COMP 448/548: Medical Image Analysis

Convolutional neural networks

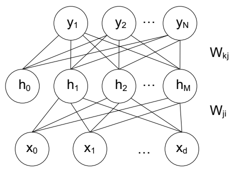
Çiğdem Gündüz Demir
cgunduz@ku.edu.tr

1

Multilayer perceptrons

(revisited)

- Contain hidden layers



Hidden units h_j's can be viewed as new "features" obtained by combining x_i's

A deeper architecture with nonlinear activations is more expressive than a shallow one

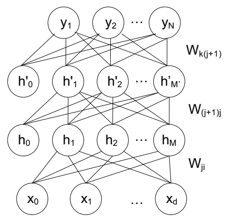
<u>In this network</u>

1. Each hidden unit computes its net activation
   $net_j^t = \sum_i x_i^t W_{ji}$
2. Each hidden unit emits an output that is a nonlinear function (e.g., sigmoid, ReLU) of its activation
   $h_j^t = non\!-\!linear\!-\!function(net_j^t)$
3. Each output unit computes its net activation
   $net_k^t = \sum_j h_j^t W_{kj}$
4. Each output units emits an output (using a linear, a sigmoid or a softmax function)
   $y_k^t = output\!-\!function(net_k^t)$

2

More hidden layers

(revisited)



$$\frac{\partial loss^t(W)}{\partial W_{ji}} = \frac{\partial loss^t(W)}{\partial net_j^t} \cdot \frac{\partial net_j^t}{\partial W_{ji}}$$

$$\frac{\partial loss^t(W)}{\partial W_{ji}} = \delta_j^t \cdot x_i^t$$

$$\delta_j^t = \sum_{(j+1)} \frac{\partial loss^t(W)}{\partial net_{(j+1)}^t} \cdot \frac{\partial net_{(j+1)}^t}{\partial net_j^t}$$

$$\delta_j^t = \left[\sum_{(j+1)} \delta_{(j+1)}^t \cdot W_{(j+1)j}\right] \cdot \sigma'(net_j)$$

δ_j may vanish after repeated multiplication. This makes deep architectures hard to train (when initial weights are not "good" enough)

<u>Approaches for alleviating underfitting and overfitting problems</u>

- Better network designs: Sparse connections, weight sharing, convolutional nets, long/short skip connections, activation functions, …
- Better network training: Regularization, loss function definitions, larger datasets, data augmentation, …
- Previously, layerwise pretraining (restricted Boltzmann machines, autoencoders)

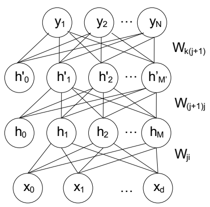
3

Layerwise pretraining

4

Vanishing gradient problem

- This makes hard to train deep architectures (with many hidden layers) by backpropagation
- When the initial weights are good enough, backpropagation works well
- Layerwise pretraining
  - Restricted Boltzmann machines
  - Autoencoders



$$\frac{\partial loss^t(W)}{\partial W_{ji}} = \frac{\partial loss^t(W)}{\partial net_j^t} \cdot \frac{\partial net_j^t}{\partial W_{ji}}$$

$$\frac{\partial loss^t(W)}{\partial W_{ji}} = \delta_j^t \cdot x_i^t$$

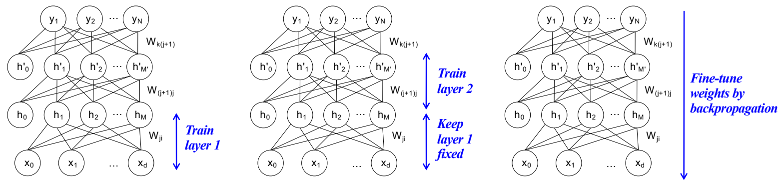
$$\delta_j^t = \sum_{(j+1)} \frac{\partial loss^t(W)}{\partial net_{(j+1)}^t} \cdot \frac{\partial net_{(j+1)}^t}{\partial net_j^t}$$

$$\delta_j^t = \left[\sum_{(j+1)} \delta_{(j+1)}^t \cdot W_{(j+1)j}\right] \cdot \sigma'(net_j)$$

5

Layerwise pretraining

- First, train one layer at a time, optimizing P(x)
- Then, fine-tune weights, optimizing P(y|x) by backpropagation



