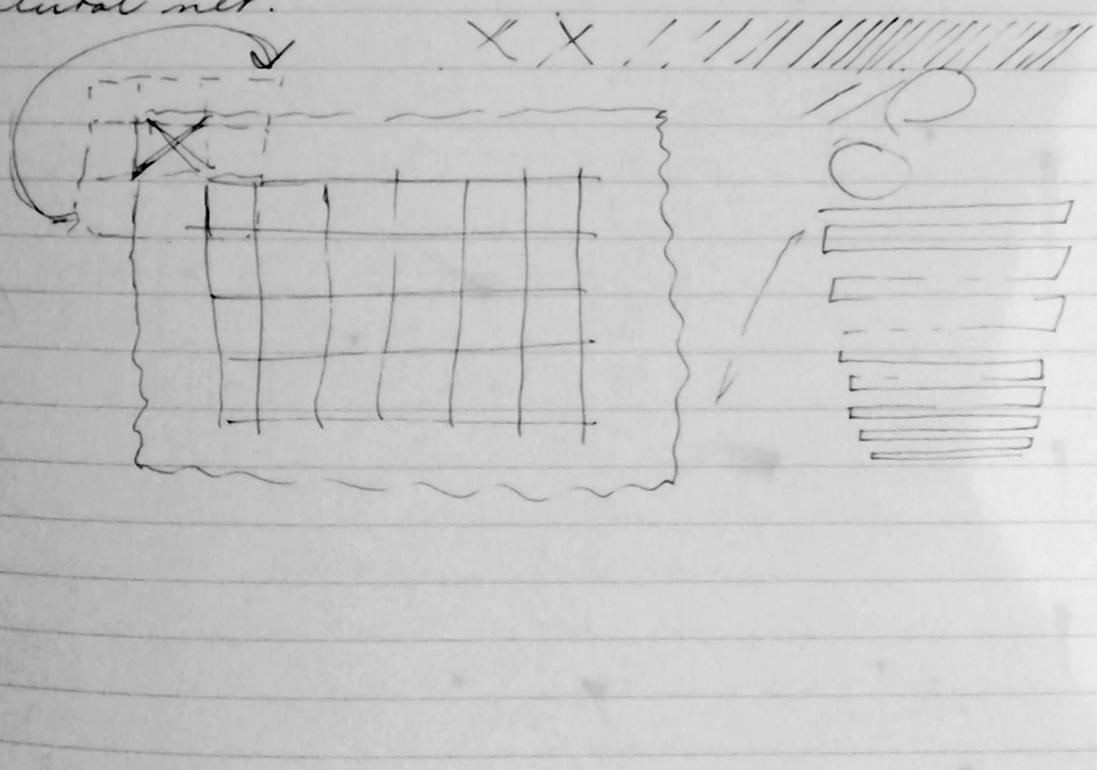
U-net

Convolutional neural network:
  setosa.io/ev/image-kernels
Convolute*n   Each node is a convolution.

Convolve 3x3 → ReLU
Copy and crop
Max pool 2x2 stride 2
Up convolution 2x2
1x1 Convolution
Back propagation

Understanding Backpropagation.

Given error function E, determine a minimum for this, ie, minimize E(f net) where net is our function representation of our neural net.



Backpropagation, cont'

4/2/2018

Error function: J (Unet uses E)

$$J \leftarrow -\sum_{x \in \Omega} w(x) \log(p_{\ell(x)}(x))$$

$$p_k(x) = \exp(a_k(x)) / \left(\sum_{k'=1}^{K} \exp(a_{k'}(x))\right)$$

$$\mathcal{D} \leftarrow \{(\vdash \div 2 \dagger \vdash) * \circledcirc \omega\}$$

~~$\mathcal{E} \leftarrow \{+/ \ldots\}$~~

$$\mathcal{E} \leftarrow \{+/ \circledast (\alpha \times +/\omega) \div (\sim\alpha) \times 0 \llbracket \omega\}$$

