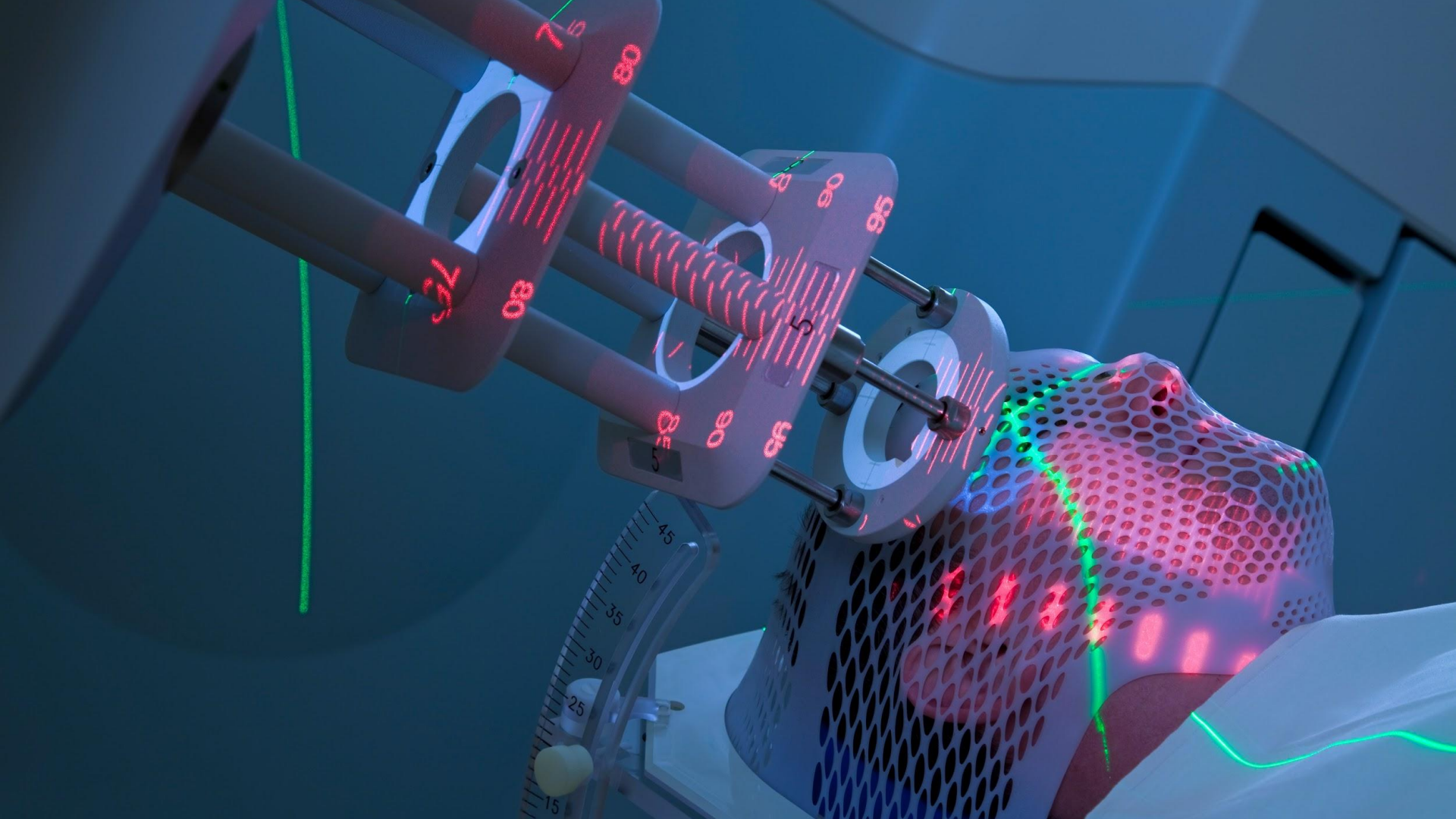
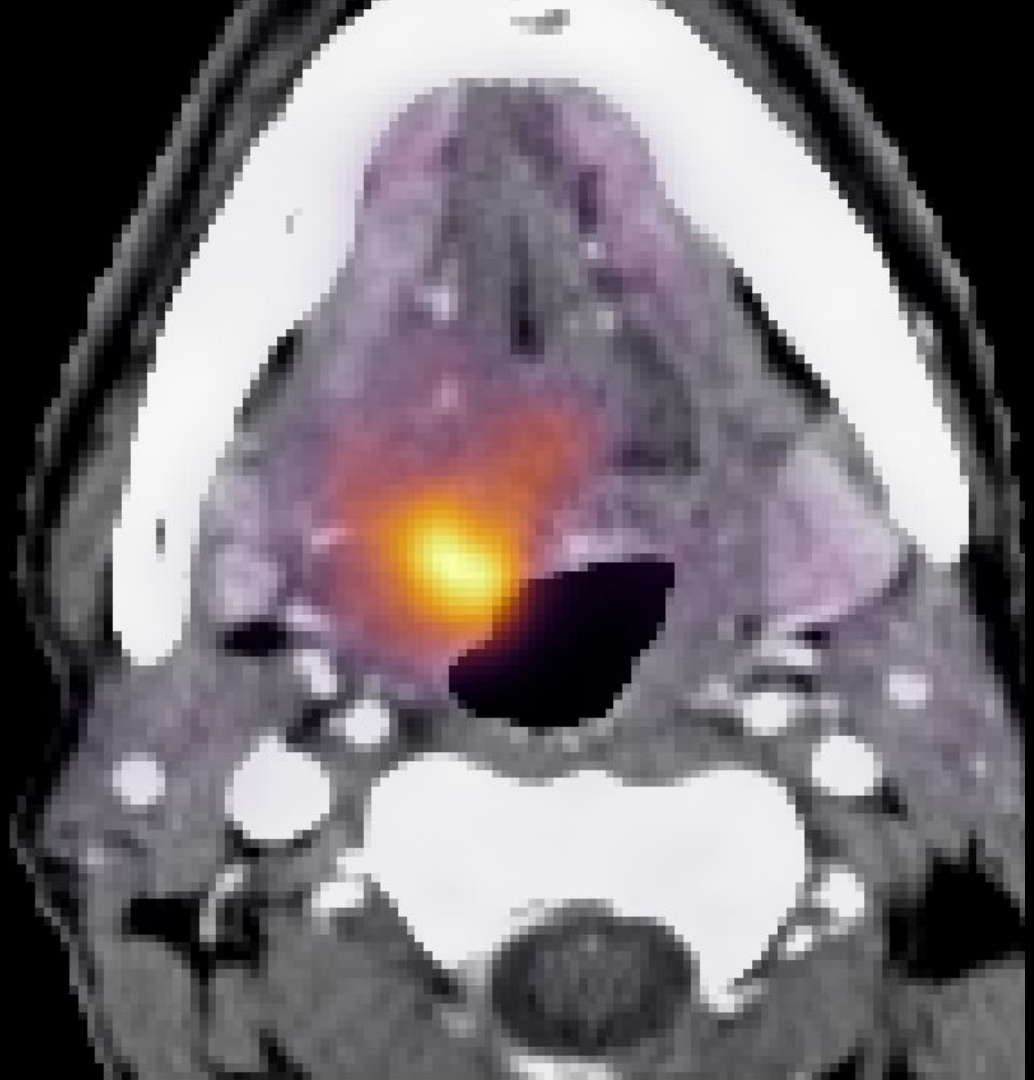
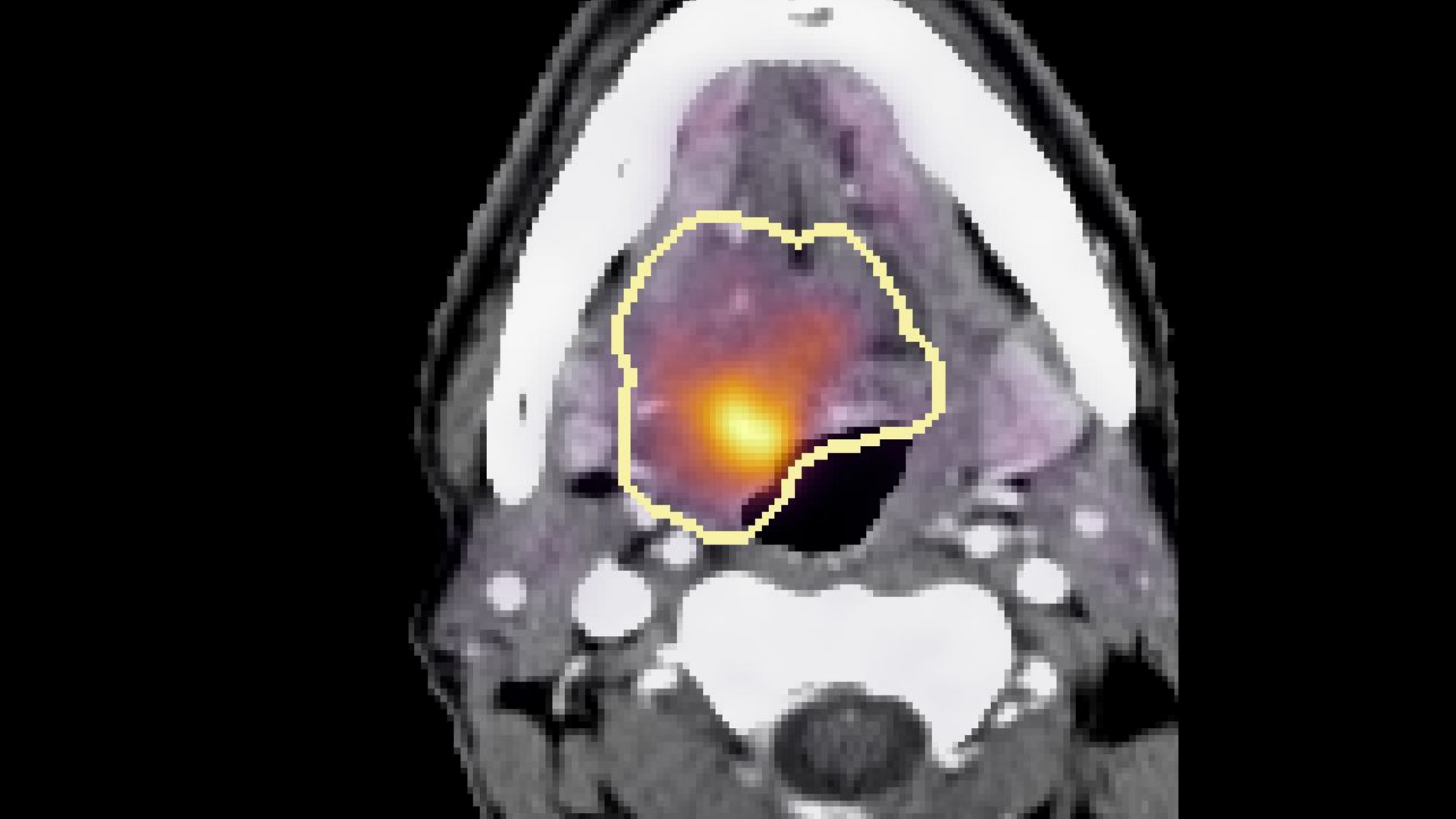
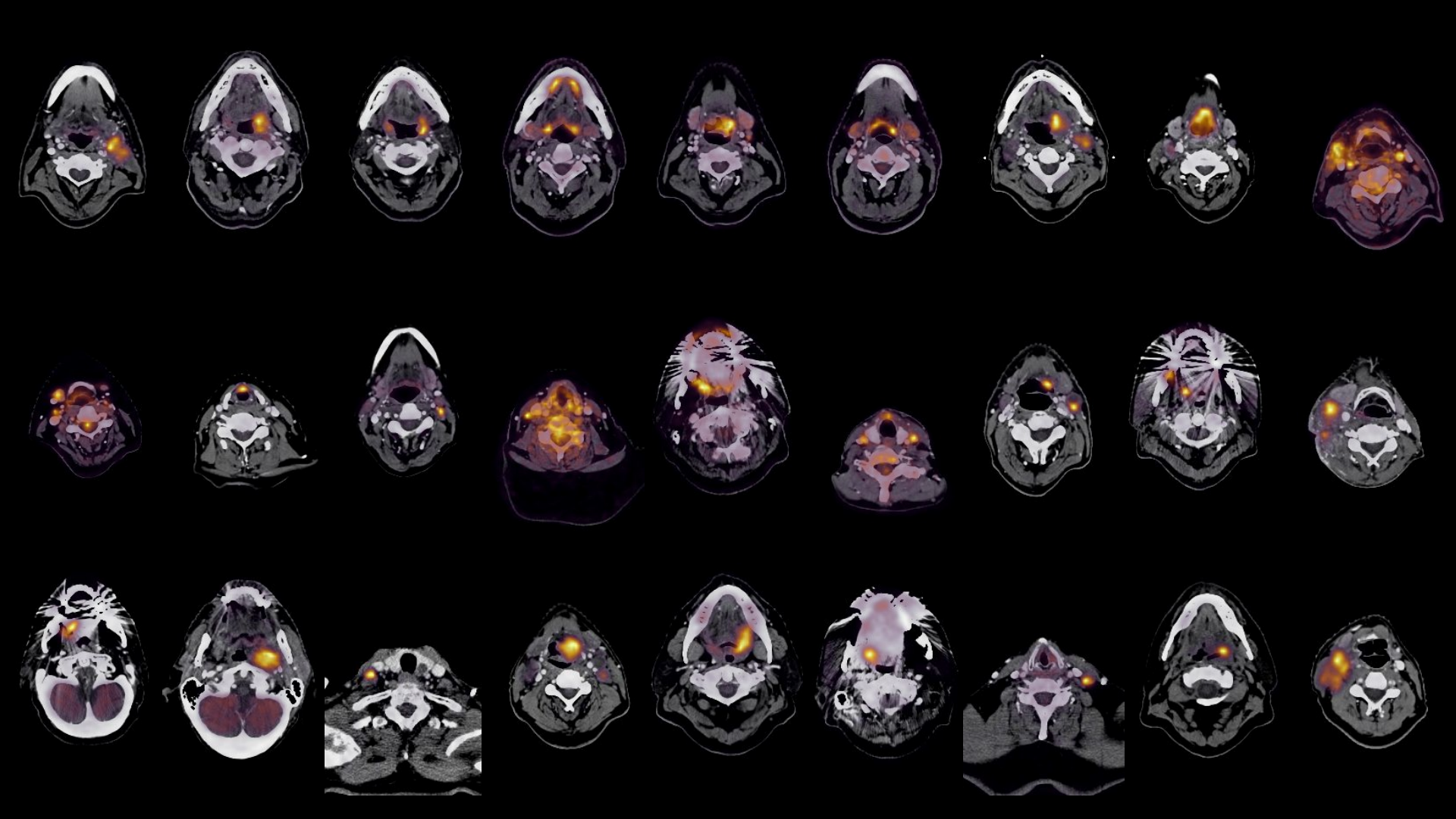
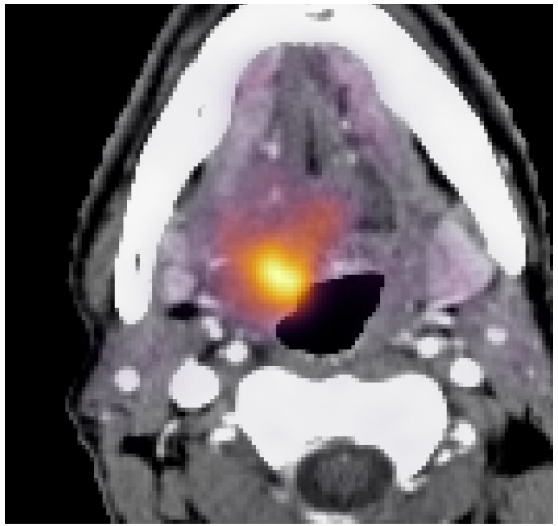