6

Restricted Boltzmann machines (RBMs)

- RBM is a simple energy-based model

$$p(x,h) = \frac{1}{Z_\theta}\exp(-E_\theta(x,h))$$

$$E_\theta(x,h) = -x^T\ W\ h - b^T\ x - d^T\ h$$

It only allows h-x interactions

$$Z_\theta = \sum_{(x,h)} \exp(-E_\theta(x,h)) \leftarrow \textit{normalizer}$$

- Train an RBM optimizing P(x)

Maximize the log-likelihood of data

$$\underbrace{\partial_{W_{ji}} \log P_W(x = x^t)}_{\text{Derivative of the log-likelihood}} = \partial_{W_{ji}} \log \sum_h P_W(x = x^t,\ h)$$

$$= -\partial_{W_{ji}} \log Z_w + \partial_{W_{ji}} \log \sum_h \exp(-E_W\ (x^t, h))$$

$$= \underbrace{-\mathrm{E}_{p(x,\,h)}\left[\,x_i\ \ h_j\,\right]}_{\text{Negative phase comes from the model}} + \underbrace{\mathrm{E}_{p(h\,|x = x^t)}\left[\,x_i^t\ \ h_j\,\right]}_{\text{Positive phase comes from the data}}$$

The negative phase term is expensive to calculate since it requires sampling (x, h) from the model. **Contrastive divergence** is a faster solution.

7

Autoencoders

- They learn to “compress” and “reconstruct” the input data

Encoder :	$h\ =\ \sigma(\ W\ x\ +\ b\ )$
Decoder :	$x'\ =\ \sigma(\ W'\ h\ +\ d\ )$

- Learn the weights to minimize the reconstruction loss

$$\text{loss} = \sum_t (x^t - x'^t)^2$$

- This is the same backpropagation for a network with one hidden layer, where x^t is both input and output
- They can be stacked to form a deep neural network
  - Cheaper alternatives to RBMs

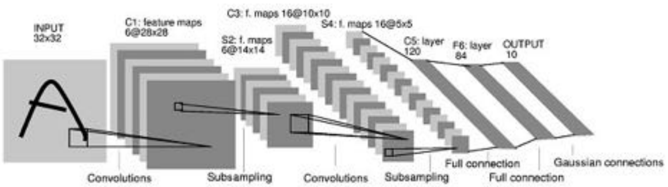
8

Basics of convolutional neural networks

9

Convolutional neural networks (CNNs)

- A CNN consists of a number of convolutional and pooling (subsampling) layers optionally followed by fully connected layers

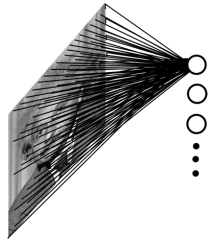


LeNet-5 by LeCun et al., 1998

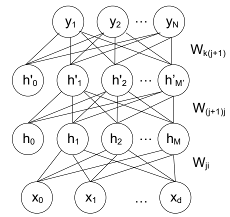
10

Fully connected layers

- When the input data is an image, a fully connected layer will produce a huge number of weights (parameters) to be learned



Example:
200x200 image
25K hidden units
→ ~1B parameters

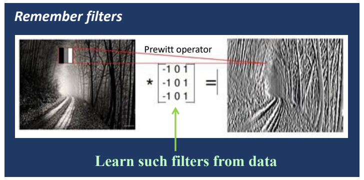


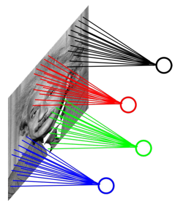
Slide credit: M.A. Ranzalo

11

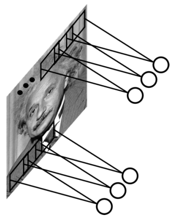
Convolutional layer

- However, spatial correlation is local and statistics is similar at different locations
- Thus, small kernels are defined and their parameters are shared by all pixels
- **It is convolution with learned kernels**





Example:
200x200 image
25K hidden units
10x10 kernels
→ ~2.5M parameters
(instead of 1B parameters)

