Convolutional Neural Networks and Recurrent Neural Networks

Maschinelles Lernen 1 -
Grundverfahren WS20/21

Prof. Gerhard Neumann
KIT, Institut für Anthrophomatik und Robotik

Learning Outcomes

We will learn today…

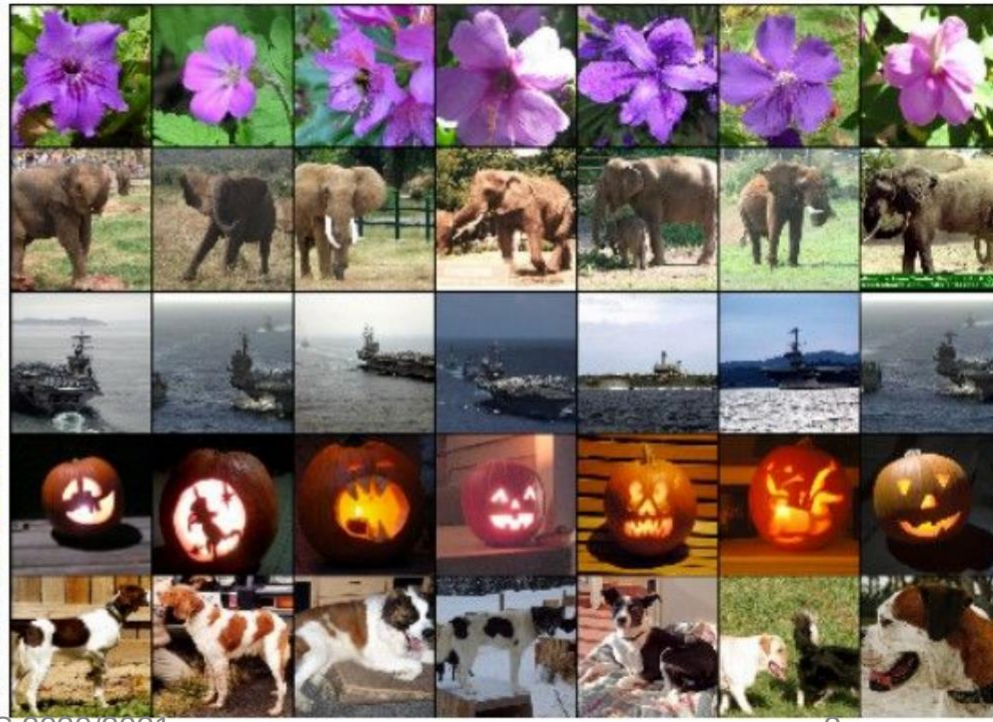
- How to process images with neural networks
- Why do we need convolutions?
- What kind of architectures have been successful?
- How can we use CNNs with a “reasonable” amount of training data?

CNNs are everywhere nowadays…

Slides based on slides from Fei-Fei Li, Justin Johnson and Serena Yeoung, Stanford