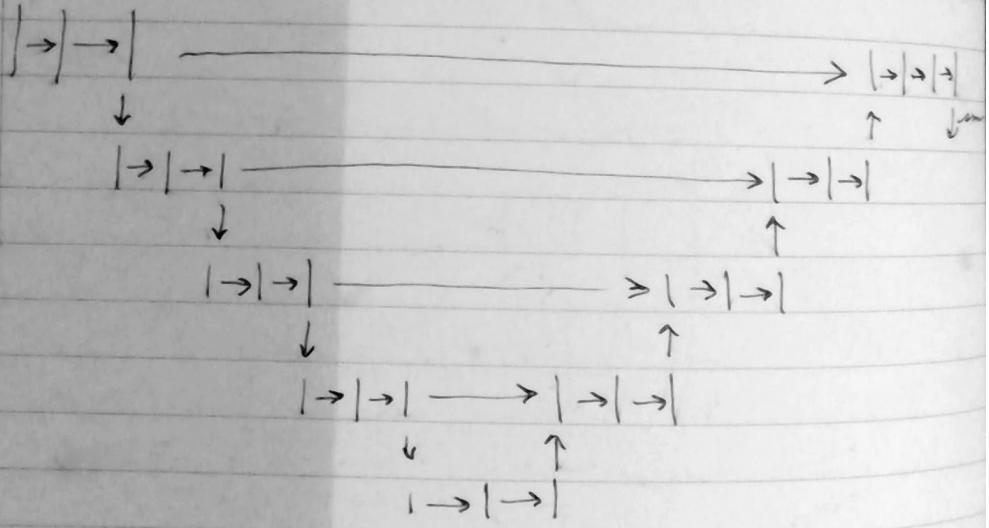
$$\{+/ \circledast (\alpha \times 1 \llbracket \omega) + (\sim\alpha) \times 0 \llbracket \omega\}$$

We use no biasing and no edge weighting in the loss function.

Learning rate is 0.99.

Updating W weights is $W = W - \alpha \frac{\Delta E}{\Delta W}$

$\alpha \leftarrow$ Learning rate. ← ←



conv 1×1, softmax

conv 3×3, relu

up-conv 2×2

copy and crop

max-pool

Backpropagations con't

4/3/2020

Conv 1×1, soft max $\frac{\delta \mathcal{E}}{}$

$(\rho W) \equiv 2\ n\ 1\ 1 \quad a^{-1} \rightarrow (1{\times}1)\ z \rightarrow sm \rightarrow a$

$W \nearrow$

$W = W - \alpha \frac{\delta \mathcal{E}}{\delta W} \qquad W \xrightarrow{1\times 1} z \xrightarrow{sm} a \quad a\ a\ a$

$\frac{\delta \mathcal{E}}{\delta W} = \frac{\delta \mathcal{E}}{\delta z} \times \frac{\delta z}{\delta W} = \frac{\delta \mathcal{E}}{\delta a} \times \frac{\delta a}{\delta z} \times \frac{\delta z}{\delta W}$

$\frac{\delta z}{\delta a^{-1}}$

Softmax $\frac{\delta \mathcal{E}}{\delta z}$ is just $a - y$.

$y \leftarrow (\alpha, \sim\alpha) \qquad \omega, (\sim\omega) \div [^{-}.5]\ \omega$

x can be our input.