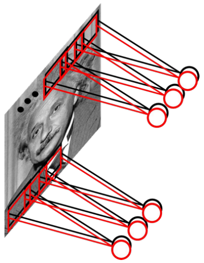


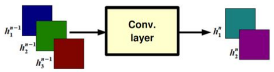
Sparse connections and weight sharing

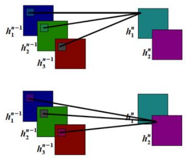
Slide credit: M.A. Ranzalo

12

Convolutional layer







$$net_j^n = \underbrace{W_{j0}}_{\text{Bias term}} + \sum_{k=1}^{K} \underbrace{h_k^{n-1}}_{\text{Input feature map}} * \underbrace{W_{jk}}_{\text{Kernel}}$$

Bias term

Input feature map
k^{th} feature map of the previous (n-1)th layer

Kernel for the k^{th} feature map of the previous layer to the j^{th} feature map of the current layer

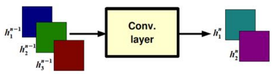
$$\underbrace{h_j^n}_{\text{Output feature map}} = non\!-\!linear\!-\!function(net_j^n)$$

Output feature map
j^{th} feature map calculated for the current n^{th} layer

Slide credit: M.A. Ranzalo

13

Convolutional layer

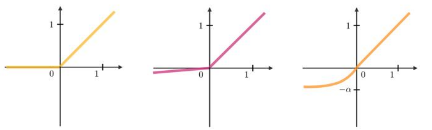


$$net_j^n = W_{j0} + \sum_{k=1}^{K} h_k^{n-1} * W_{jk}$$

$$h_j^n = non\!-\!linear\!-\!function(net_j^n)$$

Choosing the architecture (the number of feature maps, size of kernels, and number of convolutional layers) is task dependent

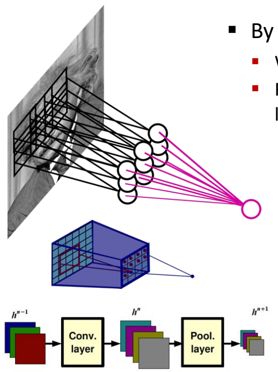
- The rectified linear unit (ReLU) is a commonly used activation function that provides nonlinearity
  - Fast to compute
  - Reduces the likelihood of the gradient to vanish
  - Better sparsity



$f(z) = \max(0, z)$ | $f(z) = \max(\epsilon z, z)$, $\epsilon \ll 1$ | $f(z) = \max(\alpha(\exp(z) - 1), z)$, $\alpha \ll 1$

14

Pooling layer



- By pooling the filter responses at different locations
  - We gain robustness to the exact location of features
  - Receptive field becomes larger for the next layer (the next layer will look at a larger spatial region)

Max-pooling:

$$h_j^n(x, y) = \max_{\bar{x} \in N(x), \bar{y} \in N(y)} h_j^{n-1}(\bar{x}, \bar{y})$$

Average-pooling:

$$h_j^n(x, y) = 1/K \sum_{\bar{x} \in N(x), \bar{y} \in N(y)} h_j^{n-1}(\bar{x}, \bar{y})$$

L2-pooling:

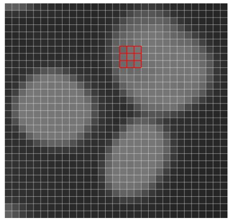
$$h_j^n(x, y) = \sqrt{\sum_{\bar{x} \in N(x), \bar{y} \in N(y)} h_j^{n-1}(\bar{x}, \bar{y})^2}$$

Slide credit: M.A. Ranzalo

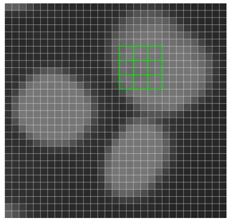
15

Receptive field

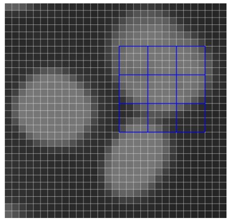
- It is the region in the input space that a convolution can "*see*"



What a 3x3 convolutional filter sees



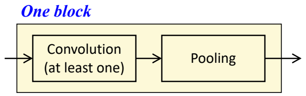
What the 3x3 convolutional filter sees after 2x2 pooling

