicit surfaces, we map surfaces very well and can be used for training robots while grabbing something as we would need to know how surface of the object is.

Learning with 3D Representations :-

With Point clouds ⇒

PointNet ⇒

→ classification → part segmentation → semantic segmentation.

→ Point cloud is basically a list of values (for all pts. of a surface)

X	Y	Z
—	—	—
—	—	—

→ can have −ve as well as +ve values

Challenges with Point Cloud:-

- → Unordered
- → Interaction among points → we don't know this! ↓ helps with CNNs. but we don't know!
- → Invariance under transformation.

↓

Consider adding 10 to everything, point cloud's origin is arbitrary. So, nothing changes (invariance) but it would impact brightness/something for the image. (we need it to change/vary mostly!)

Q How to build a NNET for this?

→ what about sorting?

— Pick a deterministic order, and feed results into a CNN? (still not able to work with invariance under transformation!)

(Also, if there's noise, it might change the ordering drastically, which might impact relationships extracted by CNNs).

Thus, this won't work well!

Q → what about invariance?

1. Treat the order as data augmentation.
2. Shuffle sequence, feed into RNN & train. Shuffle again, feed into RNN and train, etc.
3. Will this provide invariance to the order? ↳ no, as we can't go thru all the permutations possible.

PointNets:- Invariants → tries to fix these problems above!

$$f(\{x_1, \ldots, x_n\}) = g(h(x_1), \ldots, h(x_n))$$

h → simple neural network
g → order-invariant func^n

(Examples:- sum, mult, max)

⇒ Technically, this is a universal approximator, and it could learn to create voxels by partitioning the space. But it learns something more powerful!

→ Scene Segmentation & Classification can be done using this.

Q What are the critical points?

Which I/p points are contributing to the global feature?

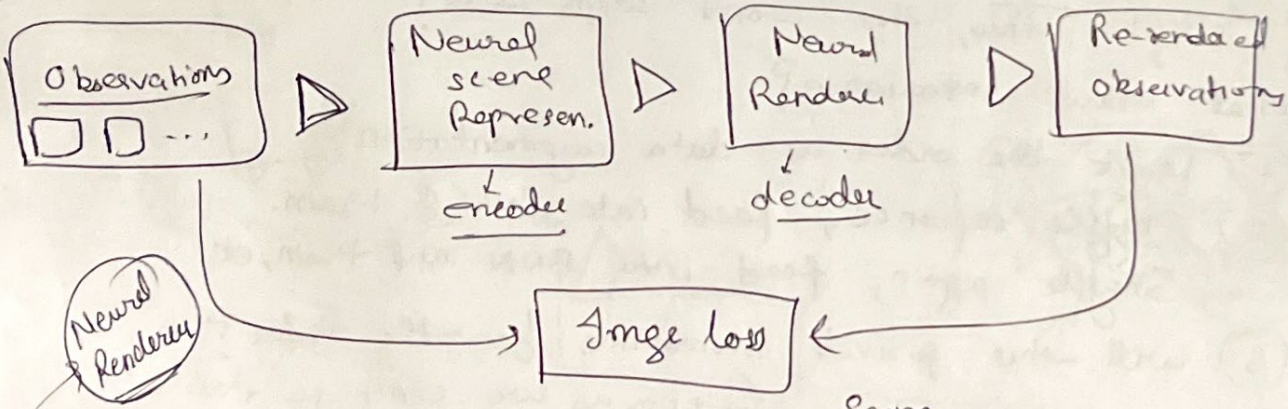
(from the layers) → (global features) → {we need to get these!}
↳ to see how model is learning.

generally determines the shapes.

