Interview Session

CNNs

(Q0)

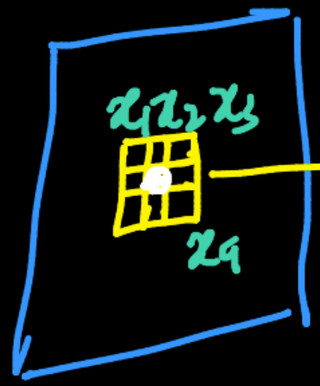
CNN

↓

#params (straight forward)

(Q1)

Maxpool $_{3\times3}$ → no params [How does back-prop over on max pool layer]



Input - - - - → Maxpool → max over 9 values (x_i's) → N (let)

$\frac{\partial L}{\partial N}$

$\frac{\partial L}{\partial N} \cdot \frac{\partial N}{\partial Max} \cdot \frac{\partial Max}{\partial x_i}$

$\frac{\partial Max(all\ x_i)}{dx_i} = \begin{cases} 1 & \text{max elem} \\ 0 & \text{non-max ele.} \end{cases}$ ✓

$\frac{\partial N}{\partial Max} \rightarrow 1$

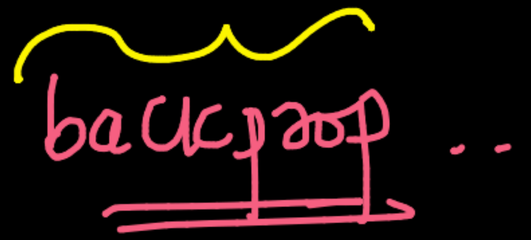
operation

conV

Maxpool

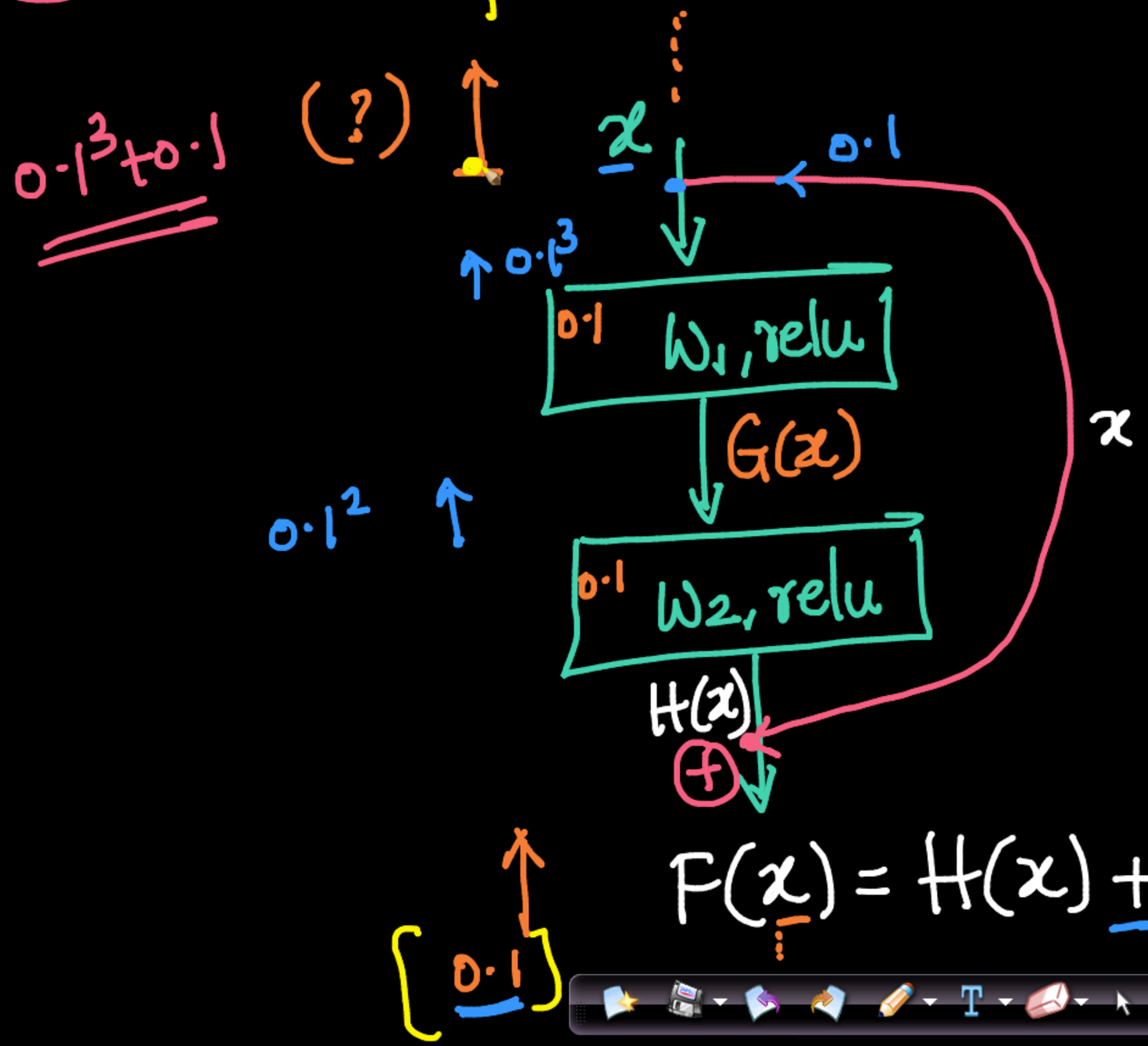
Doopout

Residual...