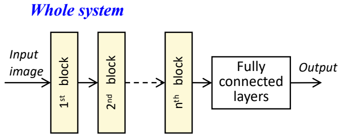


What the 3x3 convolutional filter sees after applying 2x2 pooling twice

16

Typical design





All layers are differentiable so that standard backpropagation can be used

After one block

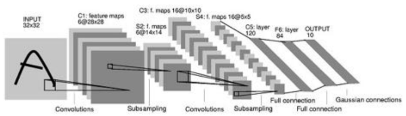
- Number of feature maps is usually increased (conv. layer)
- Spatial resolution is usually decreased (pooling layer and/or stride in conv. layer)
- Receptive field gets larger

After several blocks

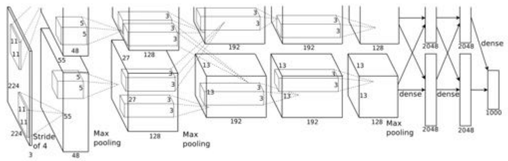
- Spatial resolution is greatly reduced and number of feature maps is large so convolution would not make any sense
- Next layer(s) will consist of fully connected layers (with or without hidden layers)
- Sigmoid and softmax layer is used at the end for binary and multi-class classification, respectively

17

Example CNN architectures



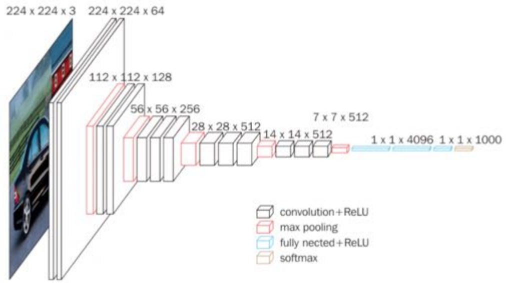
LeNet-5 by LeCun et al., 1998



AlexNet by Krizhevsky et al., 2012. ImageNet classification with deep convolutional neural networks.
https://www.cs.toronto.edu/~kriz/imagenet_classification_with_deep_convolutional.pdf

18

Example CNN architectures



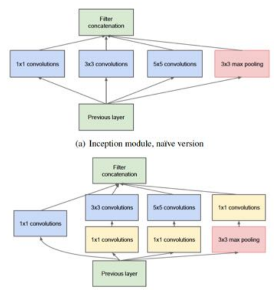
VGG16 by Simonyan and Zisserman, 2015. Very deep convolutional networks for large-scale image recognition.
https://arxiv.org/pdf/1409.1556.pdf

19

Example CNN architectures

type	patch size/ stride	output size	depth	#1×1	#3×3 reduce	#3×3	#5×5 reduce	#5×5	pool proj	params	ops
convolution	7×7/2	112×112×64	1							2.7K	34M
max pool	3×3/2	56×56×64	0								
convolution	3×3/1	56×56×192	2		64	192				112K	360M
max pool	3×3/2	28×28×192	0								
inception (3a)		28×28×256	2	64	96	128	16	32	32	159K	128M
inception (3b)		28×28×480	2	128	128	192	32	96	64	380K	304M
max pool	3×3/2	14×14×480	0								
inception (4a)		14×14×512	2	192	96	208	16	48	64	364K	73M
inception (4b)		14×14×512	2	160	112	224	24	64	64	437K	88M
inception (4c)		14×14×512	2	128	128	256	24	64	64	463K	100M
inception (4d)		14×14×528	2	112	144	288	32	64	64	580K	119M
inception (4e)		14×14×832	2	256	160	320	32	128	128	840K	170M
max pool	3×3/2	7×7×832	0								
inception (5a)		7×7×832	2	256	160	320	32	128	128	1072K	54M
inception (5b)		7×7×1024	2	384	192	384	48	128	128	1388K	71M
avg pool	7×7/1	1×1×1024	0								
dropout (40%)		1×1×1024	0								
linear		1×1×1000	1							1000K	1M
softmax		1×1×1000	0								

Table 1: GoogLeNet incarnation of the Inception architecture



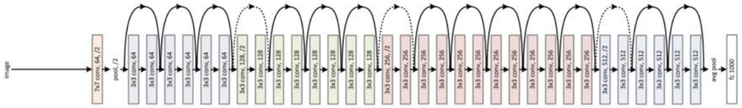
(b) Inception module with dimensionality reduction

Figure 2: Inception module

GoogLeNet by Szegedy et al., 2014. Going deeper with convolutions.
https://arxiv.org/abs/1409.4842

20

Example CNN architectures



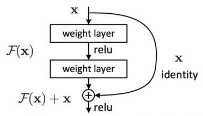


Figure 2. Residual learning: a building block.