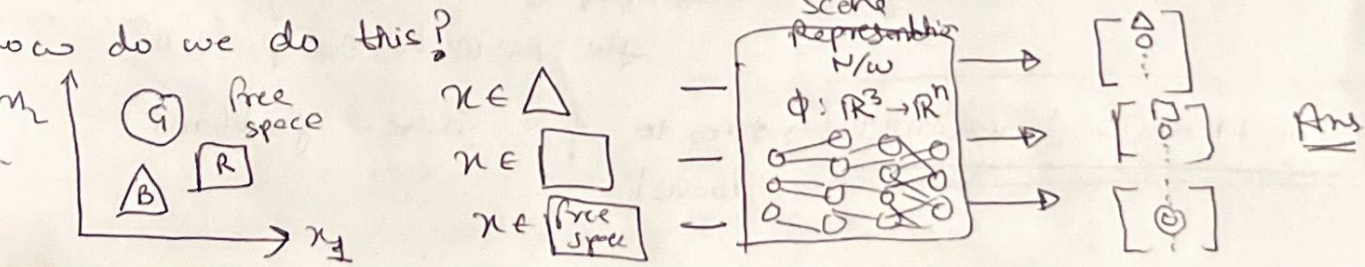
{Upper-bound shape - critical pts ⇒ points that do not matter!}

Implicit Surfaces

Scene Representation N/Ws (write an auto-encoder)



Q how do we do this?



Ans

The scene is converted into parameters of the neural network.

Neural Renderer Step 1:- Intersection Testing.

To represent a scene, we need to input the image and figure out what the weights are of this NN. Since NNs are universal approximators, there is a way to take any 2D/3D scene & compress it into the weights of a NN.

The Decoder is going to learn to query that representation.
(renderer)

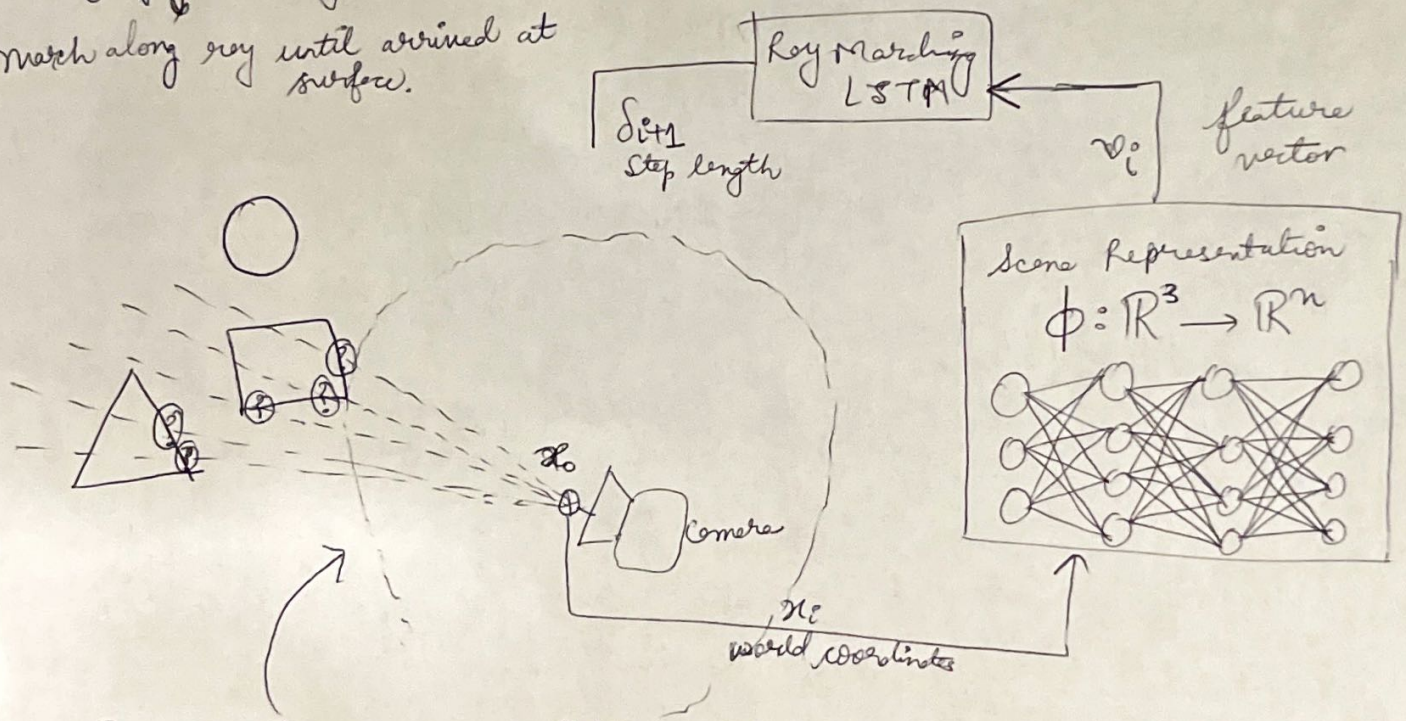
We can combine this renderer with classical algorithms in computer graphics. Eg of intersection of NN + graphics.