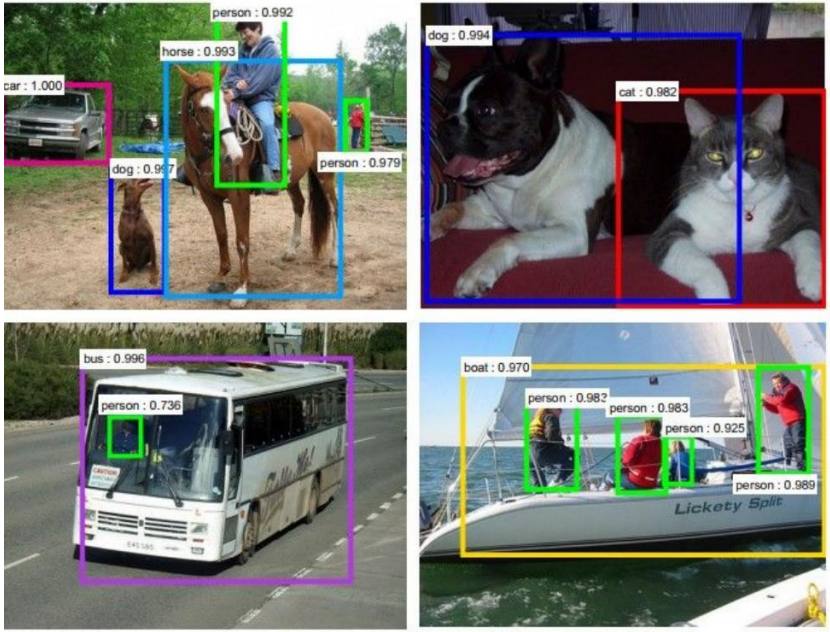
Classification

Retrieval



CNNs are everywhere nowadays…

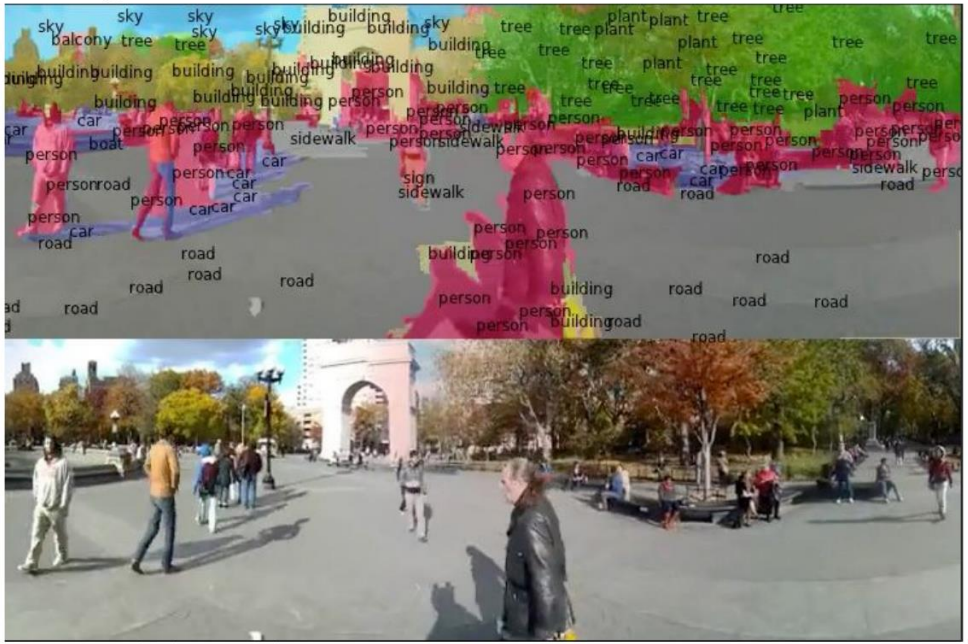
Detection



Figures copyright Shaoqing Ren, Kaiming He, Ross Girschick, Jian Sun, 2015. Reproduced with permission.

[Faster R-CNN: Ren, He, Girshick, Sun 2015]

