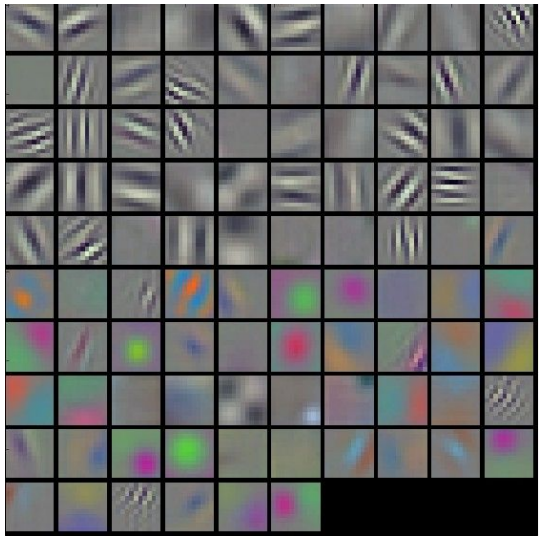
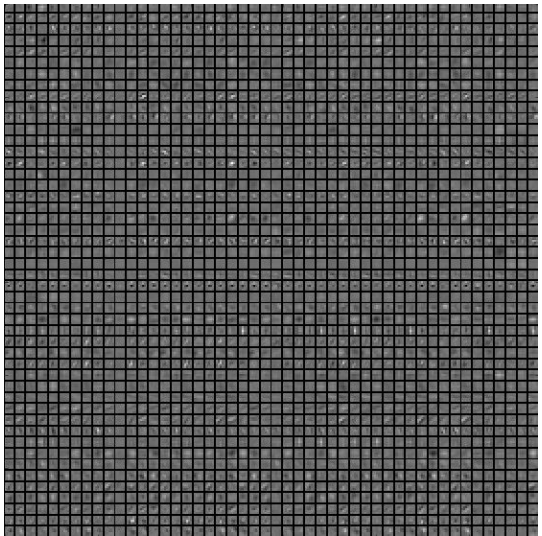
Representation learning

Representations in deep nets

- The by far most common setting in DL is supervised
- In this case, there is usually no restriction on the “representation” learned inside the net
- However, good representations often still emerge





http://cs231n.github.io/understanding-cnn

Representation in deep nets

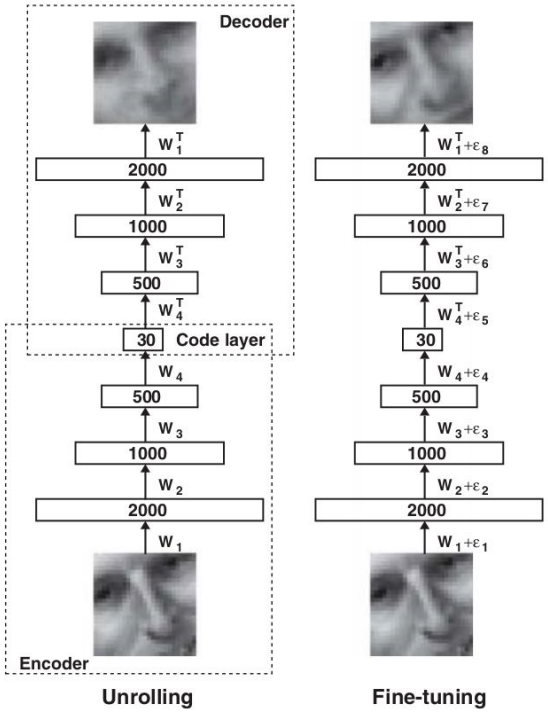
- One reason for the success of deep nets is their flexibility
- You have great freedom in the design process
  - Topology
  - Activation functions
  - Loss function(s)
  - Etc.
- With modern interfaces, like pytorch and Keras, it's child's play to add your own loss function to a model:

  Keras: https://www.tensorflow.org/api_docs/python/tf/keras/layers/Layer#add_loss

  Pytorch: take the relevant part of a layer (weights/outputs) and compute whatever loss on it you like

Unsupervised pretraining

- Remember from lecture 8 on AEs
- The 2006 paper by Hinton & Salakhutdinov that helped start the deep learning wave
- This algorithm was considered unsupervised because the classification labels were not used during (pre)training



