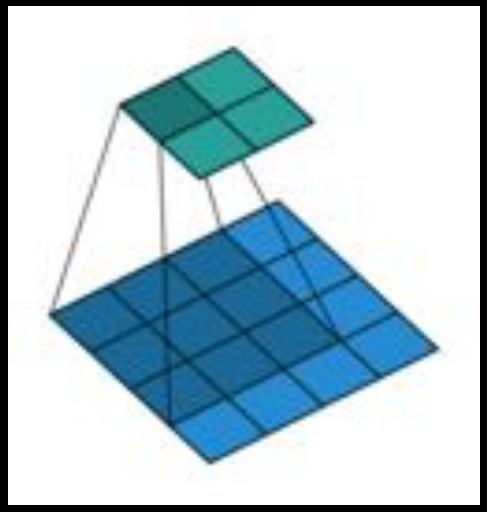
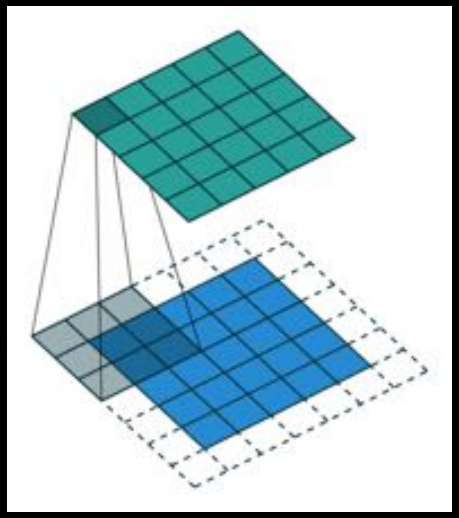
Stride

Kernel

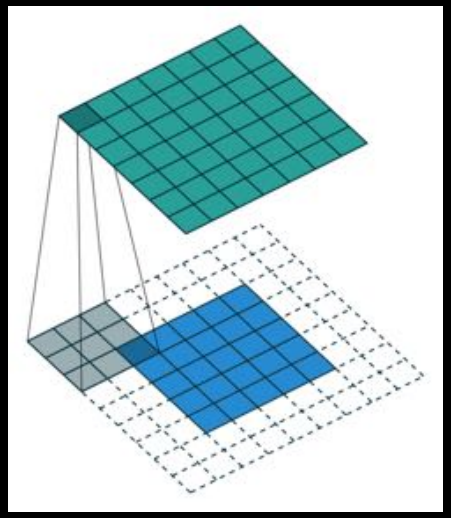
Padding



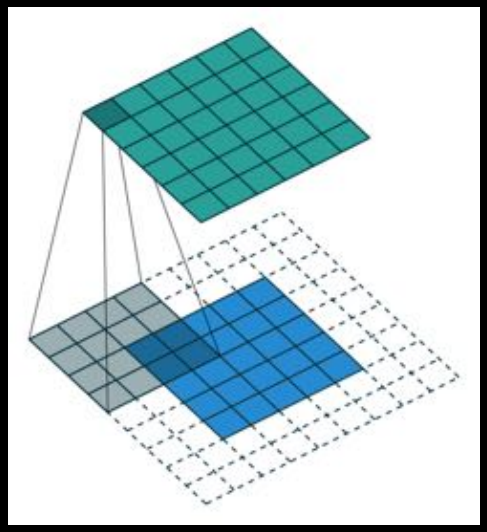
S=1, K=3, P=0



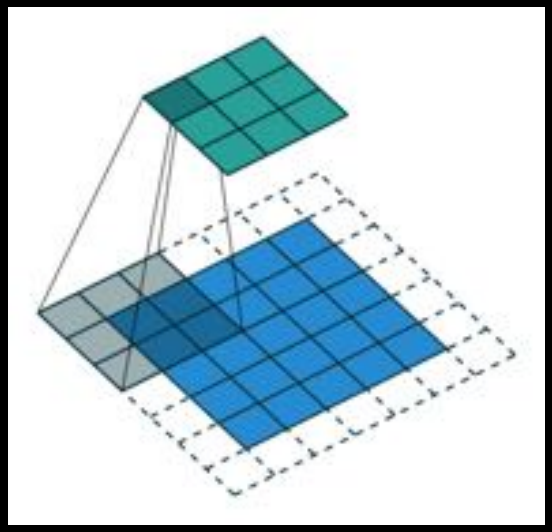
S=1, K=3, P=1



S=1, K=3, P=2



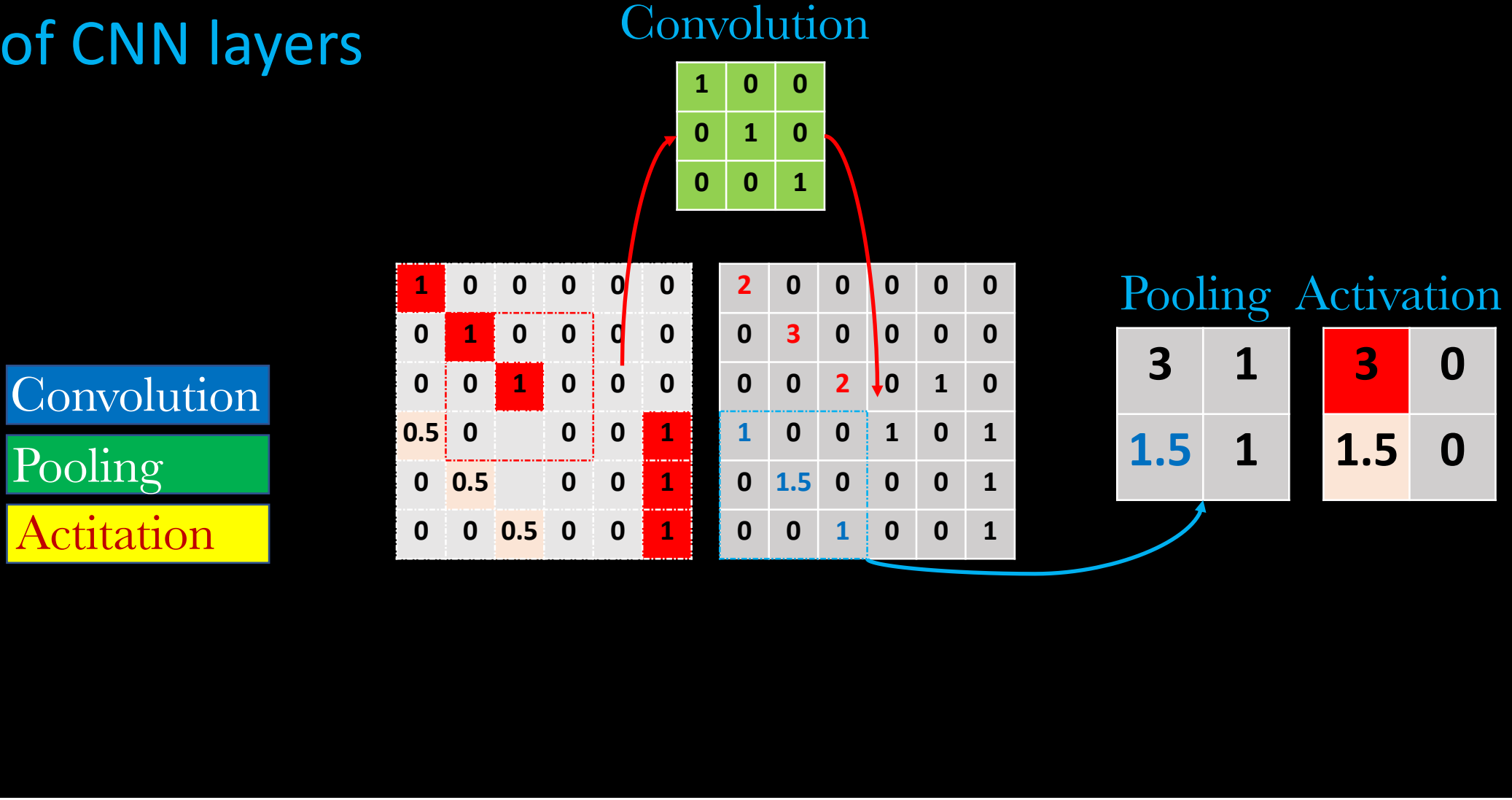
S=1, K=4, P=2

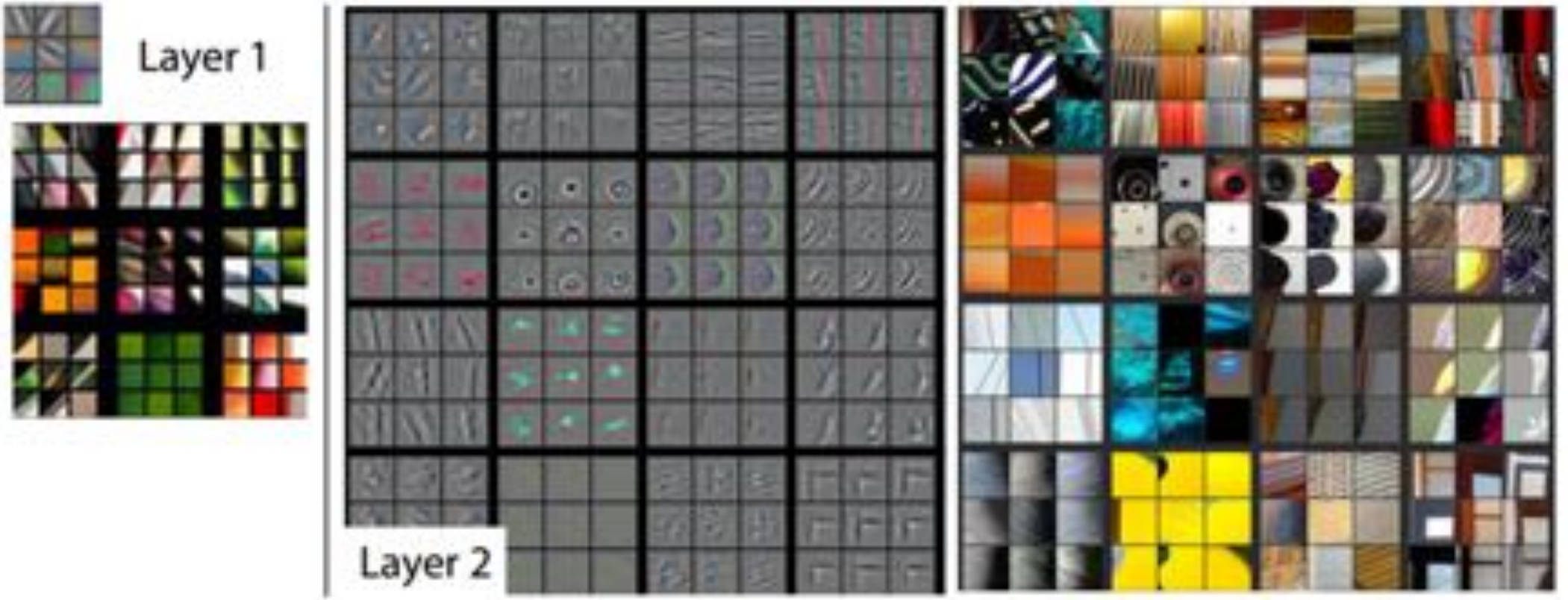


S=2, K=3 P=1

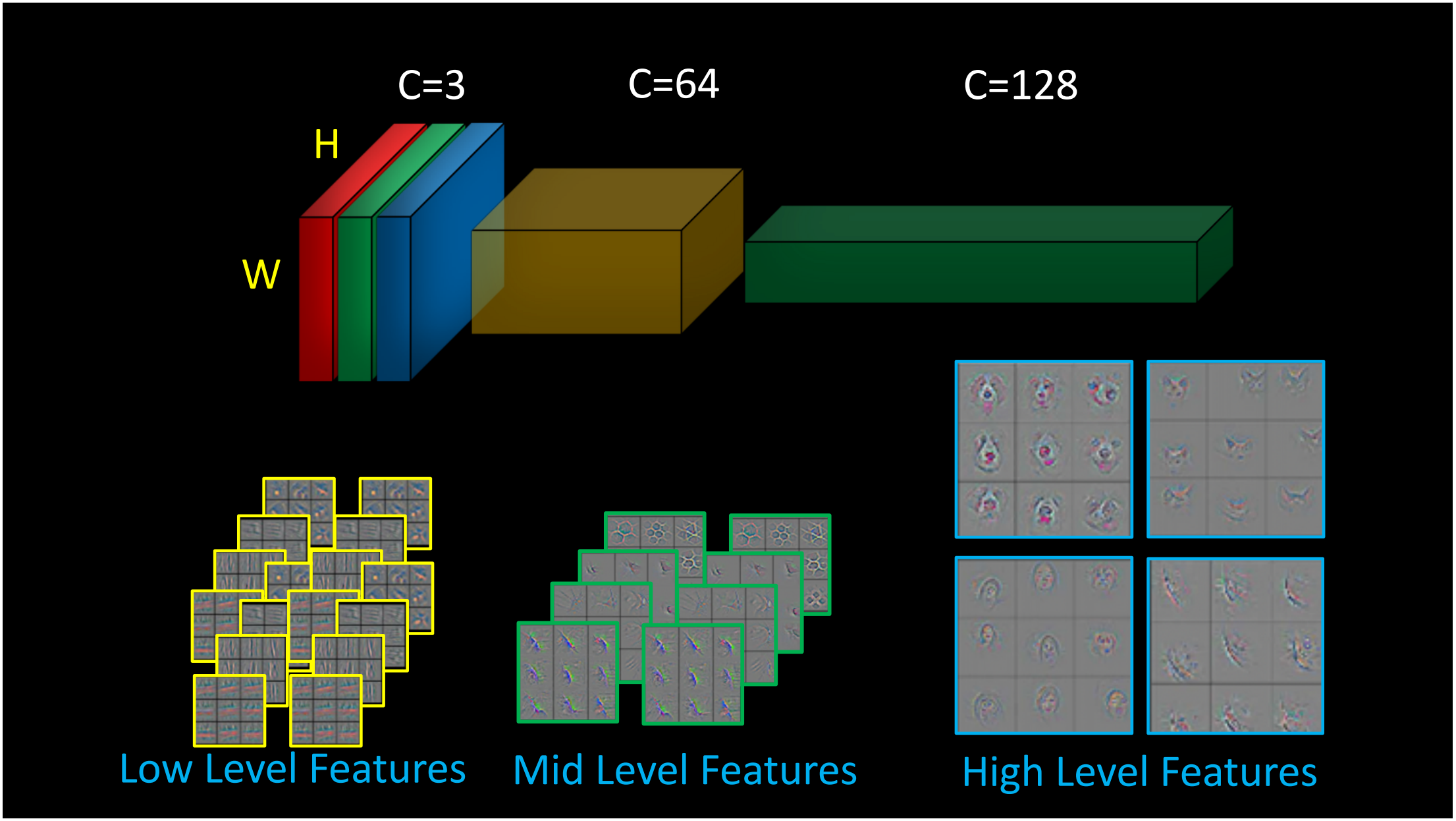
https://github.com/vdumoulin/conv_arithmetic

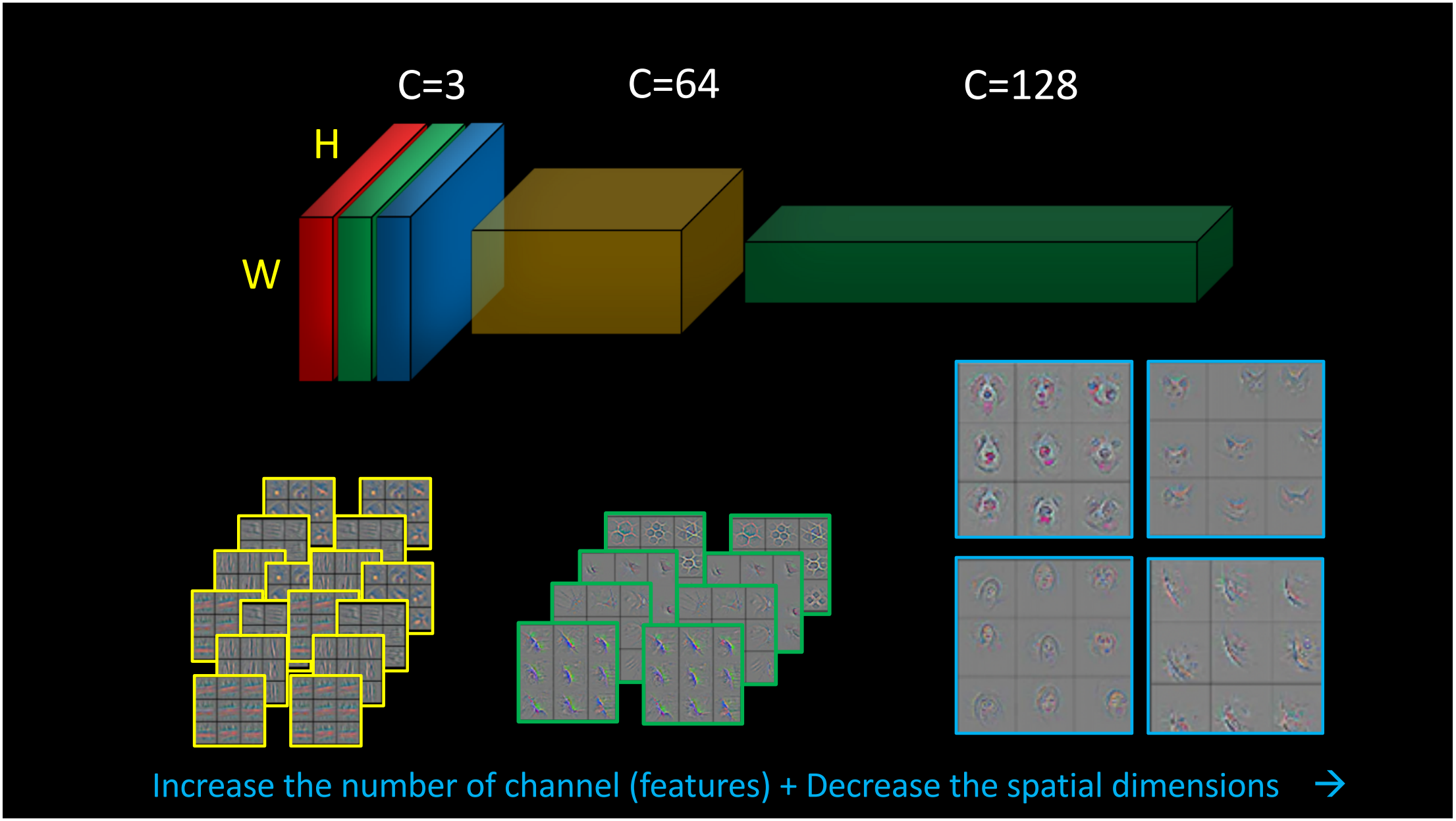
Basic "Stack" of CNN layers

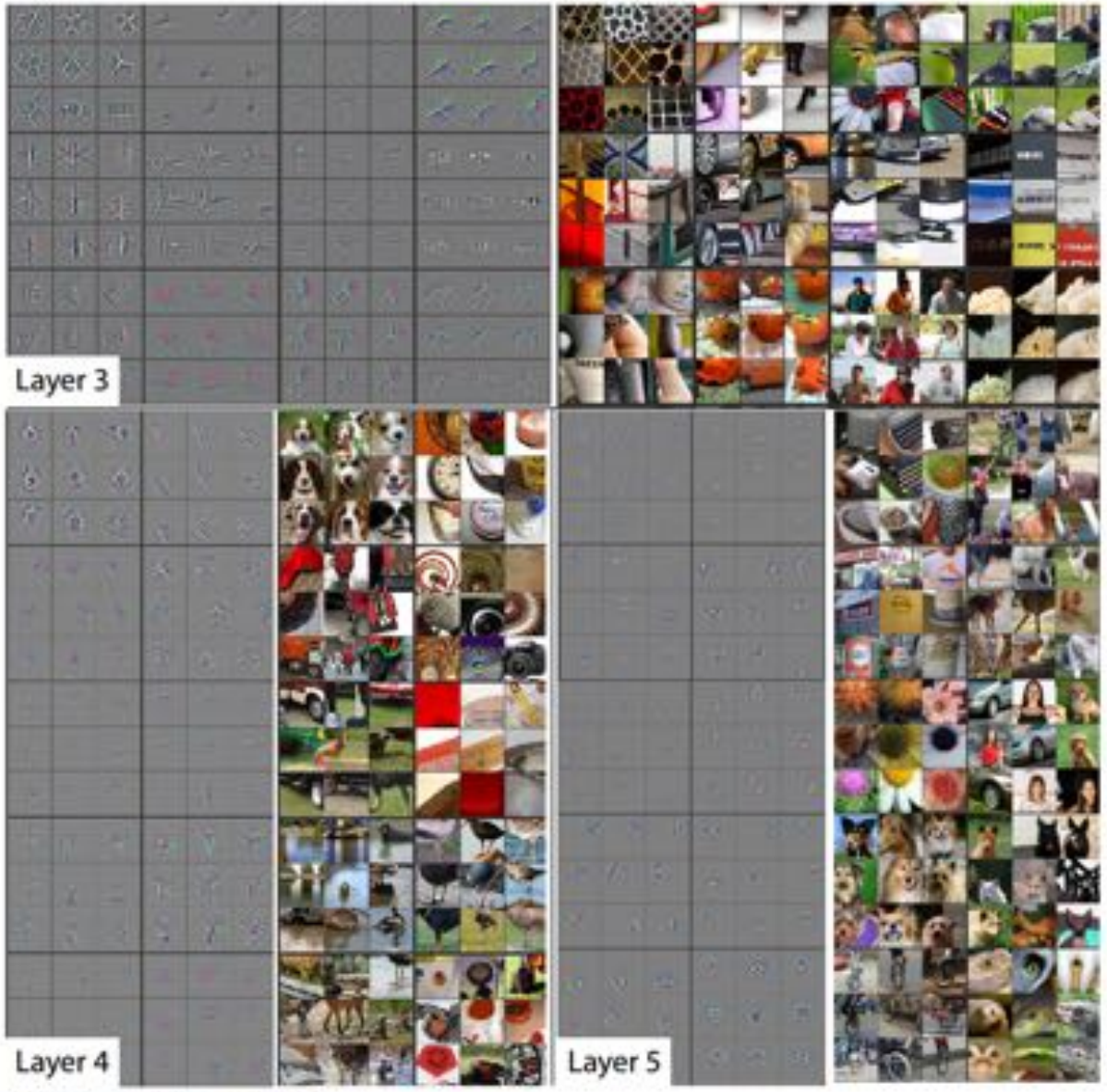




https://towardsdatascience.com/transfer-learning-using-pytorch-part-2-9c5b18e15551

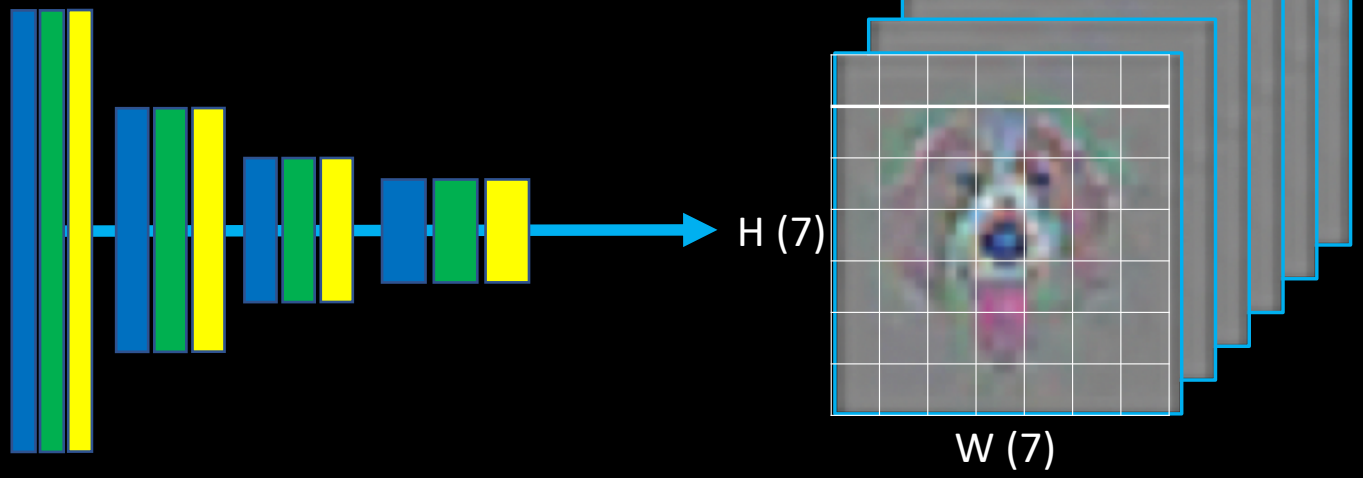

C=3
C=64
C=128
H
W
Low Level Features
Mid Level Features
High Level Features


C=3
C=64
C=128
H
W
Increase the number of channel (features) + Decrease the spatial dimensions →



Visualizations of Layers 3, 4, and 5

Encoded Feature map



C (512)
(number of
features)

H (7)

W (7)

Final Encoded
Feature map (512 x 7 x 7)

?????