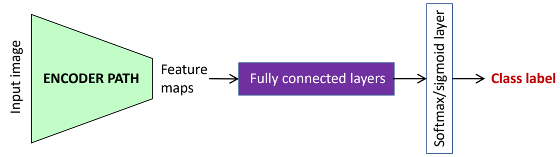
ResNet by He et al., 2015. Deep residual learning for image recognition.
https://arxiv.org/pdf/1512.03385.pdf

CNNs for medical image classification

Please check the following survey paper for more about deep learning in medical images and more references.
Litjens et al., A survey on deep learning in medical image analysis, Medical Image Analysis, 2017.
https://www.sciencedirect.com/science/article/pii/S1361841517301135

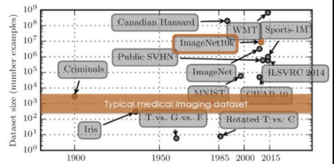
CNNs for image classification

- A CNN compresses an image into a set of feature maps to capture semantic/contextual information from the image
- This compression corresponds to downsampling the image using convolution and pooling layers
- Then it puts fully connected layers on the top of the feature maps to predict a class for the entire image



Transfer learning

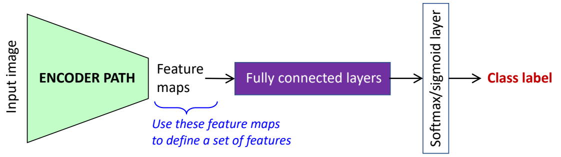
- Dataset sizes are typically small
- Thus, it is popular to use transfer learning, which employs networks (and thus, their learned weights) previously trained on large datasets



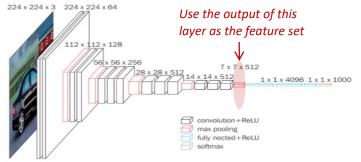
www.quantib.com/blog/deep-learning-radiology-and-challenges-radiology-ai

- Two main approaches
  1. Use a pretrained network as a feature extractor
  2. Finetune a pretrained network on the medical data

Use a pretrained network as a feature extractor

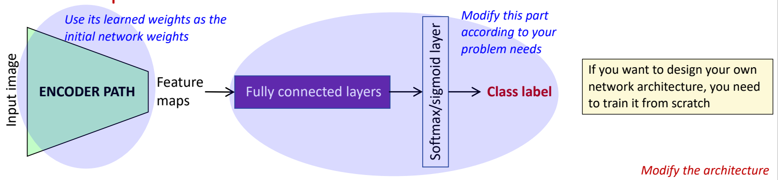


1. Feed your image to the pretrained network (may need to resize the image)
2. Calculate feature maps using the learned weights
3. Use the feature maps at the last layer of the encoder to define a feature set for your image (also possible to use feature maps of the other layers)
4. Use a classifier on this feature set

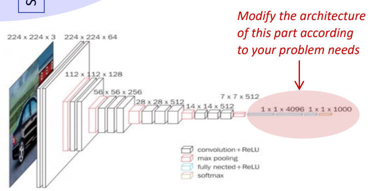


25

Finetune a pretrained network on the medical data

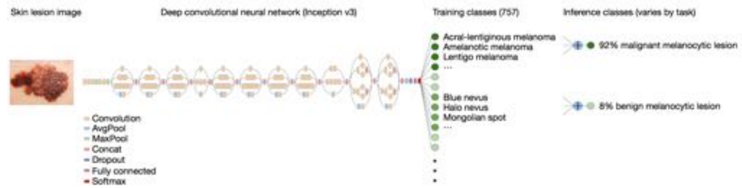


1. Feed your image to the pretrained network (may need to resize the image)
2. Modify the network architecture after the encoder path. Changing the last softmax/sigmoid layer according to your classification problem is a must. May also need to change the fully connected layers.
3. Use the learned weights (of the encoder) as the initial network's weights
4. Finetune the weights by backpropagation on your own medical data