Segmentation

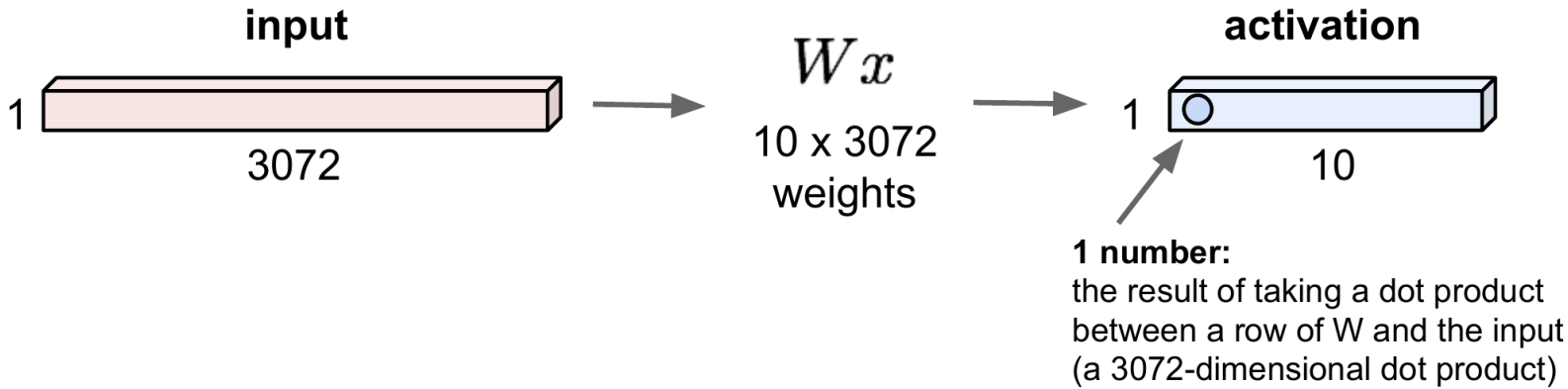


Figures copyright Clement Farabet, 2012. Reproduced with permission.

[Farabet et al., 2012]

Image-based inputs

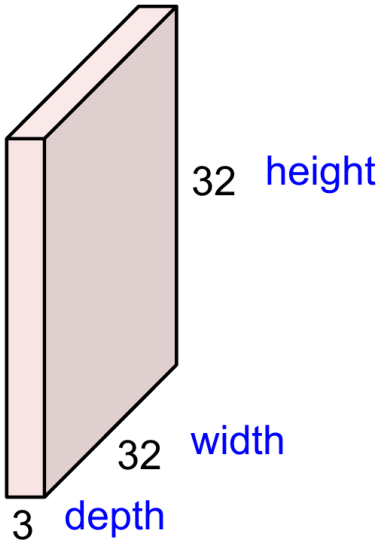
Fully connected layer: 32x32x3 image -> stretch to 3072 x 1



- We need a huge amount of weights using a FC layer
- How can we better exploit the spatial structure of an image?
  - I.e. neighbored pixels are more correlated / contain more similar information than distant pixels

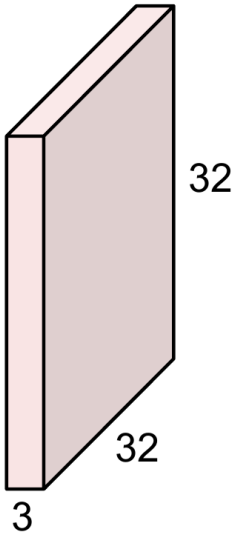
Image-based inputs

Convolutional Layer: 32x32x3 image -> preserve spatial structure

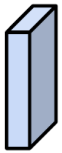


Convolutional Layer

32x32x3 image



5x5x3 filter

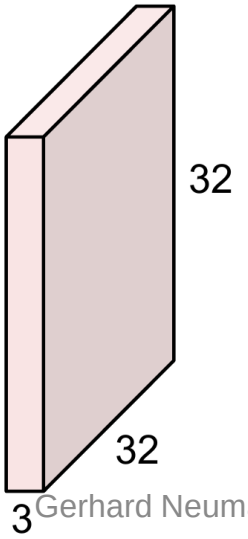


Convolve the filter with the image
i.e. “slide over the image spatially,
computing dot products”

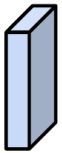
Convolutional Layer

Filters always extend the full depth of the input volume

32x32x3 image

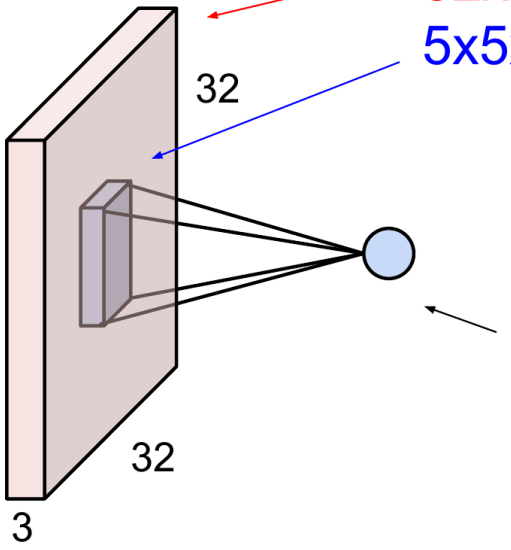


5x5x3 filter



Convolve the filter with the image
i.e. “slide over the image spatially,
computing dot products”