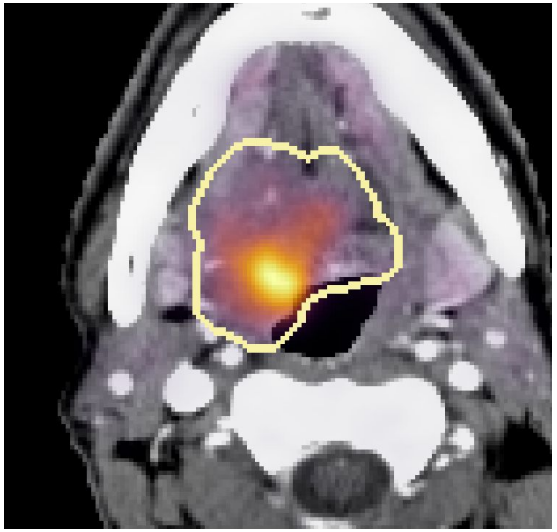




Automatisk inntegning av kreftsvulster i medisinske bilder av pasienter med hode- og nakkekreft



Maskinlæring

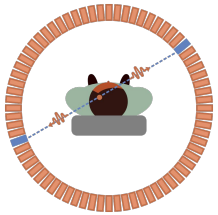


Yngve Mardal Moe

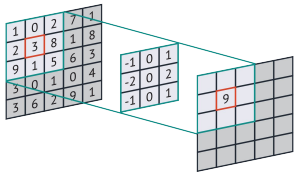
Norges miljø- og biovitenskapelige universitet

UNIVERSITETET I OSLO

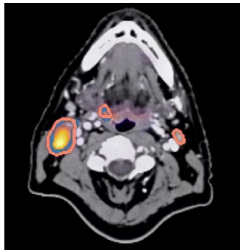
Oslo universitetssykehus



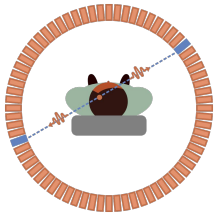
Medisinske bilder



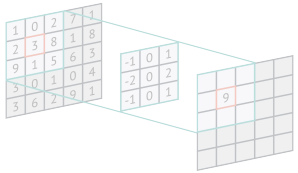
Digitale bilder, dyp læring og konvolusjonelle nett



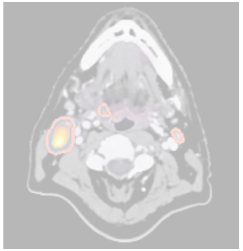
Resultater og observasjoner



Medisinske bilder

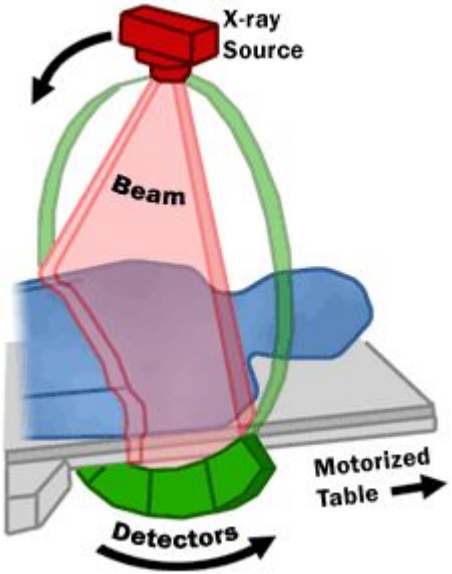


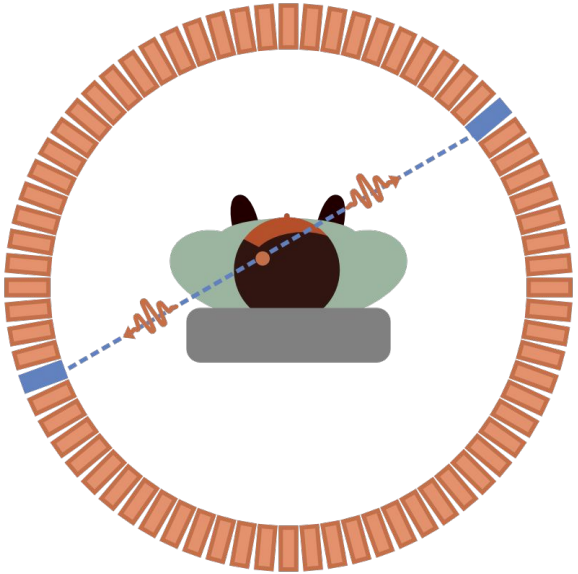
Digitale bilder, dyp læring
og konvolusjonelle nett



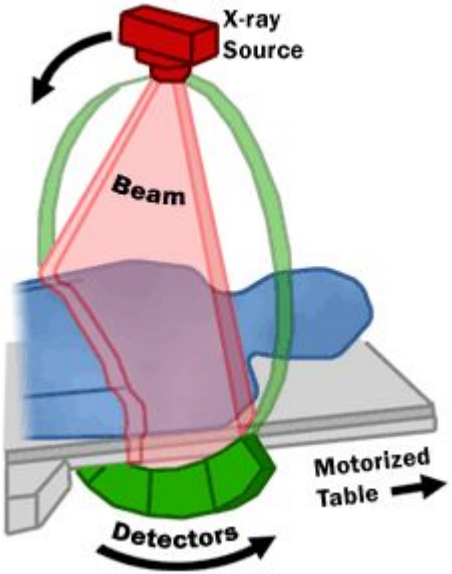
Resultater og
observasjoner

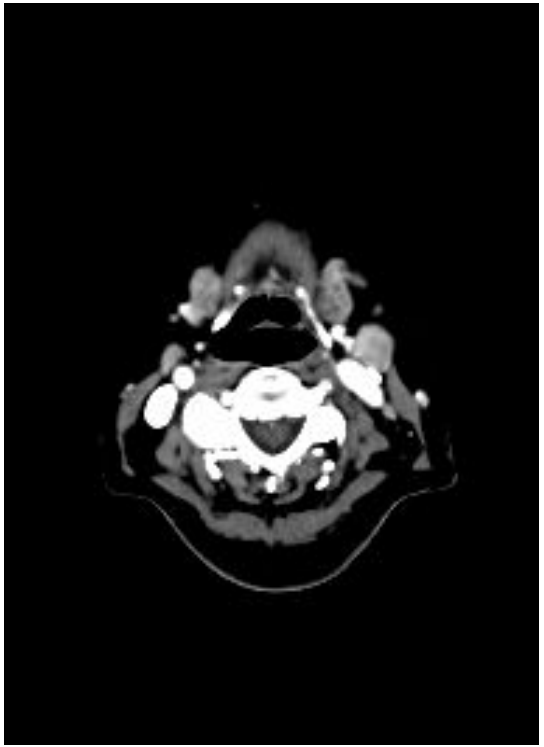
For å finne kreftsvulster brukes en kombinasjon av PET- og CT-bilder