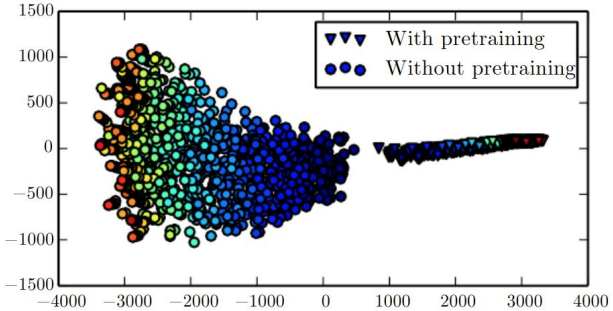
Unsupervised pretraining in general

- The objective is to learn some function *f* that generates good representations of our data
- We can train each layer of our function using a "learner" (AE with SGD)
- At each stage, the current input is the representation output of the previously learned stage
- Notice that only during (optional) fine-tuning, the **Y** is in use

$f \leftarrow$ Identity function
$\tilde{\boldsymbol{X}} = \boldsymbol{X}$
for $k = 1, \ldots, m$ **do**
  $f^{(k)} = \mathcal{L}(\tilde{\boldsymbol{X}})$
  $f \leftarrow f^{(k)} \circ f$
  $\tilde{\boldsymbol{X}} \leftarrow f^{(k)}(\tilde{\boldsymbol{X}})$
end for
if *fine-tuning* ***then***
  $f \leftarrow \mathcal{T}(f, \boldsymbol{X}, \boldsymbol{Y})$
end if
Return f

Some advantages of pretraining

- There are indications that this greatly helps initialization:



- You can train the final part (e.g. classifier) very quickly and with few parameters
- You potentially even replace the final part and *transfer* to other tasks

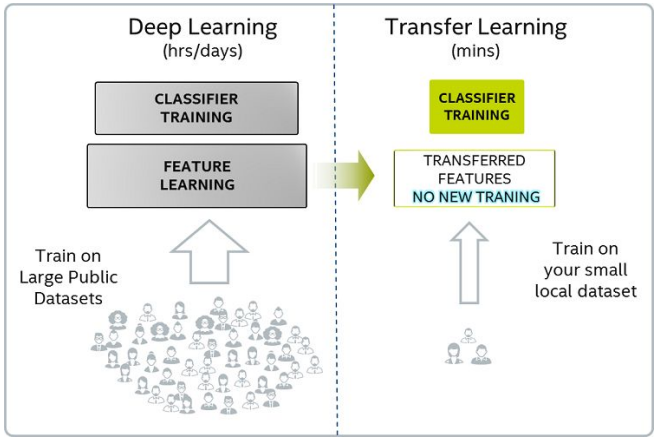
The rise of supervised learning

- Remember that, especially for image classification, supervised learning is still the winner
- In many cases, it has also been shown that the representations learned this way are very good for other tasks
- Unsupervised or semi-supervised learning is still a heavily active topic:

  https://arxiv.org/pdf/1905.09272 (CPC)
  https://arxiv.org/pdf/1911.05722 (MoCo)
  https://arxiv.org/pdf/2002.05709 (SimCLR)
  https://arxiv.org/pdf/2006.07733 (BYOL)
  https://arxiv.org/abs/2011.10566 (SimSiam)

Transfer learning

- Transfer learning can be seen as transferring a learned model to a new task, i.e. a task with new outputs
- While the output changes, in transfer learning you often rely on similar inputs between the two datasets



https://medium.com/free-code-camp/asl-recognition-using-transfer-learning-918ba054c004

Transfer learning

- A simple example:
  - Train a CNN on MNIST to classify handwritten numbers
  - Extend or adapt the model to classify handwritten letters (A to Z)
  - It is conceivable that the first part of the net is still a good feature extractor
  - Thus, you can “freeze” this part and just train a new 26-way classifier

Domain adaptation

- The hypothesis in transfer learning was that early processing stages could be transferred to new data, because the input semantics are shared
  - Only the “head” of the network needed replacement to transfer to a new target
- In domain adaptation, this is reversed
- The input semantics are now not shared any longer (sort of)
  - Book example: an LSTM-based text classifier is trained on written online consumer reviews of books, but now we want to adapt it to parse reviews of TVs
  - Another example: adaptation of an ImageNet backbone (which was trained to classify rich, full images) to a more restricted input, e.g. CIFAR

Transfer/adaptation

- An example: https://colab.research.google.com/drive/1-64gy13pQohB8j9MgL7AqUgH1ARwizS0?usp=sharing