Convolutional Layer



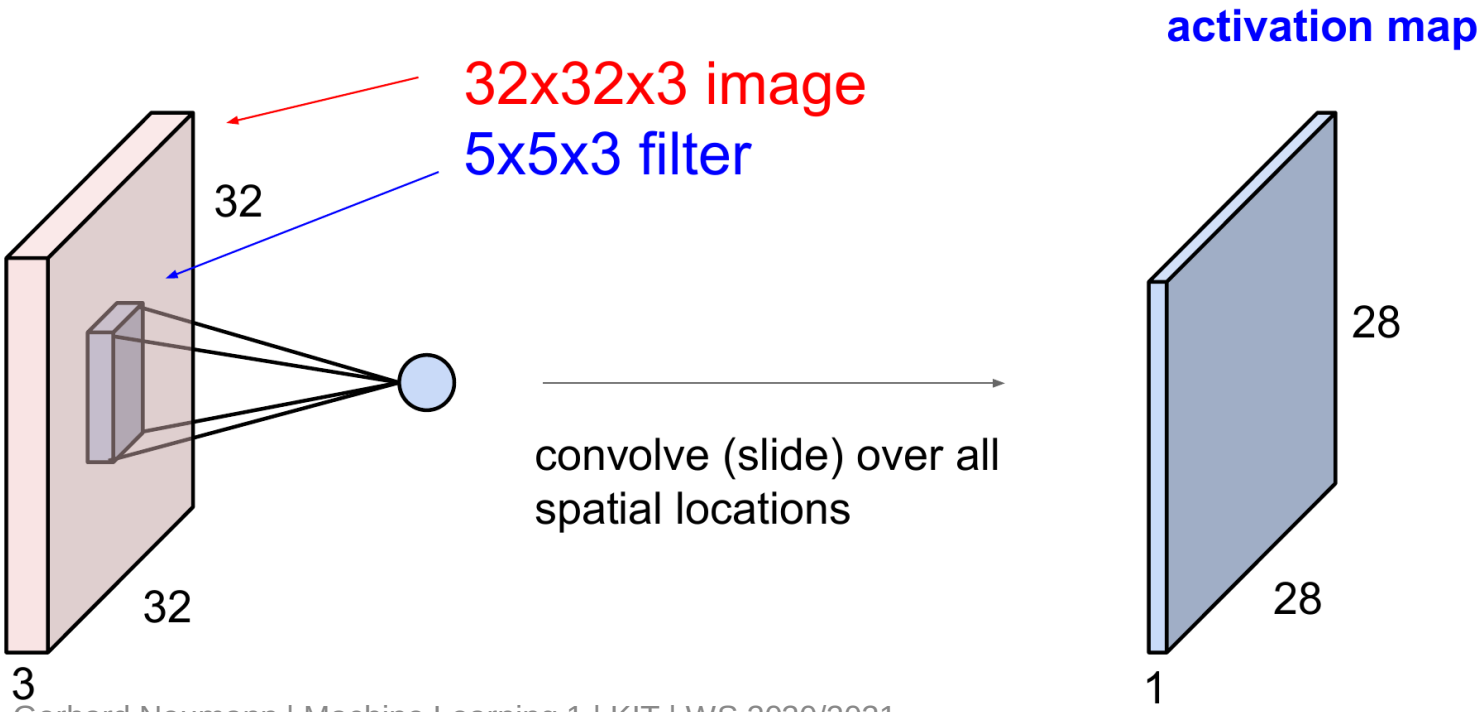
32x32x3 image

5x5x3 filter w

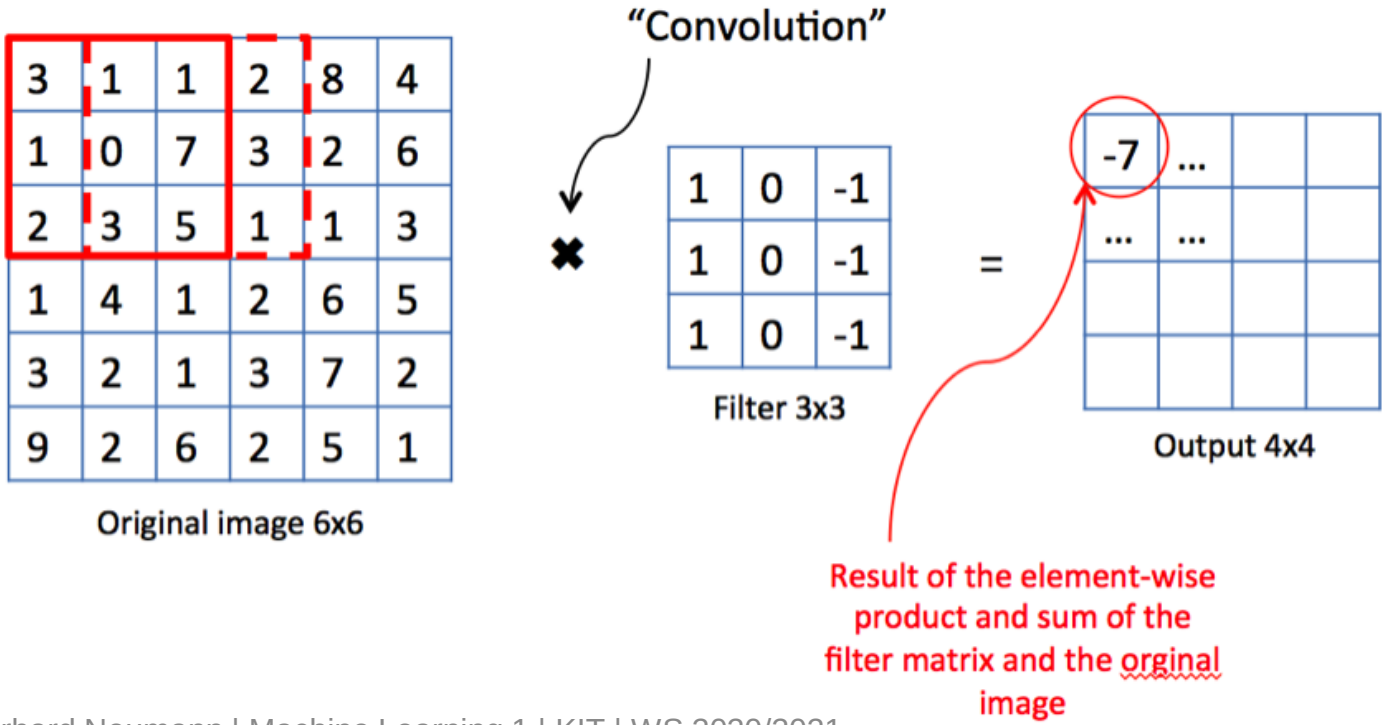
1 number:
the result of taking a dot product between the filter and a small 5x5x3 chunk of the image
(i.e. 5*5*3 = 75-dimensional dot product + bias)