Transformer. module/block

② skip-connection : back-prop



0.1^3 to 0.1

(g) 0.11 — 0.001

(h) 0.1

(a) 0.1 x 0.1 x 0.1

(b) 1 + 0.1 x 0.1 x 0.1

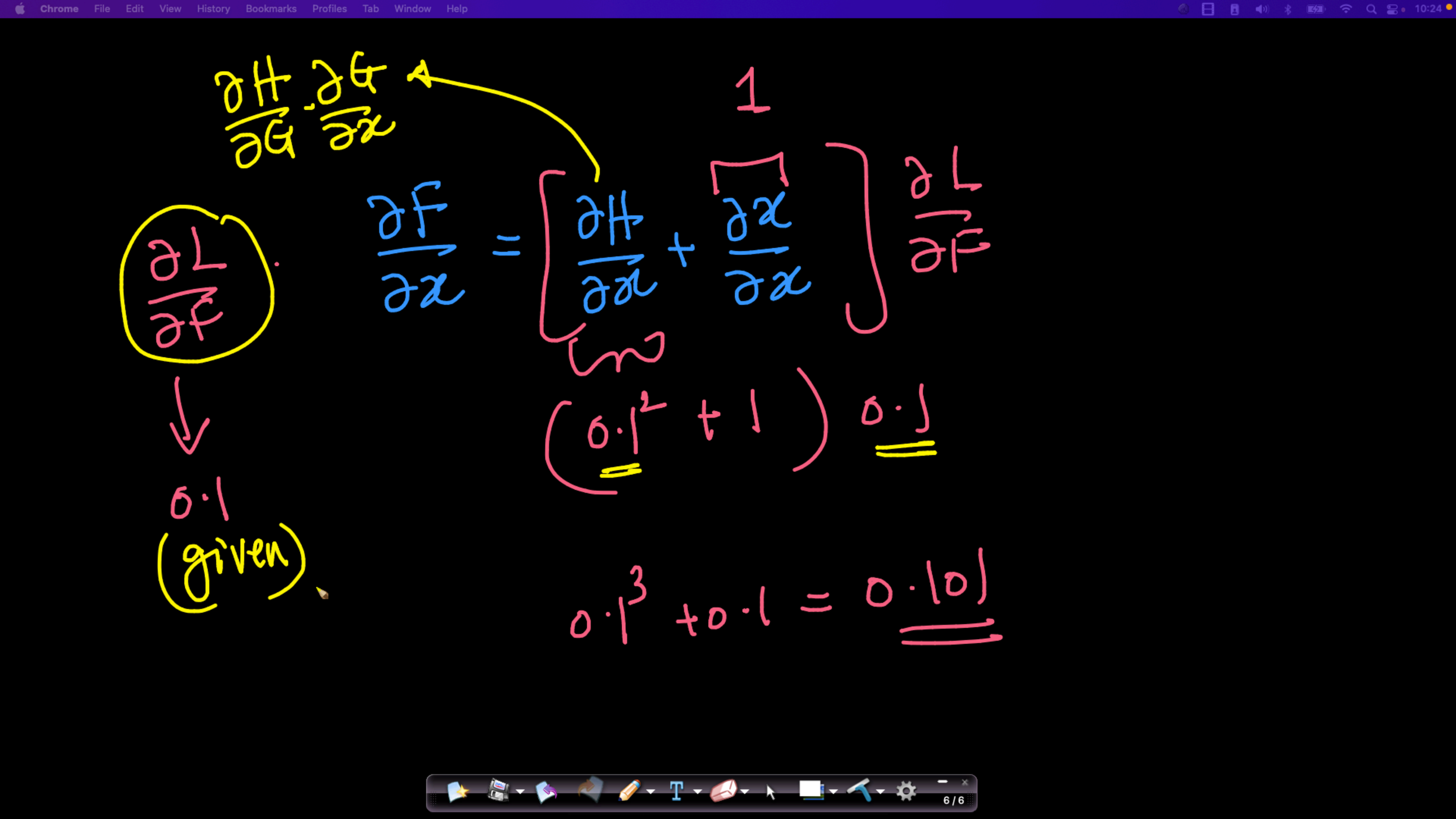
(c) 0.3

(d) 0

(e) random value

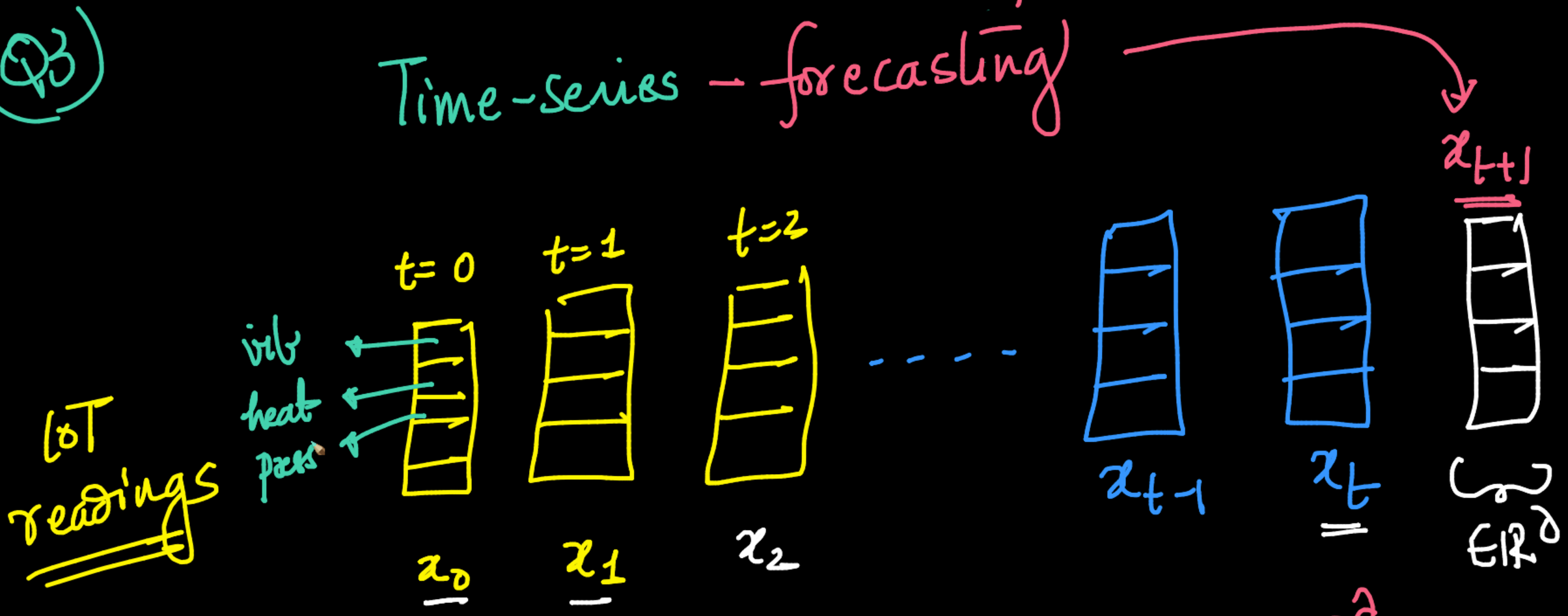
(f) 0.2

(i) 0.101 ✓


$\frac{\partial H}{\partial G} \cdot \frac{\partial G}{\partial x}$
1
$\frac{\partial L}{\partial F}$
$\frac{\partial F}{\partial x} = \left[\frac{\partial H}{\partial x} + \frac{\partial x}{\partial x} \right] \frac{\partial L}{\partial F}$
$(0.1^2 + 1)\, 0.1$
0.1
(given)
$0.1^3 + 0.1 = 0.101$