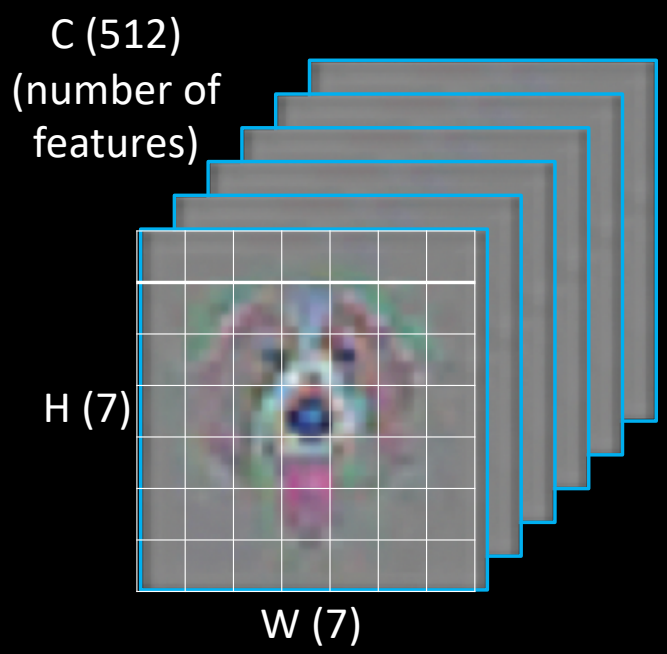
dog

car

tree

Probability

Encoded Feature map



(512 x 7 x 7)

Average over pixels
(mean pooling)

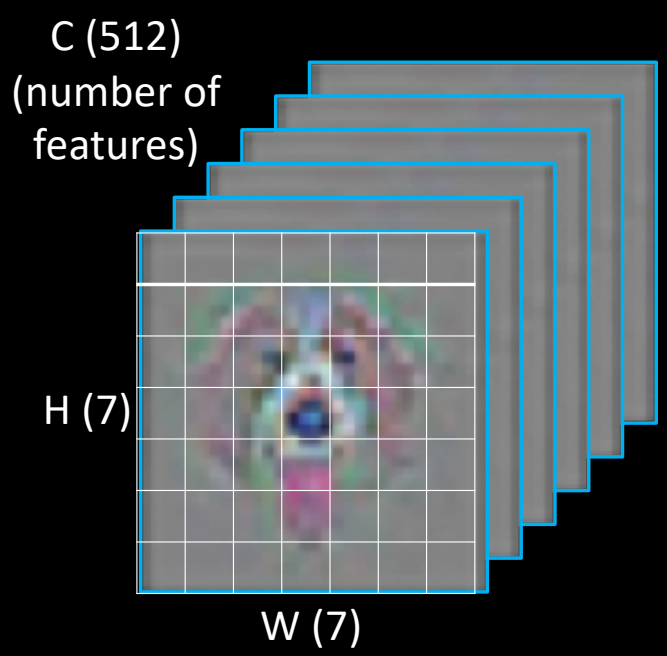
C=512

0.3
0.1
-3
4
0.9
-2

dog
car
tree

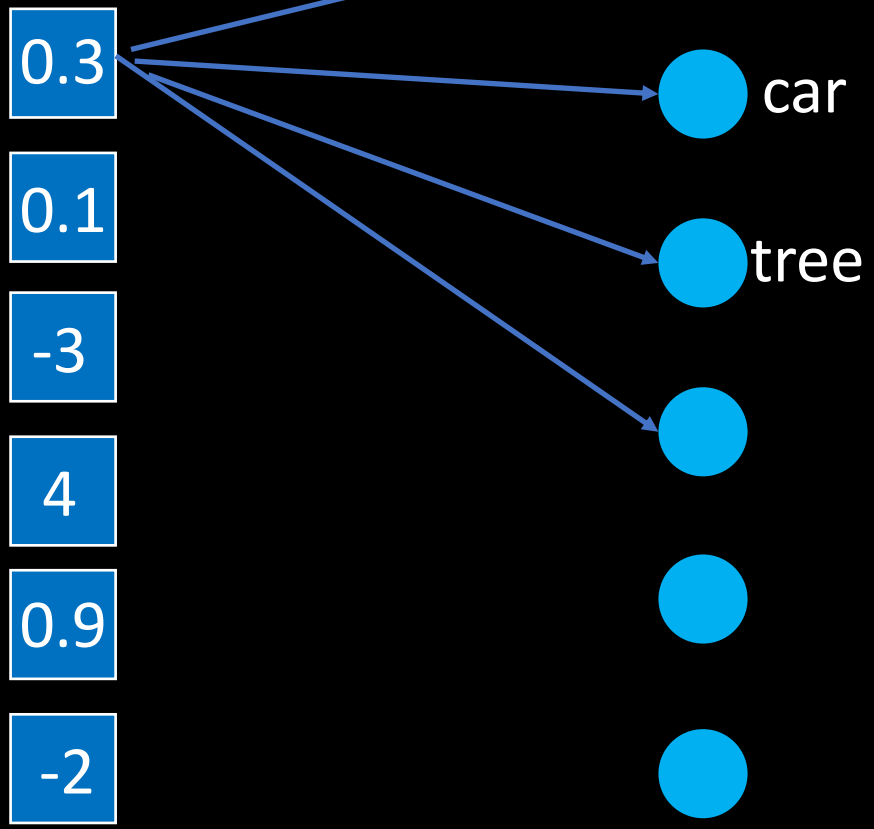
Probability

Encoded Feature map



Average over pixels
(mean pooling)

C=512

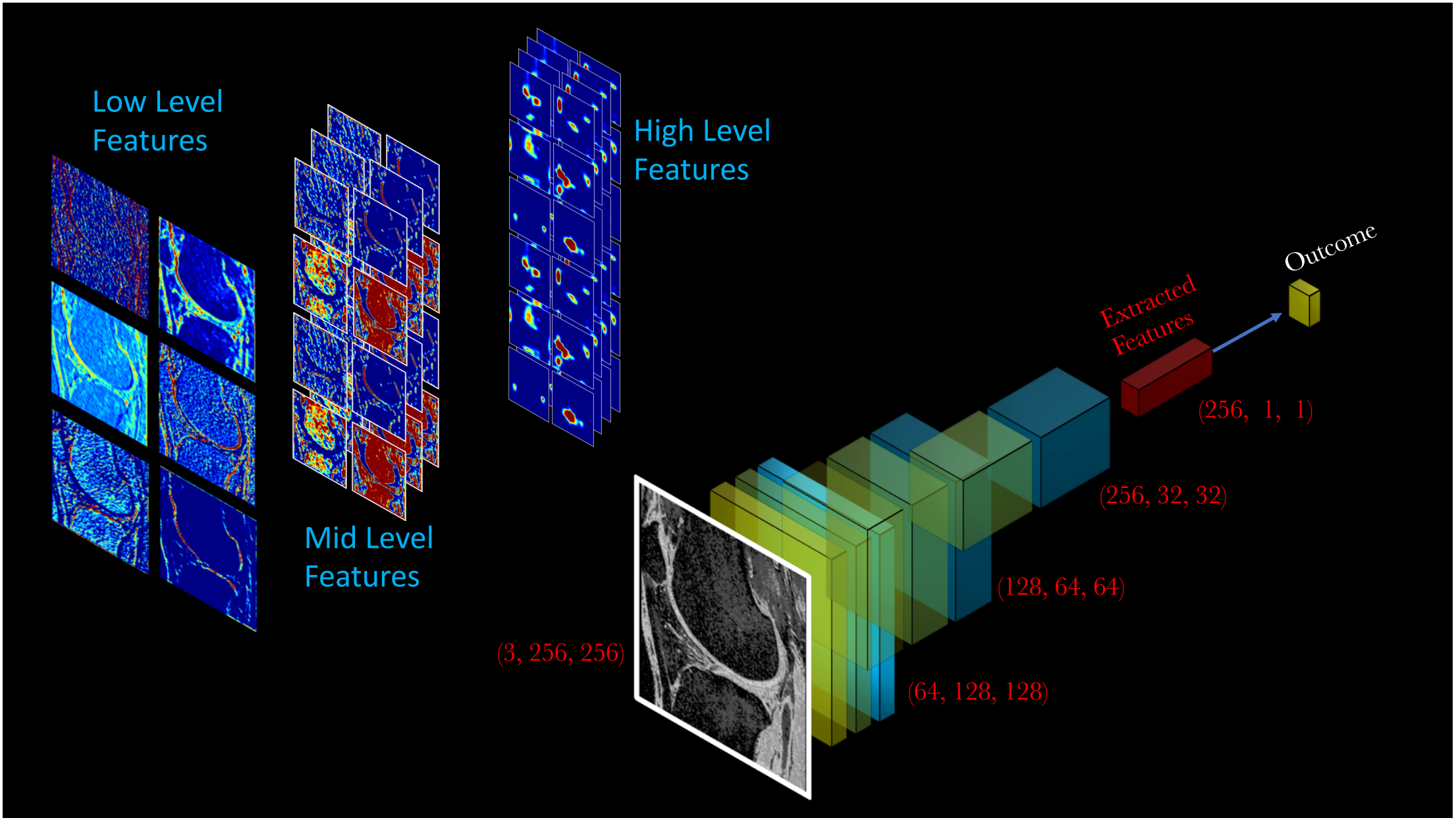


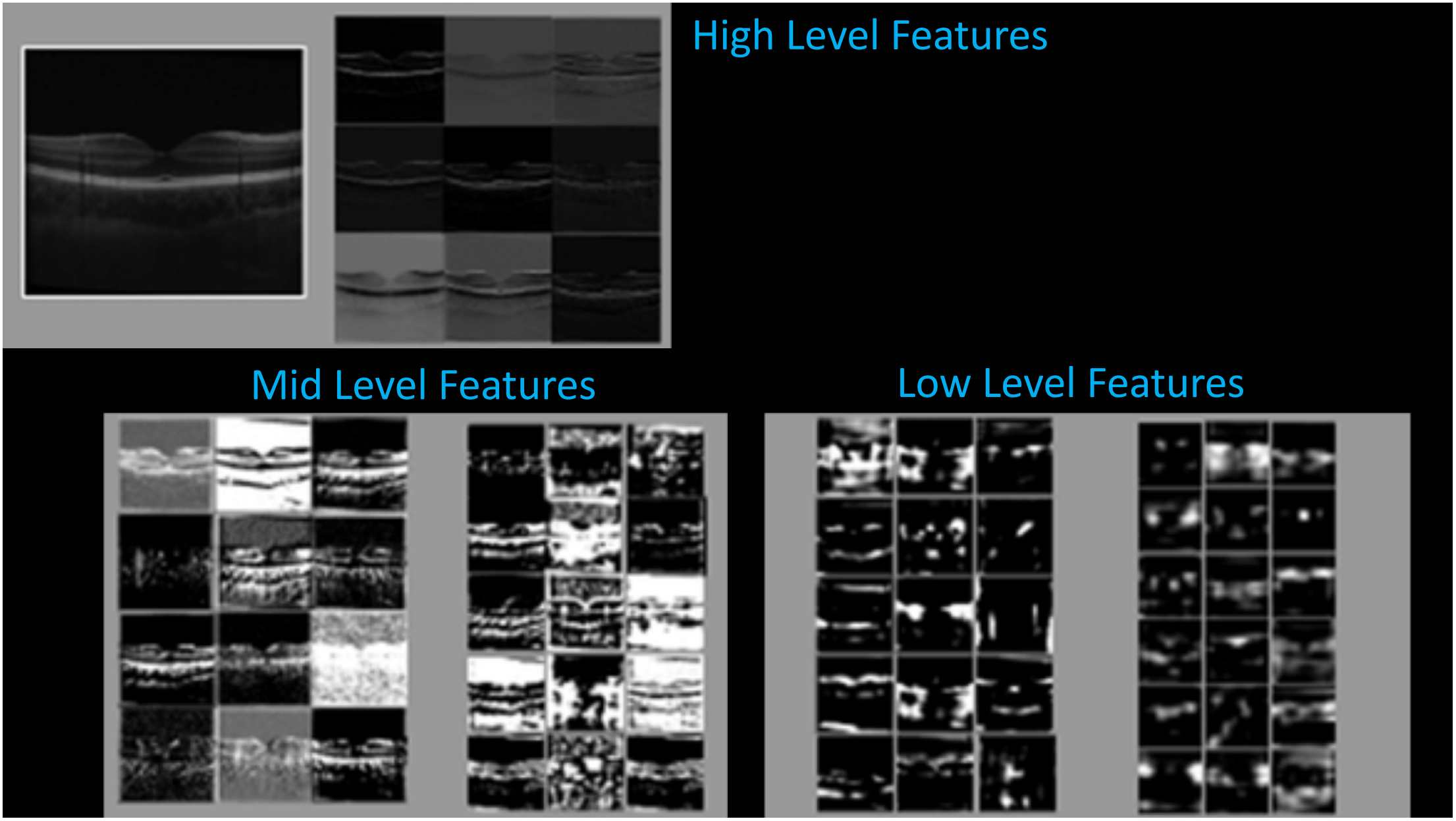
Feature Extractor

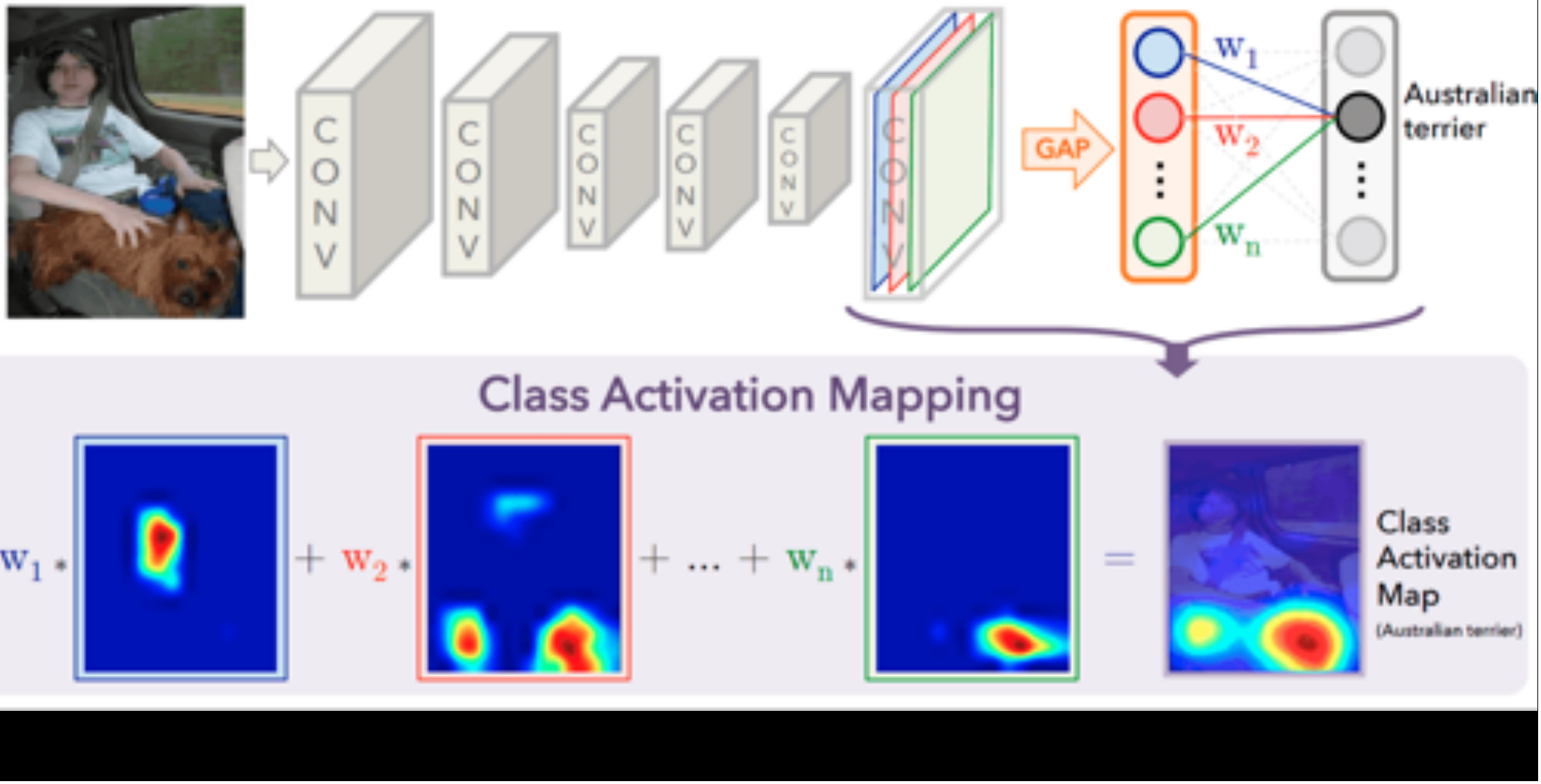
Classifier

ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224 × 224 RGB imag)					
conv3-64	conv3-64 **LRN**	conv3-64 **conv3-64**	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 **conv3-128**	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool					
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 **conv1-256**	conv3-256 conv3-256 **conv3-256**	conv3-256 conv3-256 conv3-256 **conv3-256**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

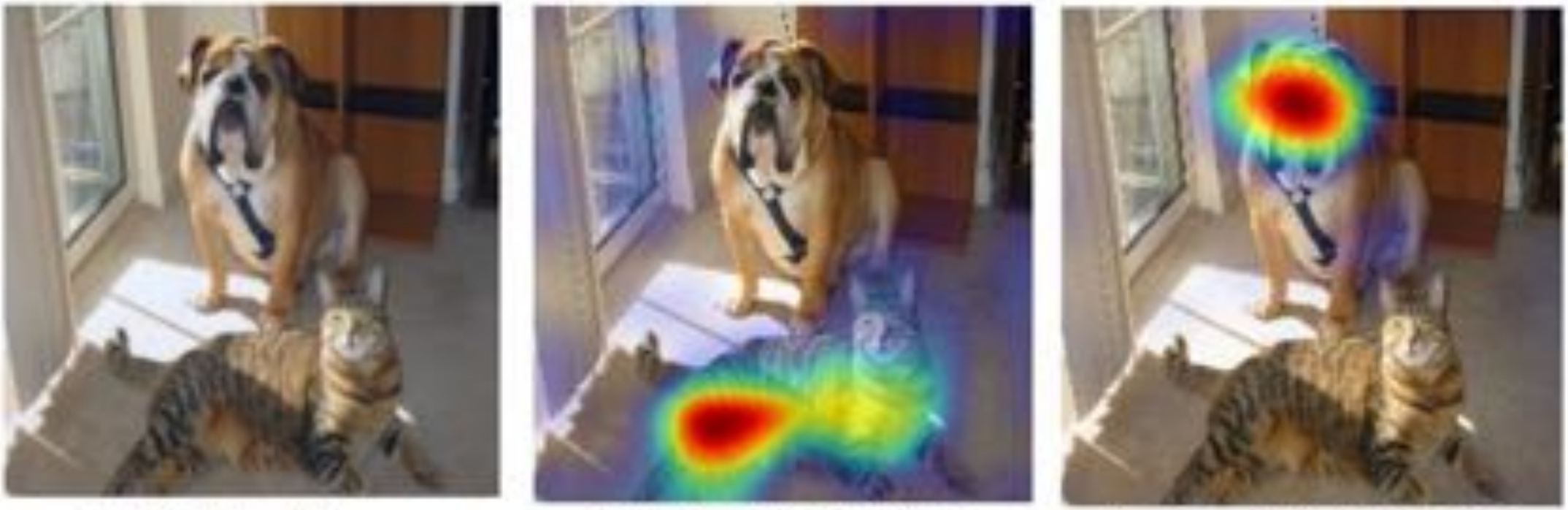
```
torchvision.models.vgg11()
VGG(
  (features): Sequential(
    (0): Conv2d(3, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): ReLU(inplace=True)
    (2): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (3): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (4): ReLU(inplace=True)
    (5): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (6): Conv2d(128, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (7): ReLU(inplace=True)
    (8): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (9): ReLU(inplace=True)
    (10): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (11): Conv2d(256, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (12): ReLU(inplace=True)
    (13): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (14): ReLU(inplace=True)
    (15): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (16): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (17): ReLU(inplace=True)
    (18): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (19): ReLU(inplace=True)
    (20): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  )
```


Low Level Features
Mid Level Features
High Level Features
(3, 256, 256)
(64, 128, 128)
(128, 64, 64)
(256, 32, 32)
Extracted Features
(256, 1, 1)
Outcome


High Level Features
Mid Level Features
Low Level Features

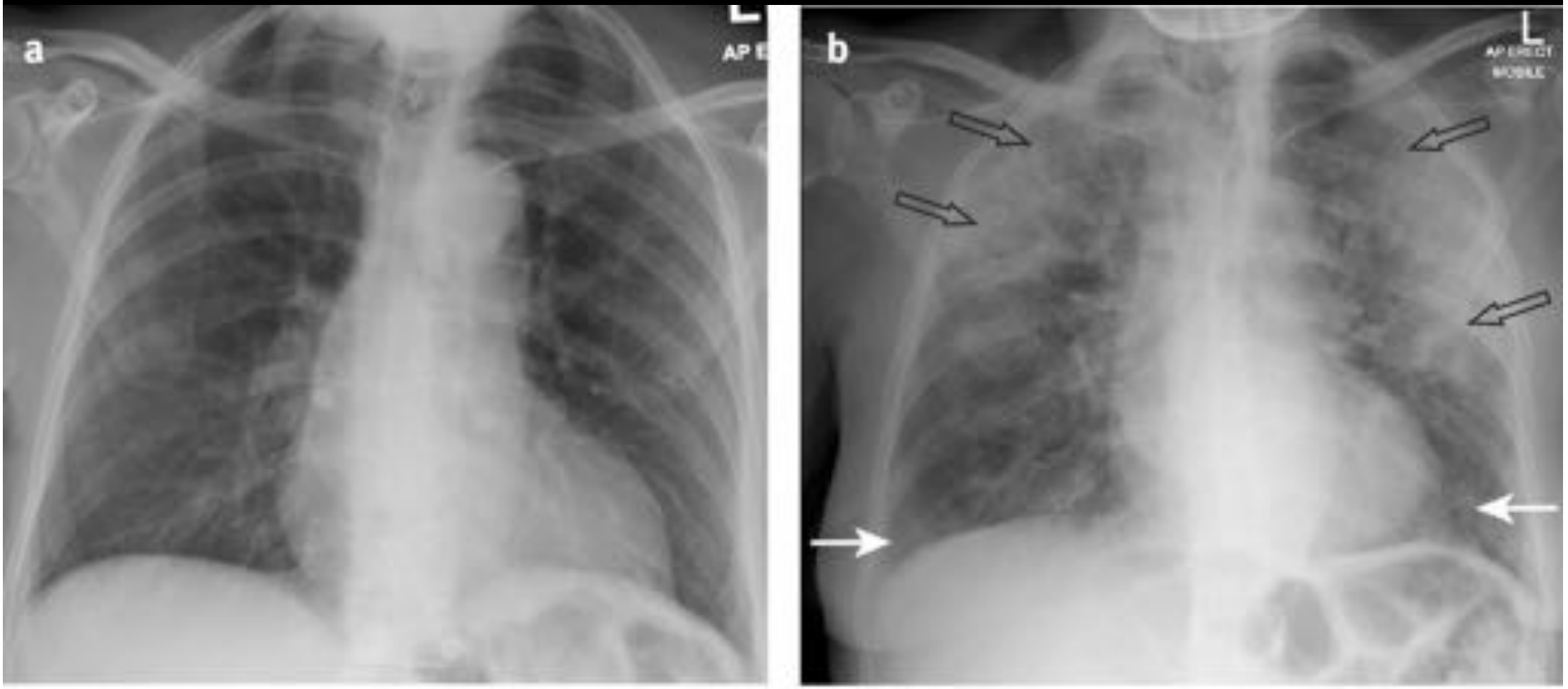

CONV
CONV
CONV
CONV
CONV
CONV
GAP
w_1
w_2
w_n
Australian terrier
Class Activation Mapping
w_1 *
+ w_2 *
+ ... + w_n *
=
Class Activation Map
(Australian terrier)

Nature Imaging: Often Large Discriminative Parts



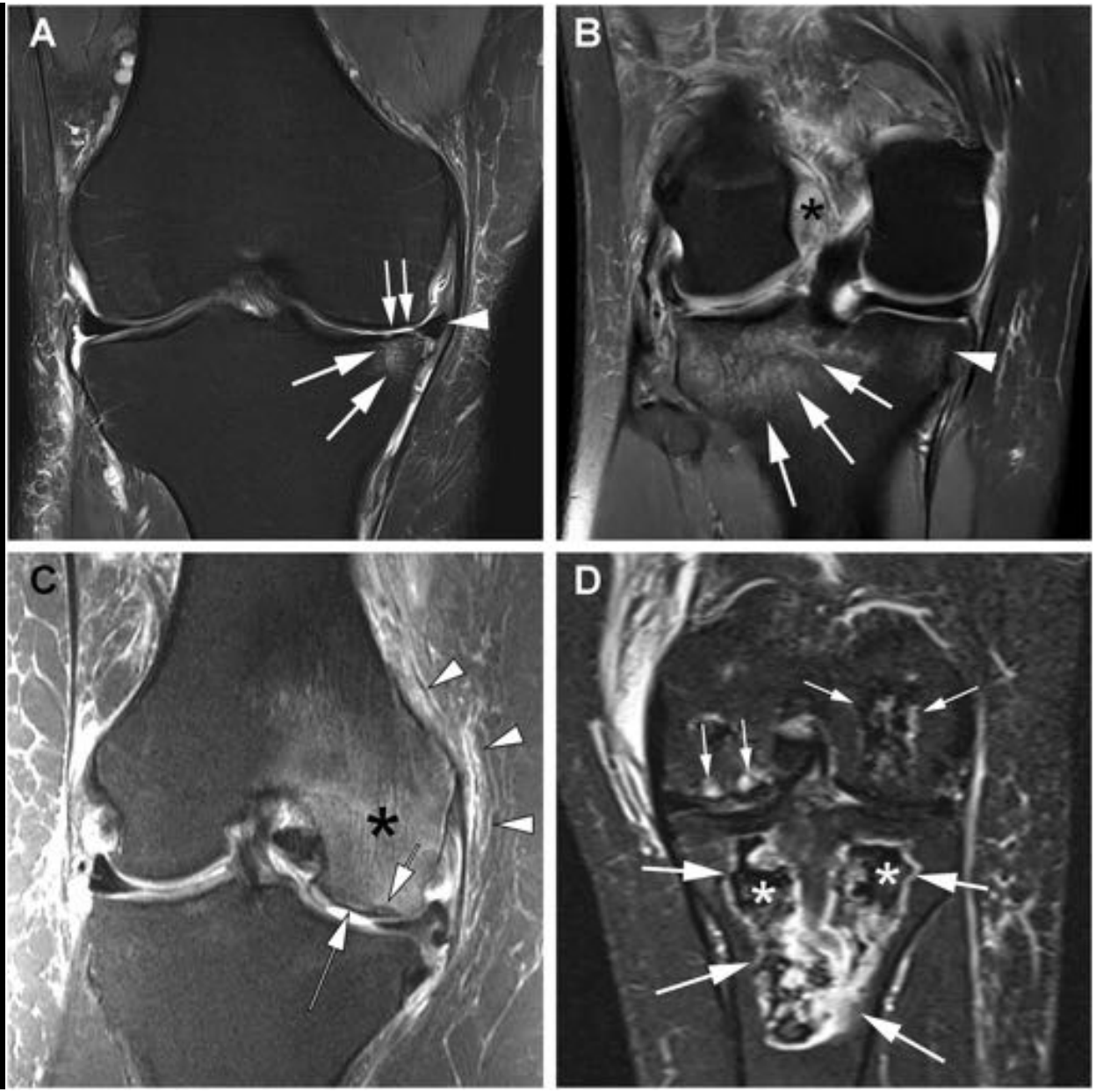
(a) Original Image (b) Grad-CAM 'Cat' (c) Grad-CAM 'Dog'

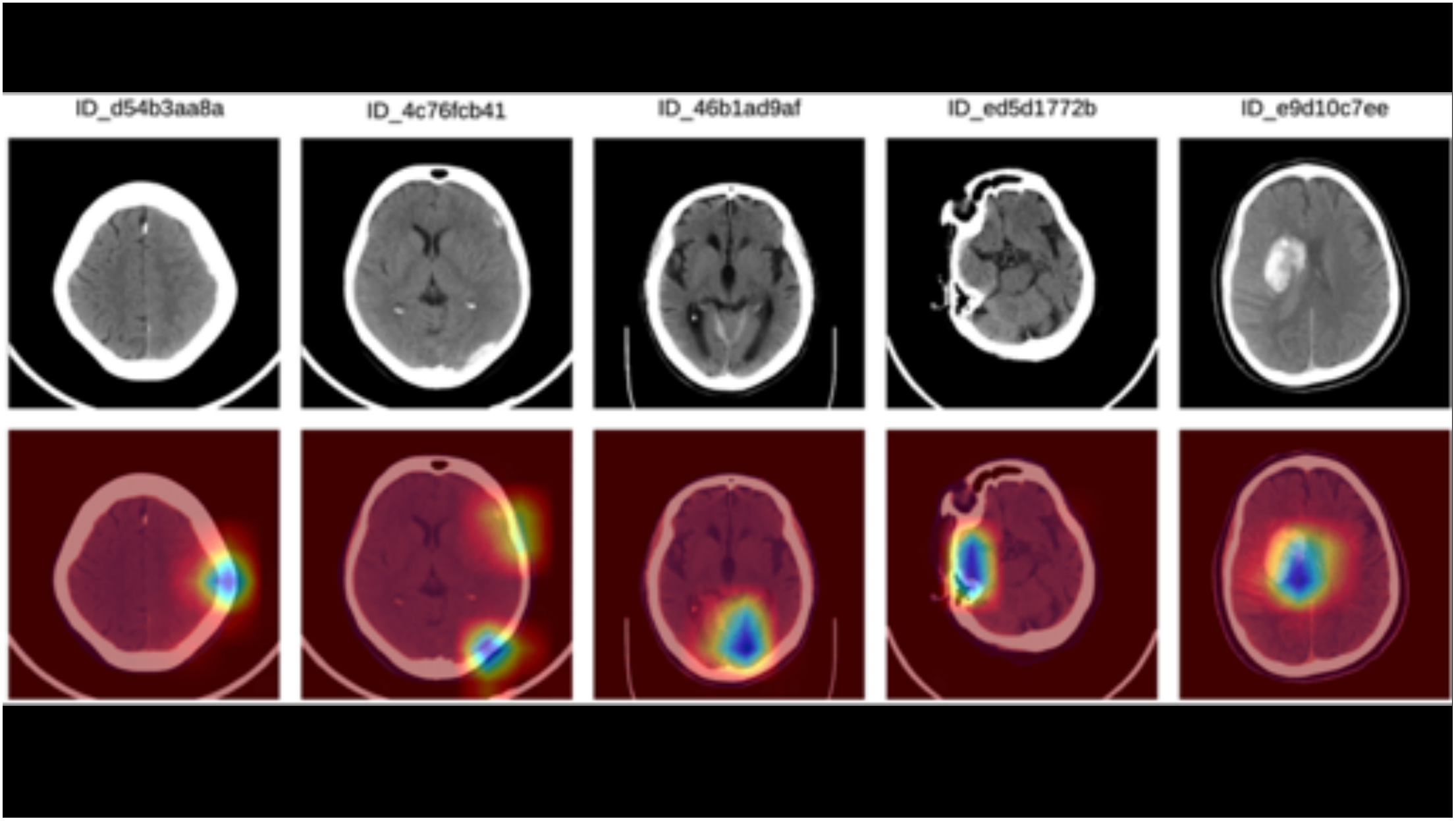
Medical Imaging: Fine-grained Discriminative Parts