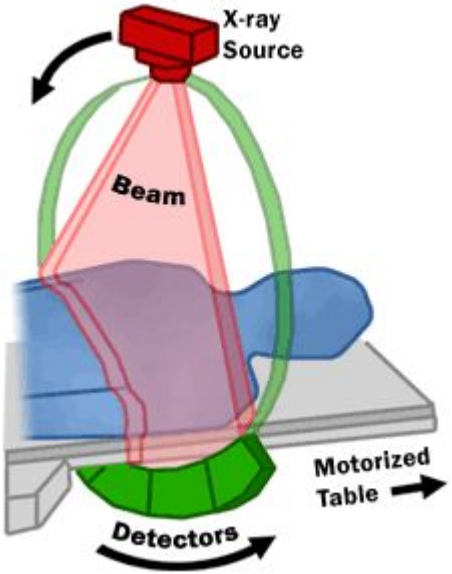


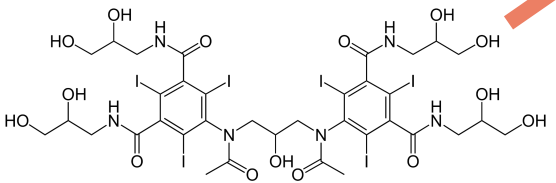
CT bilder måler massetettheten i kroppen

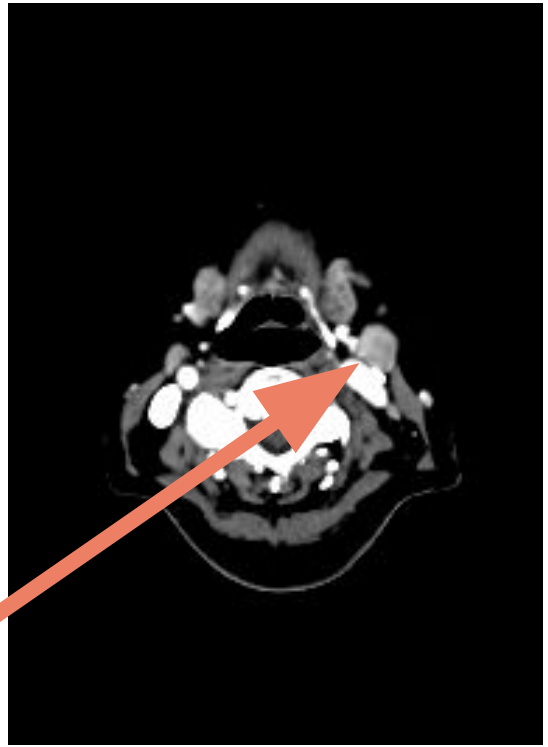




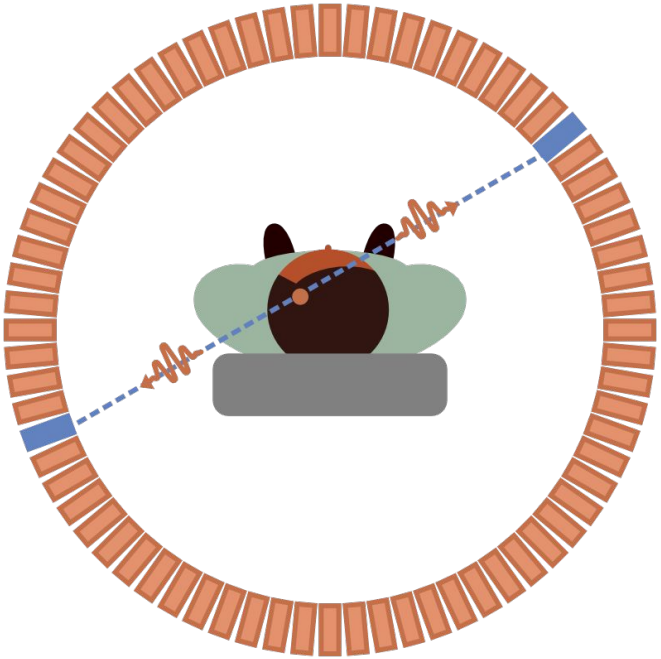
For å gjøre svulsten tydeligere, injiseres pasientene med kontrastvæske

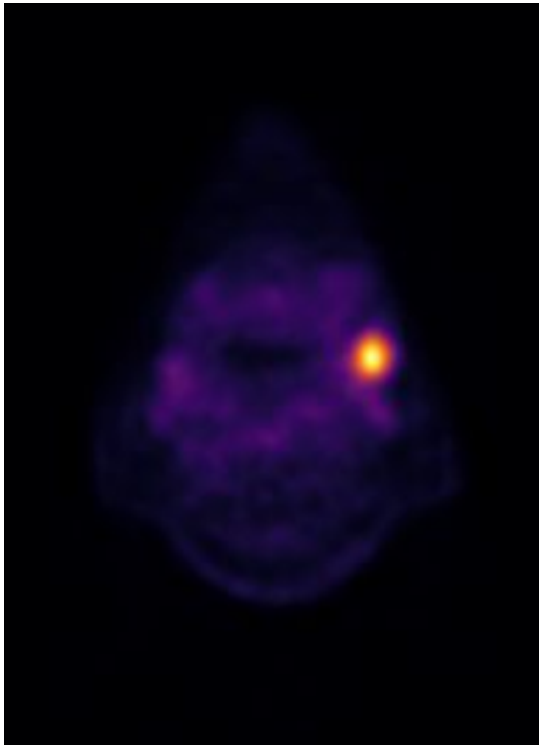


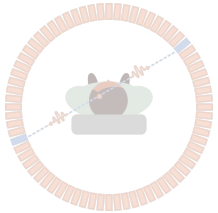




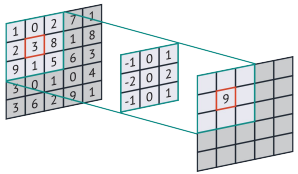
PET-metoden som brukes gir et mål på energiforbruk i kroppen



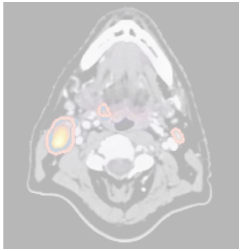




Medisinske bilder

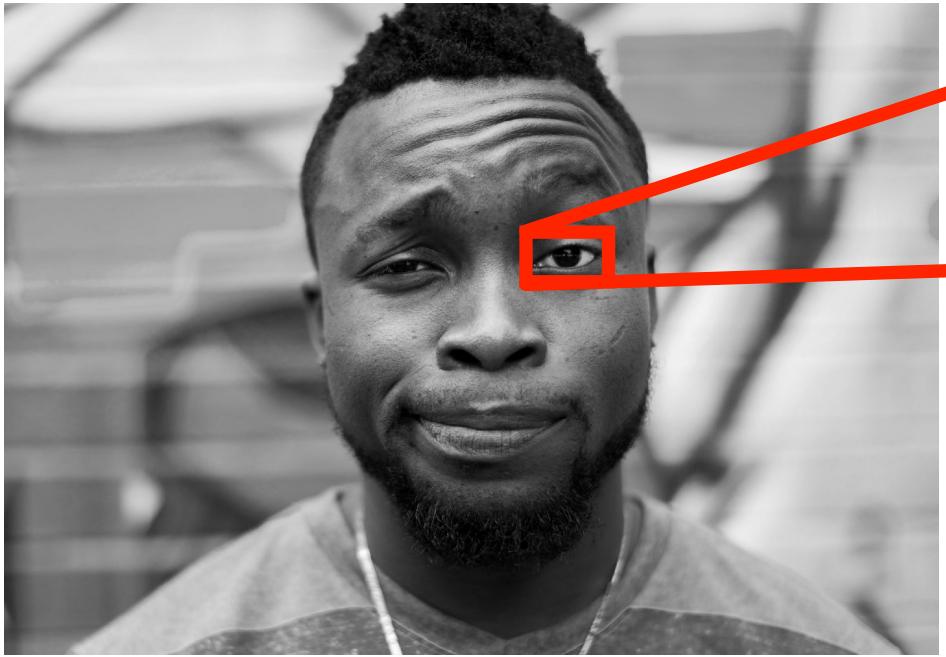


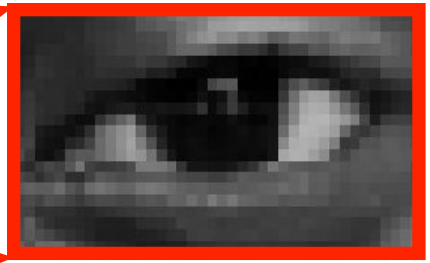
Digitale bilder, dyp læring og konvolusjonelle nett



Resultater og observasjoner

Datamaskiner representerer digitale bilder som en mengde tall



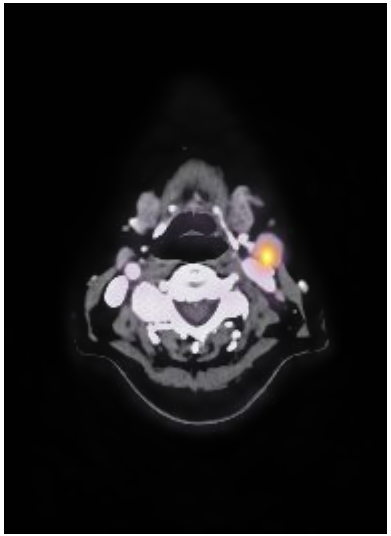


16	11	10	16	24	40	51	61
12	12	14	19	26	58	60	55
14	13	16	24	40	57	69	56
14	17	22	29	51	87	80	62
18	22	37	56	68	109	103	77
24	35	55	64	81	104	113	92
49	64	78	87	103	121	120	101

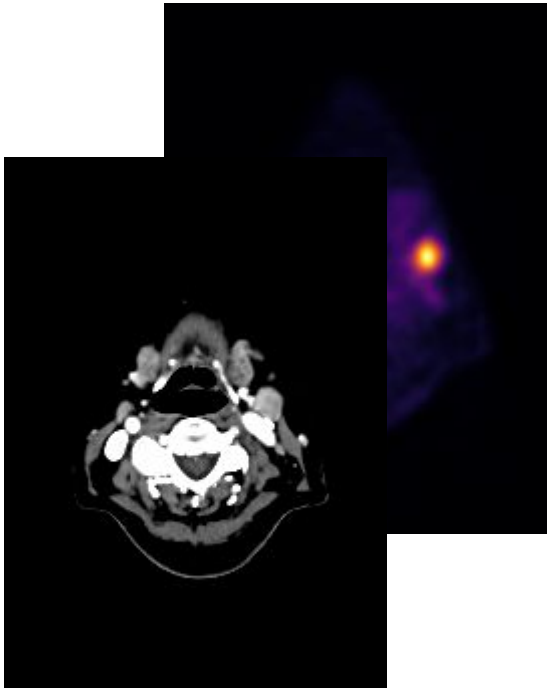
Og fargeinformasjon i forskjellige *fargekanaler*