~~$x \rightarrow 1{\times}1 \rightarrow sm \rightarrow a$~~

$x \rightarrow 1{\times}1 \rightarrow z \rightarrow sm \rightarrow a \rightarrow \mathcal{E}$

$W \nearrow$

training data

$\frac{\delta \mathcal{E}}{\delta z} \rightarrow a - (\sim\omega) \div [^{-}.5]\ \omega \quad a\ \omega$ is predicted

many

$\frac{\delta \mathcal{E}}{\delta W_{a,b,c,k}} \rightarrow \sum_i \sum_j \frac{\delta \mathcal{E}}{\delta z^{k}_{i,j}} \times x_{i+a,j+b,c} \rightarrow \frac{\delta \mathcal{E}}{\delta W_{c,k}} = \sum_i \sum_j \frac{\delta \mathcal{E}}{\delta z_{i,j,k}} \times x_{i,j}$

$\Delta z \leftarrow a - (\sim y) \div [^{-}.5]\ y$

$\Delta W \leftarrow (\div \Delta z) +.\times \div ⍉ x$

$\frac{\delta \mathcal{E}}{\delta x_{i,j,c}} = \sum_k \sum_a \sum_b \frac{\delta \mathcal{E}}{\delta z_{i-a,j-b,k}} \times W_{a,b,c} = \sum_k \frac{\delta \mathcal{E}}{\delta z_{i,j,k}} \times W_{c,k}$

$\Delta x \leftarrow (⍉ W) +.\times \Delta z$

$\Delta C_1 \leftarrow \{\Delta z \ldots a\ a\ y \ldots$ (nature nik...)

~~$a\ y\ \omega\ x \leftarrow \omega \quad \omega\ x \leftarrow \omega$~~

$(\omega\ x)(a\ y) \leftarrow \alpha\ \omega$

$\Delta z \leftarrow a - (\sim y) \div [^{-}.5]\ y \diamond \Delta W \leftarrow (\div \Delta z) +.\times \div ⍉ x \diamond \Delta x \leftarrow (⍉ W) +. \ldots$

$\Delta W\ \Delta x\}$

Backpropagation, con't

4/3/2020

Conv 1×1, softmax summary:

$\Delta C_1 \leftarrow \{(w\,x)(a\,y) \leftarrow \alpha w \diamond \Delta z \leftarrow a-(\sim y), \ldots$

$(\Delta W \leftarrow (\ldots \Delta z) +.\times \ldots \alpha \hat{x})(\Delta b \leftarrow (\alpha \ldots w) +.\times \Delta z)\}$

Conv 3×3, Relu: $(\rho W) \equiv 64\ 64\ n\ m$

$a=3,\ b=3,\ k=64\quad c=64$

$x \rightarrow 3\times 3 \rightarrow z \rightarrow \text{Relu} \rightarrow a$

$W \nearrow$

$\frac{\delta E}{\delta x}$ is given. $\frac{\delta E}{\delta z} = z>0\ ;\ \Delta z \leftarrow y>0$

$$\frac{\delta E}{\delta W_{a,b,c,k}} = \sum_{i,j} \frac{\delta E}{\delta z_{i,j,k}} \times x_{i+a,\,j+b,\,c}$$

$\Delta W \leftarrow \{k\,c\,a\,b \leftarrow w \diamond (\ldots k \llbracket \alpha\alpha) +.\times, (c \llbracket w)[a\ b\ \ldots \alpha]\}$

$(\rho \Delta z)\{\Delta z\{k\,c\,a\,b \leftarrow w \diamond (\ldots, k \llbracket \alpha\alpha) +.\times, (c \llbracket w w)[(c\,a\,b)+\ldots\}$

$\ldots x) \ldots \rho w\}$

$\Delta W \leftarrow \{\ldots \Delta z \leftarrow w \diamond x \leftarrow \alpha$

$(\rho w)\{w\{k\,c\,a\,b \leftarrow w \diamond (\ldots, k \llbracket \Delta z) +.\times, (c \llbracket x)[(c\,a\,b)+2\alpha]\}$

$\ldots \rho\, w\}$

$$\frac{\delta J}{x_{i,j,c}} = \sum_k \sum_a \sum_b \frac{\ldots \delta J}{\delta z_{i-a,\,j-b,\,k}} \times W_{a,b,c,k}$$

$k \leftarrow \ldots \phi \ominus W \quad 0,\ddot{-}0,\ddot{-}\ddot{-}0,0;\ 0 \quad 0,\ddot{-}\ddot{-}0,\ddot{-}0,\ddot{-}0,$