Q3

Time-series - forecasting

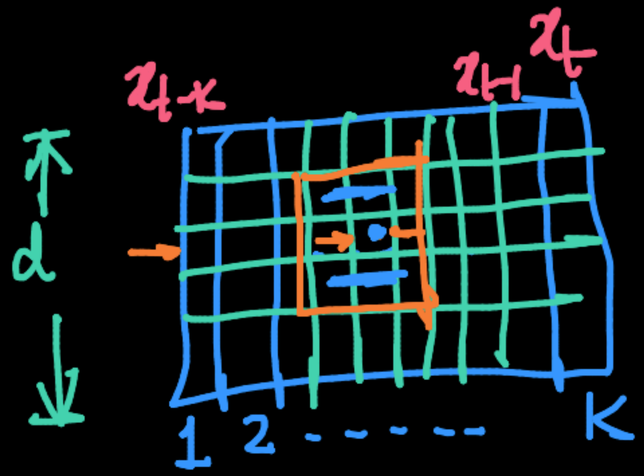


CNN

$x_i \in \mathbb{R}^d$

AI

@ time t , last k time stamps → Move window

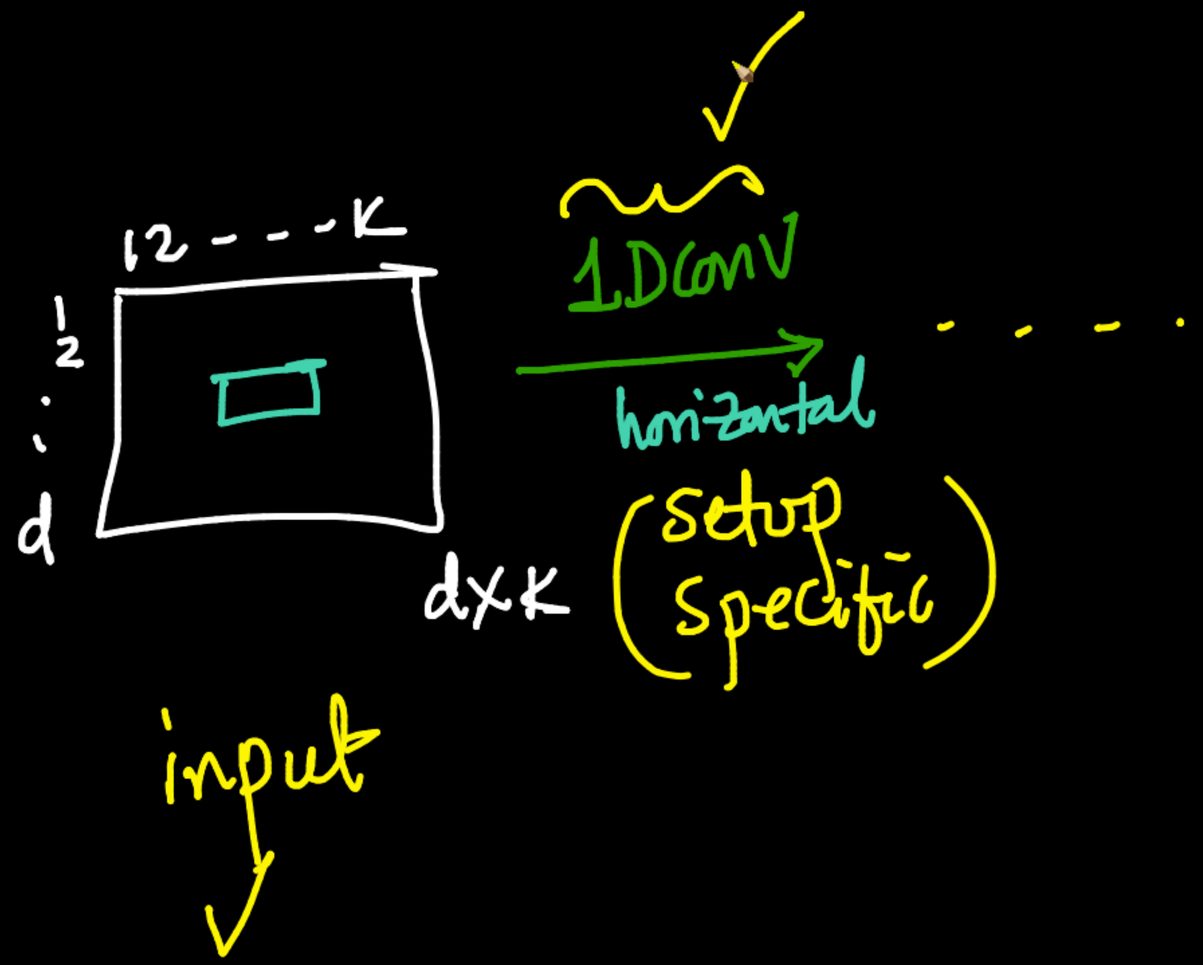


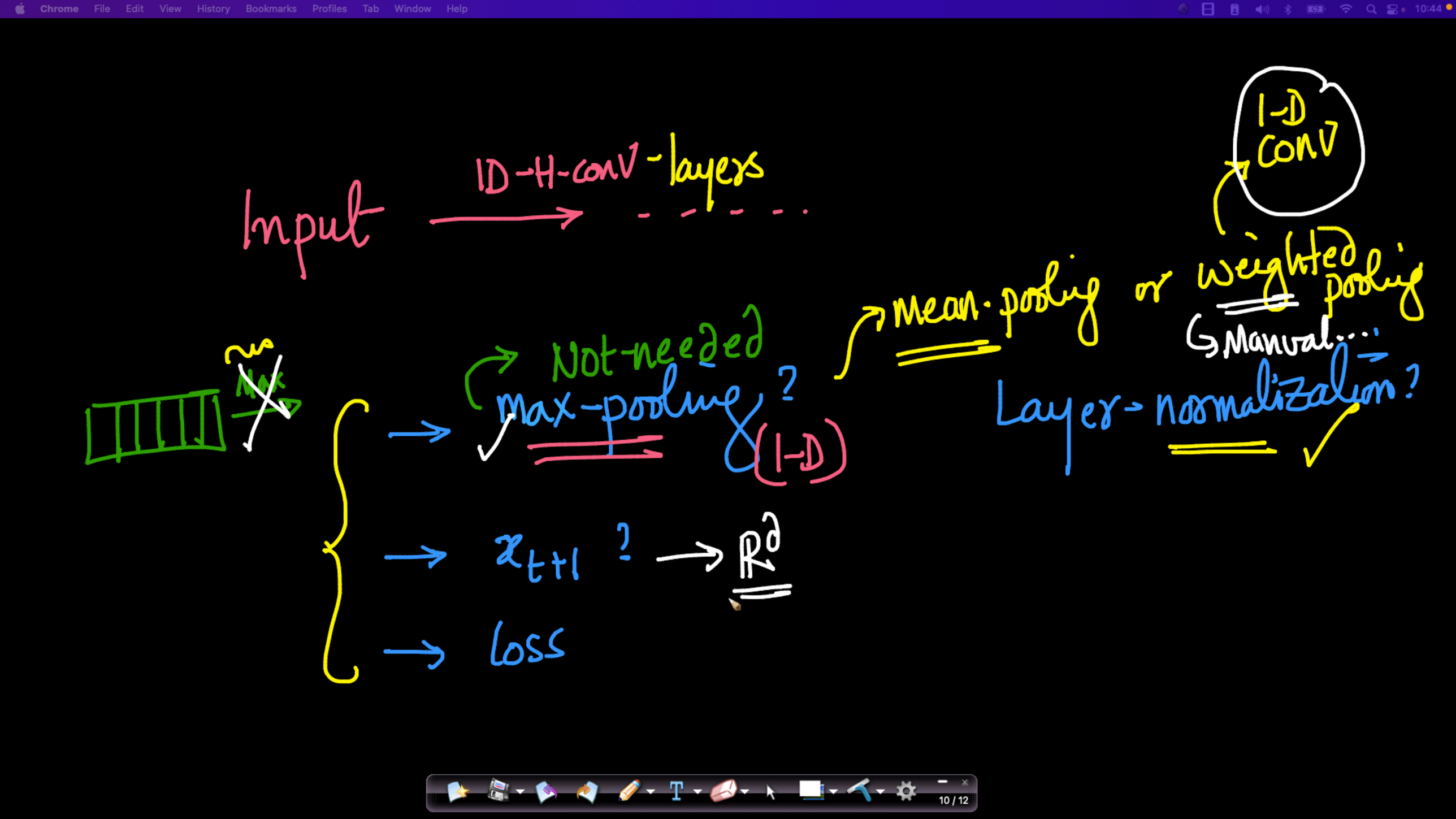
input

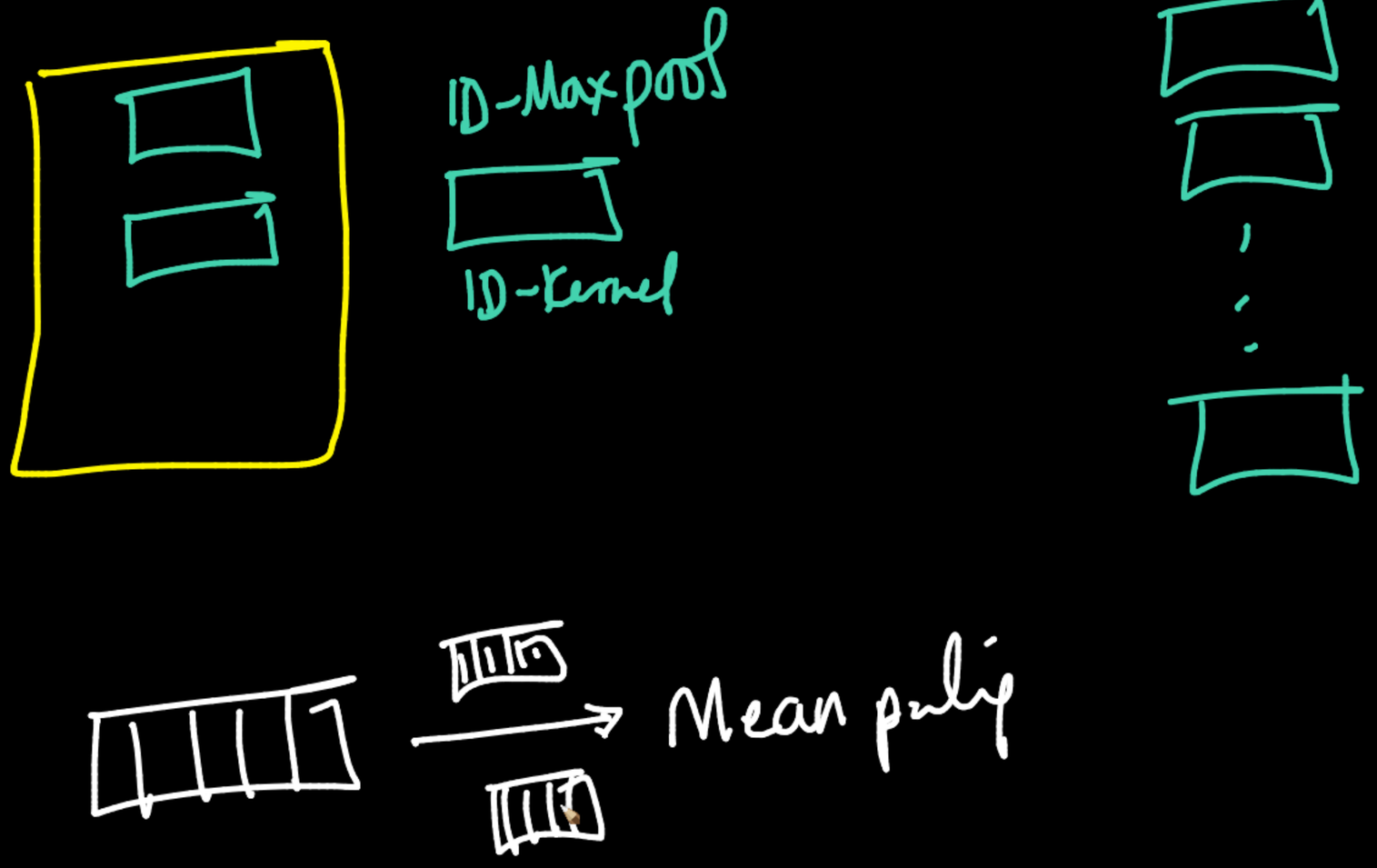
→ CNN (2D) X

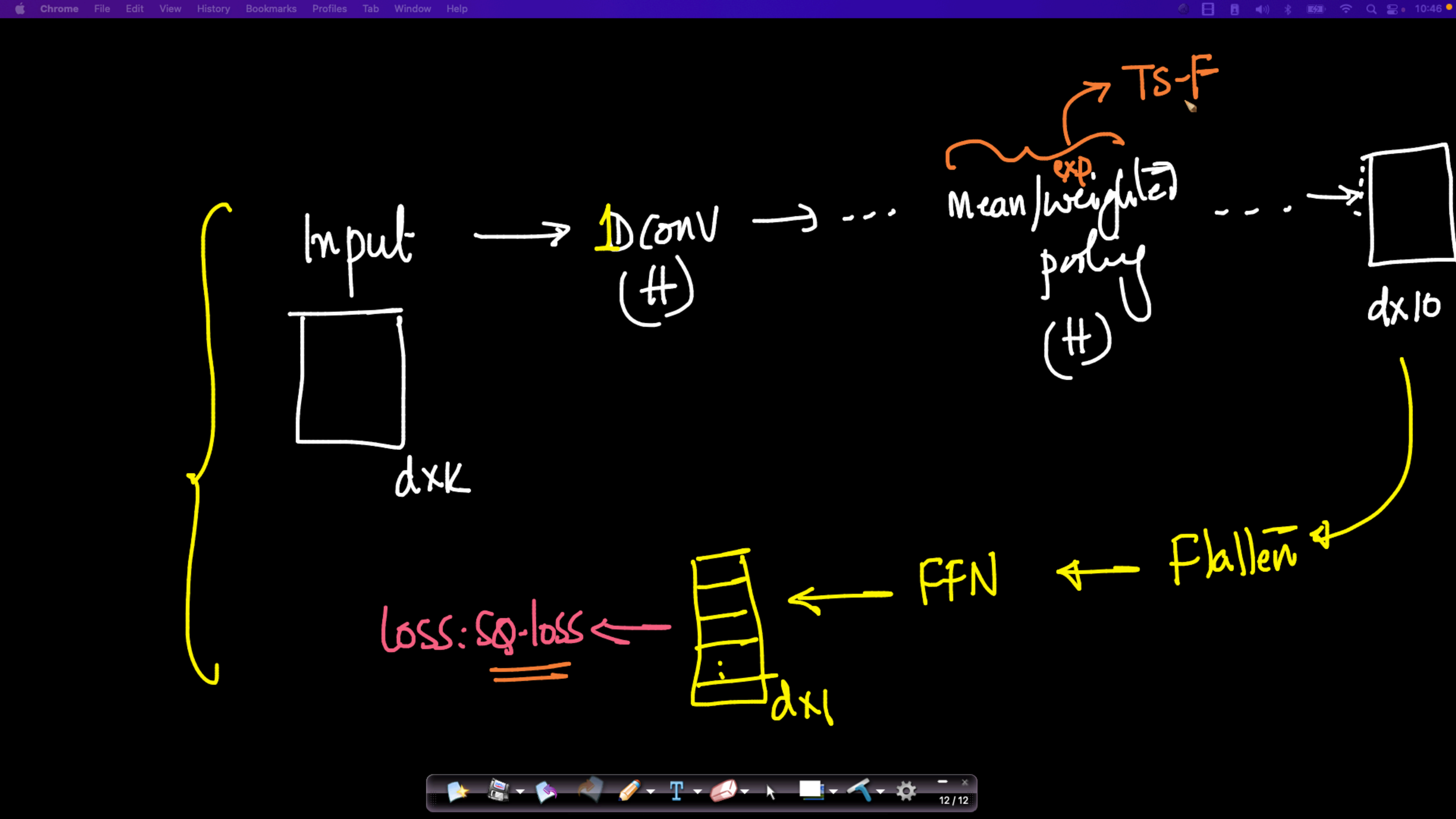
1D-conV

2D-conV


1 2 - - - K
1
2
:
d
dxK
1DConV
horizontal
(setup specific)
input

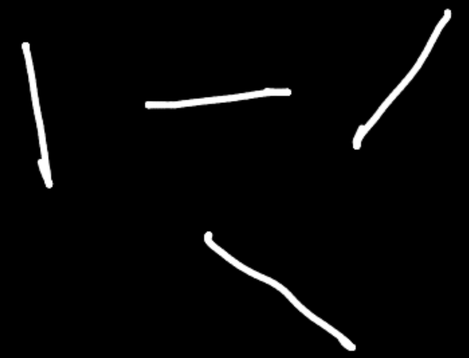

Input
1D-H-conV-layers
1-D conV
Not-needed
max-pooling ? (1-D)
mean-pooling or weighted pooling
Manual...
Layer-normalization ?
x_{t+1} ? $\rightarrow \mathbb{R}^d$
loss