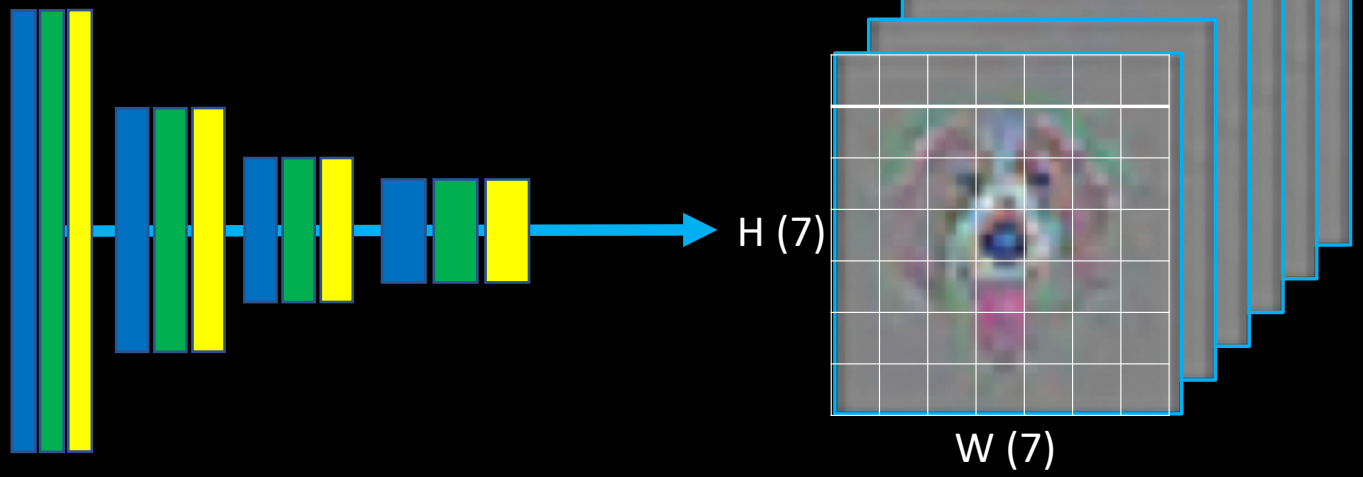


Medical Imaging:
Fine-grained Discriminative Parts

Small in scale != Low level feature

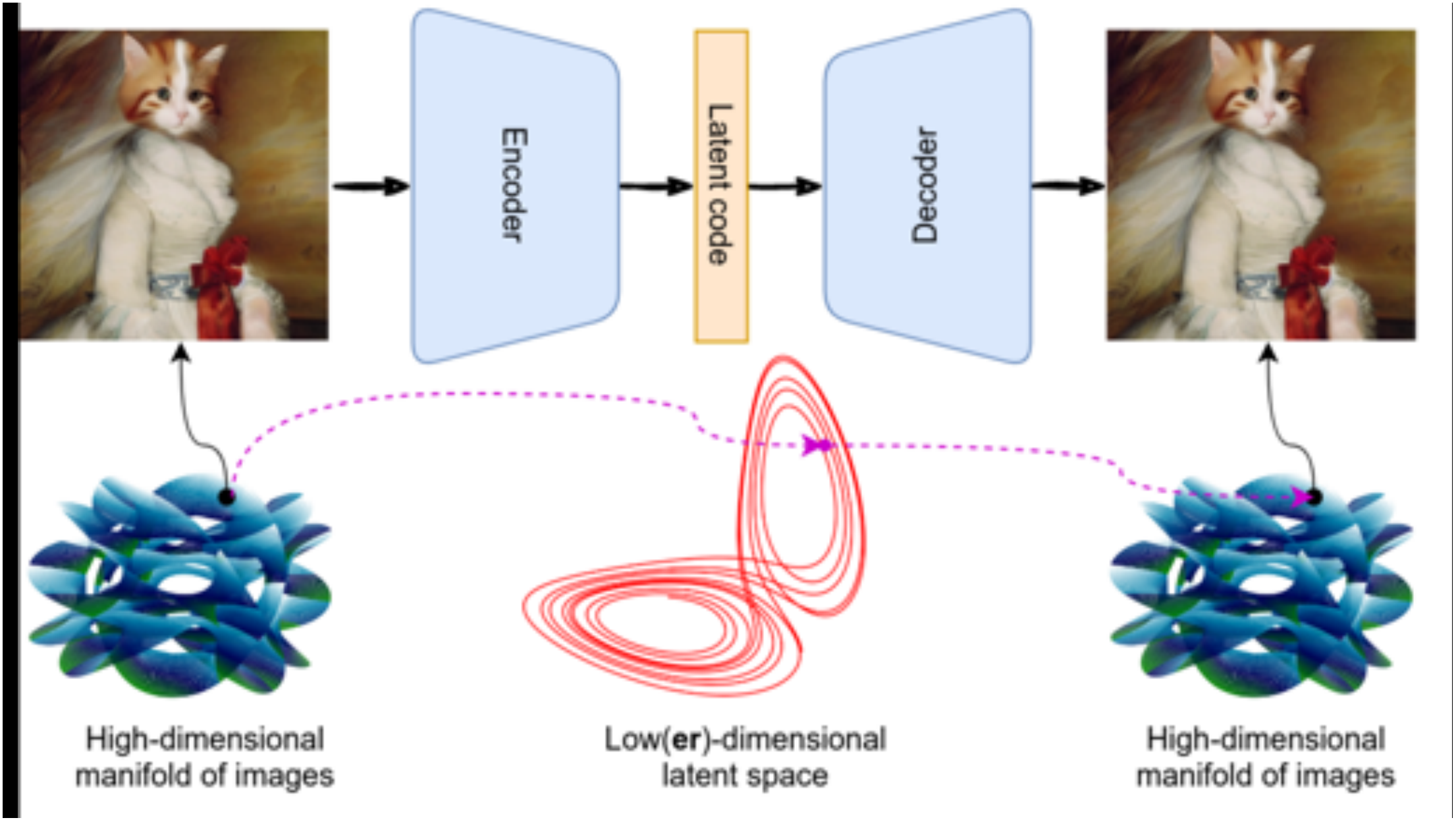


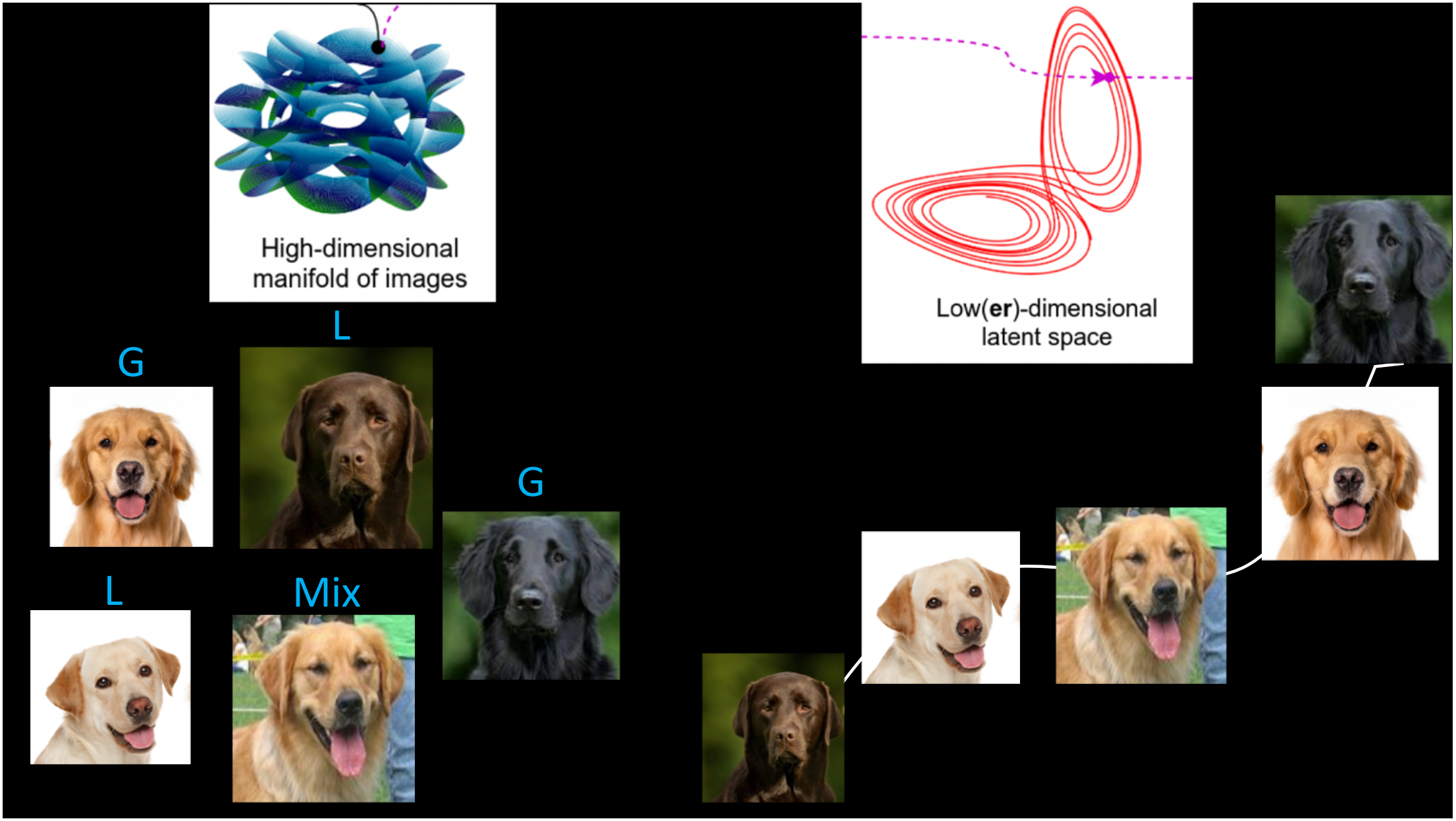

ID_d54b3aa8a
ID_4c76fcb41
ID_46b1ad9af
ID_ed5d1772b
ID_e9d10c7ee

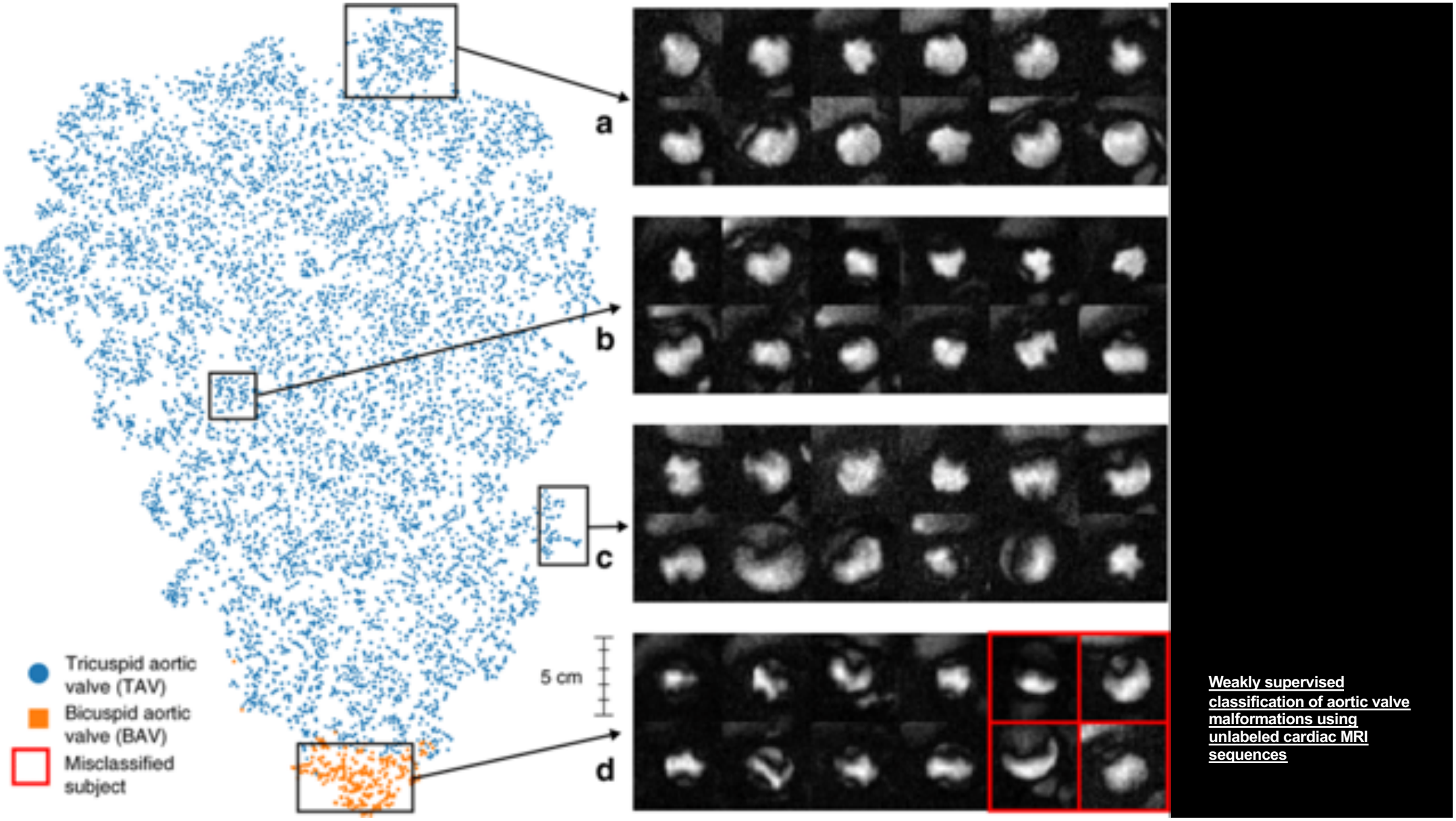


Final Encoded
Feature map (512 x 7 x 7)

“latent space”

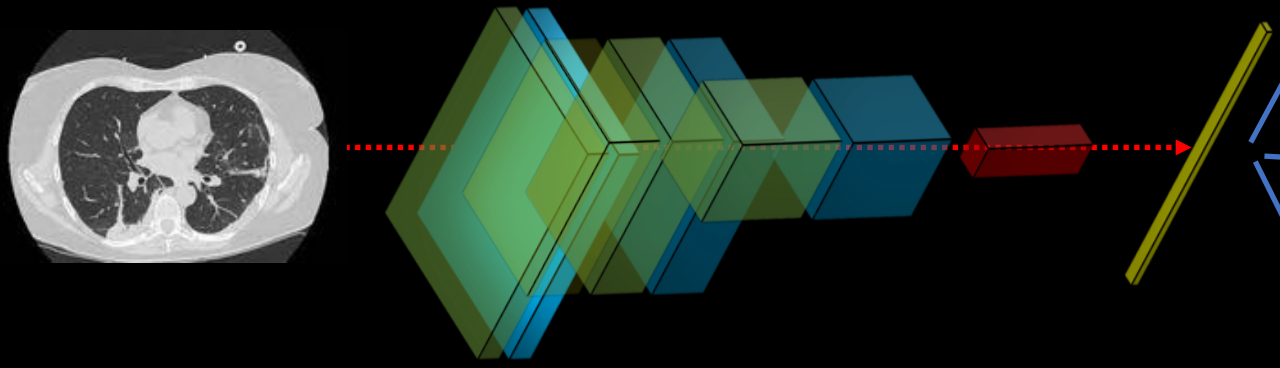

Encoder
Latent code
Decoder
High-dimensional
manifold of images
Low(er)-dimensional
latent space
High-dimensional
manifold of images


High-dimensional
manifold of images
Low(er)-dimensional
latent space
G
L
G
L
Mix



Weakly supervised classification of aortic valve malformations using unlabeled cardiac MRI sequences

A good latent space include the essential information about the data
&
Can you use for others tasks



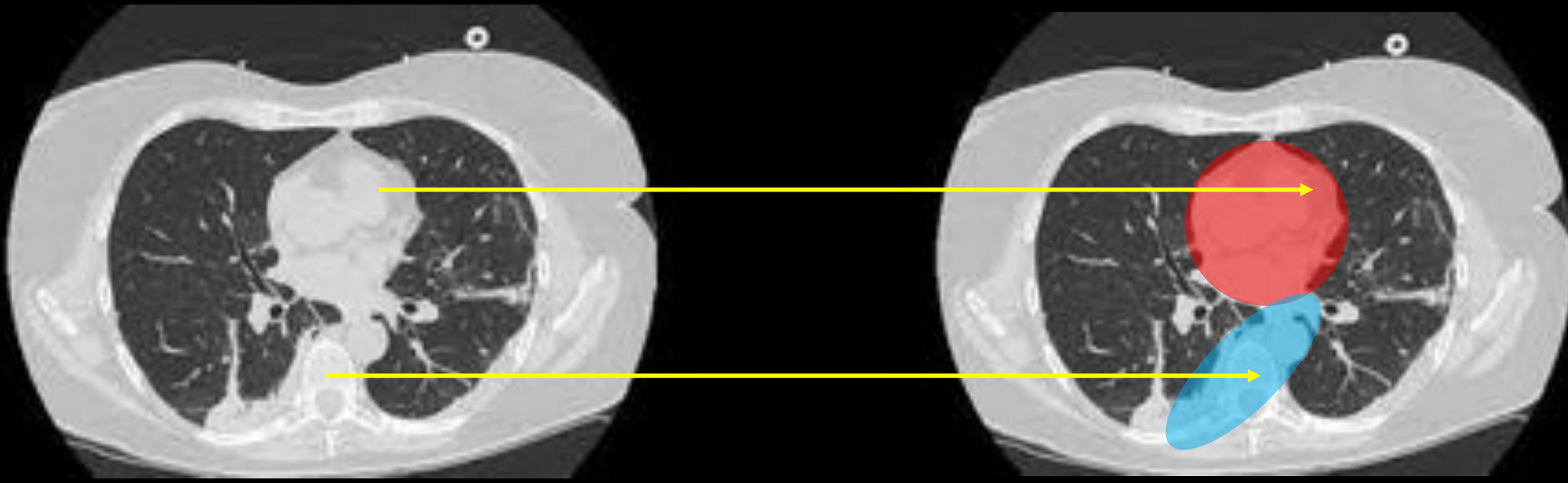
Classification

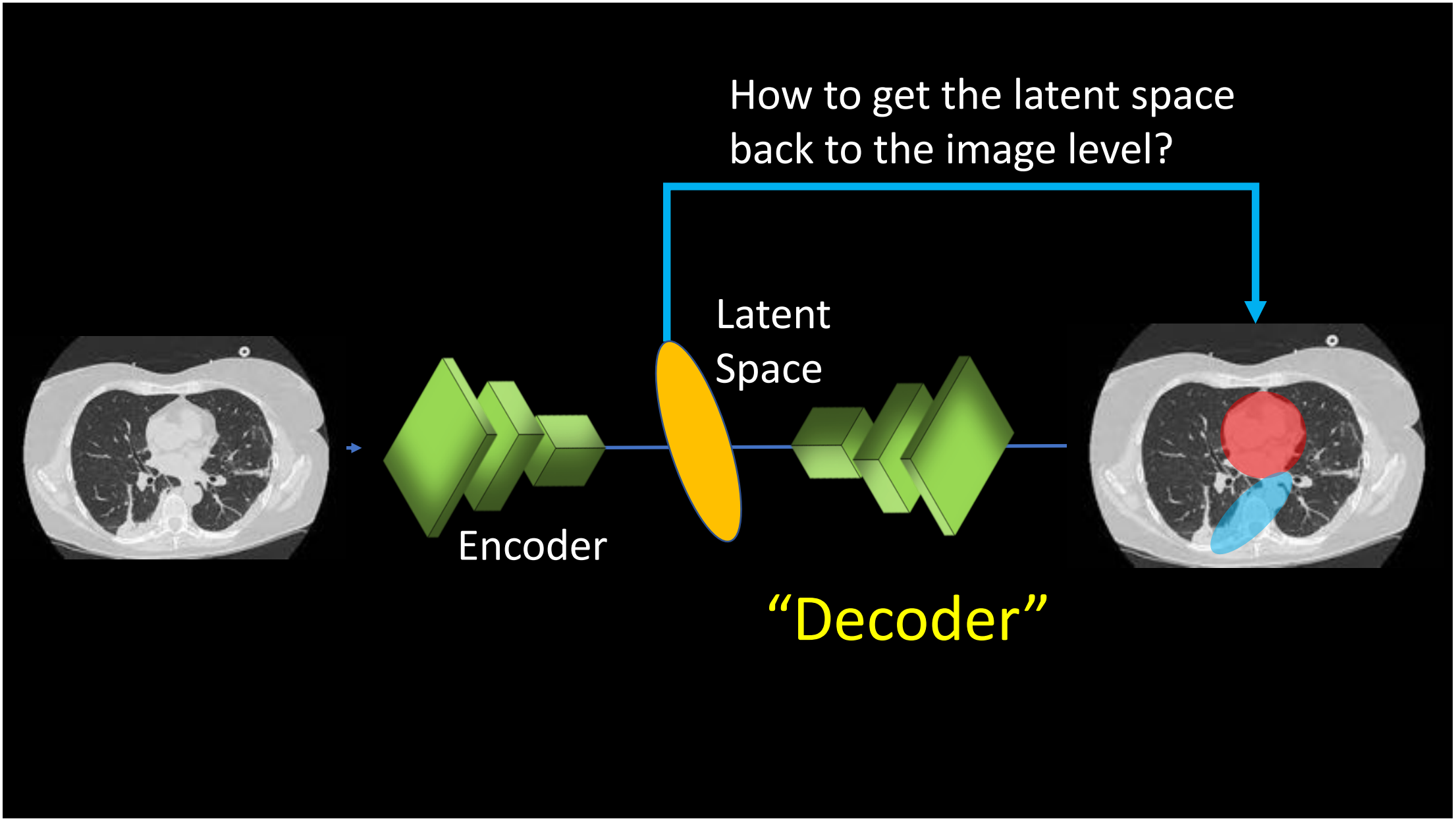
Detetion

Segmentation

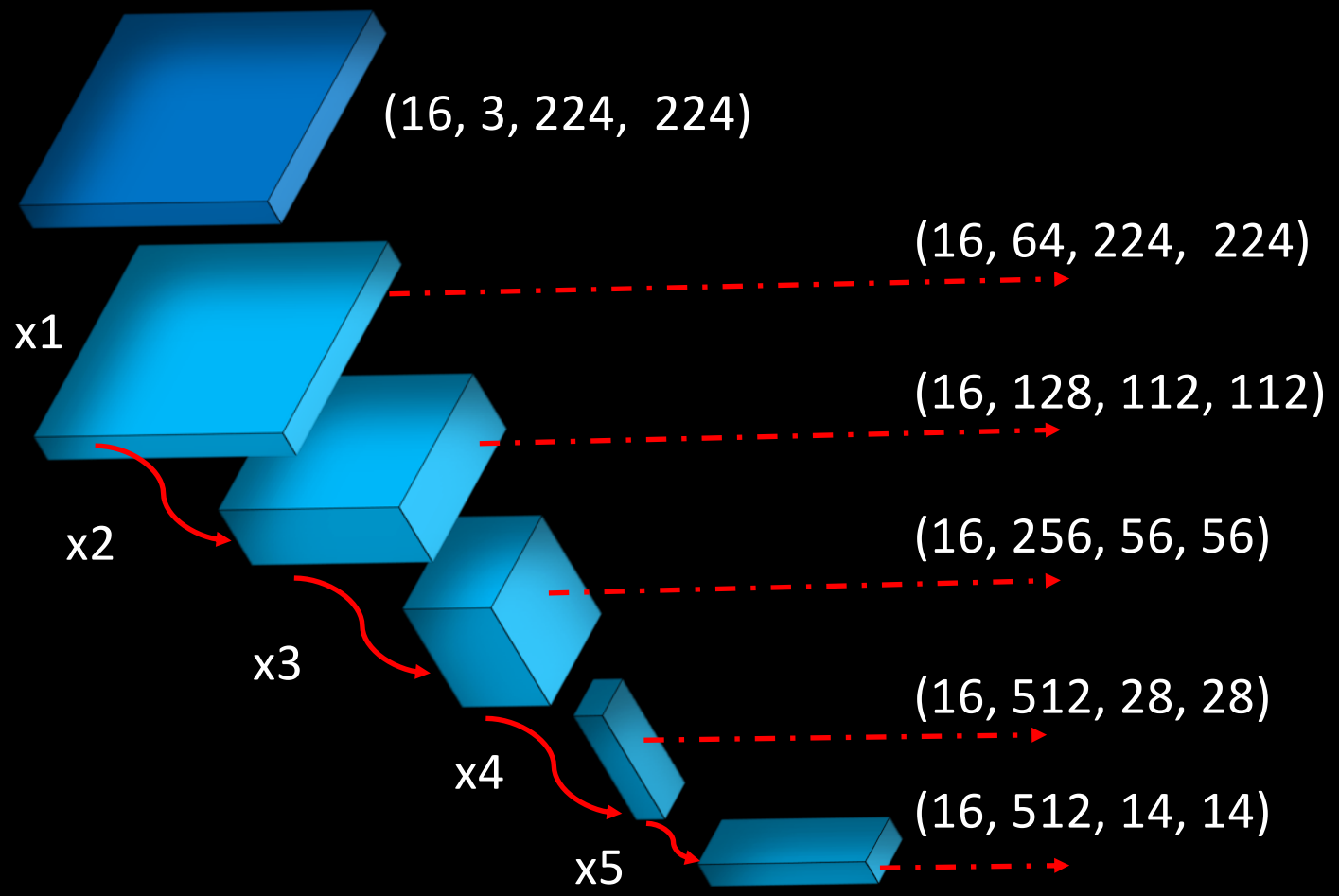
Dense prediction (ex: segmentation):

You need 1 on 1 prediction per image pixel

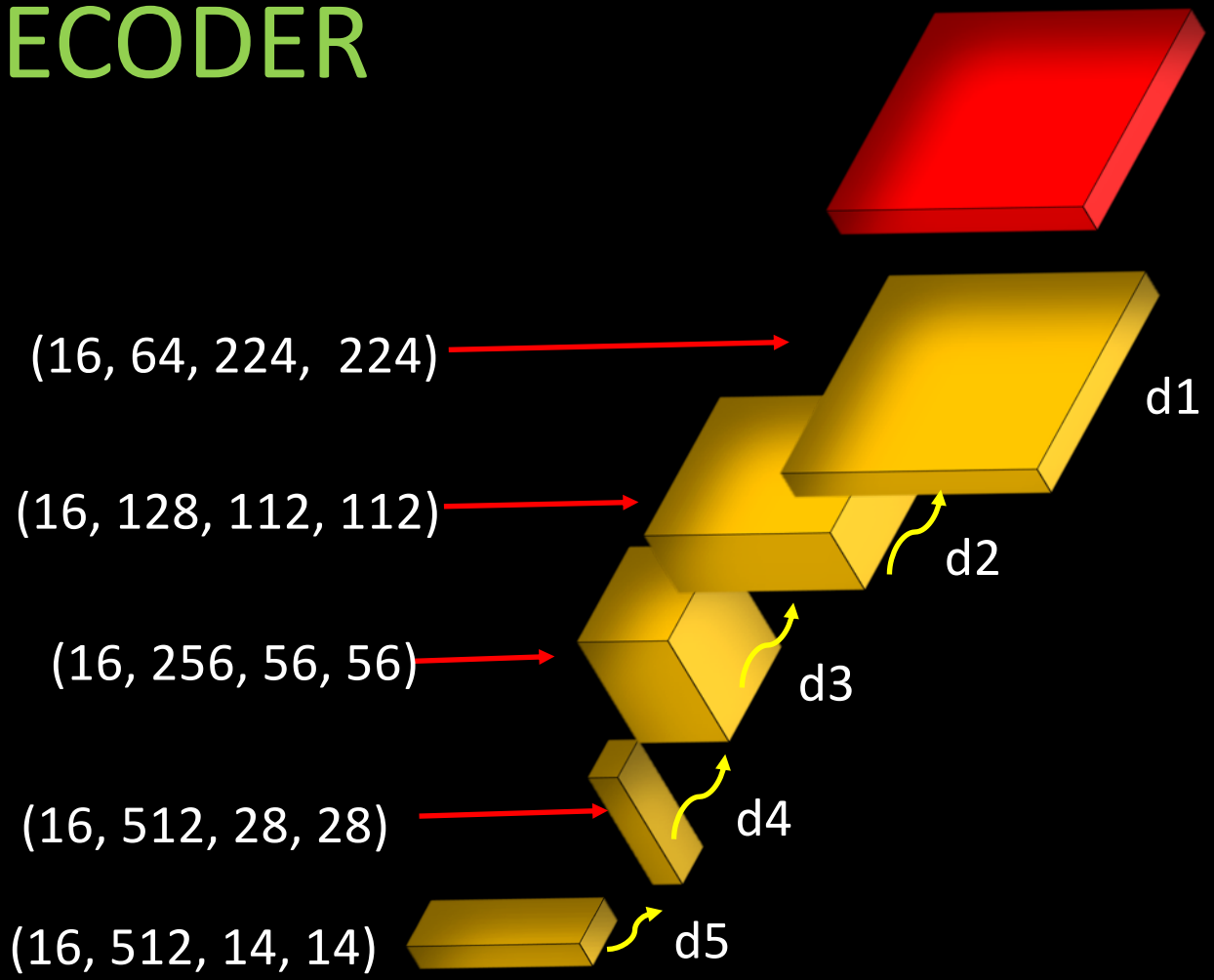



How to get the latent space
back to the image level?
Latent
Space
Encoder
“Decoder”