$((\rho \ldots \Delta z)+2) \uparrow \Delta z \quad -1\ \phi\ -1\ \ominus \quad \not\times$

$\{+/, k \times \omega\} \ldots 3\,3 \vdash \omega$

$\ldots \{\alpha \ldots \{\{+/, k \times \omega\} \ldots 3\,3 \vdash \omega\}\ddot{\circ} 2 \vdash \omega\}\ddot{\circ} 3 \vdash \omega$

$\ldots (\{+\vdash\{\{+/, k \times \omega\} \ldots 3\,3 \vdash \omega \vdash k \leftarrow \alpha\}\ddot{\circ} 2)\ddot{\circ} 3 \vdash \omega$

$(1\,0\,2\,3 \ldots \alpha \ldots)(\{+\vdash\{\{+/, k \times \omega\} \ldots 3\,3 \vdash \omega \vdash k \leftarrow \phi \ominus \alpha\}\ddot{\circ} 2)\ddot{\circ} 3 \vdash \omega$

$0,\ddot{-}\ddot{-}0,\ddot{-}0,\ddot{-}0,$

Backpropagate, con't

4/3/2020

Conv 3x3, Relu Summary: Δw Δx ← ΔCV Δz w x

```
ΔCV←{Δa w ax←⍵ ⋄ Δz←a>0 ⋄ ~~ÿ←~~
    ~~Δw←(2⍴Δz){~~
    Δw←(2⍴Δz){k c a b←⍵ ⋄ (,k⌷Δz)+.×,(c⌷x)[α+ca b]}¨2⍴w
    ~~Δz~~← pad←0⍪⍨0⍪0,⍨0,Δz
    Δx←(1 0 2 3⍉w)(+/{(+/,k×⍵}⌺3 3⊢⍵⊣k←⌽⊖⍵}⍤2⊢⍵)⍤3⊢pad
    Δw Δx}
```

Up-conv 2x2: Δw Δx ← Δup Δz w x

x' → up → x → 2x2 → z
w ↗

Δw must be same as for a 3x3
Δx seems the same as well.
Main limitation is making the code agnostic to kernel size.
We can ~~t~~ probably use ΔCV for all 2d convolutions if we also split off the Relu function.

I cannot find any off the shelf information on handling the upsampling function directly. There are no weights to worry about. Need to ensure gradient distributes to the appropriate cells. An average pooling should evenly spread concentrate the gradient from a 4x4 to 1 pixel.

Copy ~~from~~ and crop is a simple distribution through, more like a mask since some pixels are unaffected. It may not even need to be propagated along this channel.

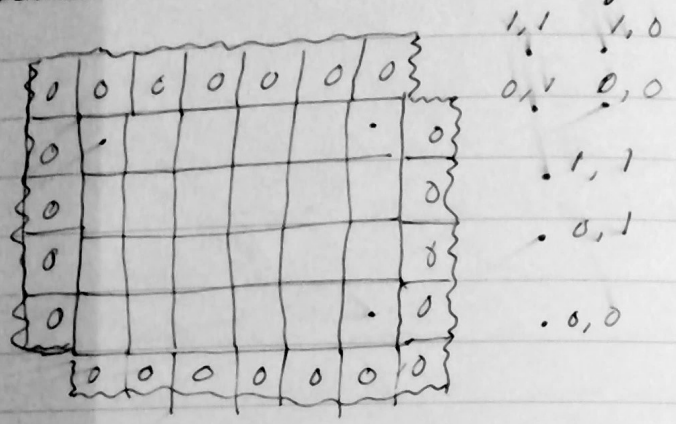
Backpropagation, cont

4/3/2023

Unpooling is a similarly simple computation that has no weights.

Distribute the gradient back along a mask indicating the max element.

Those are all the layers.



The diagram details the 2×2 transpose convolution