1D-Max pool
1D-Kernel
Mean pool

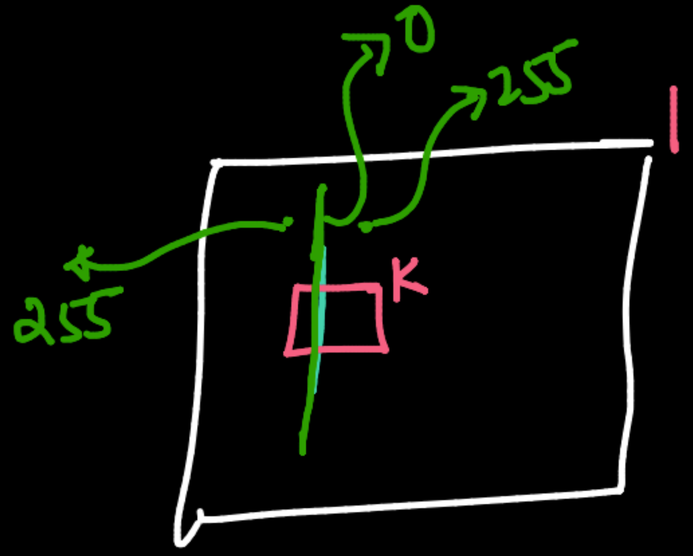

TS-F
exp
Input
1D conV
(H)
Mean/weighted
pooling
(H)
dx10
dxK
Flatten
FFN
loss: SQ-loss
dx1

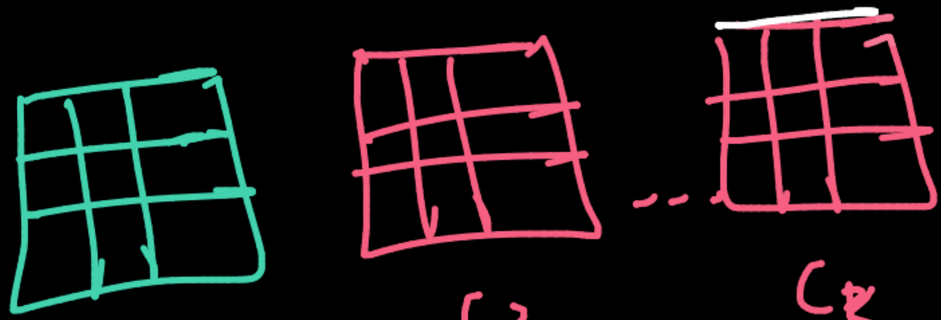
(Q)

Image - CNN



K → 1st-layer - convolutions 3x3





Specific Shape

to look like edge detects

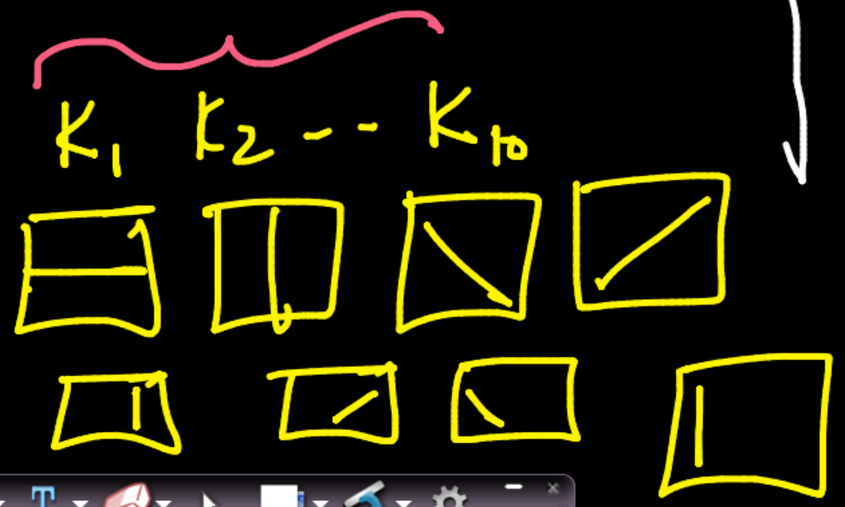
Task: build CNN model s.t 1st layer cnv look like edge detectors always

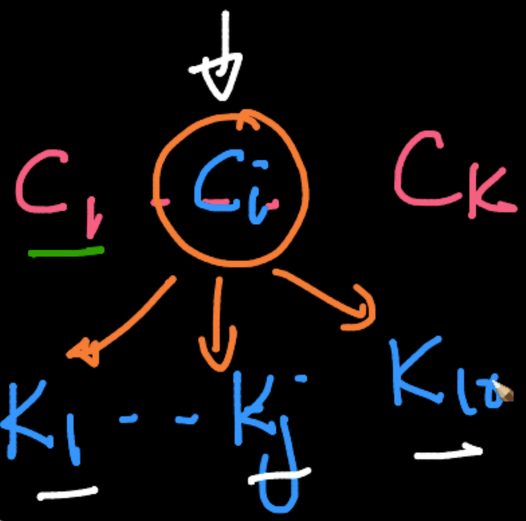
X (a) inc size of kernel

? (b) remove noise i/p image

Good Idea (c) ✓ initialize kernels to be edge-detectors ✓

(d) ✓ template kernels: K_1 K_2 -- K_{10}





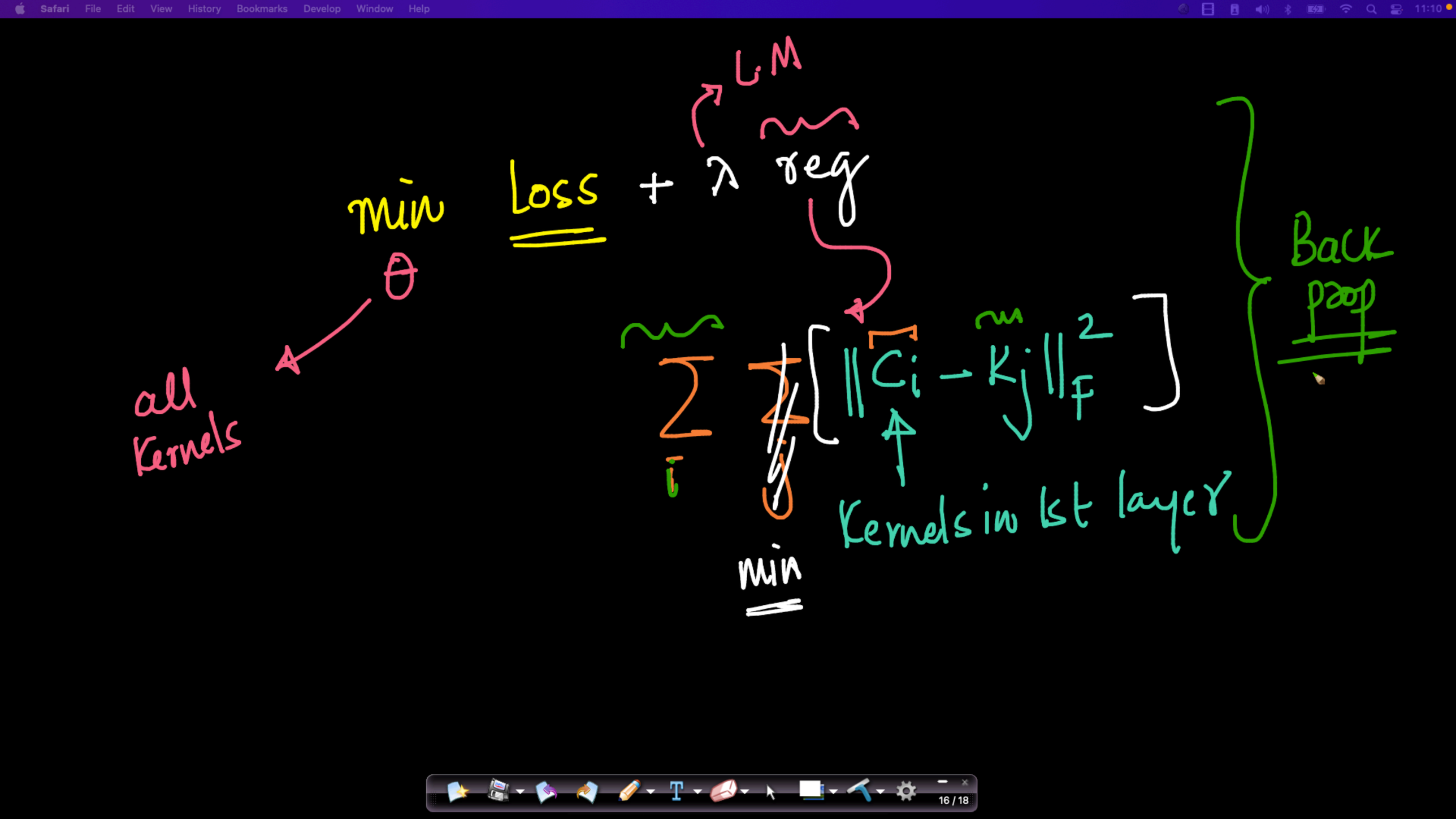
C_K ← Kernals in 1st layer

← Template Kernals

Obj:

$$\forall i : 1 \to K \quad \exists j \text{ s.t } C_i \approx K_j$$

$$\| C_i - K_j \|_F : \text{small}$$


L.M
min
θ
Loss
+ λ reg
all
Kernels
Σ
i
Σ
j
[||Ci − Kj||F2]
min
Kernels in 1st layer
Back
prop

nearest to it. These $\mathbf{x}_i$ are called *support vectors*.

Soft-margin [edit]

To extend SVM to cases in which the data are not linearly separable, the *hinge loss* function is helpful

$$\max\left(0, 1 - y_i(\mathbf{w}^T \mathbf{x}_i - b)\right).$$

Note that y_i is the *i*-th target (i.e., in this case, 1 or –1), and $\mathbf{w}^T \mathbf{x}_i - b$ is the *i*-th output.

This function is zero if the constraint in (1) is satisfied, in other words, if $\mathbf{x}_i$ lies on the correct side of the margin. For data on the wrong side of the margin, the function's value is proportional to the distance from the margin.

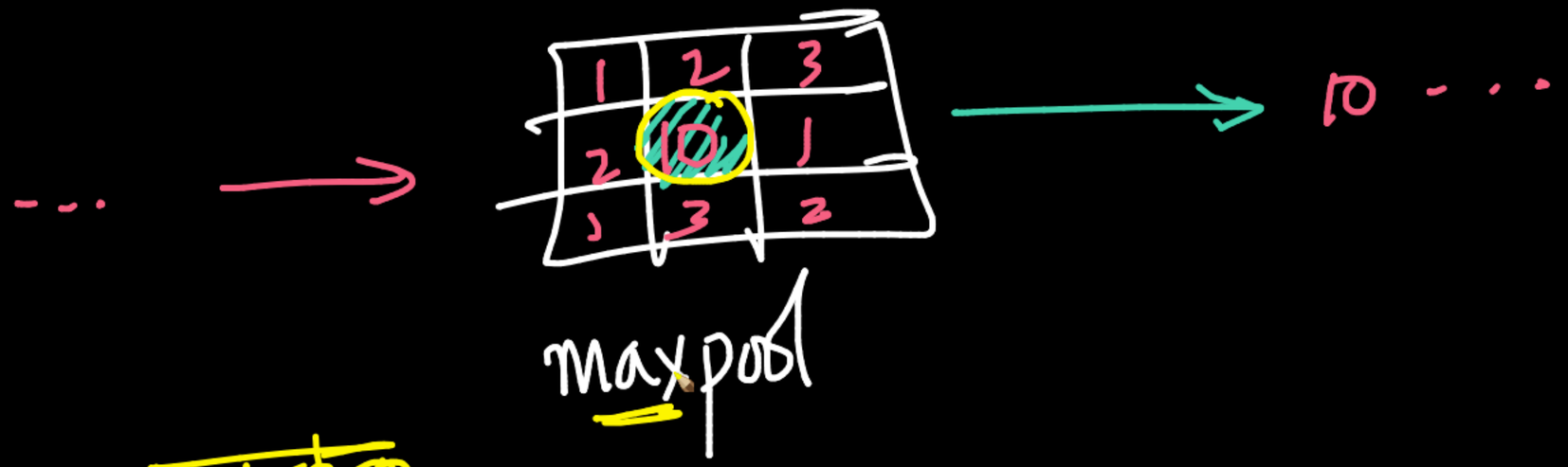
The goal of the optimization then is to minimize

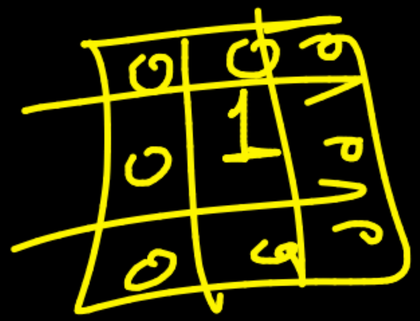
$$\lambda\|\mathbf{w}\|^2 + \left[\frac{1}{n}\sum_{i=1}^{n}\max\left(0, 1 - y_i(\mathbf{w}^T \mathbf{x}_i - b)\right)\right],$$

where the parameter $\lambda > 0$ determines the trade-off between increasing the margin size and ensuring that the $\mathbf{x}_i$ lie on the correct side of the margin. Thus, for sufficiently small values of λ, it will behave similar to the hard-margin SVM, if the input data are linearly classifiable, but will still learn if a classification rule is viable or not. (This parameter λ is often also called C, e.g. in *LIBSVM*, but usually refers to the inverse of λ.)

Nonlinear Kernels [edit]

The original maximum-margin hyperplane algorithm proposed by Vapnik in 1963 constructed a linear classifier. However, in 1992, Bernhard Boser, Isabelle Guyon and Vladimir Vapnik suggested a way to create nonlinear classifiers by applying the kernel trick (originally proposed by Aizerman et al.[formally





Q

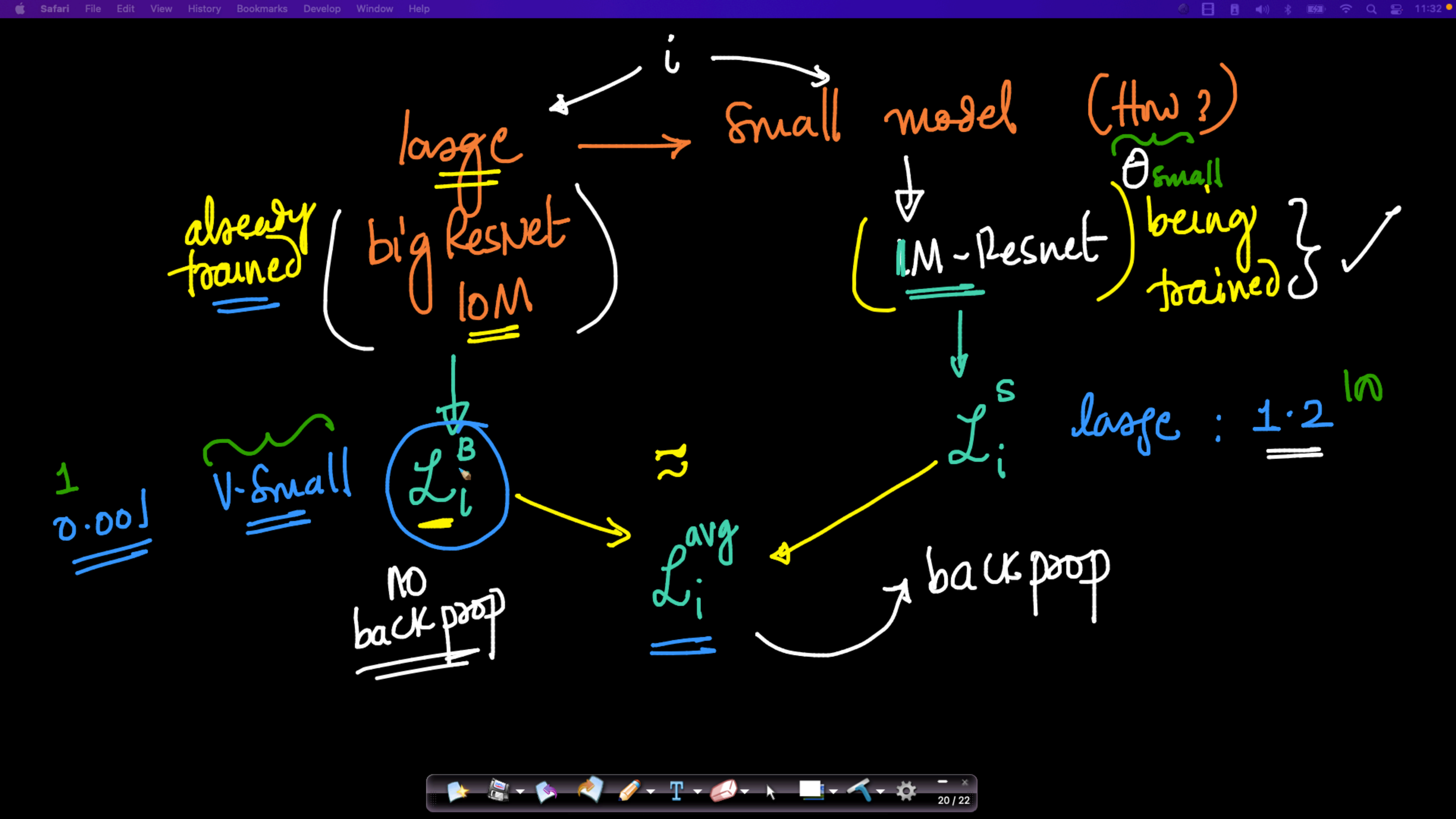
Image Classfn → CNN (1M) ResNet → well ✓