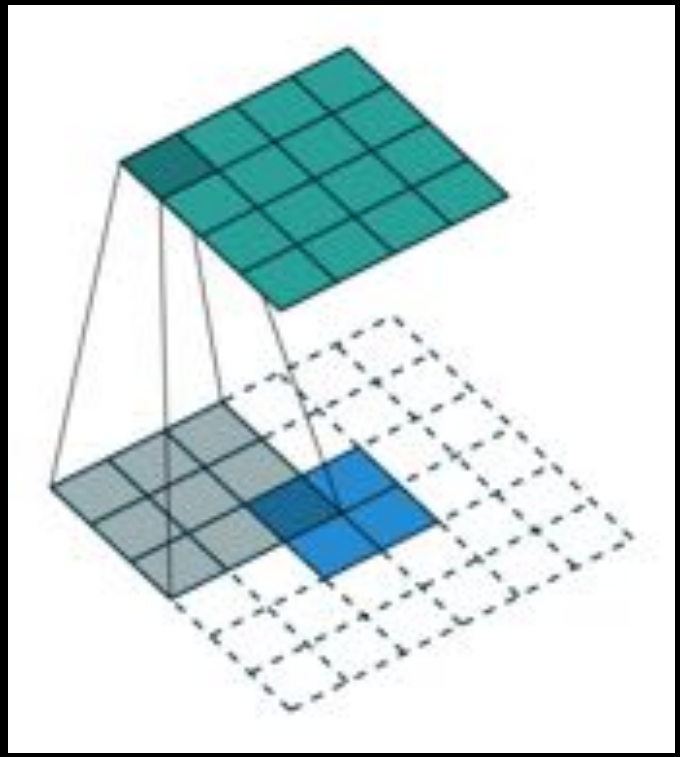
ENCODER

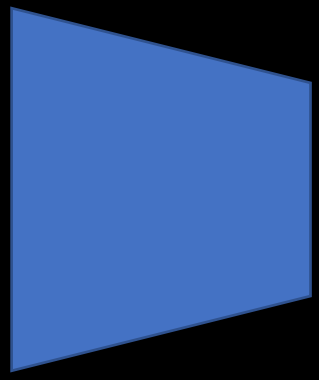

(16, 3, 224, 224)
(16, 64, 224, 224)
x1
(16, 128, 112, 112)
x2
(16, 256, 56, 56)
x3
(16, 512, 28, 28)
x4
(16, 512, 14, 14)
x5

DECODER

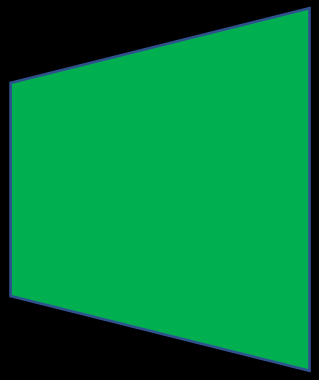


"transpose convolution"

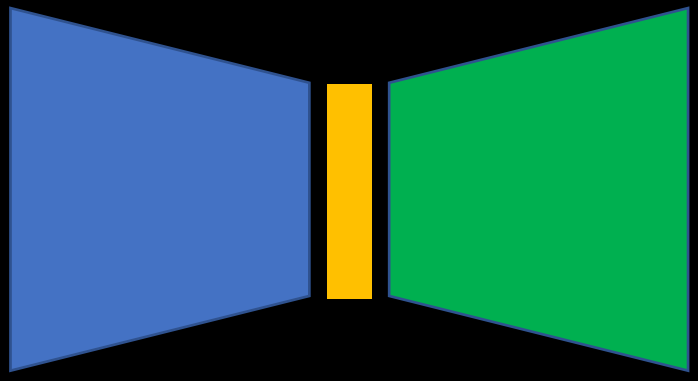




Encoder:
Image to features

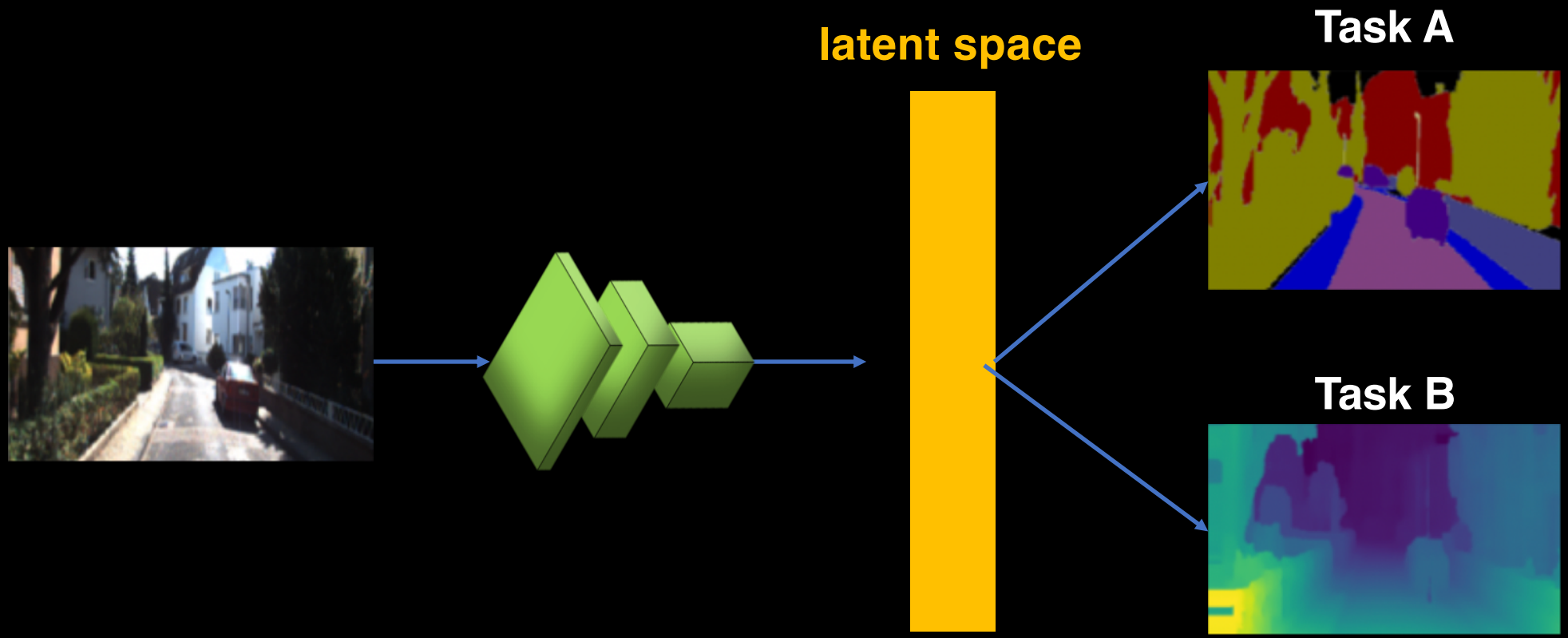


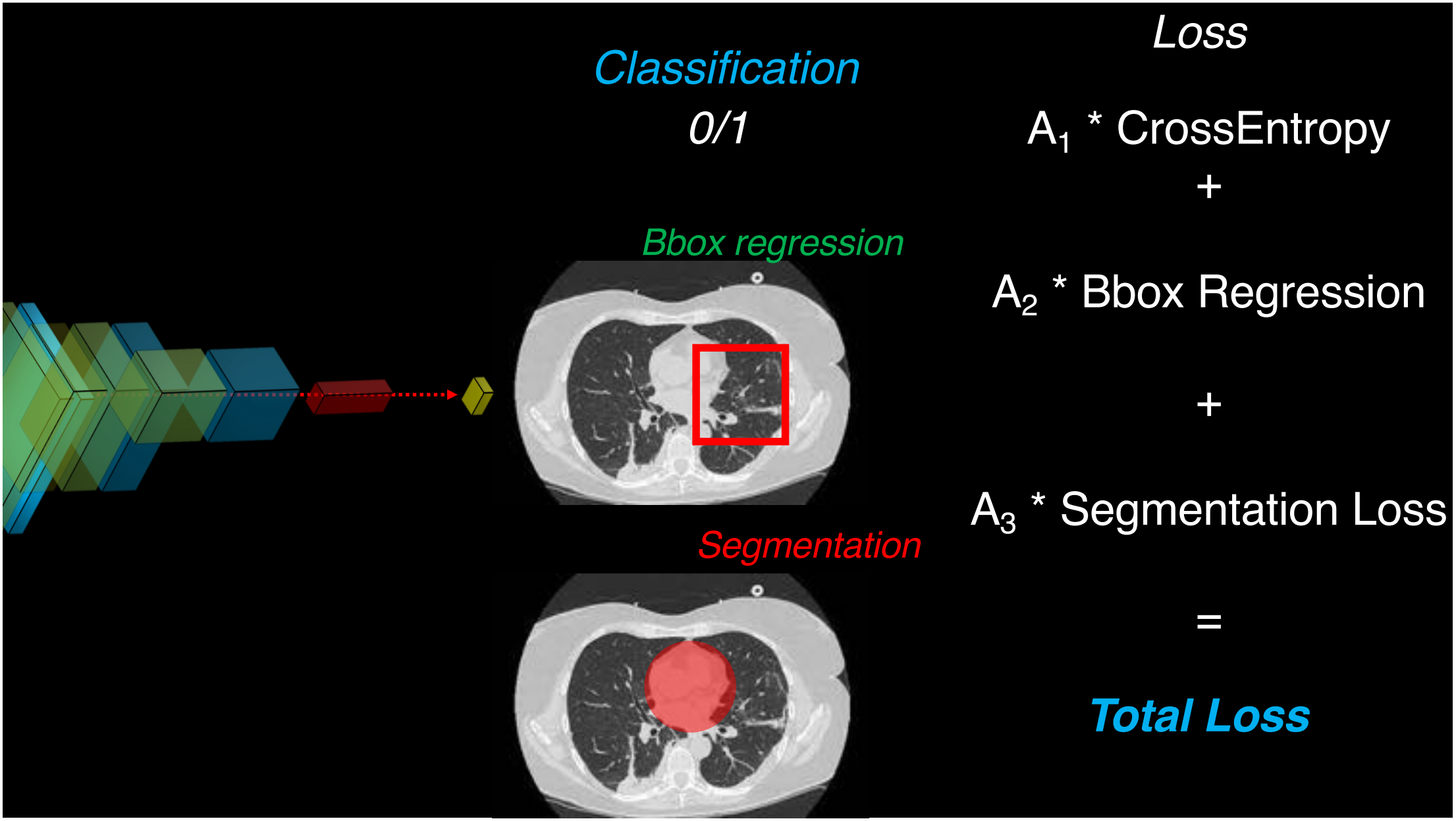
decoder:
Features to image



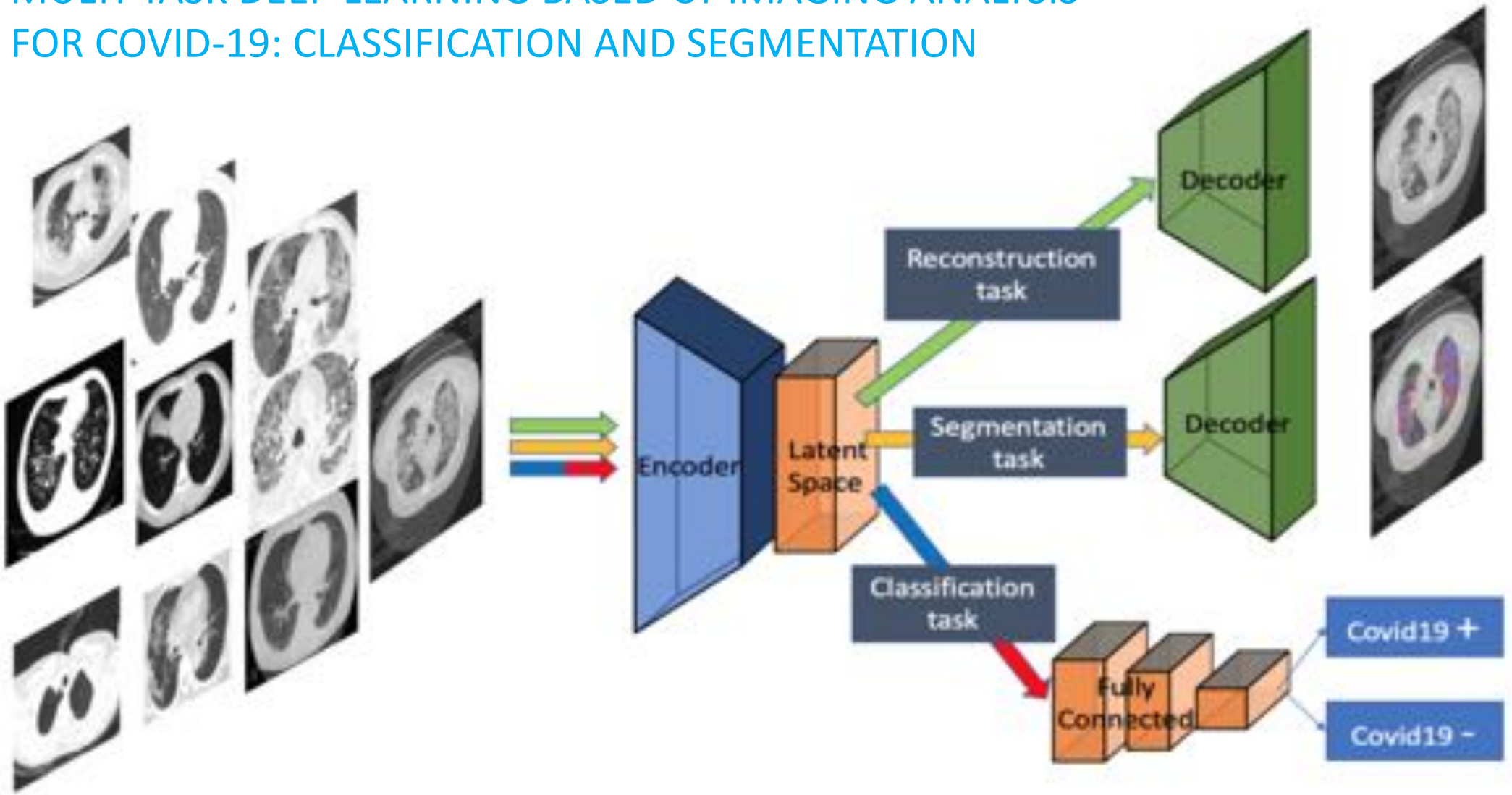
Encoder-decoder:
Image to features then to image
(why?)

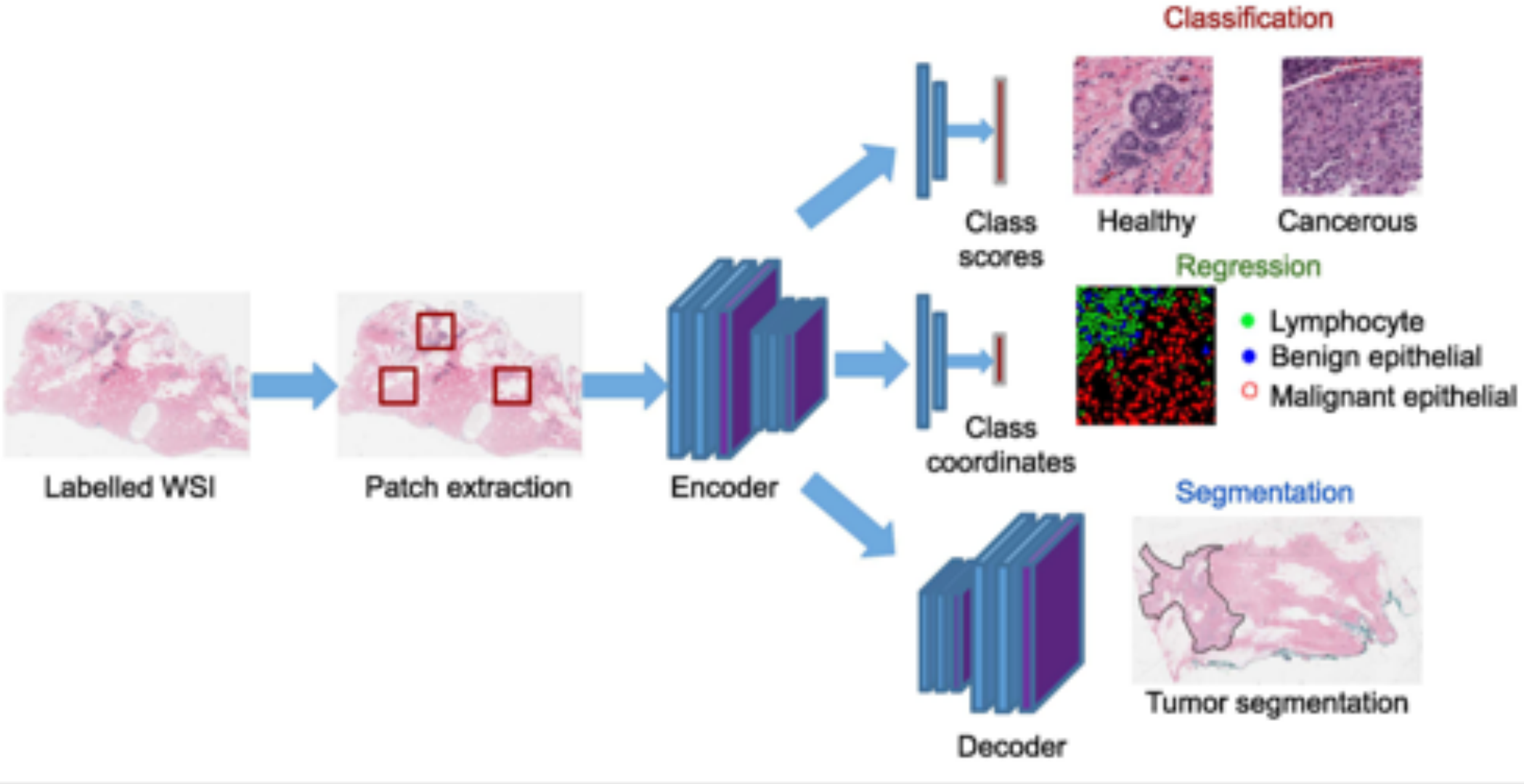
A good latent space allow you to Perform multitasking at once

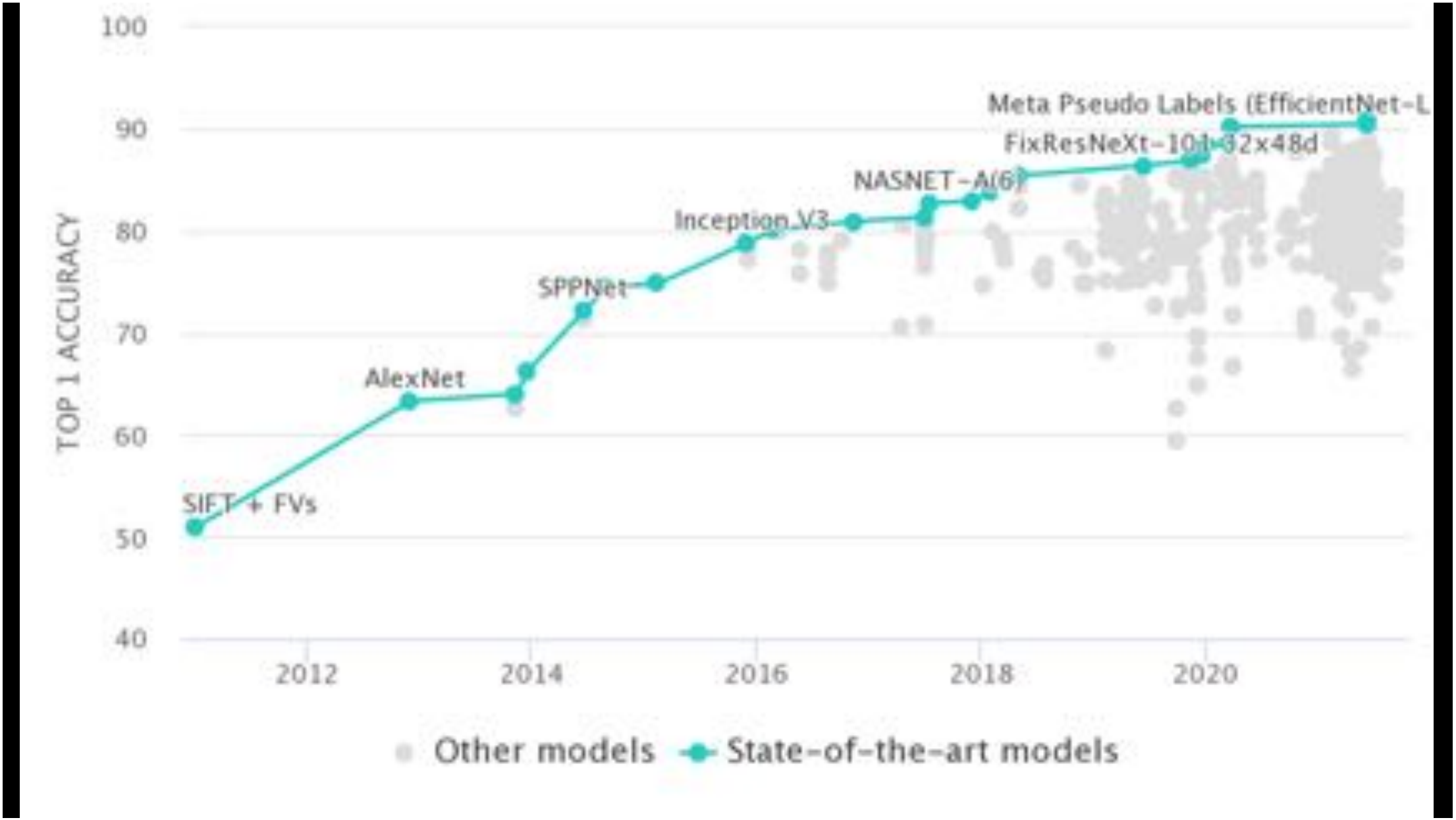



Classification
0/1
Bbox regression
Segmentation
Loss
A_1 * CrossEntropy
+
A_2 * Bbox Regression
+
A_3 * Segmentation Loss
=
Total Loss

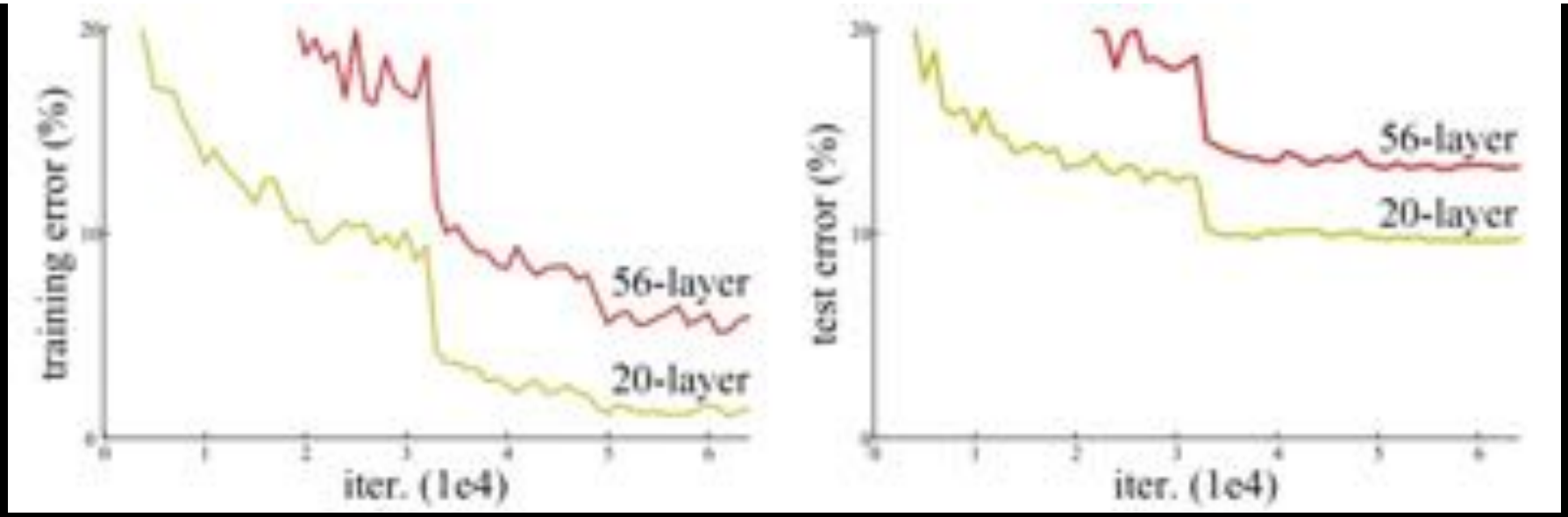
MULTI-TASK DEEP LEARNING BASED CT IMAGING ANALYSIS FOR COVID-19: CLASSIFICATION AND SEGMENTATION



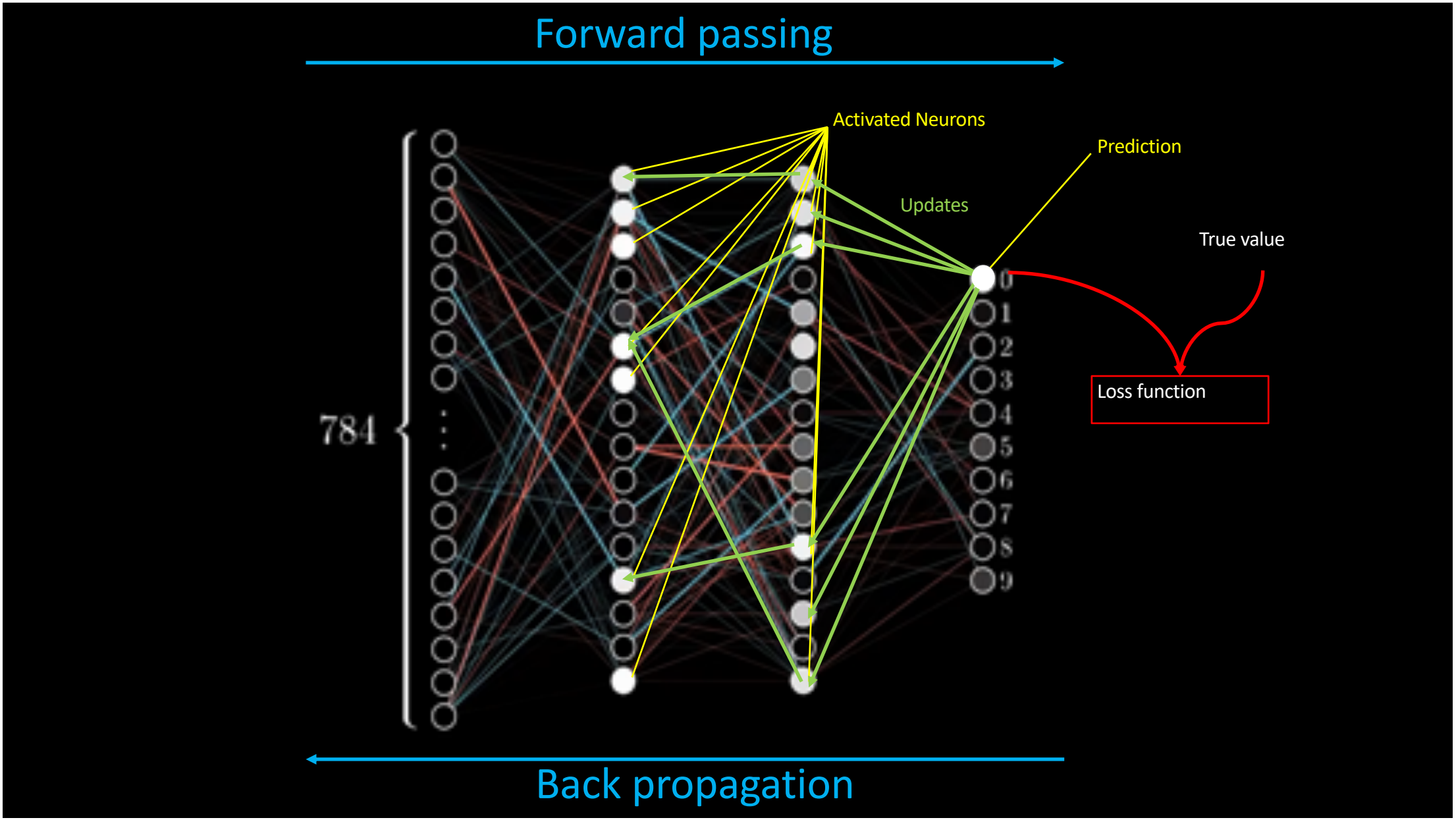

Classification
Class scores
Healthy
Cancerous
Regression
Lymphocyte
Benign epithelial
Malignant epithelial
Class coordinates
Labelled WSI
Patch extraction
Encoder
Segmentation
Tumor segmentation
Decoder