$$w^T x + b$$

Convolutional Layer



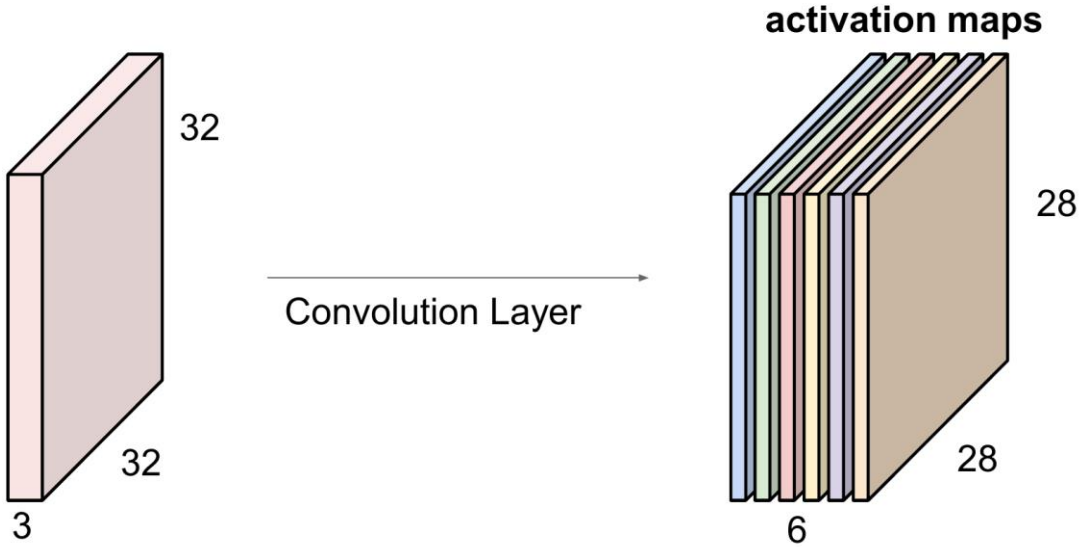
Example: 1 channel



Stacking convolutions

We can stack filters to obtain a multi-channel output “image” (28x28x6)

- For example, if we had 6 5x5 filters, we’ll get 6 separate activation maps

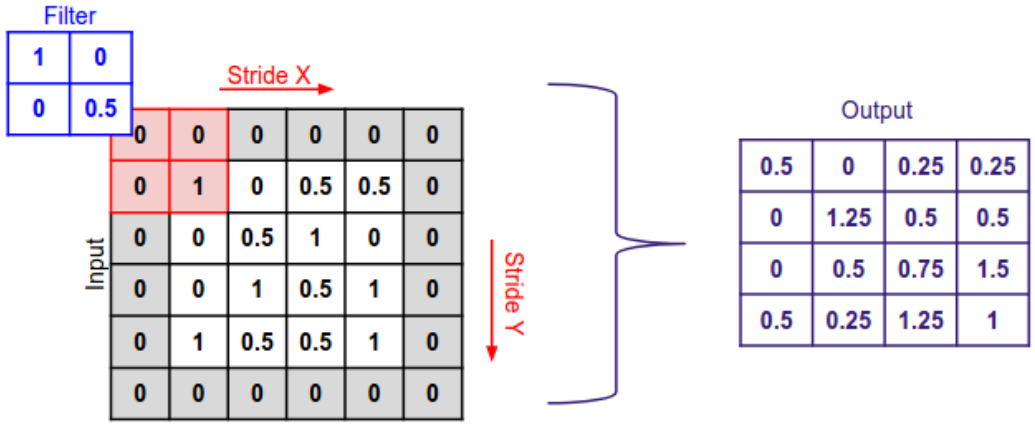


Stride and padding

- We can also set the “stride” (step-size) for our convolution
  - Stride > 1 often used to down-sample the image
- What do we do with border pixels?
  - Padding: Fill up the image borders (zero-padding is most common)

In general, common to see CONV layers with stride 1, filters of size FxF, and zero-padding with (F-1)/2. (will preserve size spatially)

- e.g. F = 3 => zero pad with 1
- F = 5 => zero pad with 2
- F = 7 => zero pad with 3

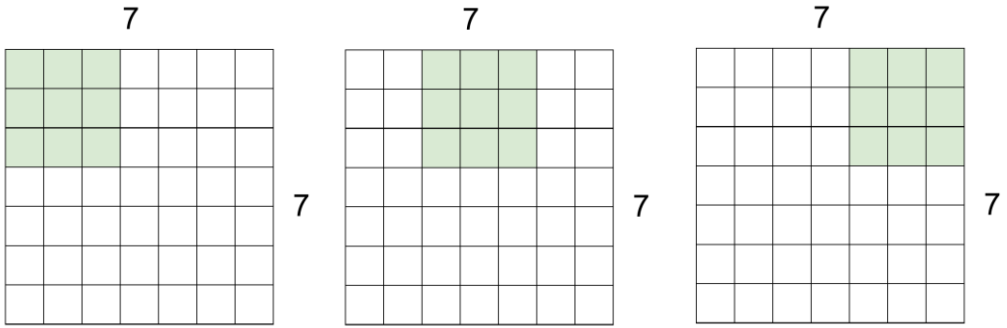


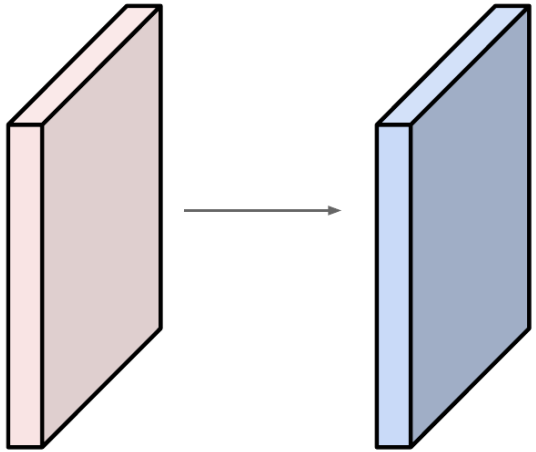
Example:

Input volume: 32x32x3

- 10 5x5 filters with stride 1, pad 2

Output volume size: ?