R-CNN

- In 2014, Girshick et al. delivered a result that - similar to what AlexNet did for classification - blew away the competition on object detection
- The algorithm was conceptually simple
- Many, many developments and improvements have been done since

Rich feature hierarchies for accurate object detection and semantic segmentation

Ross Girshick[1] Jeff Donahue[1,2] Trevor Darrell[1,2] Jitendra Malik[1]
[1]UC Berkeley and [2]ICSI
{rbg,jdonahue,trevor,malik}@eecs.berkeley.edu

Abstract

Object detection performance, as measured on the canonical PASCAL VOC dataset, has plateaued in the last few years. The best-performing methods are complex ensemble systems that typically combine multiple low-level image features with high-level context. In this paper, we propose a simple and scalable detection algorithm that improves mean average precision (mAP) by more than 30% relative to the previous best result on VOC 2012—achieving a mAP of 53.3%. Our approach combines two key insights:

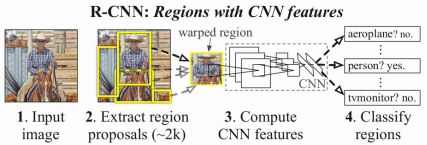
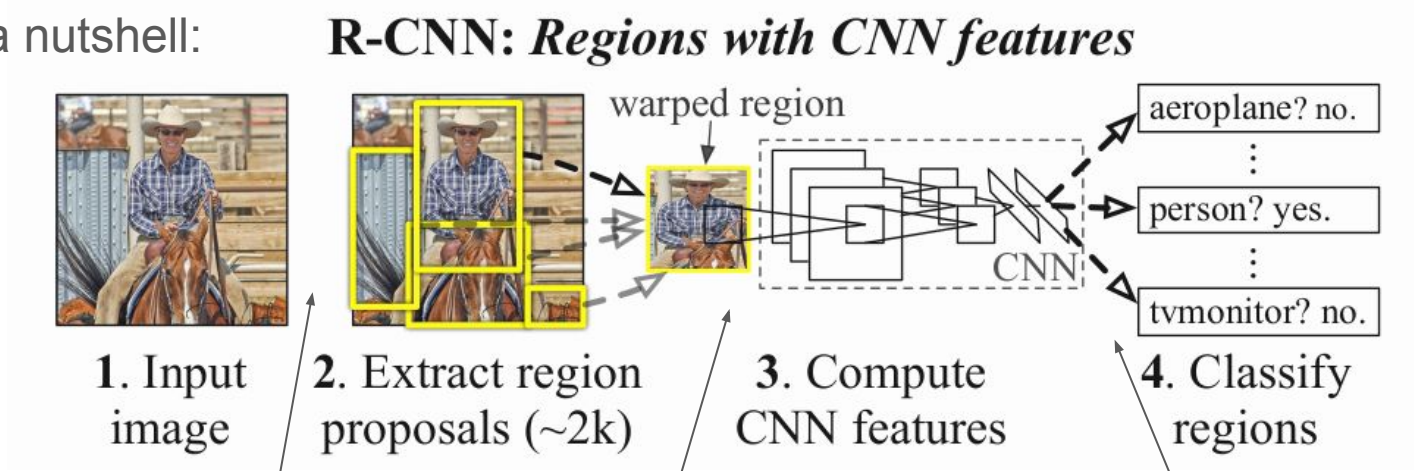


Figure 1: Object detection system overview. Our system (1) takes an input image, (2) extracts around 2000 bottom-up region proposals, (3) computes features for each proposal using a large convolutional neural network (CNN), and then (4) classifies each

R-CNN

- In a nutshell:



External (slow) algorithm

Domain adaptation!

Transfer learning!

R-CNN

- Performance:

VOC 2007 test	aero	bike	bird	boat	bottle	bus	car	cat	chair	cow	table	dog	horse	mbike	person	plant	sheep	sofa	train	tv	mAP
R-CNN $pool_5$	51.8	60.2	36.4	27.8	23.2	52.8	60.6	49.2	18.3	47.8	44.3	40.8	56.6	58.7	42.4	23.4	46.1	36.7	51.3	55.7	44.2
R-CNN fc_6	59.3	61.8	43.1	34.0	25.1	53.1	60.6	52.8	21.7	47.8	42.7	47.8	52.5	58.5	44.6	25.6	48.3	34.0	53.1	58.0	46.2
R-CNN fc_7	57.6	57.9	38.5	31.8	23.7	51.2	58.9	51.4	20.0	50.5	40.9	46.0	51.6	55.9	43.3	23.3	48.1	35.3	51.0	57.4	44.7
R-CNN FT $pool_5$	58.2	63.3	37.9	27.6	26.1	54.1	66.9	51.4	26.7	55.5	43.4	43.1	57.7	59.0	45.8	28.1	50.8	40.6	53.1	56.4	47.3
R-CNN FT fc_6	63.5	66.0	47.9	37.7	29.9	62.5	70.2	60.2	32.0	57.9	47.0	53.5	60.1	64.2	52.2	31.3	55.0	50.0	57.7	63.0	53.1
R-CNN FT fc_7	64.2	69.7	50.0	41.9	32.0	62.6	71.0	60.7	32.7	58.5	46.5	56.1	60.6	66.8	54.2	31.5	52.8	48.9	57.9	64.7	54.2
R-CNN FT fc_7 BB	**68.1**	**72.8**	**56.8**	**43.0**	**36.8**	**66.3**	**74.2**	**67.6**	**34.4**	**63.5**	**54.5**	**61.2**	**69.1**	**68.6**	**58.7**	**33.4**	**62.9**	**51.1**	**62.5**	**64.8**	**58.5**
DPM v5 [17]	33.2	60.3	10.2	16.1	27.3	54.3	58.2	23.0	20.0	24.1	26.7	12.7	58.1	48.2	43.2	12.0	21.1	36.1	46.0	43.5	33.7
DPM ST [25]	23.8	58.2	10.5	8.5	27.1	50.4	52.0	7.3	19.2	22.8	18.1	8.0	55.9	44.8	32.4	13.3	15.9	22.8	46.2	44.9	29.1
DPM HSC [27]	32.2	58.3	11.5	16.3	30.6	49.9	54.8	23.5	21.5	27.7	34.0	13.7	58.1	51.6	39.9	12.4	23.5	34.4	47.4	45.2	34.3

Table 2: Detection average precision (%) on VOC 2007 test. Rows 1-3 show R-CNN performance without fine-tuning. Rows 4-6 show results for the CNN pre-trained on ILSVRC 2012 and then fine-tuned (FT) on VOC 2007 trainval. Row 7 includes a simple bounding box regression (BB) stage that reduces localization errors (Section 3.4). Rows 8-10 present DPM methods as a strong baseline. The first uses only HOG, while the next two use different feature learning approaches to augment or replace HOG.

R-CNN

- Visualizations:

Figure 3: Top regions for six pool$_5$ units. Receptive fields and activation values are drawn in white. Some units are aligned to concepts, such as people (row 1) or text (4). Other units capture texture and material properties, such as dot arrays (2) and specular reflections (6).

Visualizing representations

- We can’t do a DL course without also at least introducing t-SNE (van der Maaten & Hinton, 2008)
- Think of the purpose of our pretrained backbone during transfer learning
  - We are provided with some high-dimensional features
  - These features are expected to be good at separating the classes of the inputs
  - This makes it easier to train a new classification “head”
- What if we could actually “see” how these features cluster together?

t-SNE

- t-SNE allows you to map a set of high-dimensional data points all the way down to <= 3 dimensions
  - No labels required!
  - One-shot method!
- The method tries to preserve relative neighbor distances while projecting down to 2D
- It’s very easy to use:
  https://scikit-learn.org/stable/modules/generated/sklearn.manifold.TSNE.html
- Now let’s see it on our ResNet features for CIFAR-10:
  https://colab.research.google.com/drive/1-64gy13pQohB8j9MgL7AqUgH1ARwizS0#scrollTo=3tX0XP14wPSS

Few-shot learning

- Finally, equipped with your fantastic high-dimensional embeddings of whatever input you have, you can now do few-shot learning
  - In the extreme case, “few” could be only a single example: one-shot learning
- Imagine you have only *k* (maybe 1-5) examples per class during training
  - Pass the training examples through the pretrained net
  - Store the high-dimensional representations
  - Now go back to lecture 1: the *k*-NN classifier
  - At test time, also pass the new images through the net
  - Classify using your *k*-NN classifier in latent space
- Let’s see how this works:
  https://colab.research.google.com/drive/1-64gy13pQohB8j9MgL7AqUgH1ARwizS0#scrollTo=j53FzOs_w5Tg