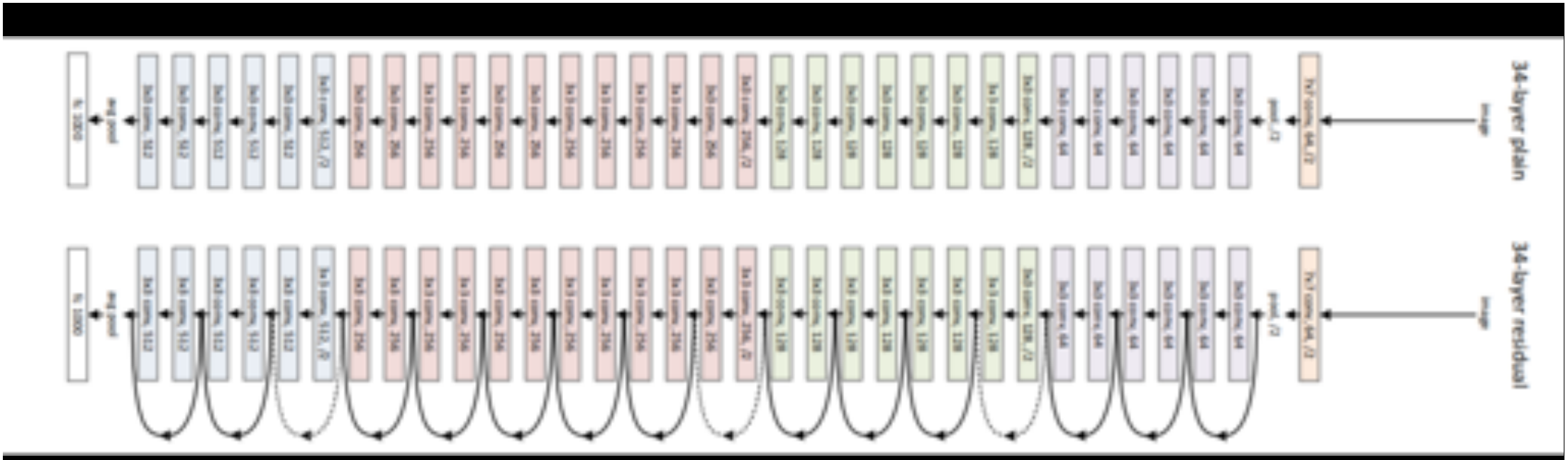

100
90
80
70
60
50
40
TOP 1 ACCURACY
2012
2014
2016
2018
2020
SIFT + FVs
AlexNet
SPPNet
Inception V3
NASNET-A(6)
FixResNeXt-101-32x48d
Meta Pseudo Labels (EfficientNet-L
Other models
State-of-the-art models

Residual Networks (Resnet)

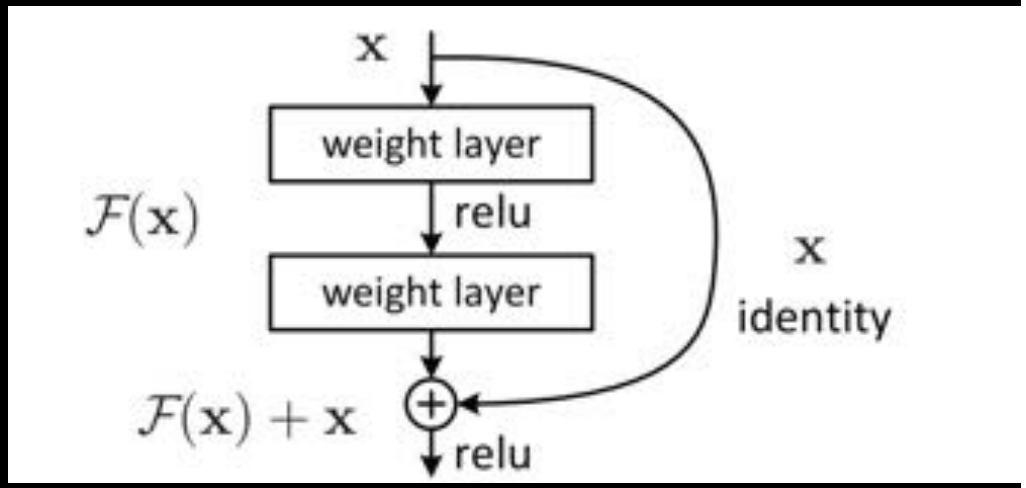


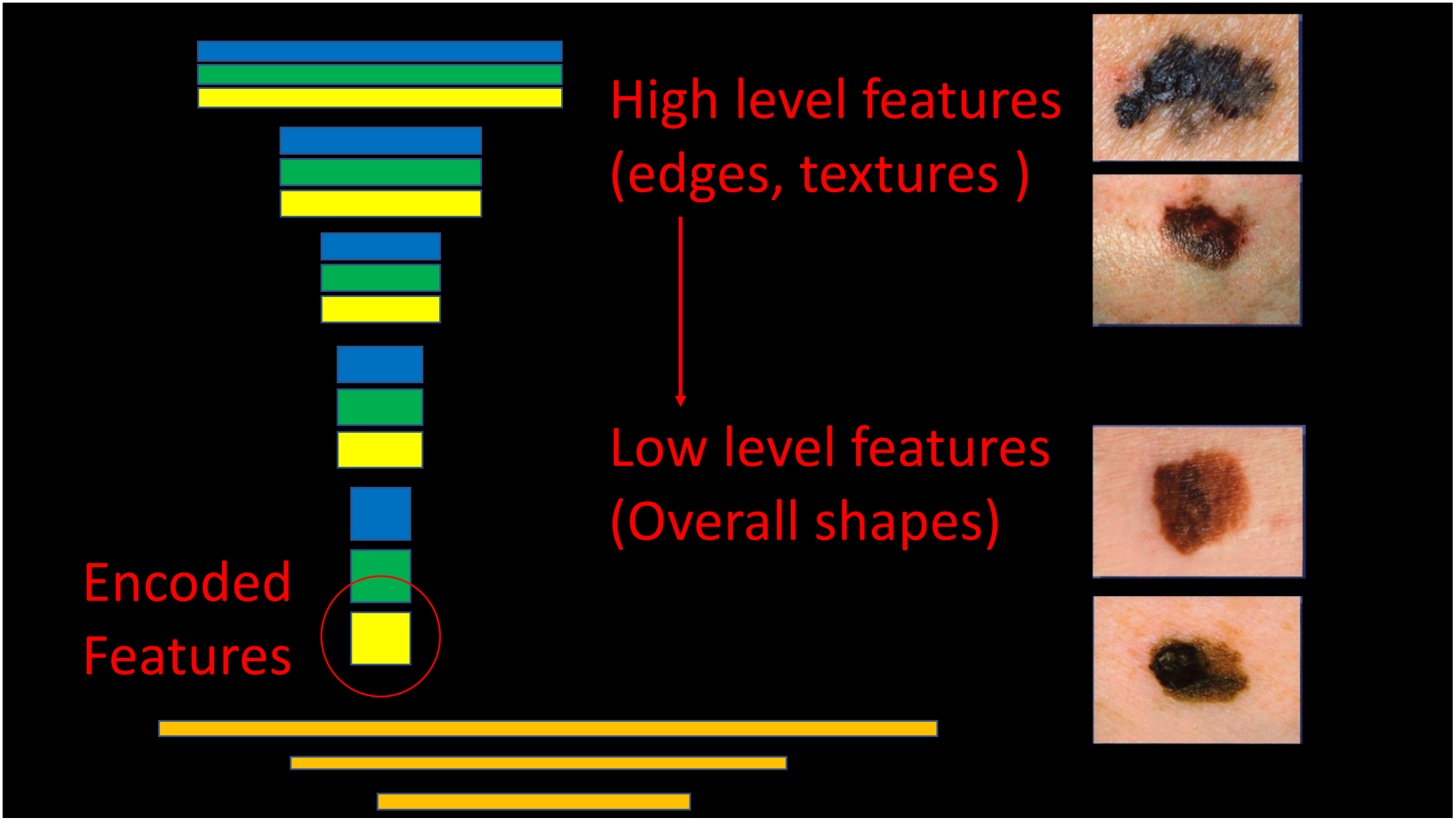
Deeper the better?

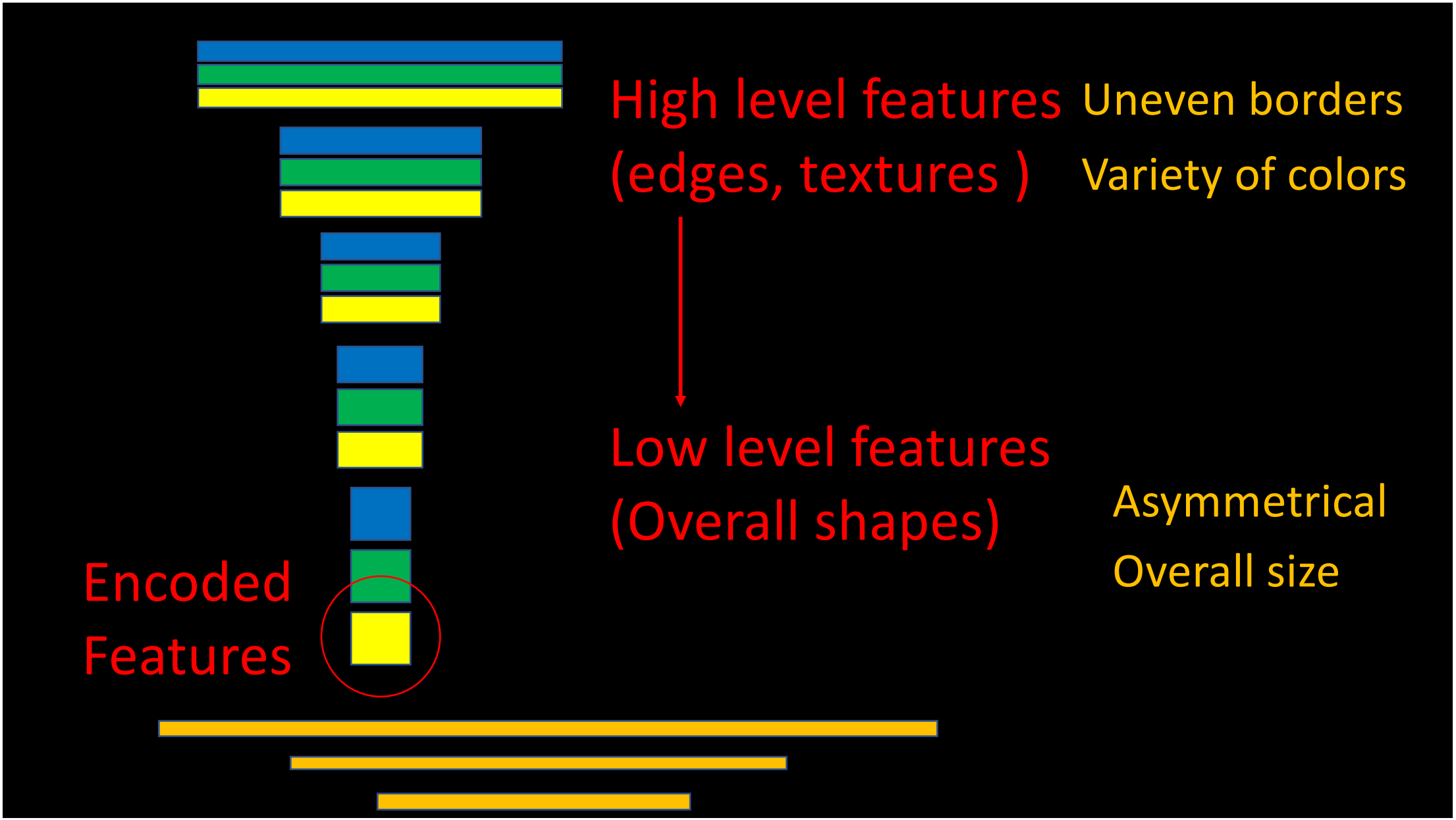

Forward passing
Activated Neurons
Prediction
Updates
True value
784
0
1
2
3
4
5
6
7
8
9
Loss function
Back propagation

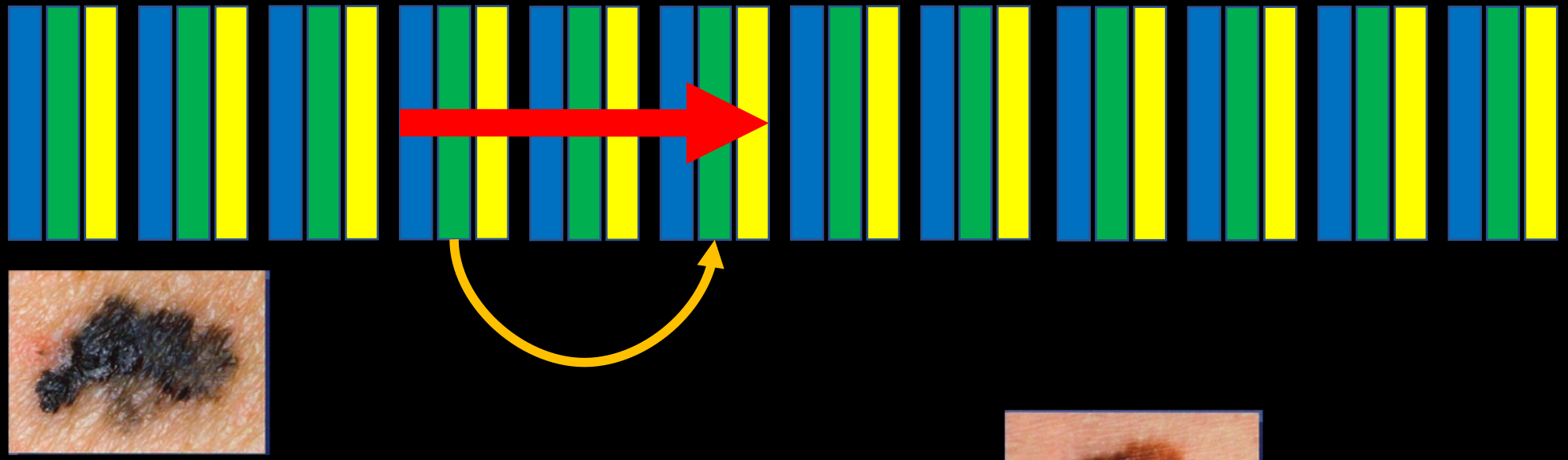


ResNet 34….up to 101
(most commonly used)




High level features
(edges, textures)
Low level features
(Overall shapes)
Encoded
Features


High level features
(edges, textures)
Uneven borders
Variety of colors
Low level features
(Overall shapes)
Asymmetrical
Overall size
Encoded
Features

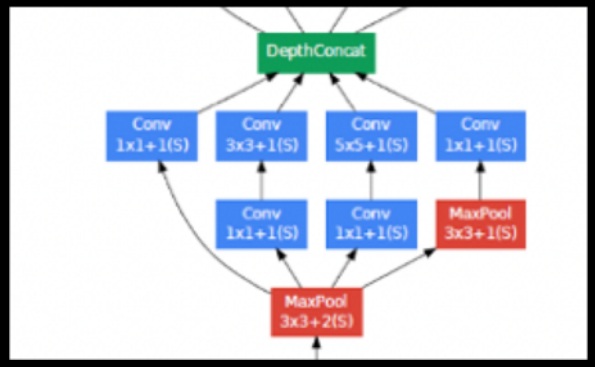


I want to have the option to keep updating it,

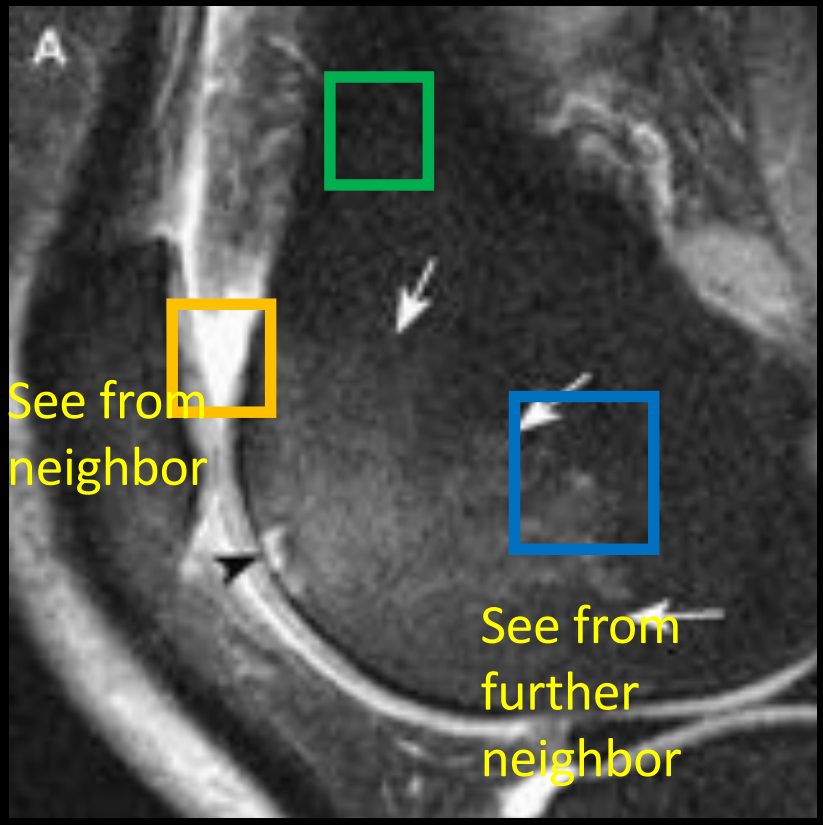
OR keep it the same

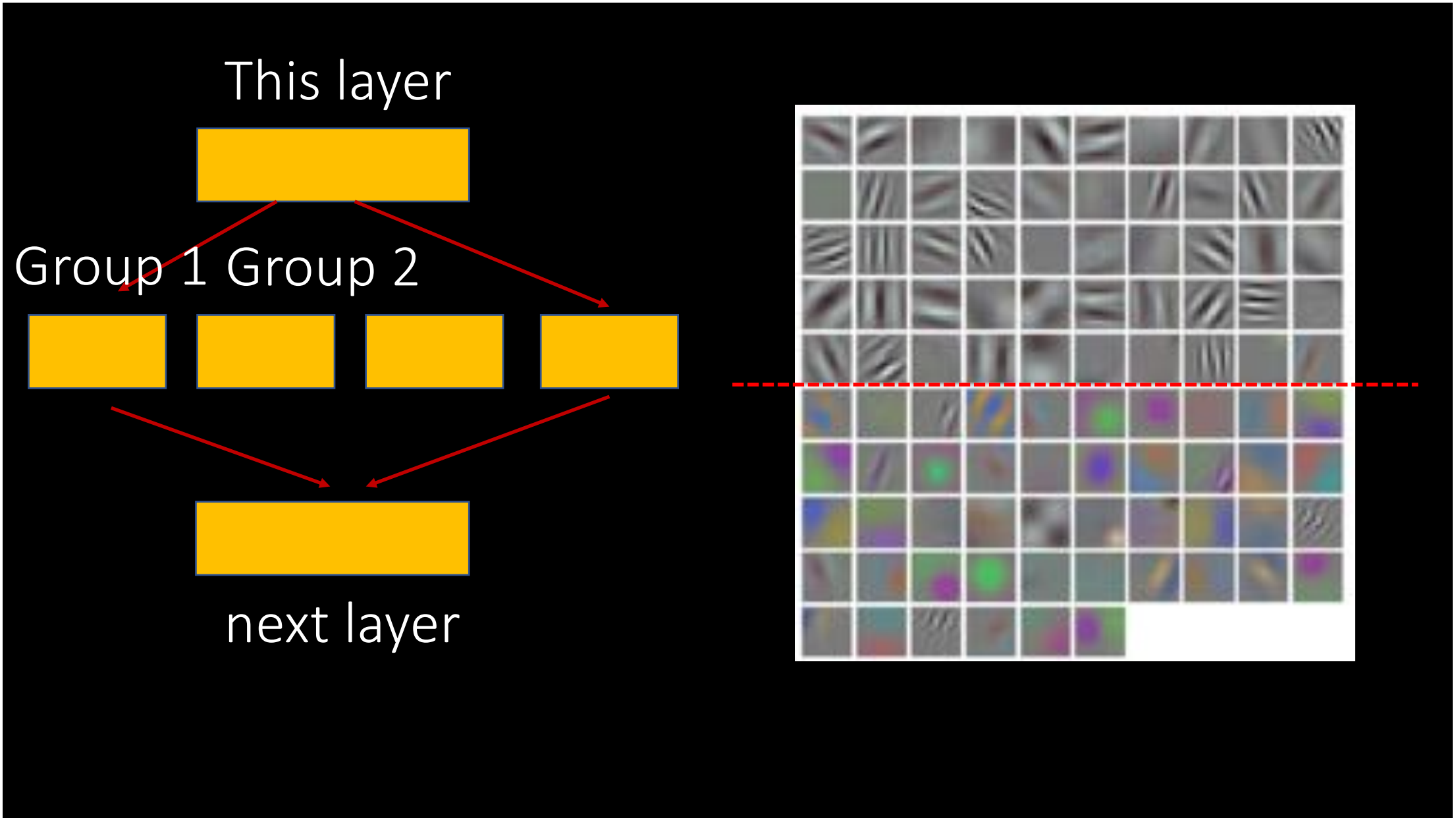
Low level may appear later / earlier!

CONVOLUTION WITH BRANCHES



Just Average over



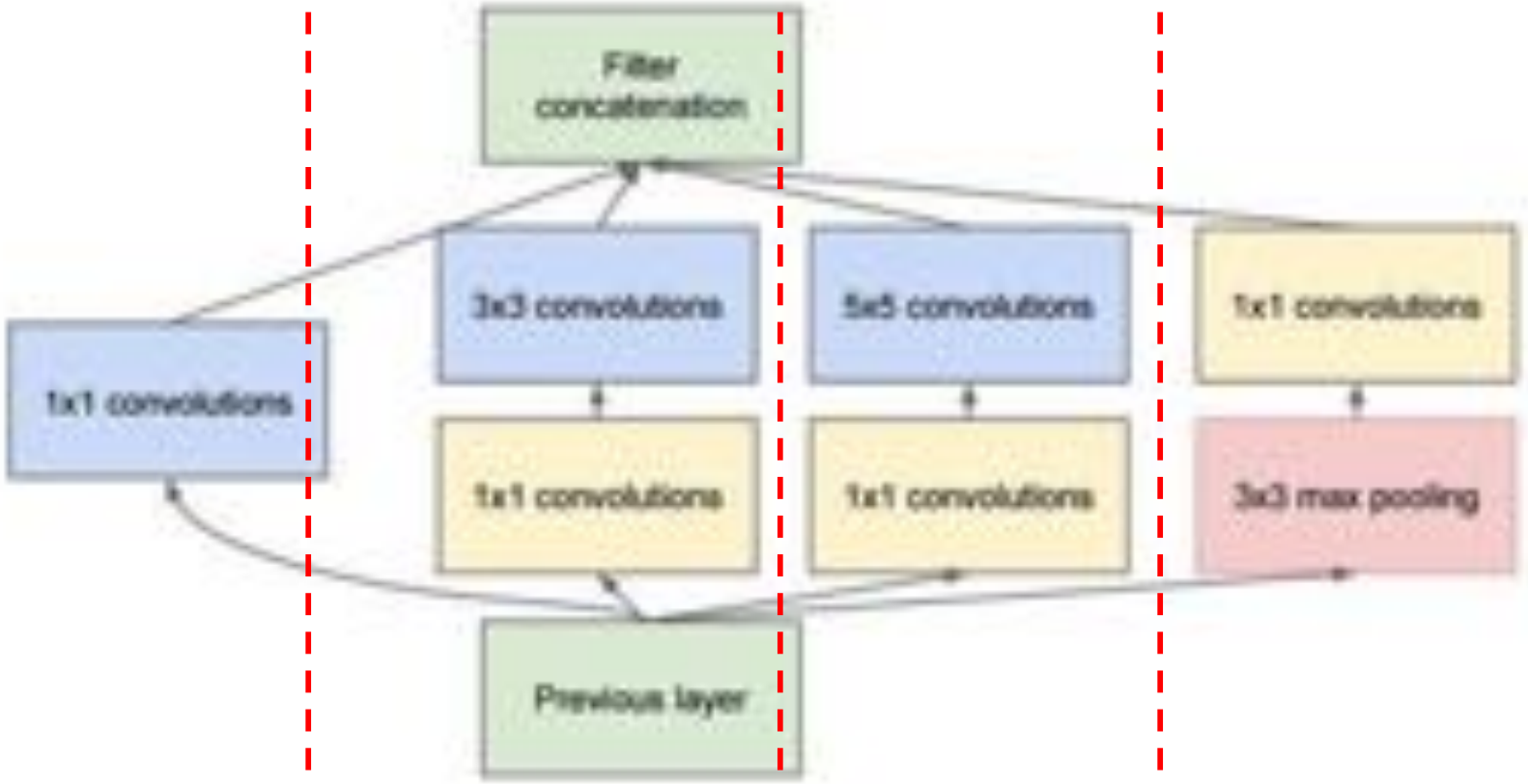

This layer
Group 1 Group 2
next layer

CONVOLUTION WITH BRANCHES

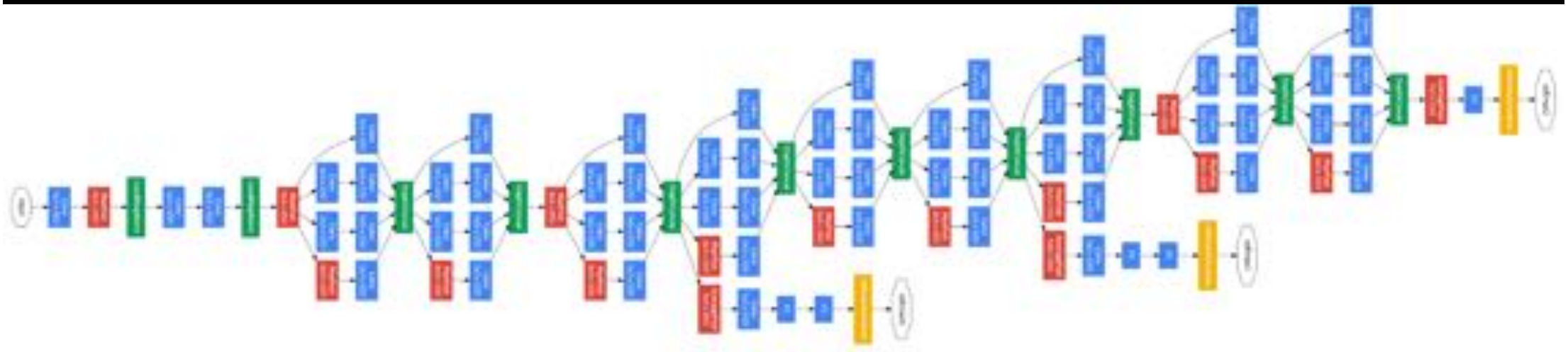
See from neighbor

See from further neighbor

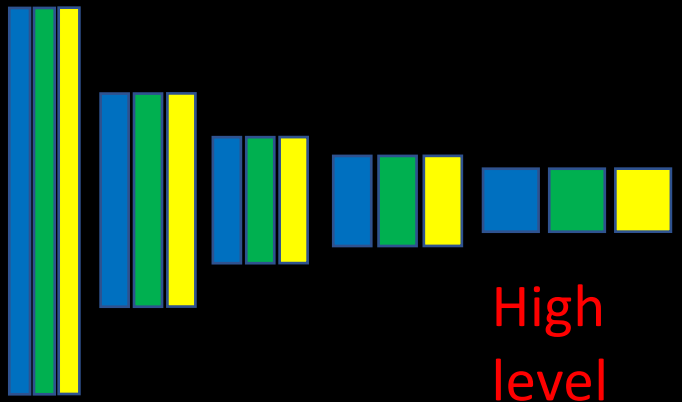
Just Pool over



Inception (2014)

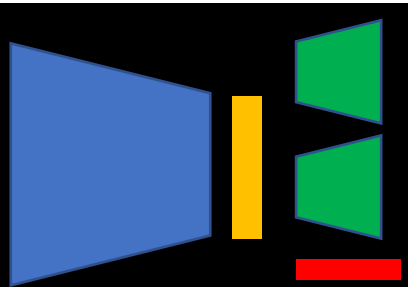


Summary

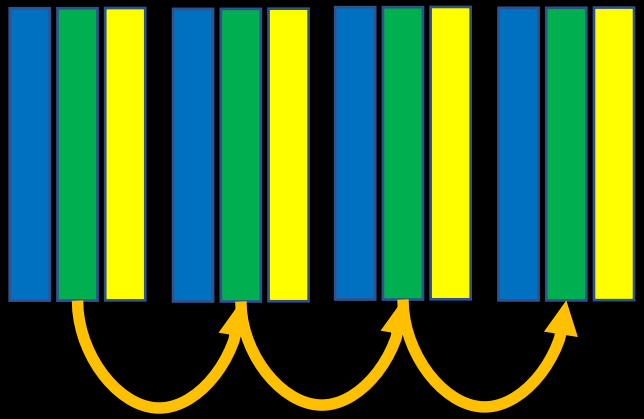


Low level features (edges, textures)

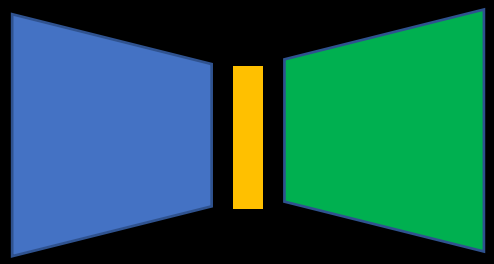
High level features (Overall shapes)