26

Example: CNN for skin cancer classification

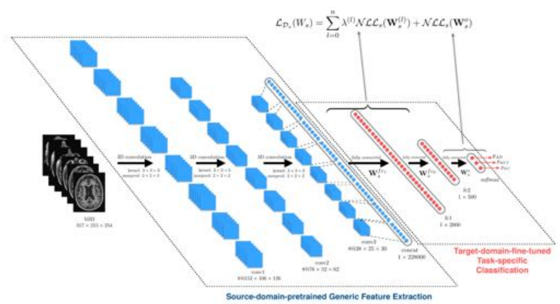


The authors used the Google Inception v3 CNN architecture pretrained on the ImageNet dataset (1.28 million images over 1,000 generic object classes) and finetuned on their own dataset of 129,450 skin lesions. They resized each image to 299x299 pixels to make it compatible with the original dimensions of the Inception v3 network architecture. They defined 757 training classes, for which the probabilities would be accumulated to infer the final inference class.

Esteva et al, 2017. Dermatologist-level classification of skin cancer with deep neural networks. Nature 542, 115–118. https://www.nature.com/articles/nature21056

27

Example: 3D CNN for Alzheimer's disease diagnostics



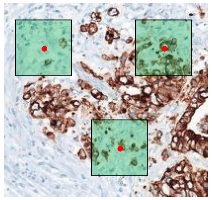
Hosseini-Asl et al., 2016. Alzheimer's disease diagnostics by a deeply supervised adaptable 3D convolutional network. https://arxiv.org/abs/1607.00556

28

CNNs for object detection and segmentation

Training:

- Small patches are cropped around individual pixels
- Each patch is labeled with the class of the pixel, around which it is cropped
- CNN is trained on these small patches



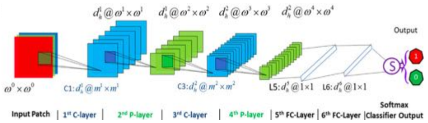
Detection/segmentation:

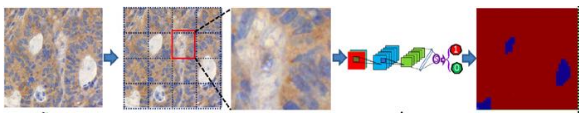
- For an entire (large) image, patches are obtained using a sliding window approach
- These patches are classified by the trained CNN
- Outputs (i.e., posteriors) generated by this CNN are commonly postprocessed

29

Example: CNN for histopathological image segmentation

- *Slide a window* over an image to obtain patches
- Using the trained CNN, classify each patch with either the epithelial or the stromal class





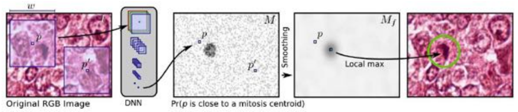
Xu et al., 2016. A deep convolutional neural network for segmenting and classifying epithelial and stromal regions in histopathological images. Neurocomputing (191), 214-223.
https://www.sciencedirect.com/science/article/pii/S0925231216001004

30

Example: CNN for mitosis detection

- *Slide a window* over an image to obtain patches
- Using the trained CNN, obtain the probability of a pixel belonging to a mitotic cell
- Find the local maxima on the smoothed probability map

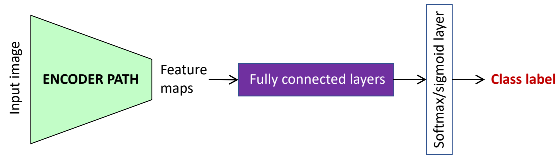
> 1. *Classifying each pixel in a sliding window fashion, used by earlier studies, is expensive as it requires lots of redundant calculations*
> 2. *Trade-off between localization accuracy and the use of context*
>   - *Larger patches require more max-pooling layers that reduce the localization accuracy*
>   - *Small patches results in seeing only little context*
>
> *Dense prediction networks, used by recent studies, have greatly improved efficiency and accuracy.*



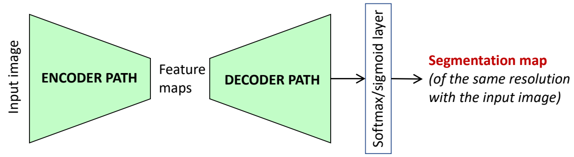
Ciresan et al., 2013. Mitosis detection in breast cancer histology images with deep neural networks. MICCAI.
https://link.springer.com/chapter/10.1007/978-3-642-40763-5_51

31

Convolutional neural networks (CNNs) for image classification



Dense prediction networks for semantic image segmentation



32

Thank you!

Next time:

Dense prediction networks

33