=

Slike bildekanaler kan representere andre ting enn farger

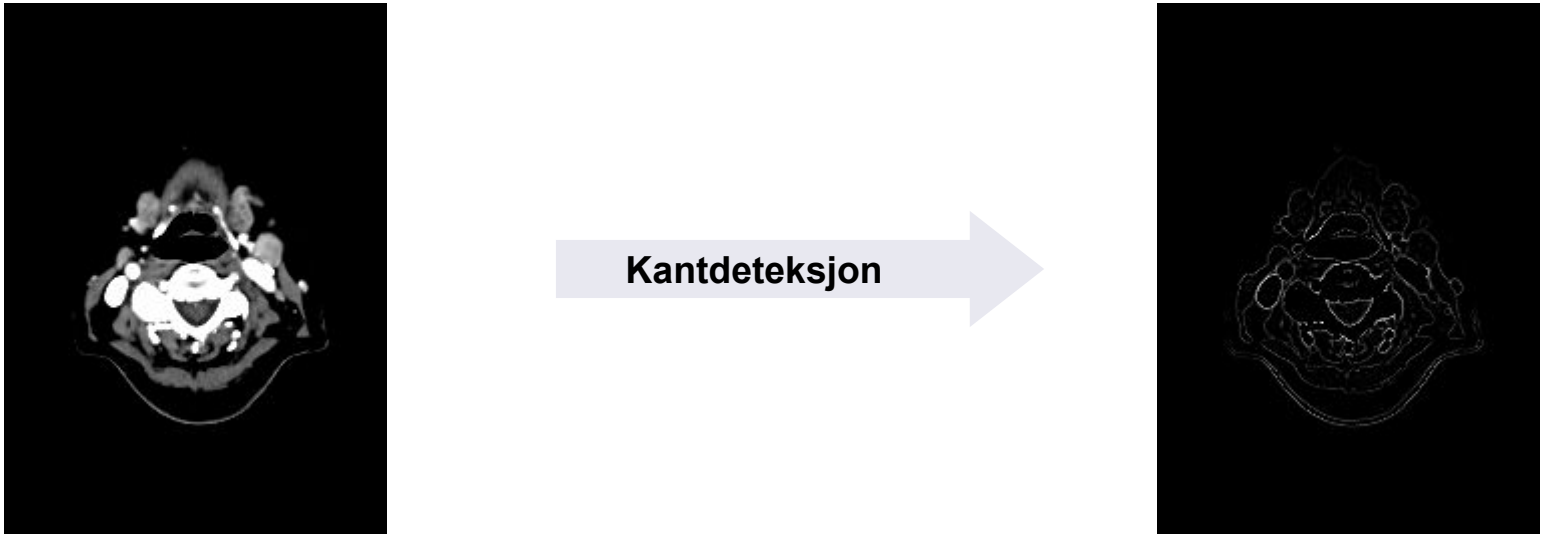


=

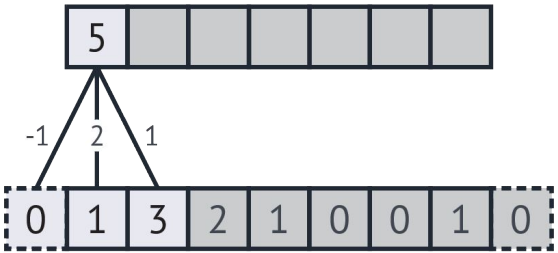


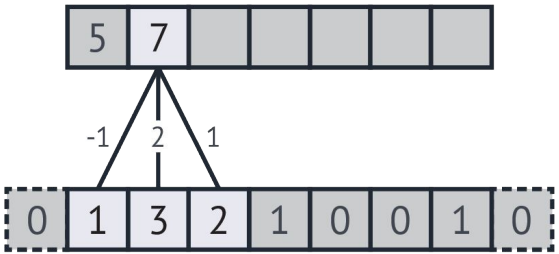
Vi ønsker metoder som finner nyttig informasjon i bilder

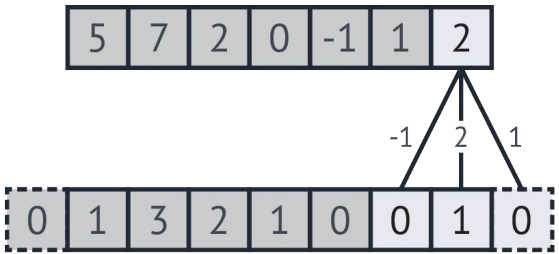
Kanter er et eksempel på nyttig informasjon i bilder

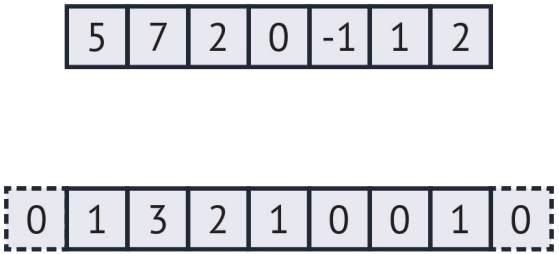


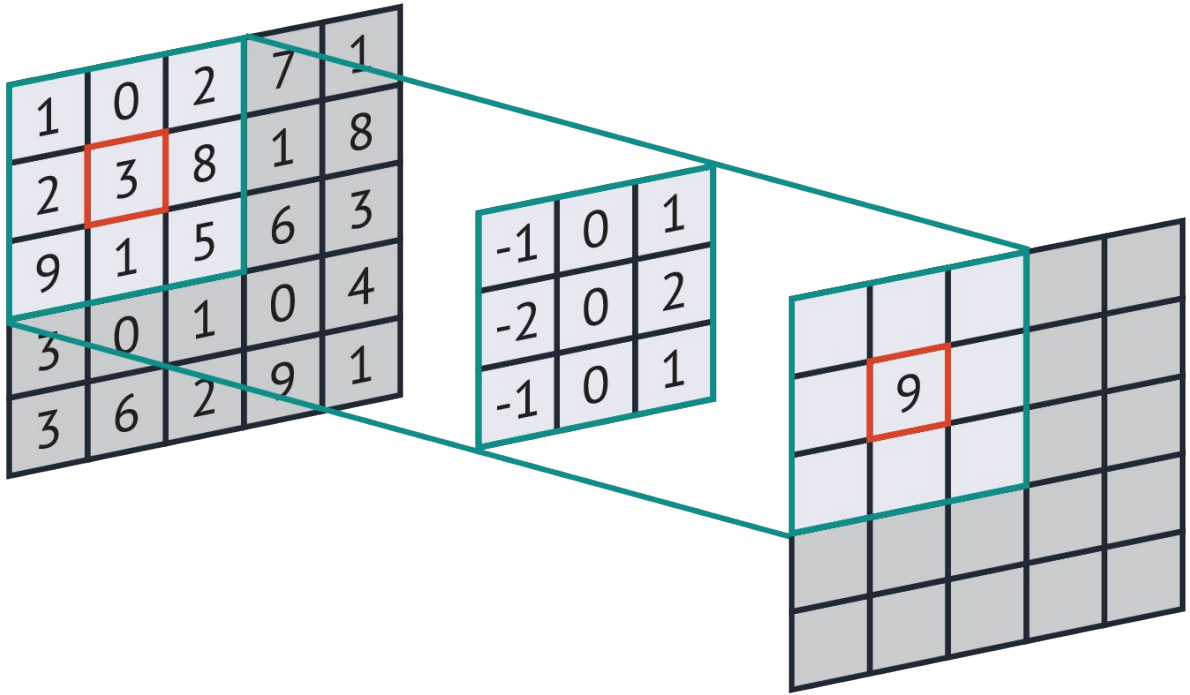
Hvordan kan datamaskiner finne informasjon i bilder?