A good latent space allow you to do many tasks simultaneously



Residual give you very deep networks



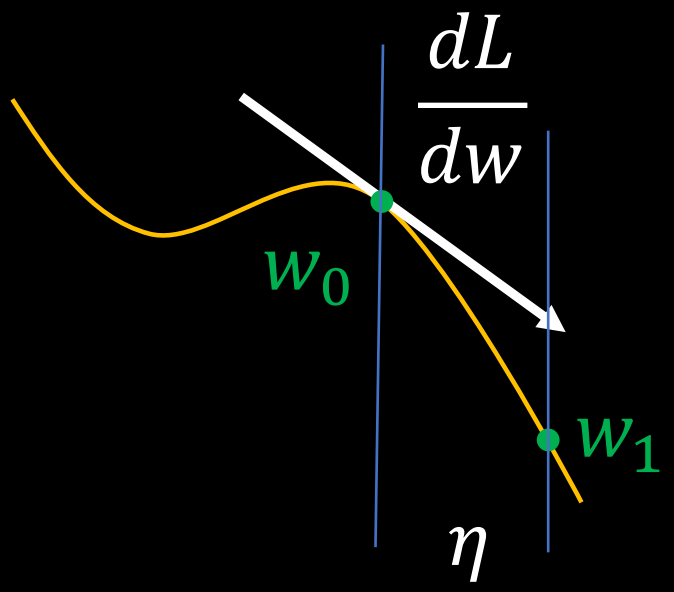
Use encoder-decoder to go back to image level

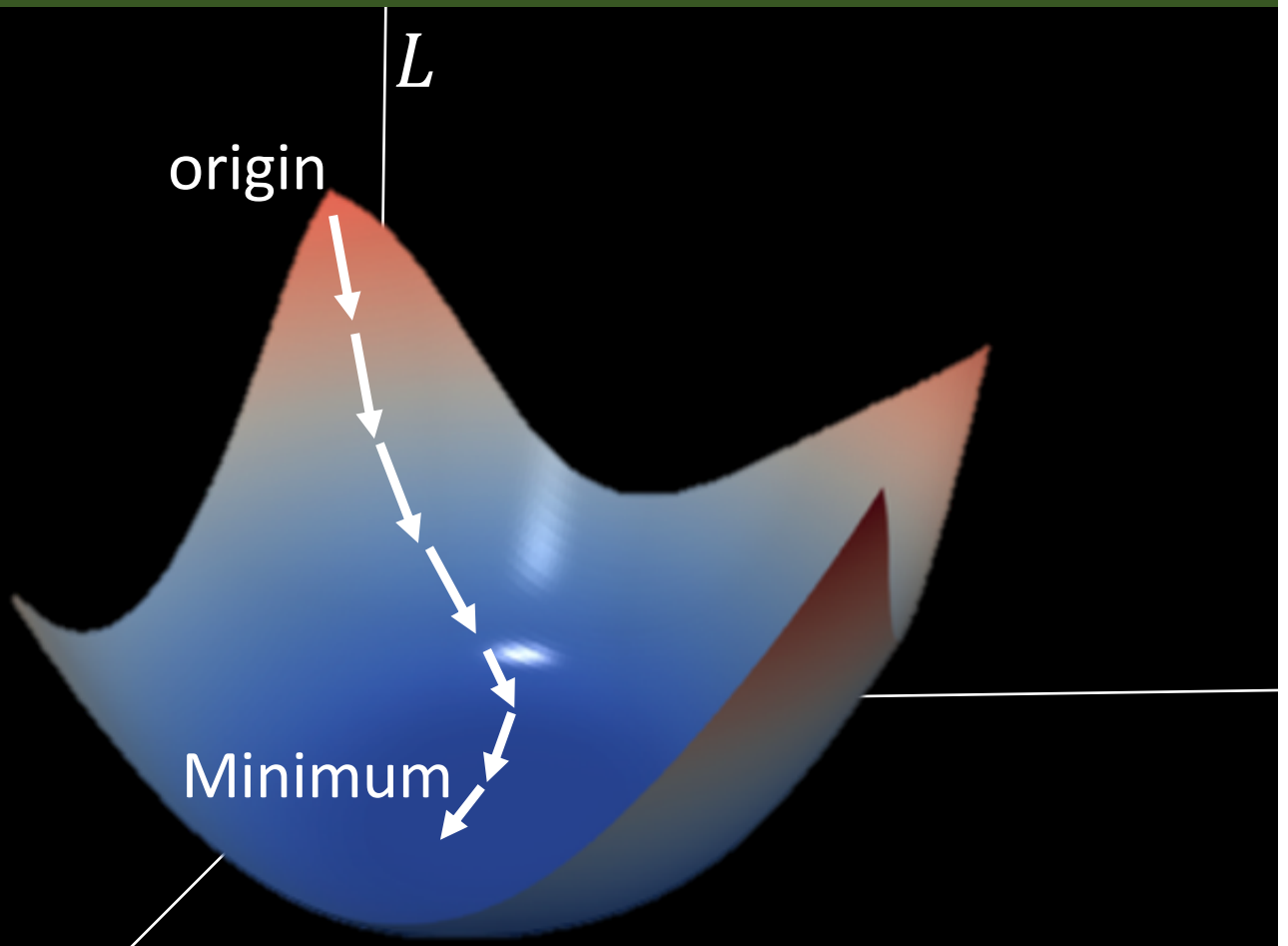
Optimization &
Stochastic Gradient Descend (SGD)

Training Neural Networks: optimization

Optimizer

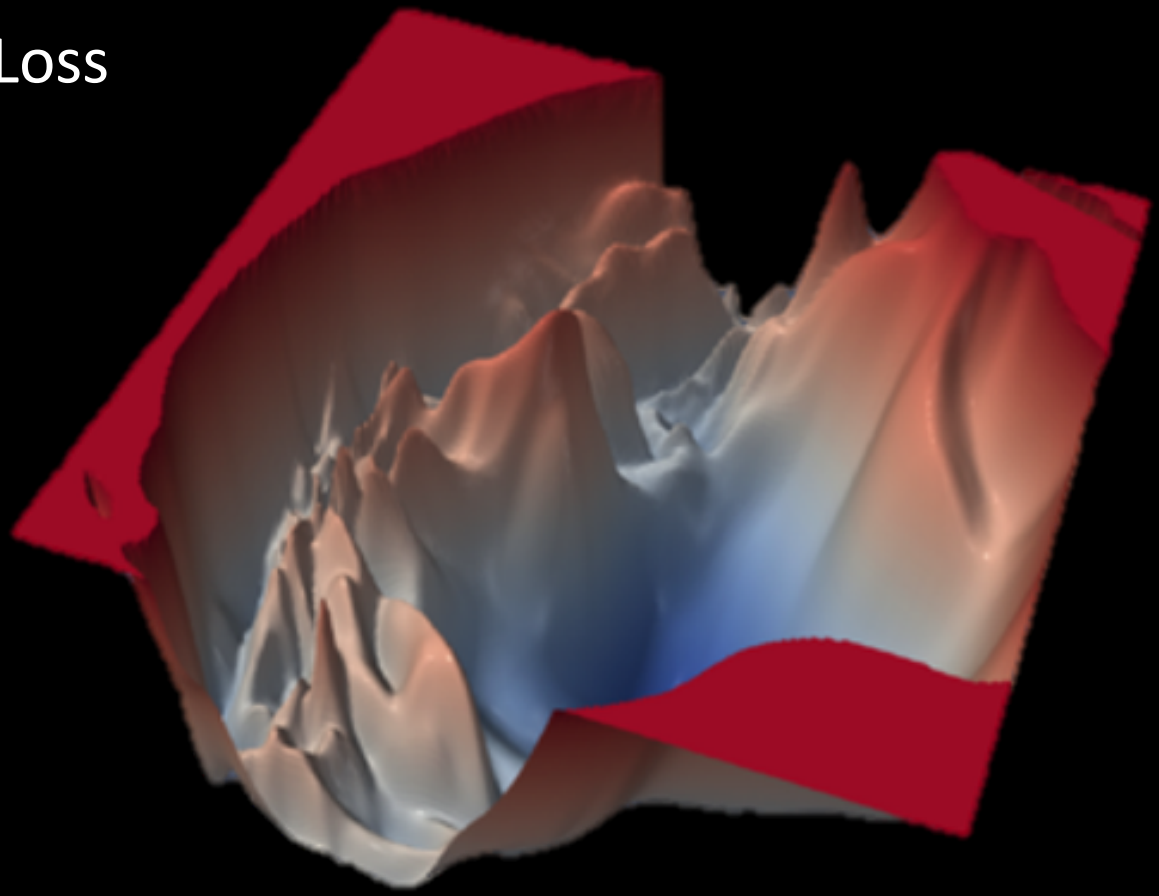
$argmin L(w_i, b)$

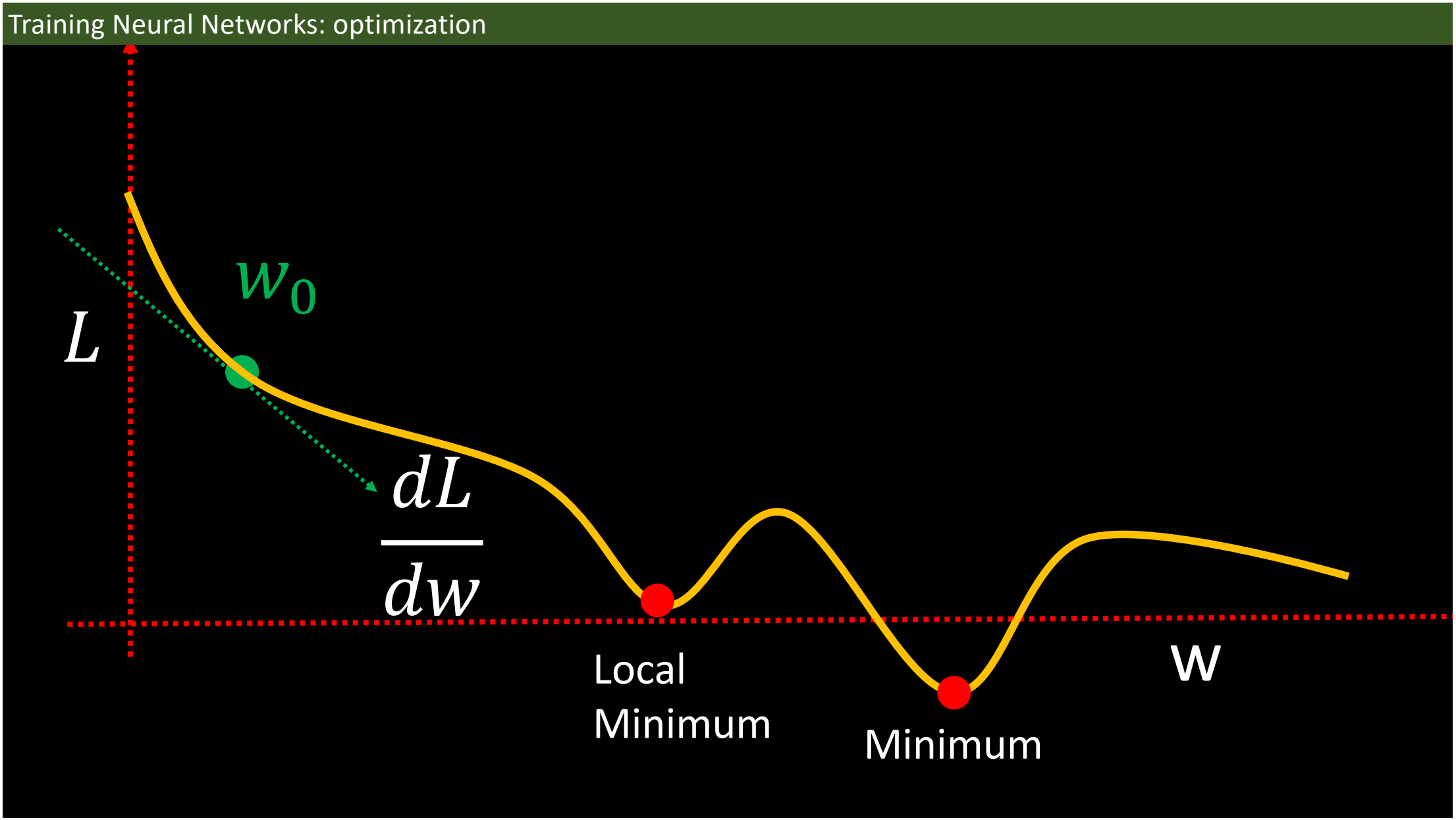




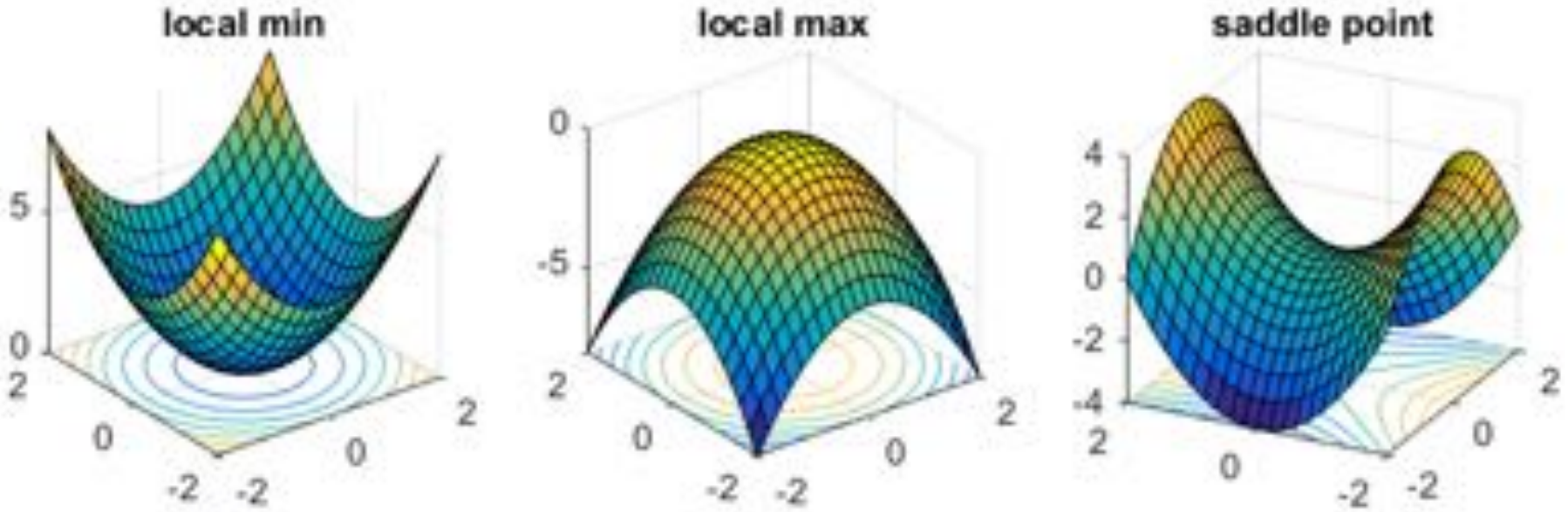
$$w_1 = w_0 - \frac{dL}{dw}\eta$$

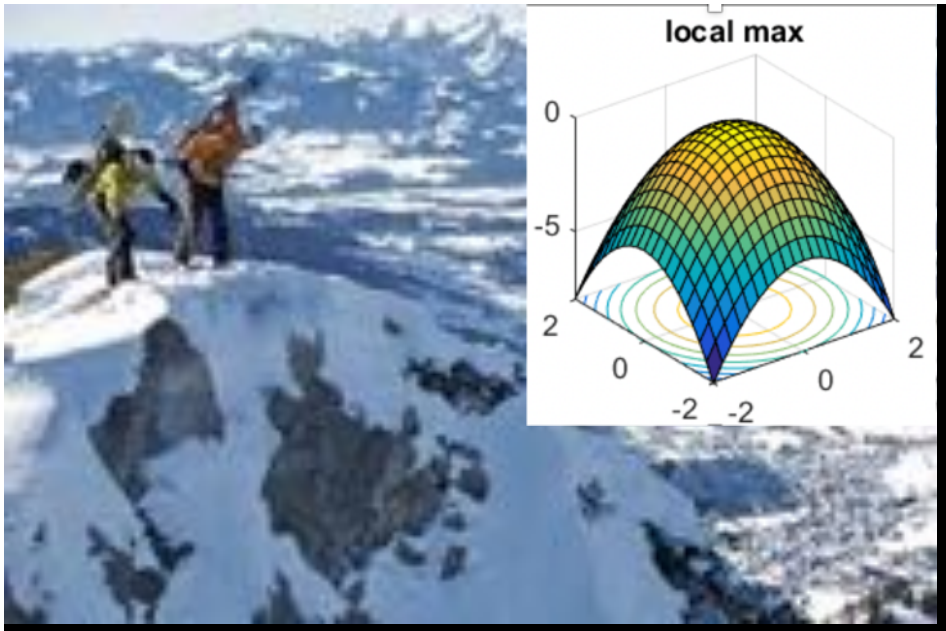
Real Surface of Loss



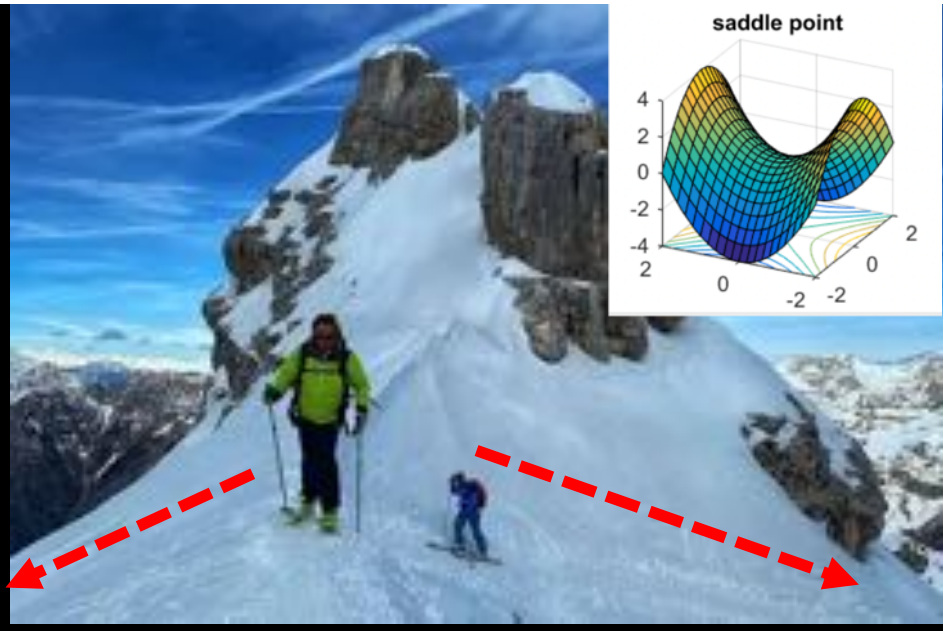

Training Neural Networks: optimization
L
w_0
$\frac{dL}{dw}$
Local
Minimum
Minimum
W

Training Neural Networks: optimization

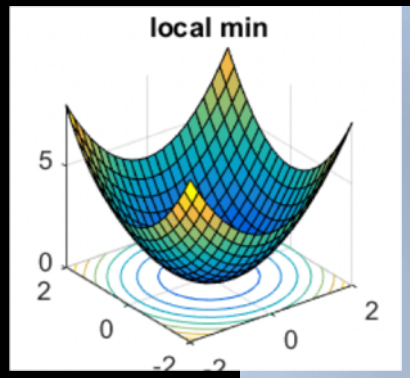




We don't care about this

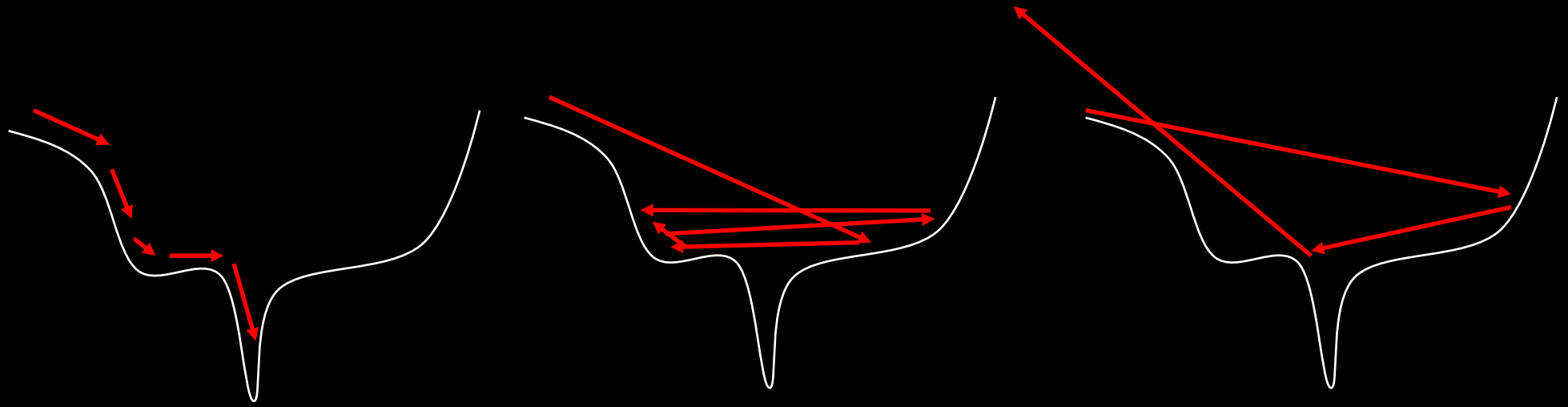


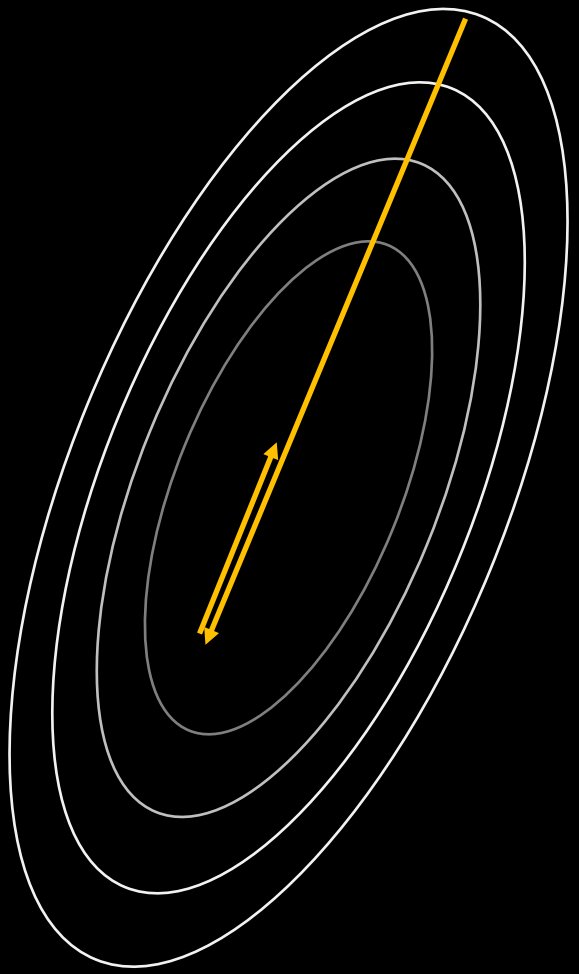
Most common



Your model is stuck

Avoiding stuck in local minimum
1: learning rate

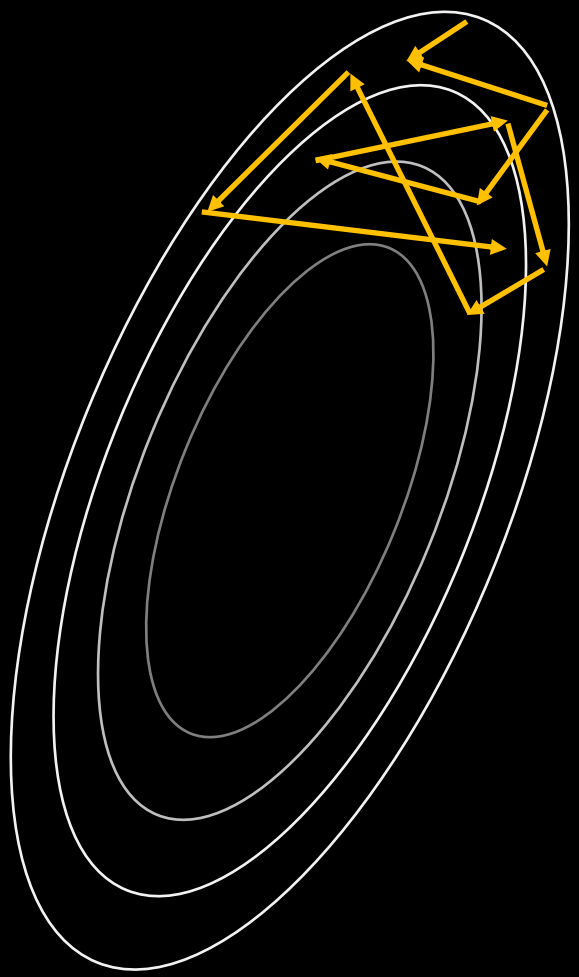




(Full)- batch:

using all the data at once

"deterministic"

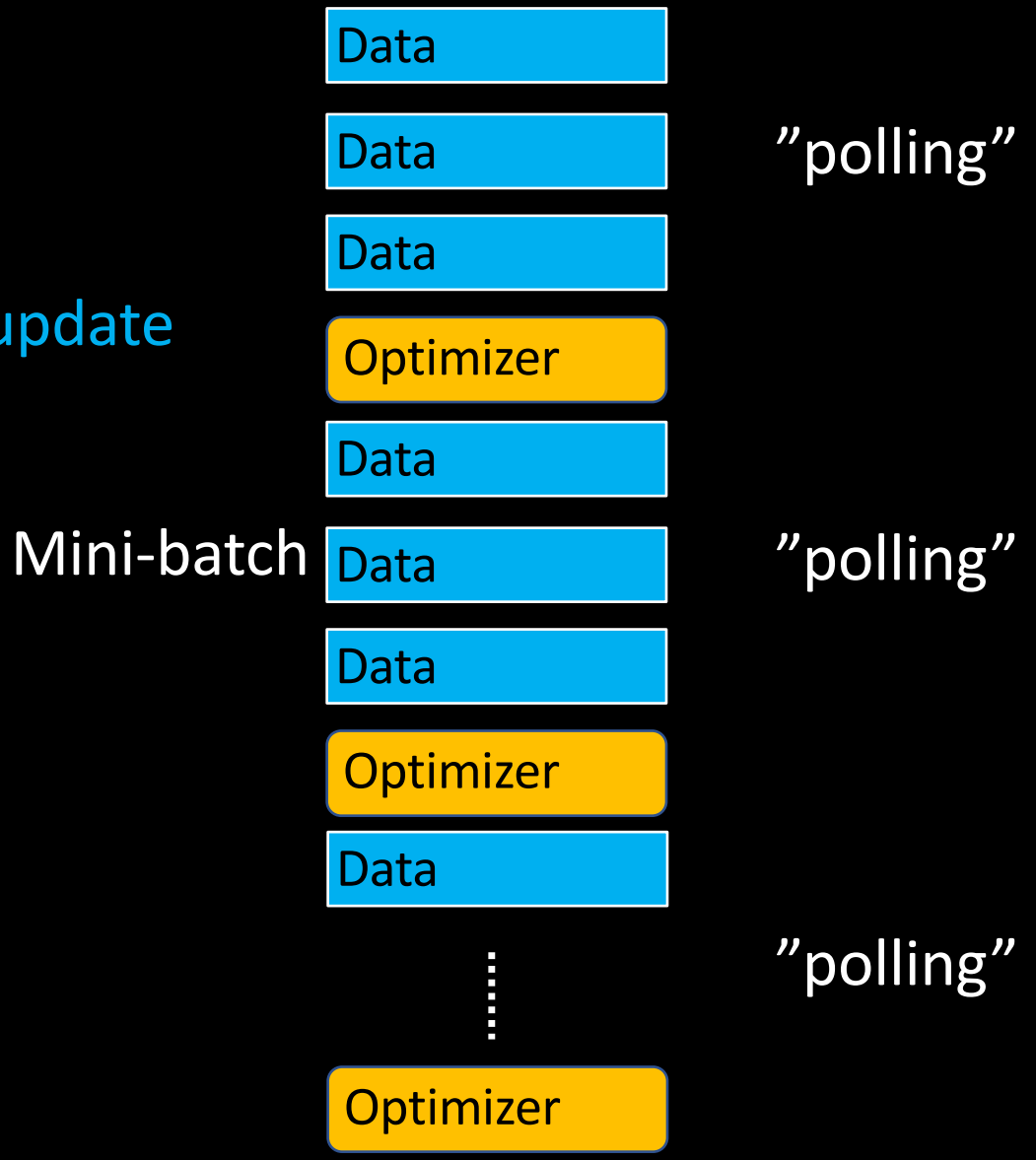


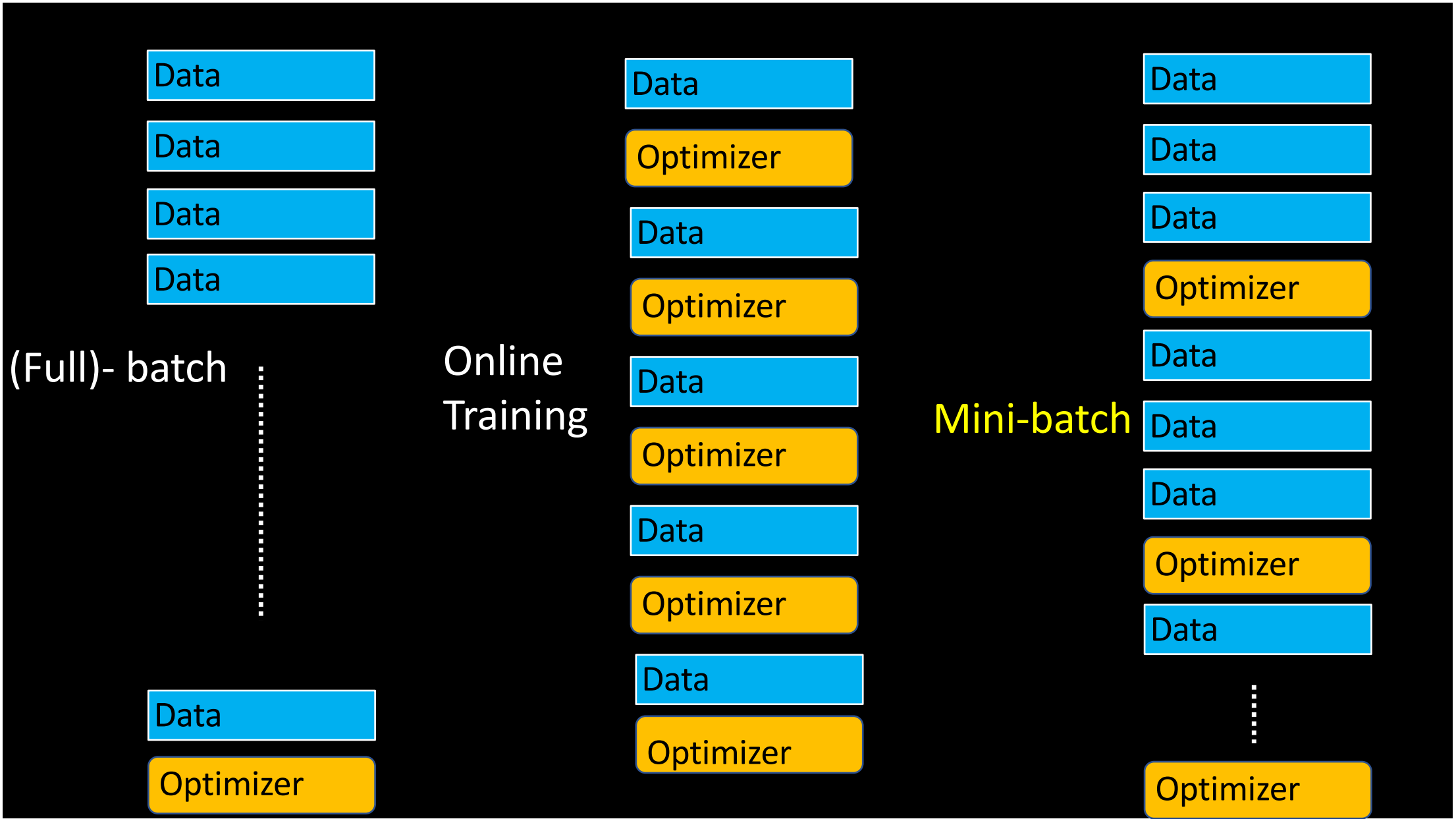
Online (one by one) Training:
using one data per update