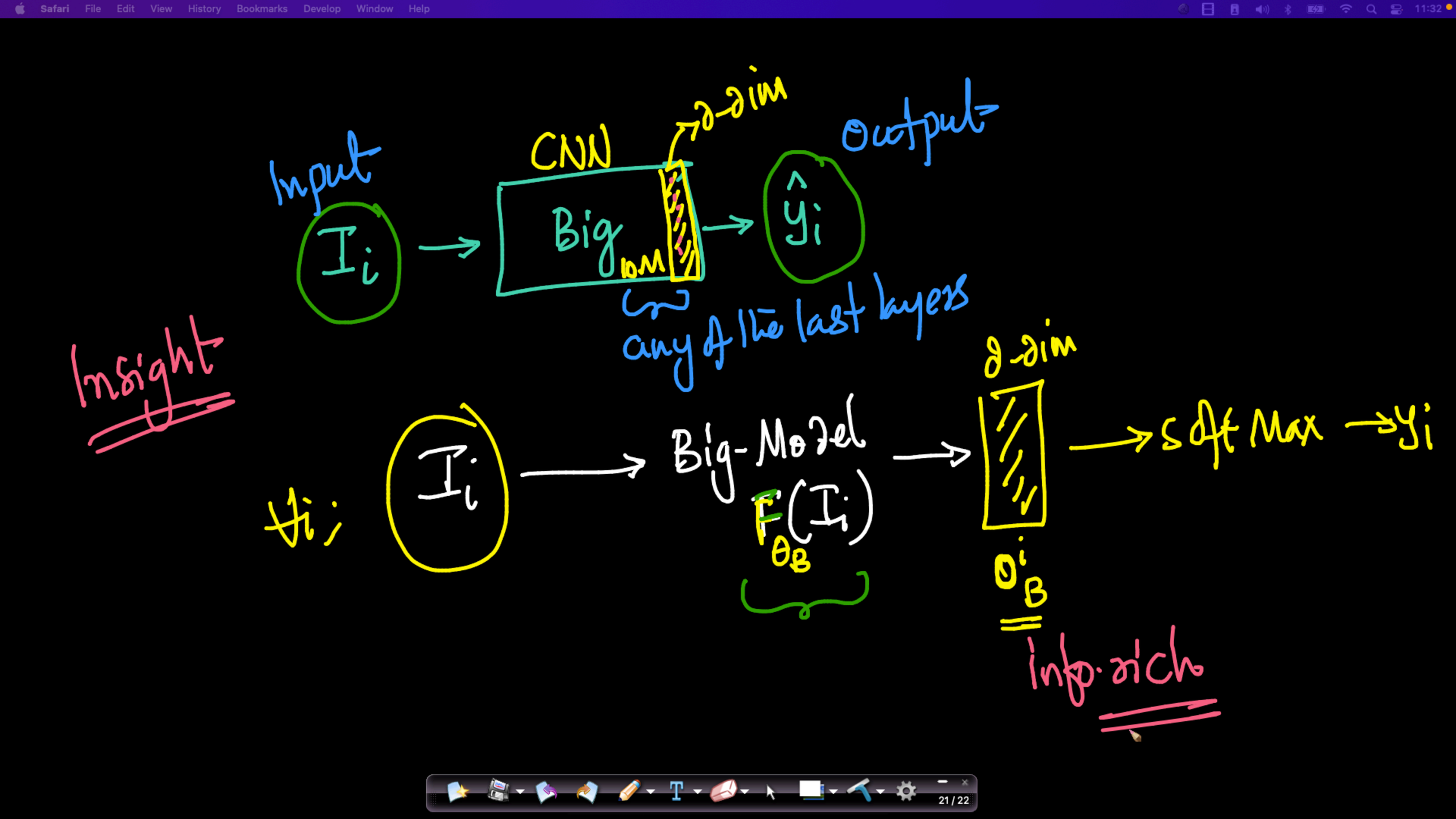
10M params

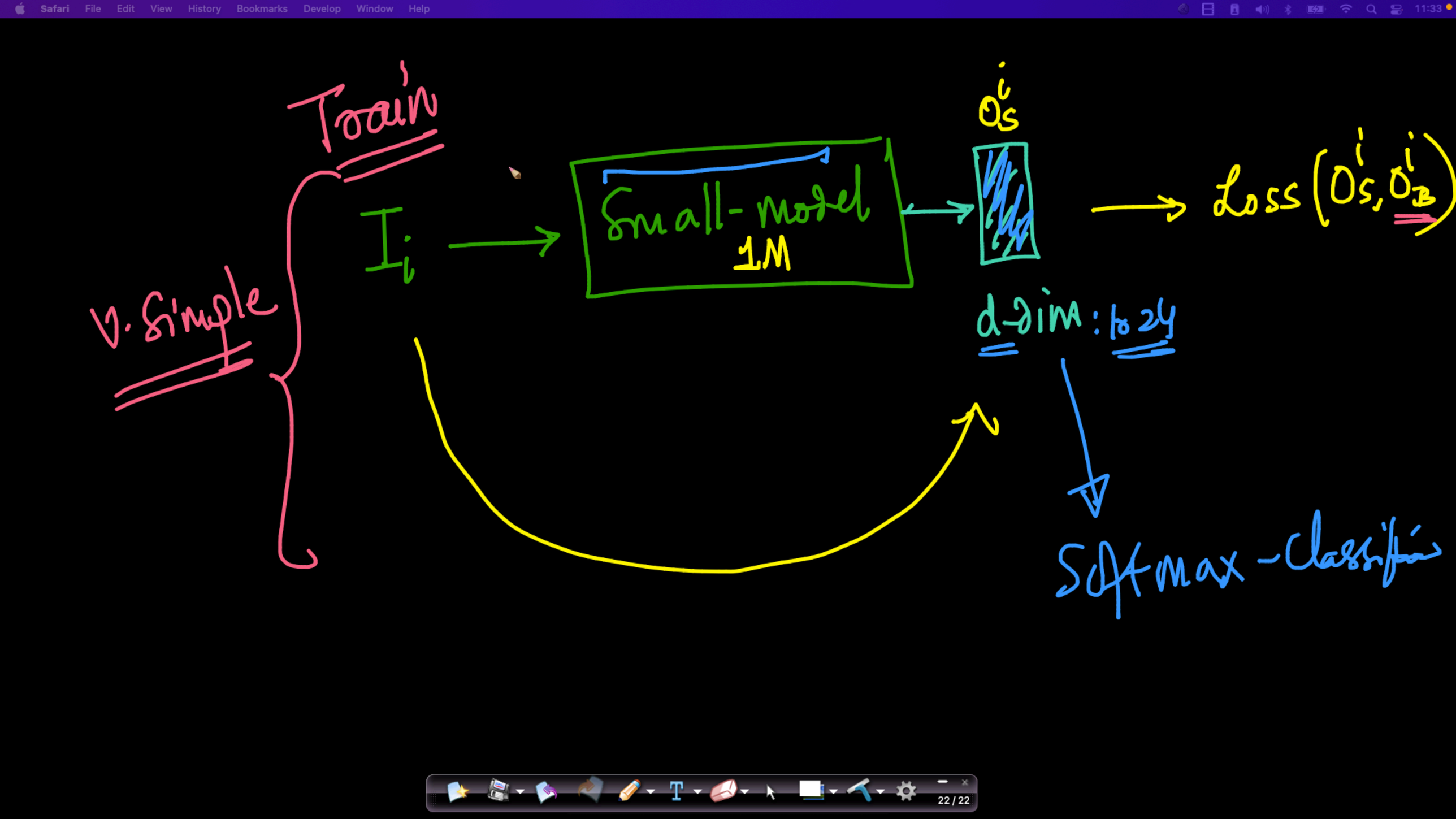
Mobile phone

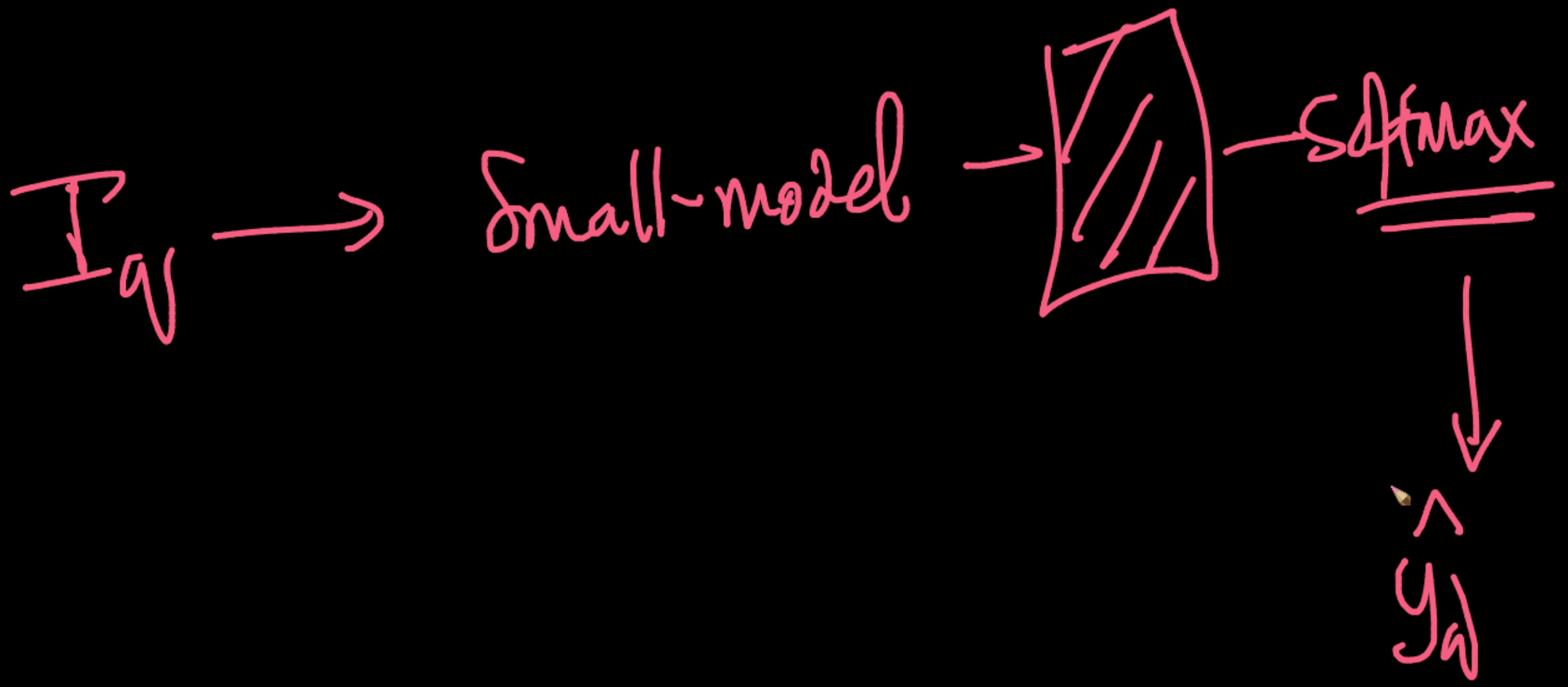
[already → engineering hacks
↓
float → int
TFLite]

not sufficient


i
large
small model
(How ?)
already trained
(big Resnet 10M)
(LM-Resnet)
θ small being trained
V-small
1
0.001
NO back prop
≈
large : 1.2 ln
back prop


Input
I_i
CNN
Big
DM
d-dim
output
$\hat{y}_i$
any of the last layers
Insight
y_i;
I_i
Big-Model
$F_{\theta_B}(I_i)$
d-dim
softMax
y_i
o_B^i
info-rich


Train
V. Simple
I_i
Small-model
1M
O_s^i
$Loss(O_s^i, O_B^i)$
d-dim: to 24
Softmax-classifi


I_q
small-model
softmax
$\hat{y}_q$

nearest to it. These $\mathbf{x}_i$ are called *support vectors*.

Soft-margin [edit]

To extend SVM to cases in which the data are not linearly separable, the *hinge loss* function is helpful

$$\max\left(0, 1 - y_i(\mathbf{w}^T \mathbf{x}_i - b)\right).$$

Note that y_i is the *i*-th target (i.e., in this case, 1 or –1), and $\mathbf{w}^T \mathbf{x}_i - b$ is the *i*-th output.

This function is zero if the constraint in (1) is satisfied, in other words, if $\mathbf{x}_i$ lies on the correct side of the margin. For data on the wrong side of the margin, the function's value is proportional to the distance from the margin.

The goal of the optimization then is to minimize

$$\lambda\|\mathbf{w}\|^2 + \left[\frac{1}{n}\sum_{i=1}^{n}\max\left(0, 1 - y_i(\mathbf{w}^T \mathbf{x}_i - b)\right)\right],$$

where the parameter $\lambda > 0$ determines the trade-off between increasing the margin size and ensuring that the $\mathbf{x}_i$ lie on the correct side of the margin. Thus, for sufficiently small values of λ, it will behave similar to the hard-margin SVM, if the input data are linearly classifiable, but will still learn if a classification rule is viable or not. (This parameter λ is often also called C, e.g. in *LIBSVM*, but usually refers to the inverse of λ.)

Nonlinear Kernels [edit]

The original maximum-margin hyperplane algorithm proposed by Vapnik in 1963 constructed a linear classifier. However, in 1992, Bernhard Boser, Isabelle Guyon and Vladimir Vapnik suggested a way to create nonlinear classifiers by applying the kernel trick (originally proposed by Aizerman et al.[formally