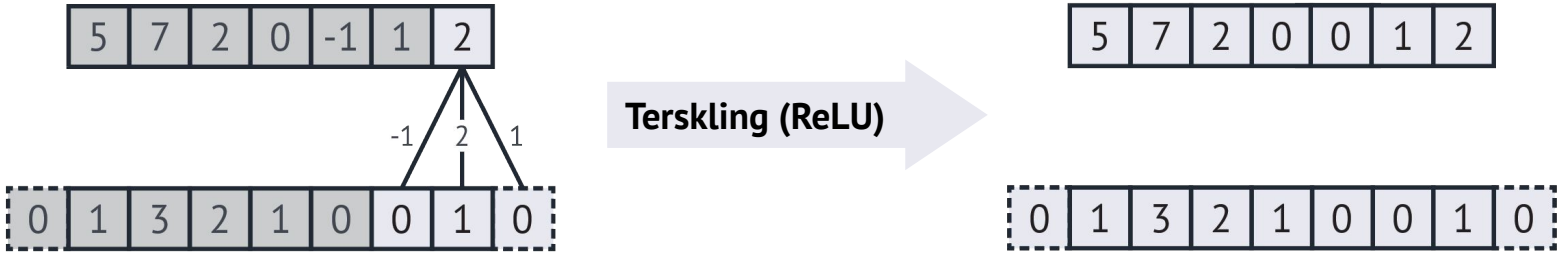




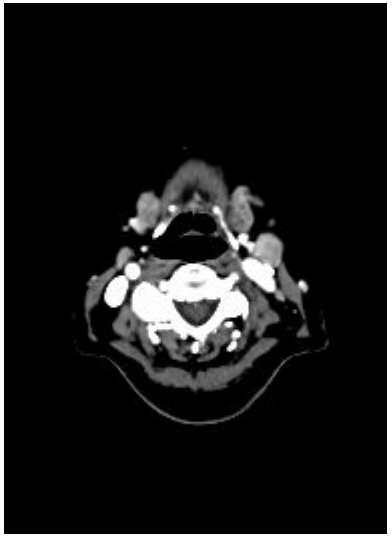



1 0 2 7 1
2 3 8 1 8
9 1 5 6 3
3 0 1 0 4
3 6 2 9 1
-1 0 1
-2 0 2
-1 0 1
9

Konvolusjonslag består av en konvolusjon etterfulgt av terskling

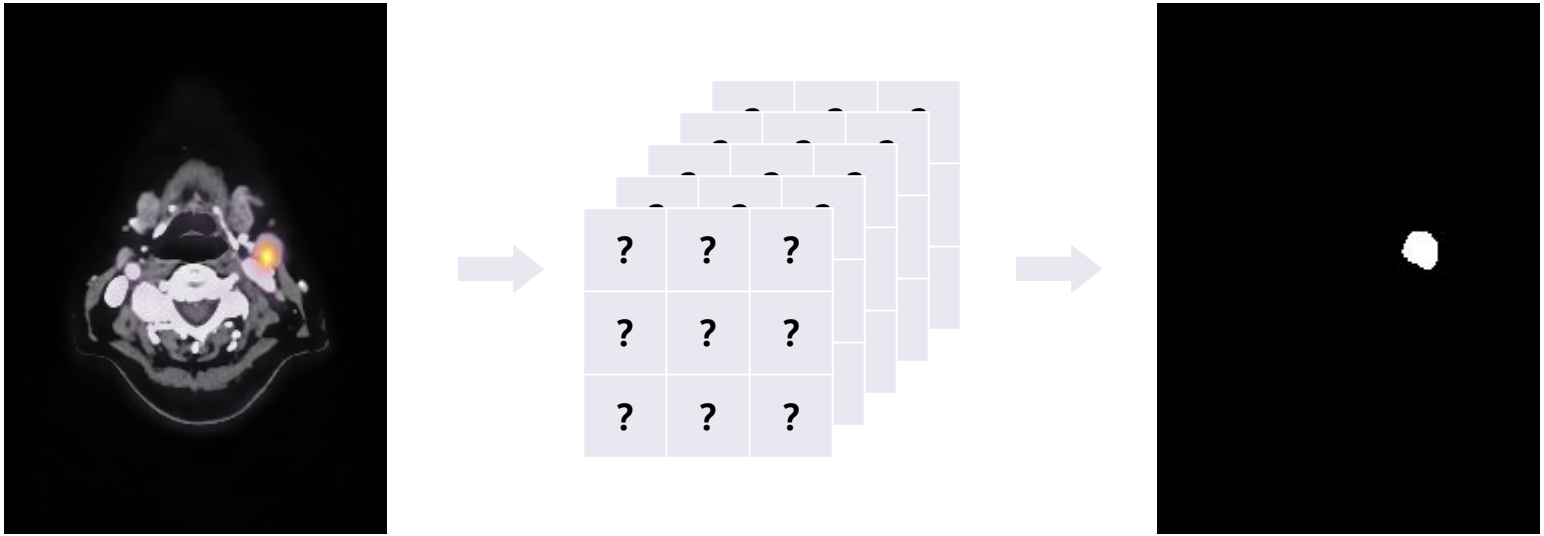


Kanter er et eksempel på nyttig informasjon i bilder

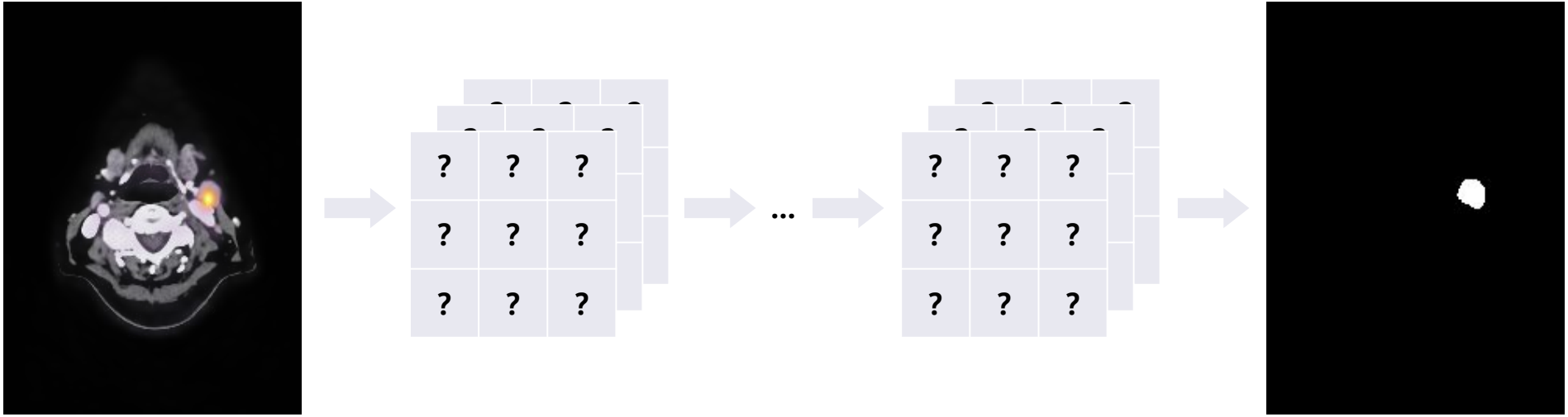


0	1	0
1	-4	1
0	1	0

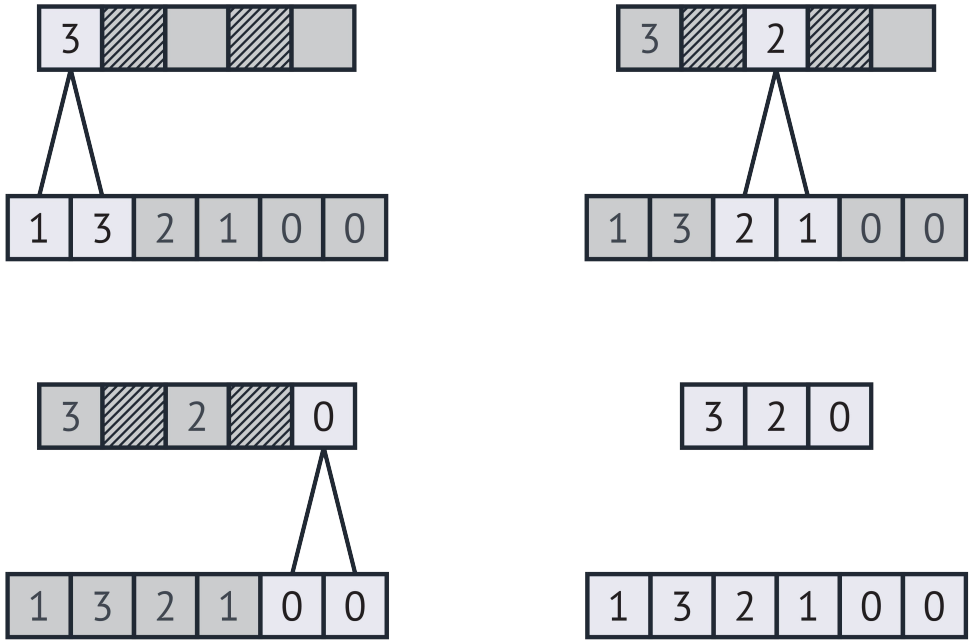
Men vi trenger mer enn bare kanter for å finne kreft



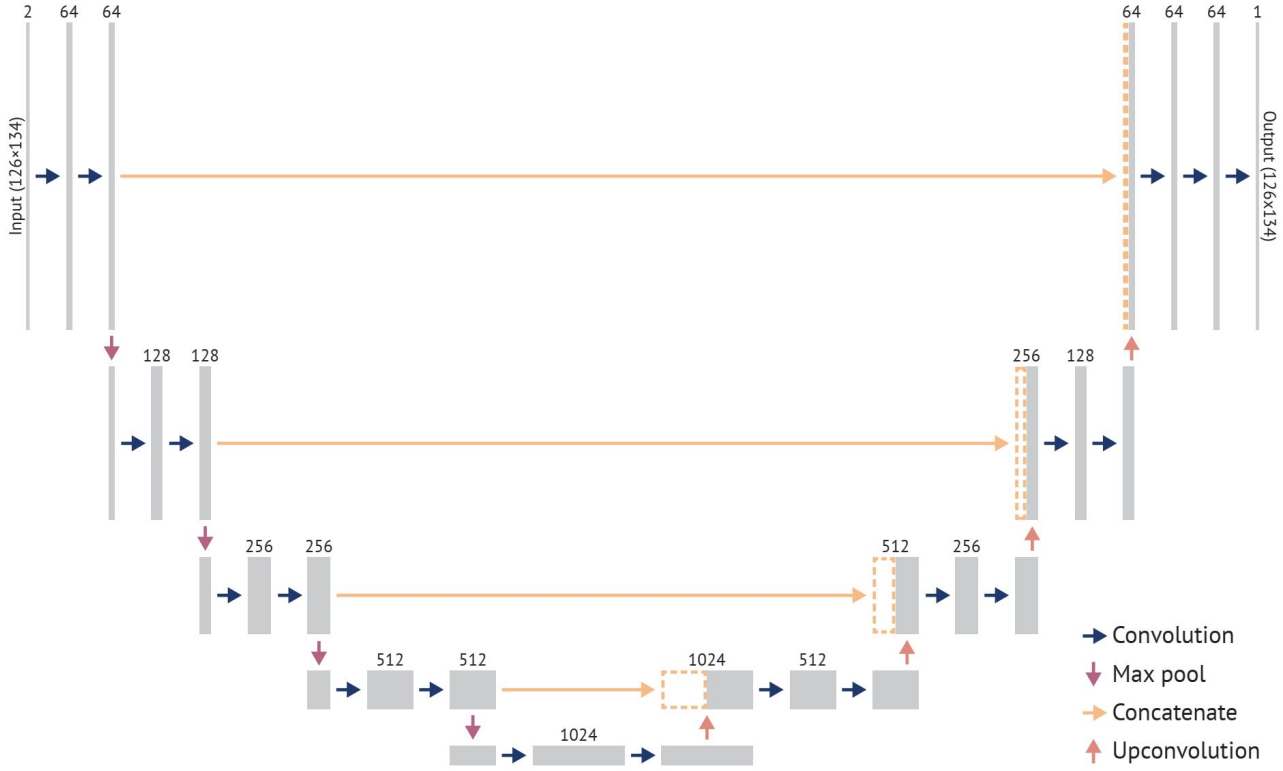
Ved å sette slike konvolusjonslag etter hverandre får vi et konvolusjonelt nett



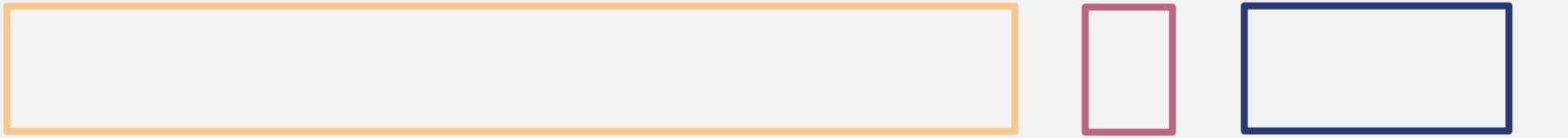
Vi pleier også nedskalere bildene mellom konvolusjonslagene