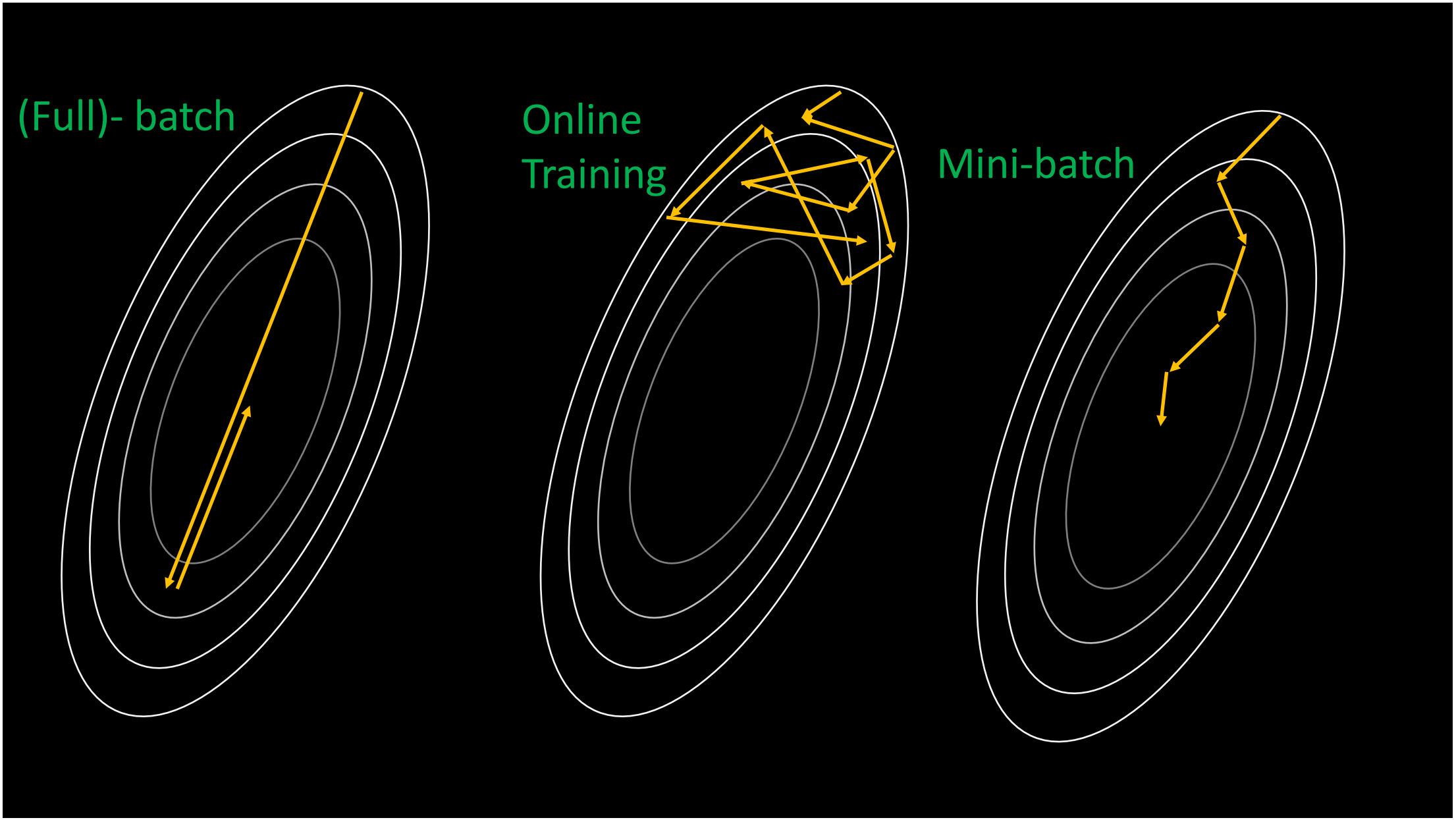
"tend to be crazy"

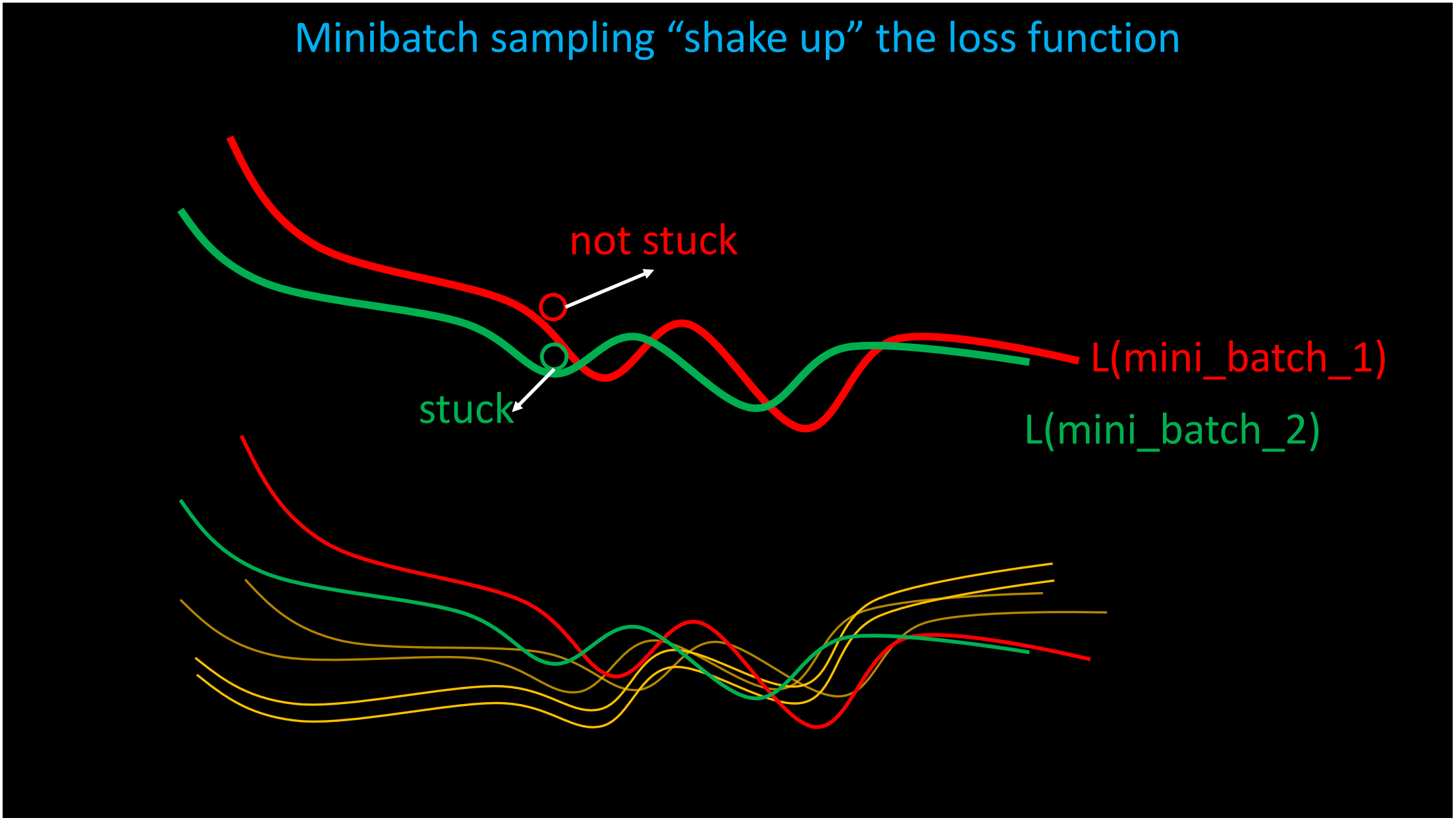
Minibatch training:

using some data (batch) per update

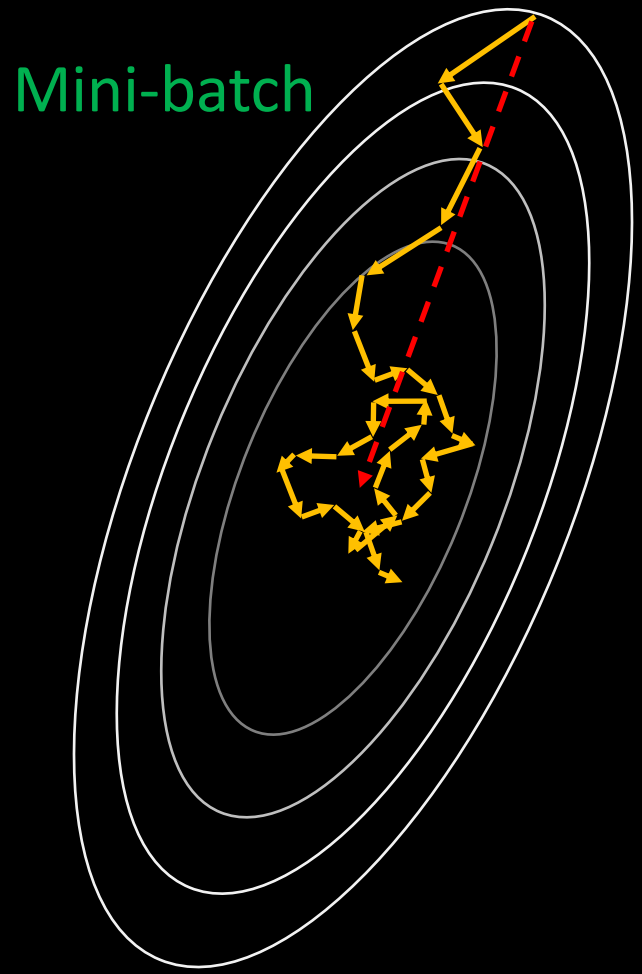


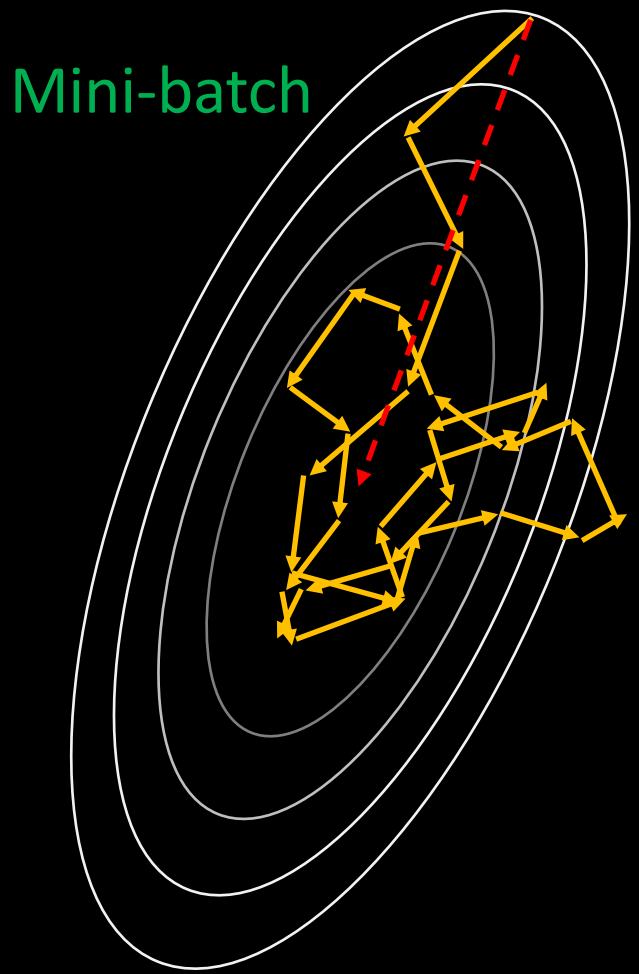

Data
Data
Data
Data
(Full)- batch
Data
Optimizer
Online
Training
Data
Optimizer
Data
Optimizer
Data
Optimizer
Data
Optimizer
Data
Optimizer
Mini-batch
Data
Data
Data
Optimizer
Data
Data
Data
Optimizer
Data
Optimizer


(Full)- batch
Online
Training
Mini-batch

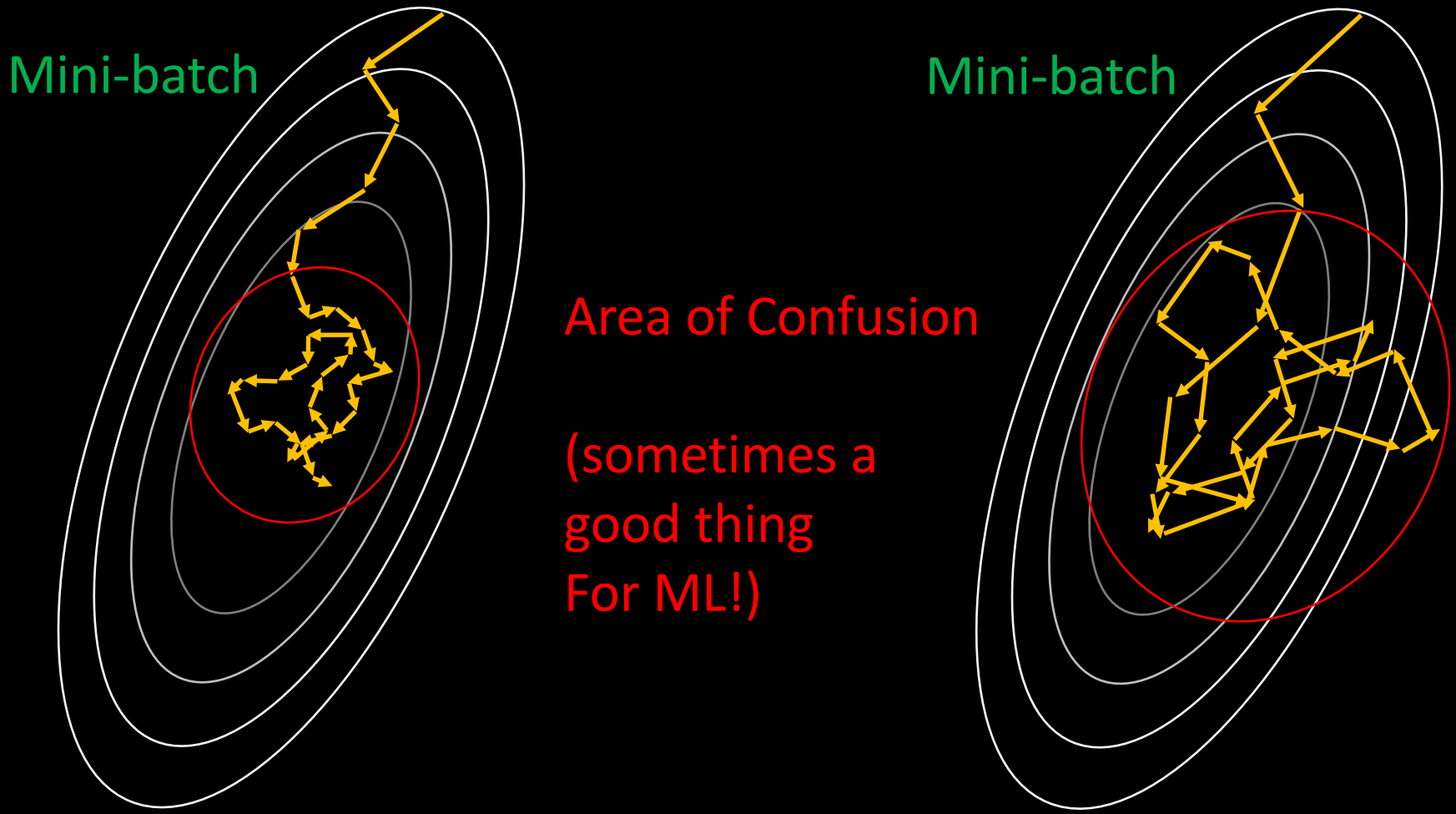

Minibatch sampling “shake up” the loss function
not stuck
stuck
L(mini_batch_1)
L(mini_batch_2)

Training Neural Networks: minibatch



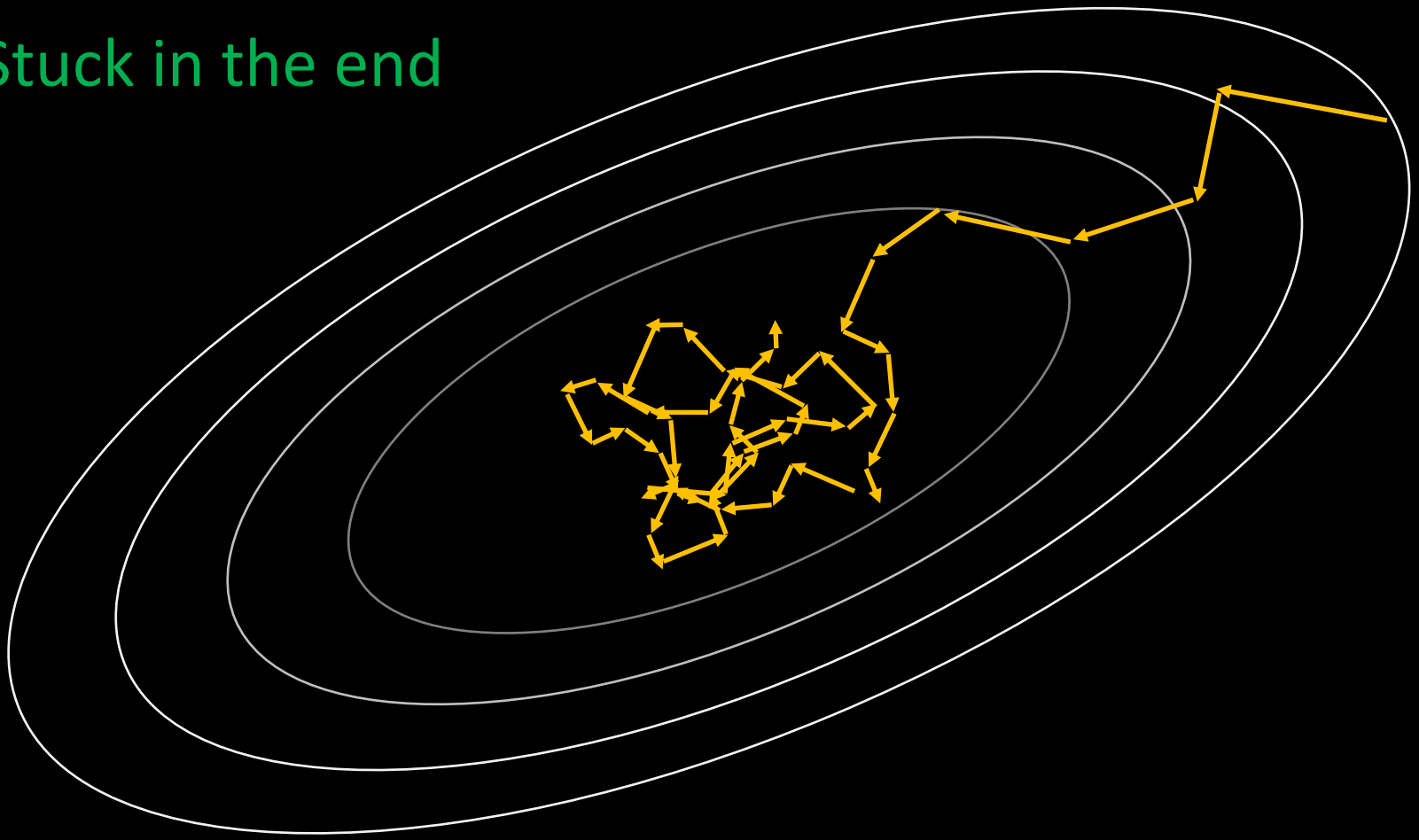


Training Neural Networks: minibatch



Fast Progress in the Beginning

Fussy / Stuck in the end



Good for big data!

One example of modern deep learning training

	Hardware	Chips	Batch	Optimizer	BN	Accuracy	Time
Goyal et al. [6]	P100	256	8192	Momentum	Local	76.3%	1 hour
Smith et al. [16]	TPU v2	128	8192 → 16384	Momentum	Local	76.1%	30 mins.
Akiba et al. [2]	P100	1024	32768	RMS + Mom.	Local	74.9%	15 mins.
Jia et al. [10]	P40	1024	65536	LARS	Local	76.2%	8.7 mins.
Baseline	TPU v2	4	1024	Momentum	Local	76.3%	8.0 hours
Ours	TPU v2	256	16384	Momentum	Local	75.1%	10 mins.
Ours	TPU v2	256	32768	LARS	Local	76.3%	8.5 mins.
Ours	TPU v3	512	32768	LARS	Local	76.4%	3.3 mins.
Ours	TPU v3	1024	32768	LARS	Distributed	76.3%	**2.2 mins.**

Mini-batch w/ shuffling

All Data

Random Shuffle

Minibatch Sampling

There are tones of research go into
how to fix crazy/stuck in the end

(not always matters in practice)

There are tones of research go into
how to fix crazy/stuck in the end

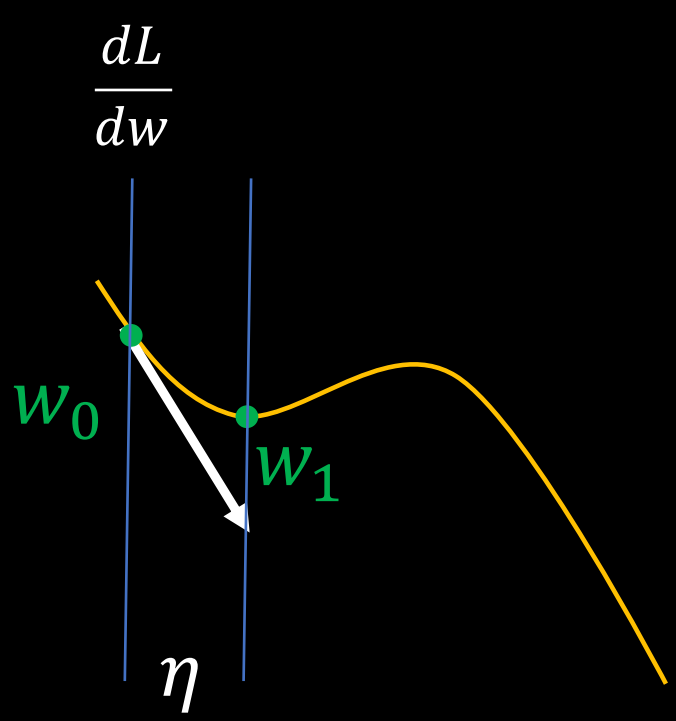
TIPS

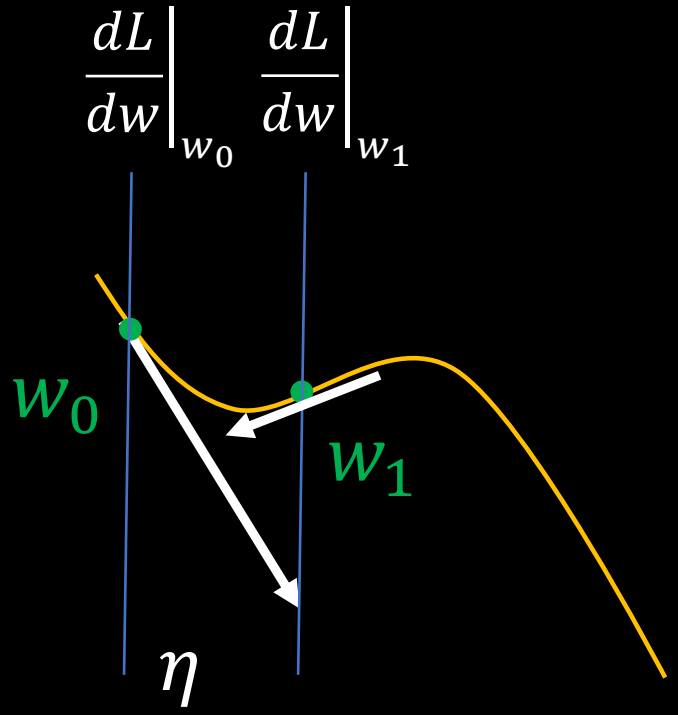
- schemes of gradient calculation

- find the right learning rate to start

- find the good way to change learning rate during training

Momentum





$$w_1 = w_0 - \frac{dL}{dw}\eta$$

$$w_1 = w_0 - \left(\left.\frac{dL}{dw}\right|_0 + \beta \left.\frac{dL}{dw}\right|_{-1}\right)\eta$$