→ It needs to read out from the encoder to reconstruct the I/P.

---

*Ray Marching used in Neural Renderer Step 1: Intersection Testing

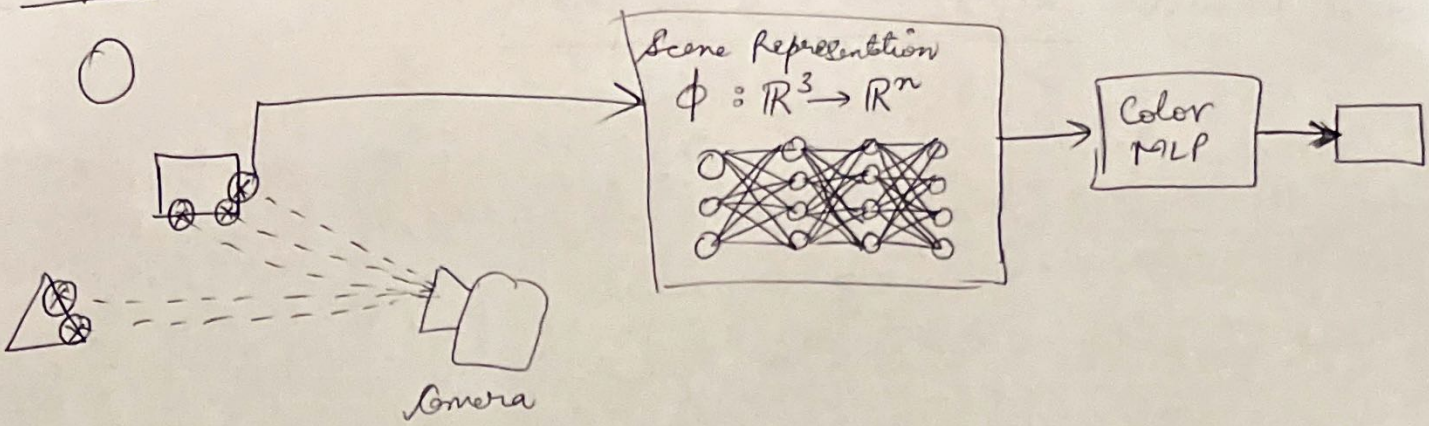
march along ray until arrived at surface.



Feasible step length : Distance to closest scene image

When you shoot your ray, march along it & hit a surface, you record that value; example: color of the object.

Step 2 : Color Generation



Can now train end-to-end with posed images only!

* You can backpropagate at each step since every step is differentiable.

Ray marching LSTM tells you WHICH DIRECTION to walk in and also retains your history of the direction you have taken. LSTM tells you how to sample along that ray. The scene representation is continuous & so are the rays.

Because the scene representation is continuous, we can easily increase the resolution of the camera (eg 1MPix camera) by increasing the # of rays.

This scene representation (encoder) ~~model~~ works well since it samples a couple different views of the object.

Ray marching assumes that we are operating w/o any transparent objects and that lighting does not matter.

→ we move the camera around if we cannot see an object much/ not at all!