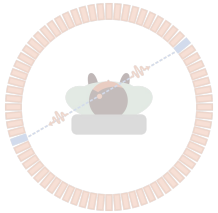


For bildesegmentering brukes ofte U-Net modellen

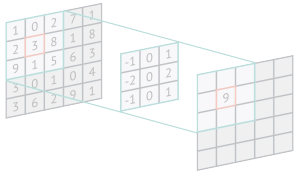


Vi delte datasettet inn i tre deler for å evaluere modellen

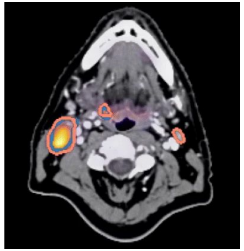




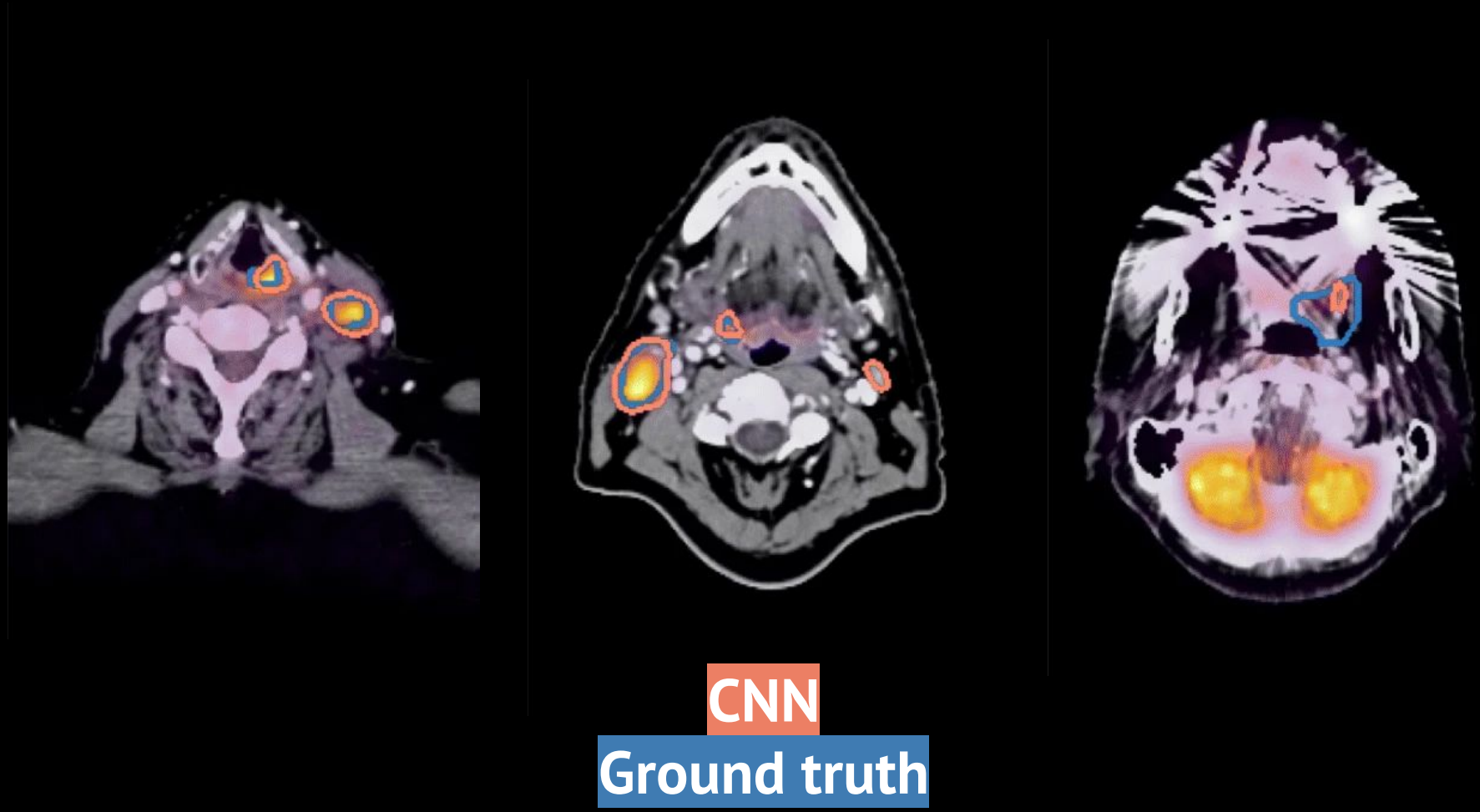
Medisinske bilder



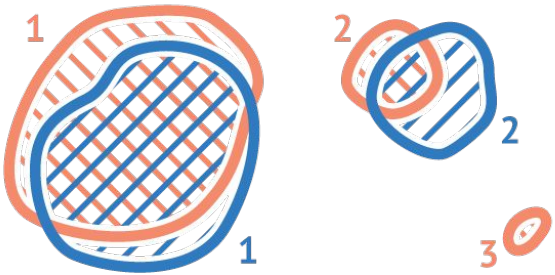
Digitale bilder, dyp læring
og konvolusjonelle nett



Resultater og observasjoner


CNN
Ground truth

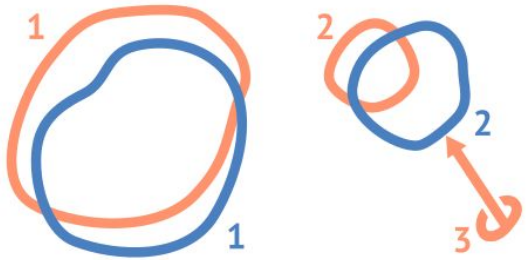
Dice score



$$\text{Dice} = \frac{\square}{\frac{\square + \square}{2}}$$

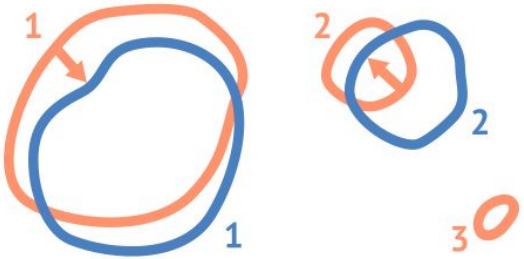
71% overlapp

Vanlig målt avstand



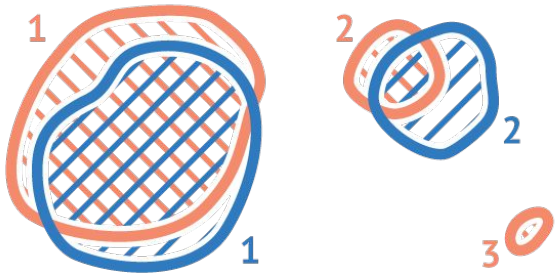
4.7 mm avstand

Strukturvis målt avstand



1.0 mm avstand

Nøyaktighet



86 % sensitiv

33 % nøyaktig

De falske positive inntegningene var ofte veldig små

Sanne inntegninger:

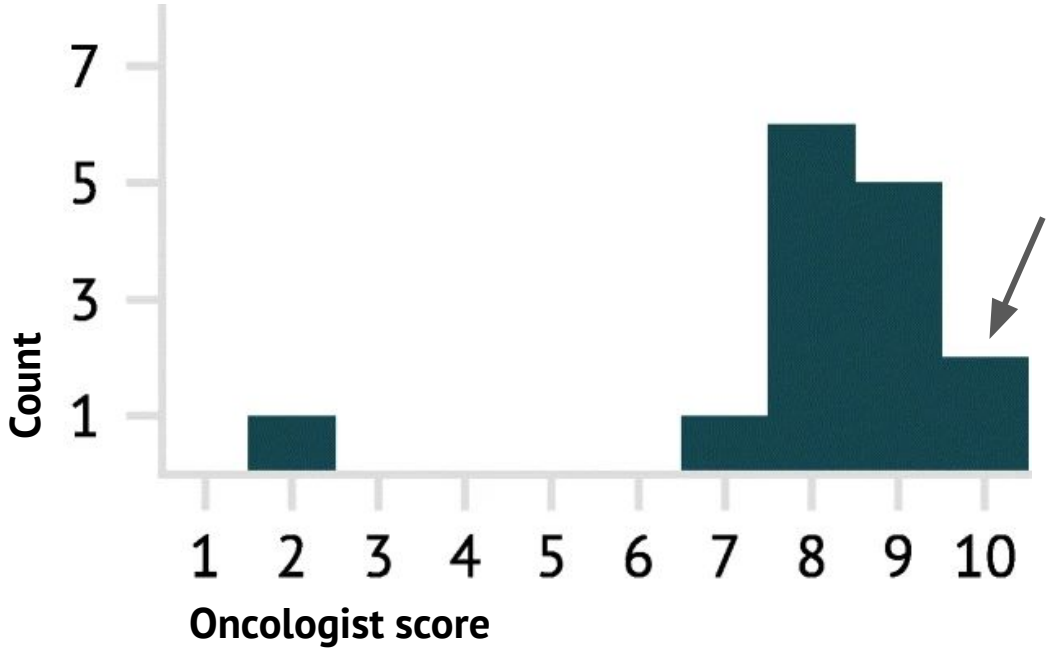
$15\ cm^3$

Falskt positive inntegninger:

$0.54\ cm^3$

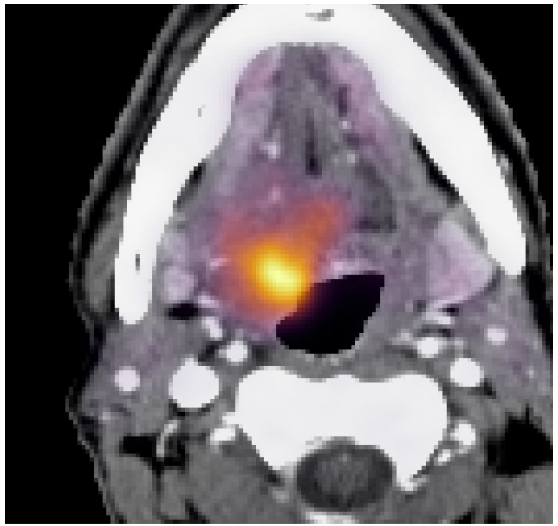
Vi hadde også hjelp av en erfaren onkolog for å evaluere inntegningene

Onkologen var generelt veldig fornøyd med auto-inntegningene

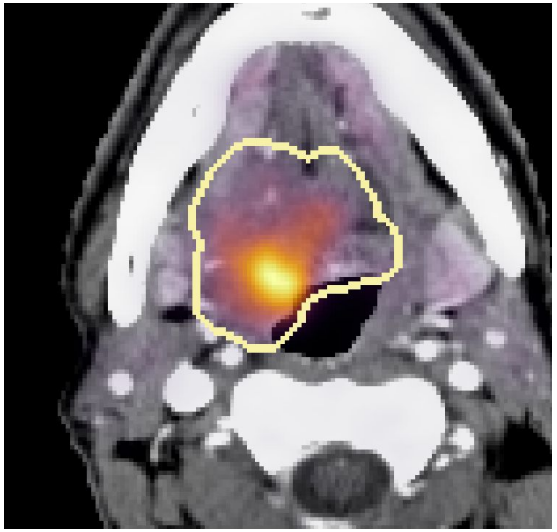


Klarte ikke se om inntegningen var manuell eller ikke

Automatiske inntegningsmetoder kan brukes for å minke arbeidsbyrden under stråleterapiplanlegging



Maskinlæring



Norges miljø- og biovitenskapelige universitet

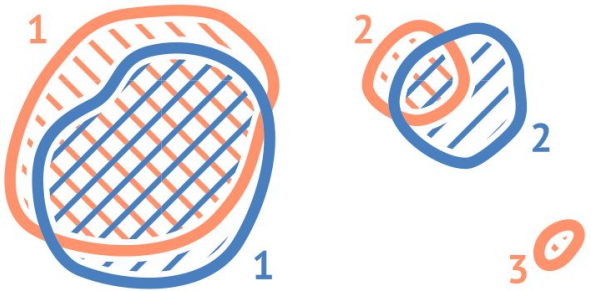
UNIVERSITETET I OSLO

Oslo universitetssykehus

Takk for meg

Additionally, we introduced new surface-wise performance metrics

Coverage Fraction (CFrac)



$$CFrac_1 = \frac{\square_1}{\square_1} = 0.7 \qquad CFrac_1 = \frac{\square_1}{\square_1} = 0.8$$

$$CFrac_2 = \frac{\square_2}{\square_2} = 0.6 \qquad CFrac_2 = \frac{\square_2}{\square_2} = 0.3$$

$$CFrac_3 = \frac{\square_3}{\square_3} = 0$$

– CNN
– Oncologist

Structure-wise sensitivity and PPV

$PPV_{CNN} = 0.7$ $\qquad$ $Sens_{GT} = 0.5$

In deep learning, we hope to find functions that utilise the structure of our data to perform a given task

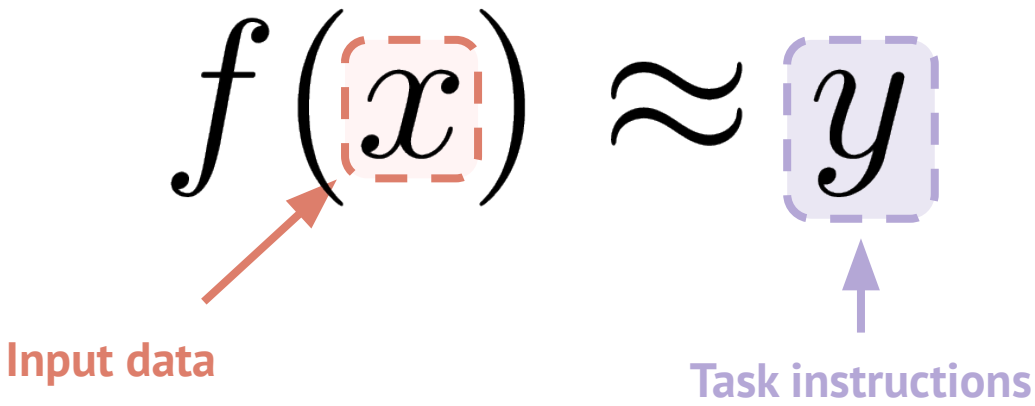
$$f(x) \approx y$$

In deep learning, we hope to find functions that utilise the structure of our data to perform a given task