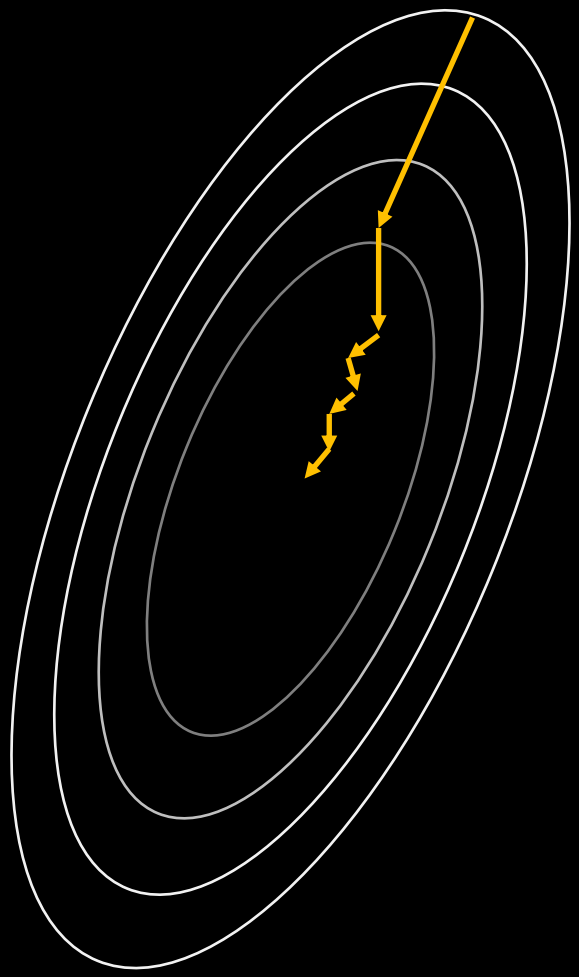
Momentum


gettyimages
Credit: Ashley Jouhar
82214305

Adagrad (adaptive)

$$w_1 = w_0 - \eta \left.\frac{dL}{dw}\right|_0 \epsilon$$

$$\epsilon = \frac{1}{\sqrt{\sum_0^t (\frac{dL}{dw})^2 + \varepsilon}}$$

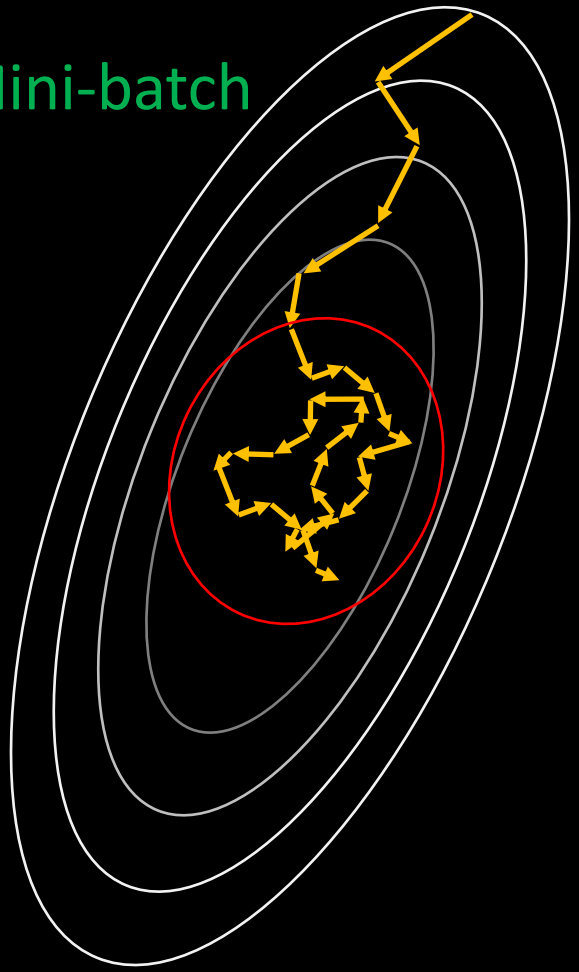


Adams

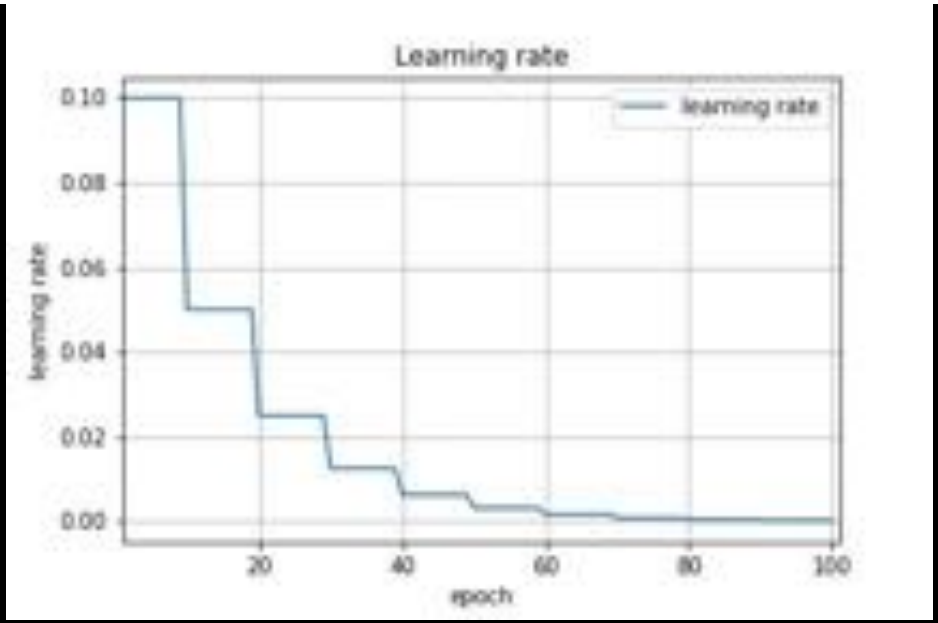
$$w_1 = w_0 - \eta \left(\left.\frac{dL}{dw}\right|_0 + \beta \left.\frac{dL}{dw}\right|_{-1}\right)\epsilon$$

Momentum Ada

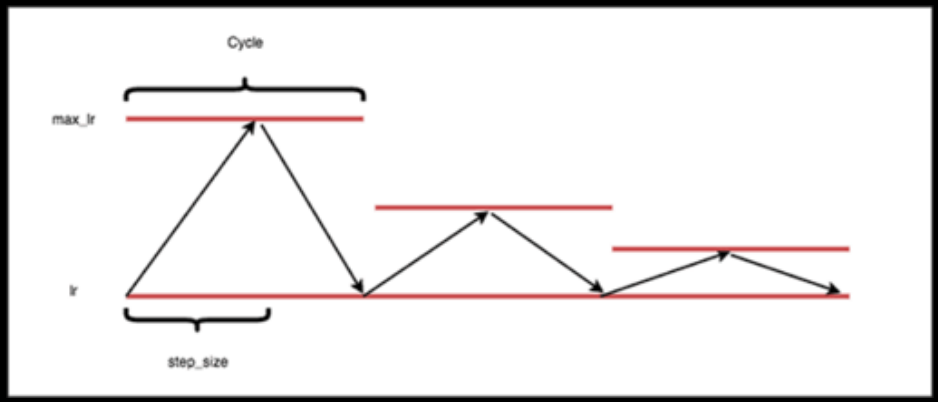
Mini-batch



Learning rate decay



Cyclic learning rate



Control Flow of updating neural network by SGD

Control Flow of SGD

Overall Loop

For epoch = 1…..n_epoch:

Train Loop

for batch = 1 ….. n_batch_training:

do training_step()
do optimize()

Validation Loop

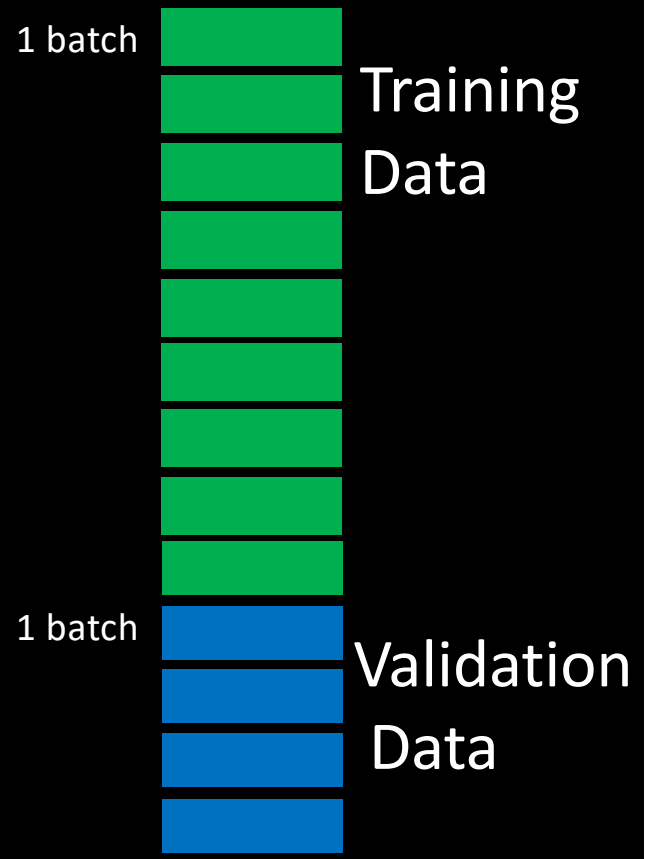
for batch = 1 ….. n_batch_validation:

do validation_step()

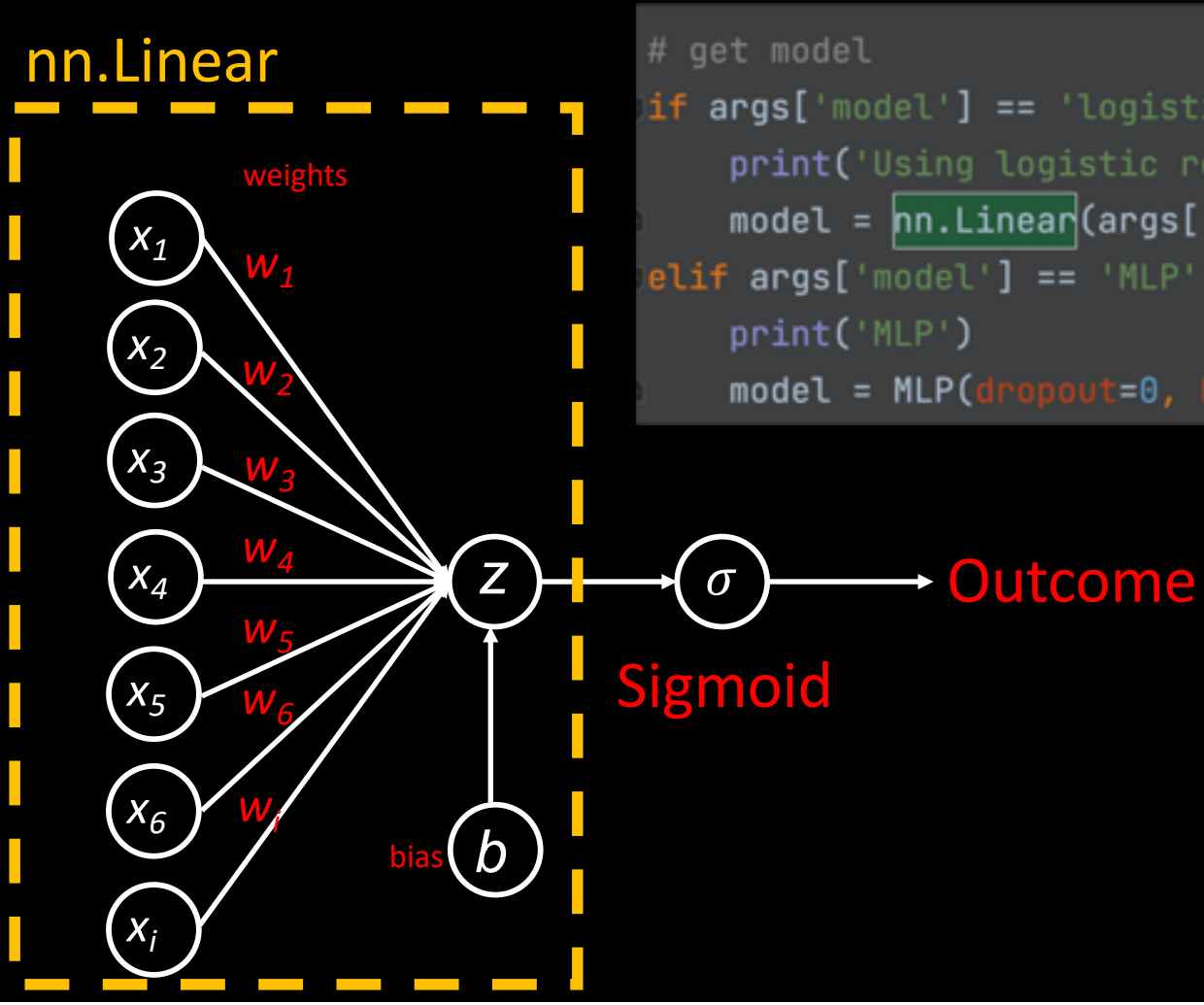
```python
# Medical MNIST dataset (images and labels)
train_loader, validation_loader = get_medical_mnist(args=args)
print('Done with data preparation')
```

```
data folder:  mmnist/
length of all images: 42000
length of all images: 16954
Done with data preparation
```

```
Length of train dataset:
42000
Length of validation dataset:
16954
Length of train dataloader:
2625
Length of validation dataloader:
1060
Why is that?
```



Graph of Logistic Regression




```python
# get model
if args['model'] == 'logistic_regression':
    print('Using logistic regression')
    model = nn.Linear(args['img_size'], args['num_classes'])
elif args['model'] == 'MLP':
    print('MLP')
    model = MLP(dropout=0, hidden_1=512, hidden_2=512)
```

```
>>> import torch
>>> import torch.nn as nn

>>> model = nn.Linear(100, 1)
>>> model

Linear(in_features=100, out_features=1, bias=True)
```

torch: basic things.

torch.nn: neural network components

a 100 features to 1 class linear model

Usually, the first dimension means the batch number (how many data were put into NN at once)

The second dimension means how many features (channels) of the data

```
>>> one_case = torch.rand(1, 100)
>>> print(one_case.shape)

torch.Size([1, 100])
```

One data with 100 features

Usually, the first dimension means the batch number (how many data were put into NN at once)

```
>>> one_case = torch.rand(1, 100)
>>> print(one_case.shape)

torch.Size([1, 100])

>>> print(model(one_case).shape)

torch.Size([1, 1])
```

One data with 100 features

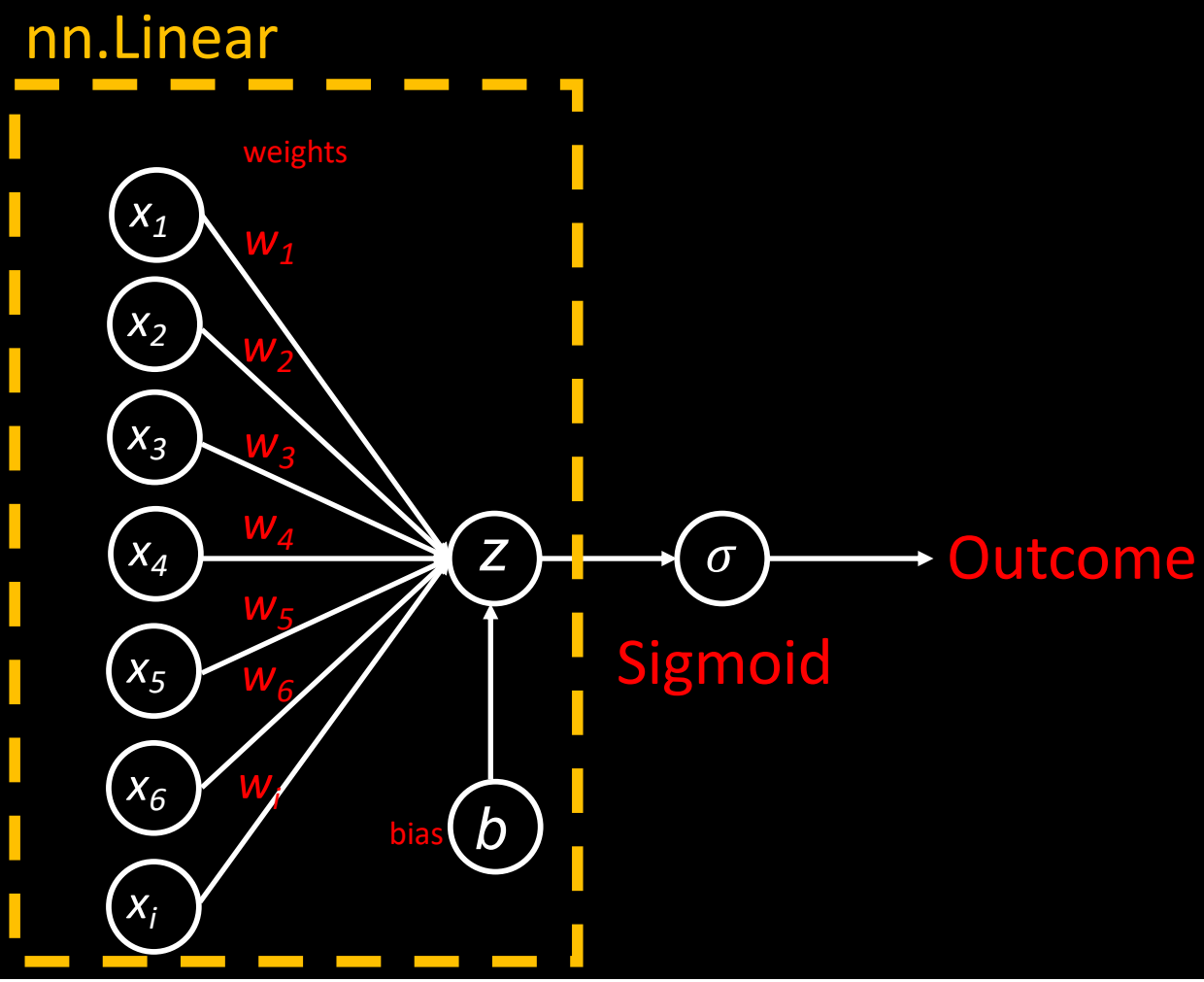
One data create one output

```
>>> three_case = torch.rand(3, 100)
>>> print(three_case.shape)

torch.Size([3, 100])

>>> print(model(three_case).shape)

torch.Size([3, 1])
```

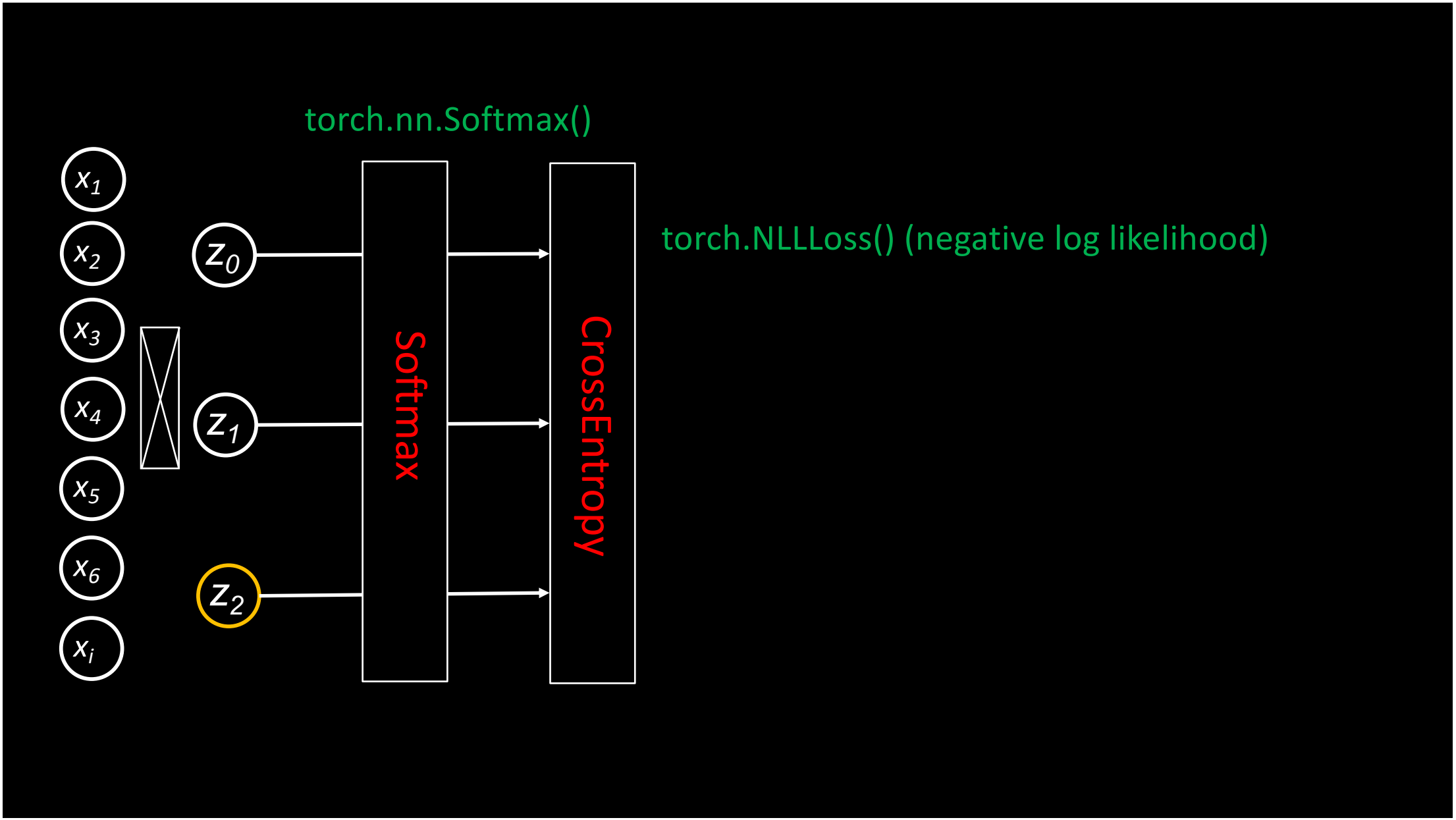
three data with 100 features

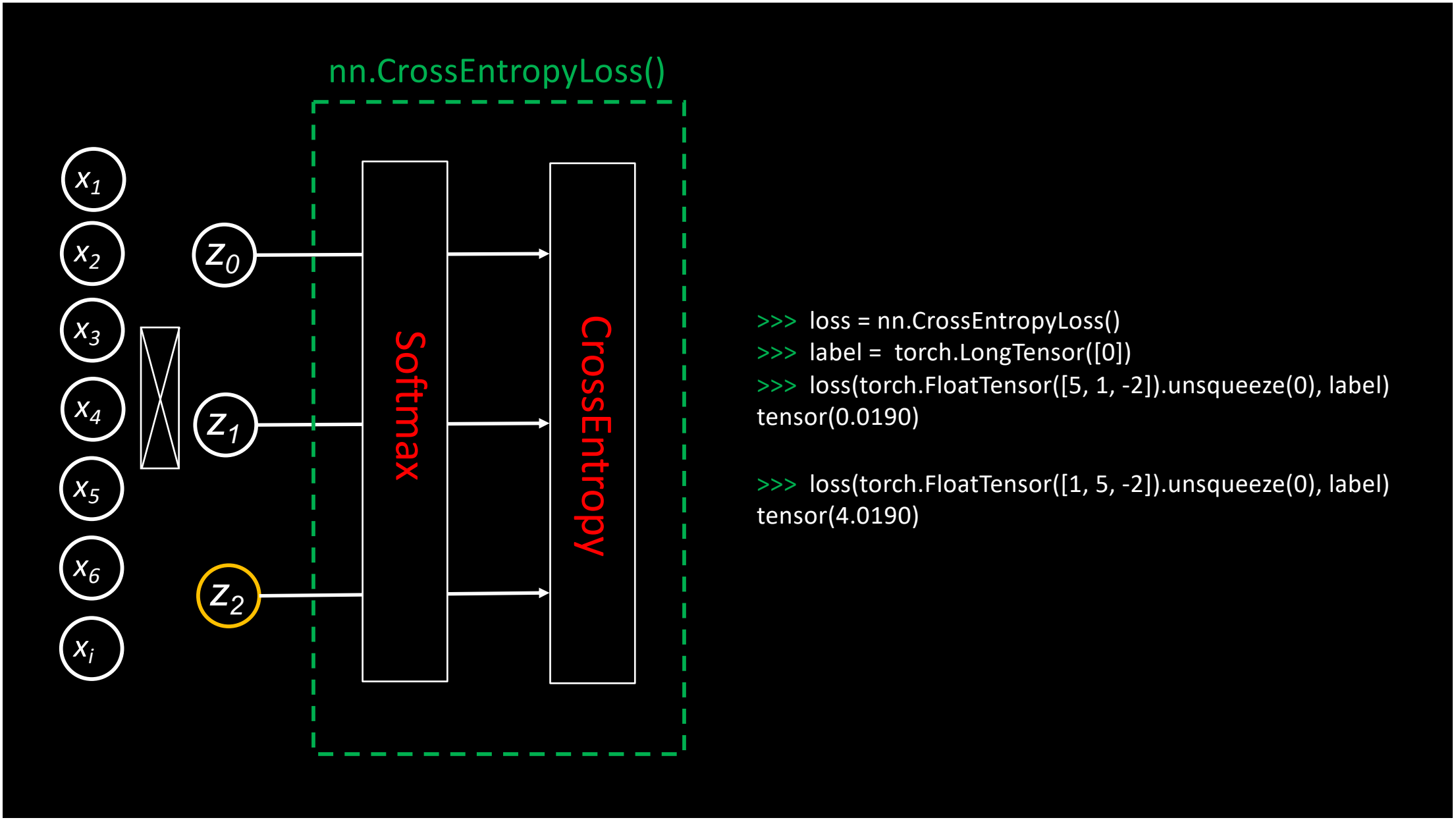
three data create three output

Graph of Logistic Regression

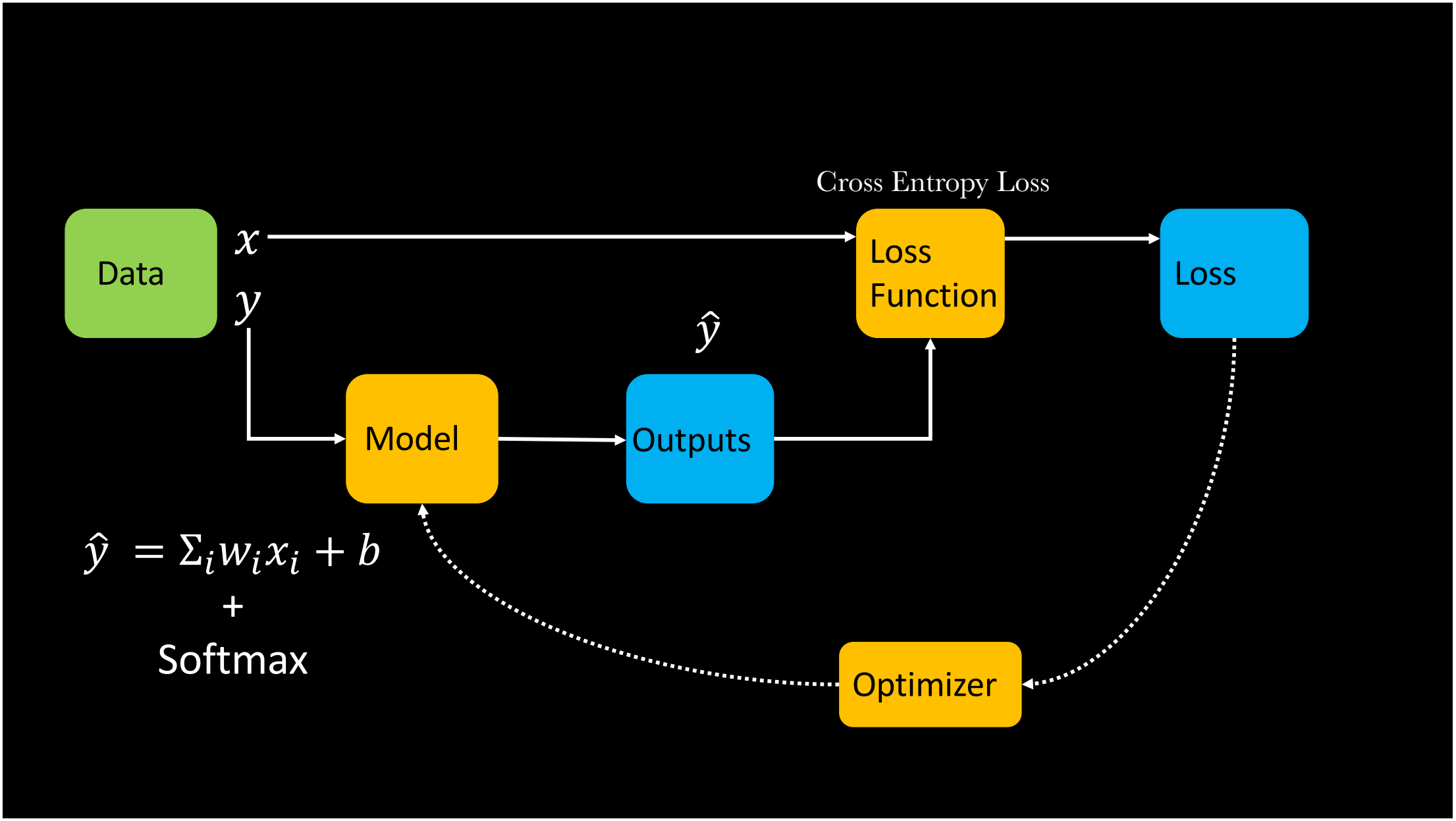


```
# Logistic regression model
model = nn.Linear(args['img_size'], args['num_classes'])
                          784 = 28 X 28        10


model
Linear(in_features=784, out_features=10, bias=True)

model.weight.shape
torch.Size([10, 784])

model.bias.shape
torch.Size([10])

model(torch.rand(3, 28*28)).shape
torch.Size([3, 10])
```


torch.nn.Softmax()
x_1
x_2
x_3
x_4
x_5
x_6
x_i
z_0
z_1
z_2
Softmax
CrossEntropy
torch.NLLLoss() (negative log likelihood)




```
>>> loss = nn.CrossEntropyLoss()
>>> label =  torch.LongTensor([0])
>>> loss(torch.FloatTensor([5, 1, -2]).unsqueeze(0), label)
tensor(0.0190)

>>> loss(torch.FloatTensor([1, 5, -2]).unsqueeze(0), label)
tensor(4.0190)
```


Cross Entropy Loss
Data
x
y
Loss Function
Loss
$\hat{y}$
Model
Outputs
$\hat{y} = \Sigma_i w_i x_i + b$
+
Softmax
Optimizer