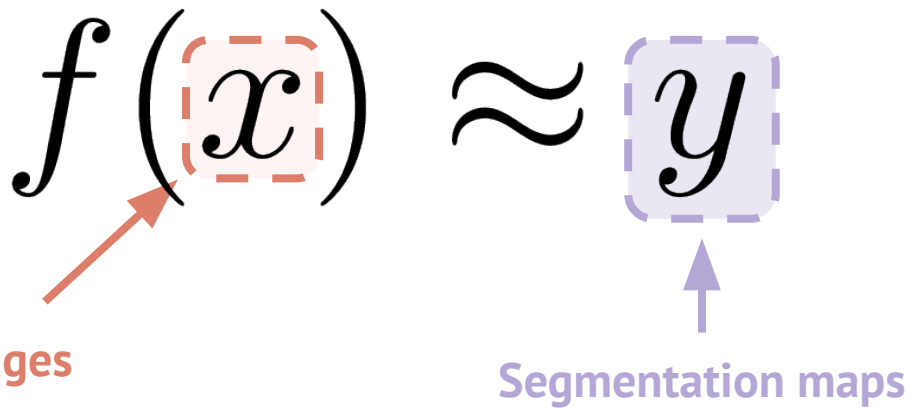
$$f(x) \approx y$$

Input data

In deep learning, we hope to find functions that utilise the structure of our data to perform a given task



In deep learning, we hope to find functions that utilise the structure of our data to perform a given task



To discover this function, we parametrise it by a set of weights and try to find the optimal weights

$$f(x; W) \approx y$$

To discover this function, we parametrise it by a set of weights and try to find the optimal weights

$$f(x; W) \approx y$$

Model weights
(parameters)

Neural networks are described as a cascade of function compositions

$$f(x; W) = \breve{f}_n(\breve{f}_{n-1}(...\breve{f}_2(\breve{f}_1(x; W_1), W_2)..., W_{n-1}), W_n)$$

Neural networks are described as a cascade of function compositions

$$f(x; W) = \breve{f}_n(\breve{f}_{n-1}(\ldots \breve{f}_2(\breve{f}_1(x; W_1), W_2)\ldots, W_{n-1}), W_n)$$

Neural network

Layer functions

Each layer is generally an affine map (linear plus constant) followed by an element-wise nonlinear activation function

$$f(x; W) = \breve{f}_n(\breve{f}_{n-1}(...\breve{f}_2(\breve{f}_1(x; W_1), W_2)..., W_{n-1}), W_n)$$

$$\breve{f}_k(x; A, b) = \phi(Ax + b)$$

Each layer is generally an affine map (linear plus constant) followed by an element-wise nonlinear activation function

$$f(x; W) = \breve{f}_n(\breve{f}_{n-1}(...\breve{f}_2(\breve{f}_1(x; W_1), W_2)..., W_{n-1}), W_n)$$

$$\breve{f}_k(x; A, b) = \phi(Ax + b)$$

Layer weights (W_k above)

Each layer is generally an affine map (linear plus constant) followed by an element-wise nonlinear activation function

$$f(x; W) = \breve{f}_n(\breve{f}_{n-1}(...\breve{f}_2(\breve{f}_1(x; W_1), W_2)..., W_{n-1}), W_n)$$

$$\breve{f}_k(x; A, b) = \phi(Ax + b)$$

Layer weights (W_k above)

Affine map

Each layer is generally an affine map (linear plus constant) followed by an element-wise nonlinear activation function

$$f(x; W) = \breve{f}_n(\breve{f}_{n-1}(...\breve{f}_2(\breve{f}_1(x; W_1), W_2)..., W_{n-1}), W_n)$$

$$\breve{f}_k(x; A, b) = \phi(Ax + b)$$

Layer weights (W_k above)

Nonlinear activation function

Affine map

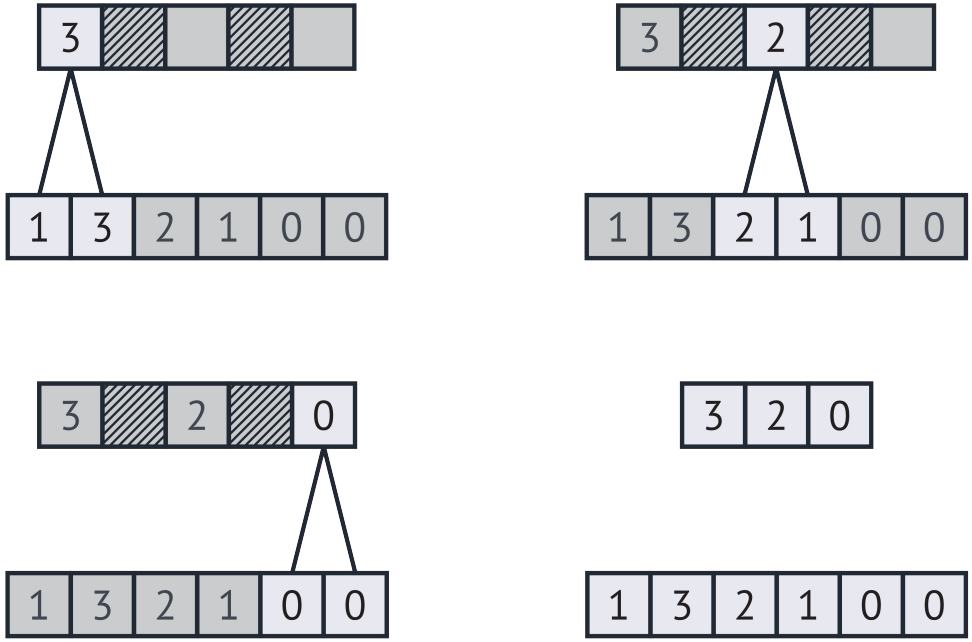
The most commonly used activation function is the ReLU function

$$\breve{f}_k(x; A, b) = \phi(Ax + b)$$

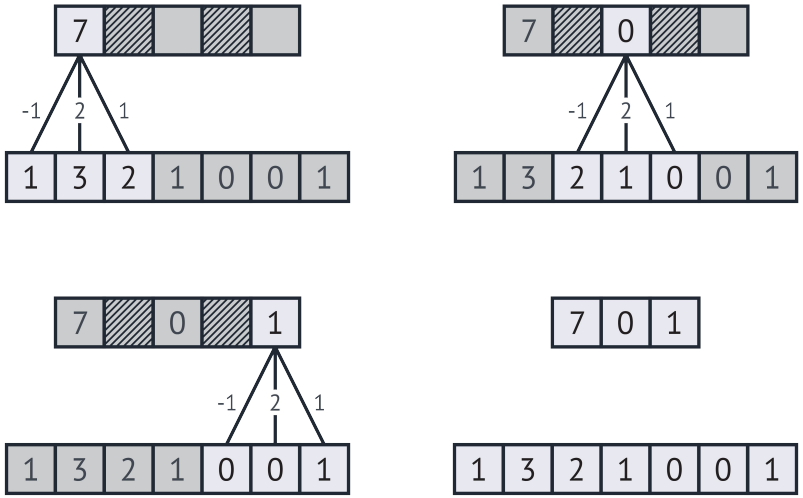
$$\phi(x) = \max(x, 0)$$

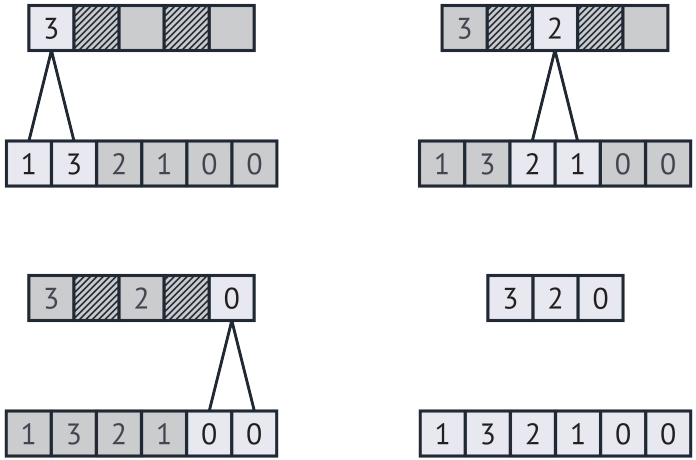
If we just stack convolutions, then the receptive field grows slowly

To combat this, we generally use downsampling, or *pooling* layers

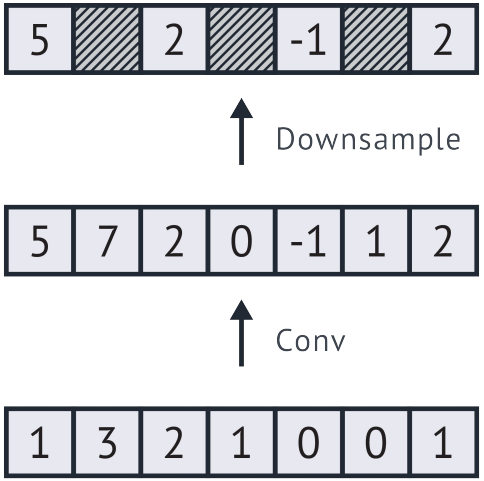


Downsampling is generally either performed via a max-pooling or a strided convolution

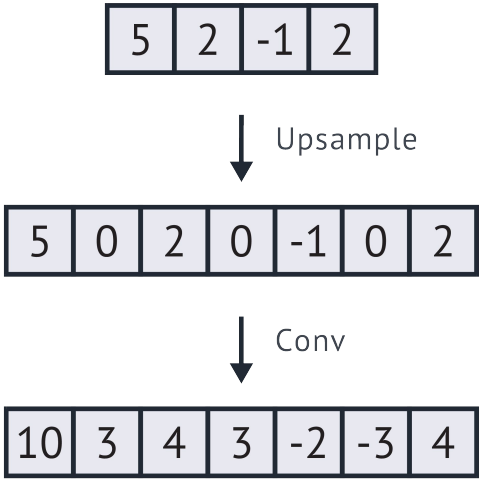




This upscaling is normally done via a transposed strided convolution



Strided convolution



Transposed strided convolution
“Upconvolution”
“Deconvolution”
[...]

We formulate a loss function that measures the severity of our prediction errors

$$L(W;\mathcal{T}) = \sum_{(x^{(i)},y^{(i)})\in\mathcal{T}} l(f(x^{(i)};W),y^{(i)})$$

We formulate a loss function that measures the severity of our prediction errors

$$L(W;\mathcal{T}) = \sum_{(x^{(i)},y^{(i)})\in\mathcal{T}} l(f(x^{(i)};W), y^{(i)})$$

Training example

We formulate a loss function that measures the severity of our prediction errors

$$L(W;\mathcal{T}) = \sum_{(x^{(i)},y^{(i)})\in\mathcal{T}} l(f(x^{(i)};W), y^{(i)})$$

Training example

Training set

We formulate a loss function that measures the severity of our prediction errors

$$L(W;\mathcal{T}) = \sum_{(x^{(i)},y^{(i)})\in\mathcal{T}} l(f(x^{(i)};W),y^{(i)})$$

Training example

Training set

Loss function

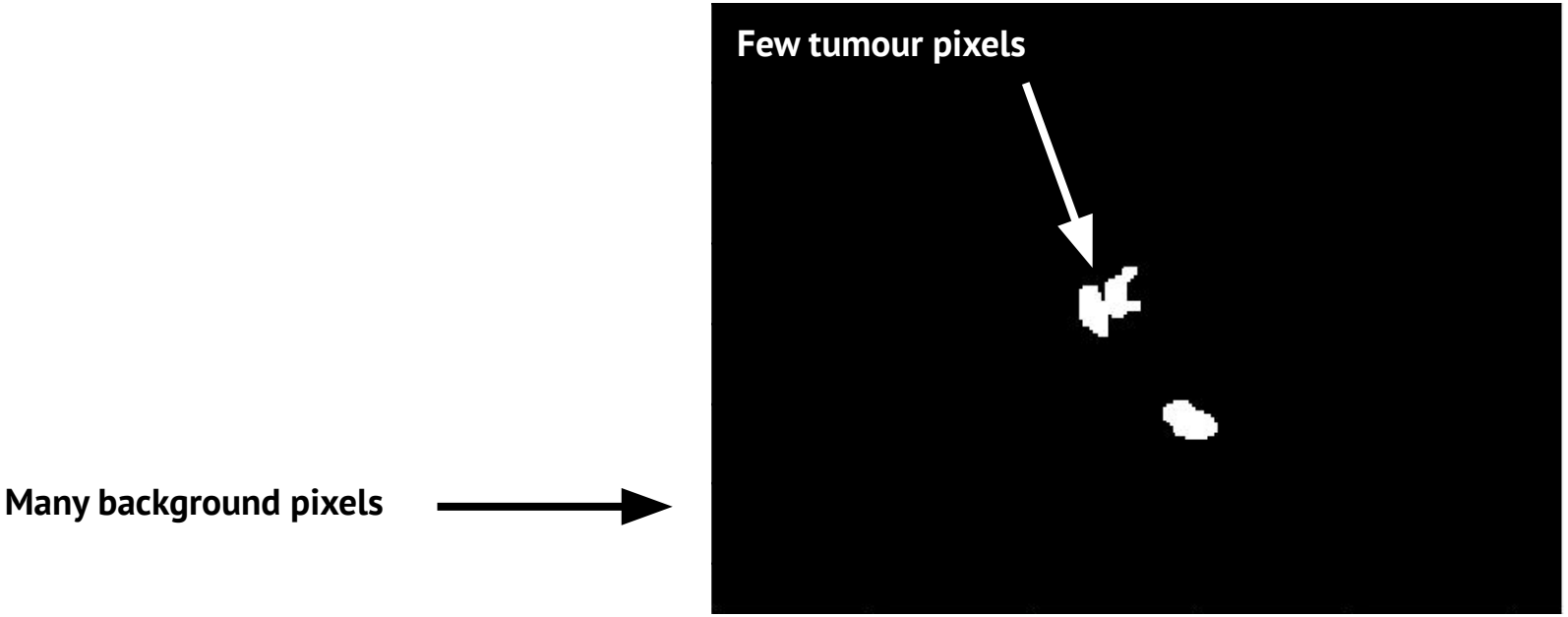
For classification problems, we often use the cross entropy error used in logistic regression

$$l_{\mathrm{CE}}(f(x^{(i)};W),y^{(i)}) = -\sum_{p}\left[y_p^{(i)} log\left(f\left(x_p^{(i)}\right)\right) + \left(1-y_p^{(i)}\right) log\left(1-f\left(x_p^{(i)}\right)\right)\right]$$

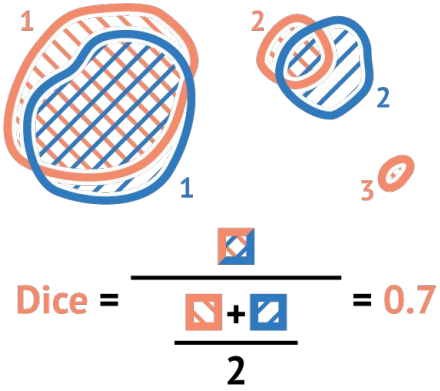
True pixel class

Predicted pixel class

However, the cross-entropy loss puts too much weight into correctly classifying background pixels

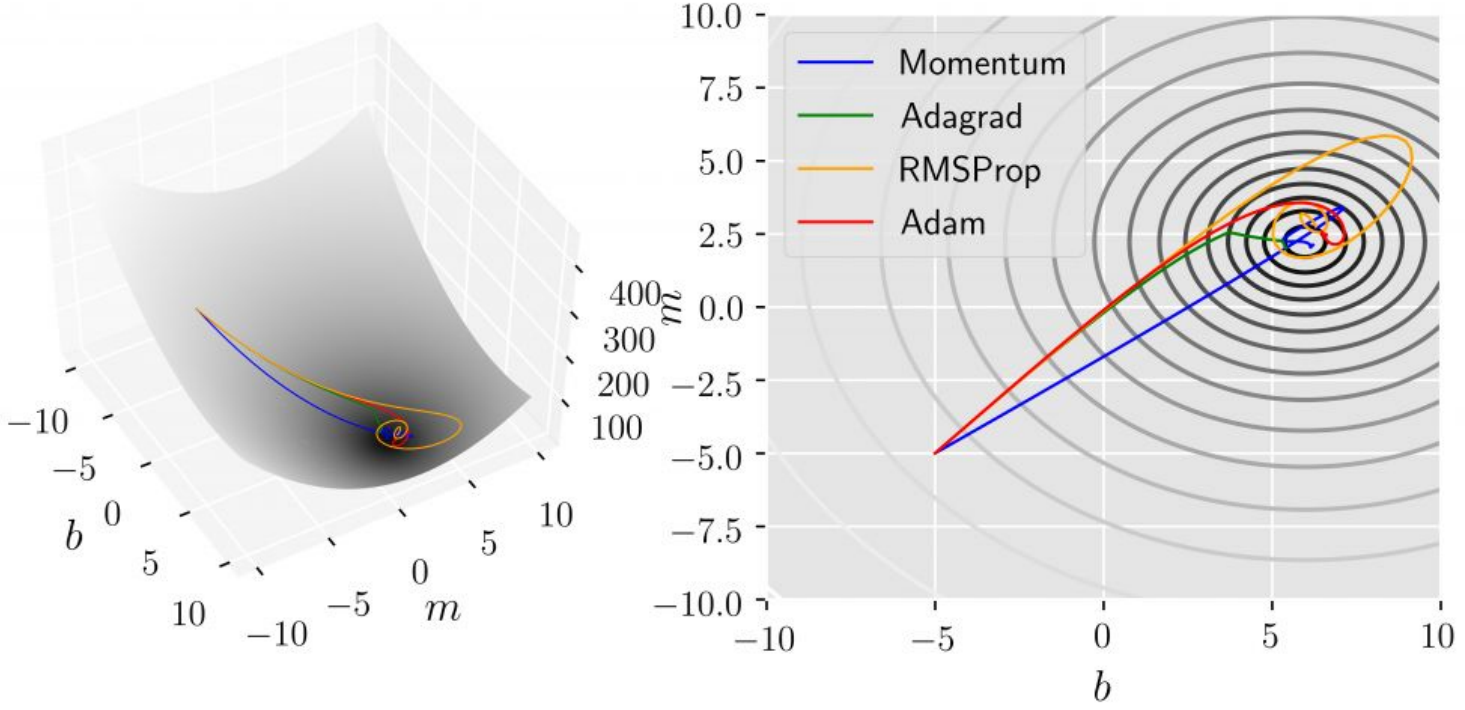


To combat this, we use a Dice loss function, that doesn't take the number of true negatives into account



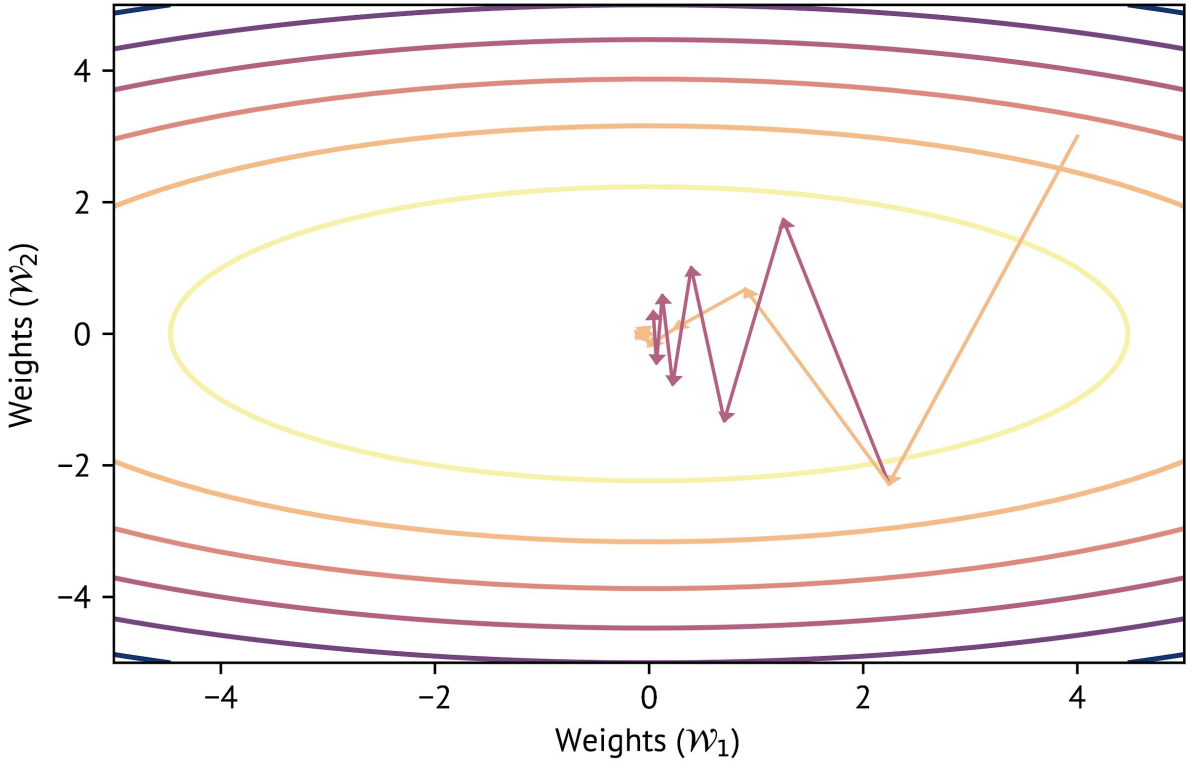
$$l_{\text{dice}}(f(x^{(i)};W),y^{(i)}) = \frac{2\sum_p \left[f(x_p^{(i)})y_p^{(i)}\right]}{\sum_p f(x_p^{(i)}) + \sum_p y_p^{(i)}}$$

First minimise the loss using an adaptive momentum method such as Adam



[Figure by Sebastian Hell]

To minimise the loss, we use a stochastic first order optimisation method such as ADAM



If the generalisation error is large, try using momentum SGD optimiser instead of an adaptive momentum method