Example:

Input volume: 32x32x3

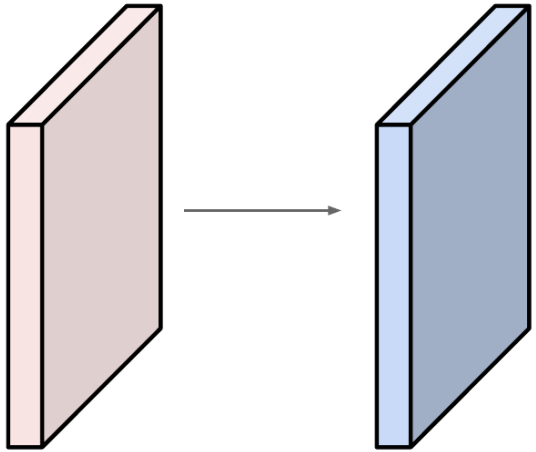
- 10 5x5 filters with stride 1, pad 2

Output volume size:

- (32+2*2-5)/1+1 = 32 spatially, so 32x32x10

Number of parameters in this layer?

- each filter has 5*5*3 + 1 (bias) = 76 params
- => 76*10 = 760



ConvLayer Summary

Accepts a volume of size $W_1 \times H_1 \times D_1$

- Four hyperparameters:
  - Number of filters K
  - Spatial extend / kernel size F
  - Stride S
  - Amount of zero padding P

Produces a volume of size $W_2 \times H_2 \times D_2$ *where*

- $W_2 = (W_1 - F + 2P)/S + 1$
- $H_2 = (H_1 - F + 2P)/S + 1$
- $D_2 = K$

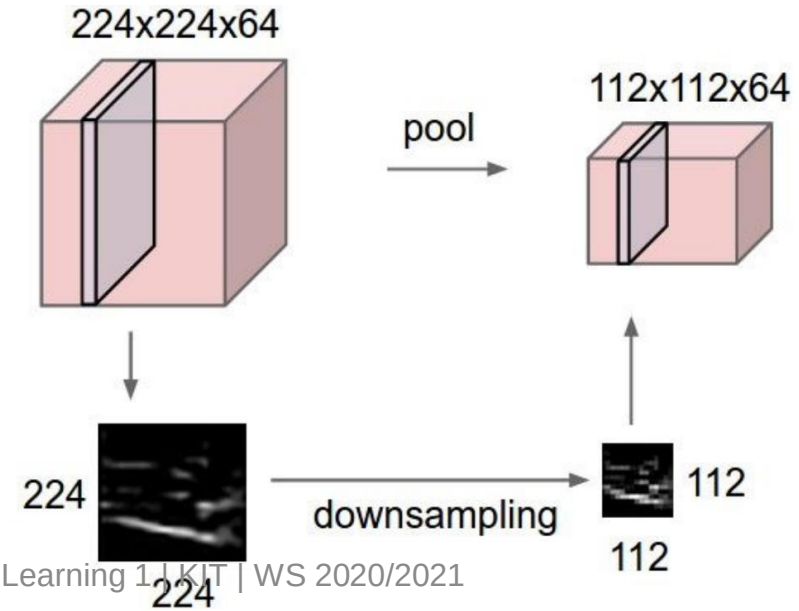
- Number of Weights: $F \cdot F \cdot D_1 \cdot K$ (and K biases)

Common settings:

- K = (powers of 2, e.g. 32, 64, 128, 512)
- F = 3, S = 1, P = 1
- F = 5, S = 1, P = 2
- F = 5, S = 2, P = ? (whatever fits)
- F = 1, S = 1, P = 0

Pooling Layers

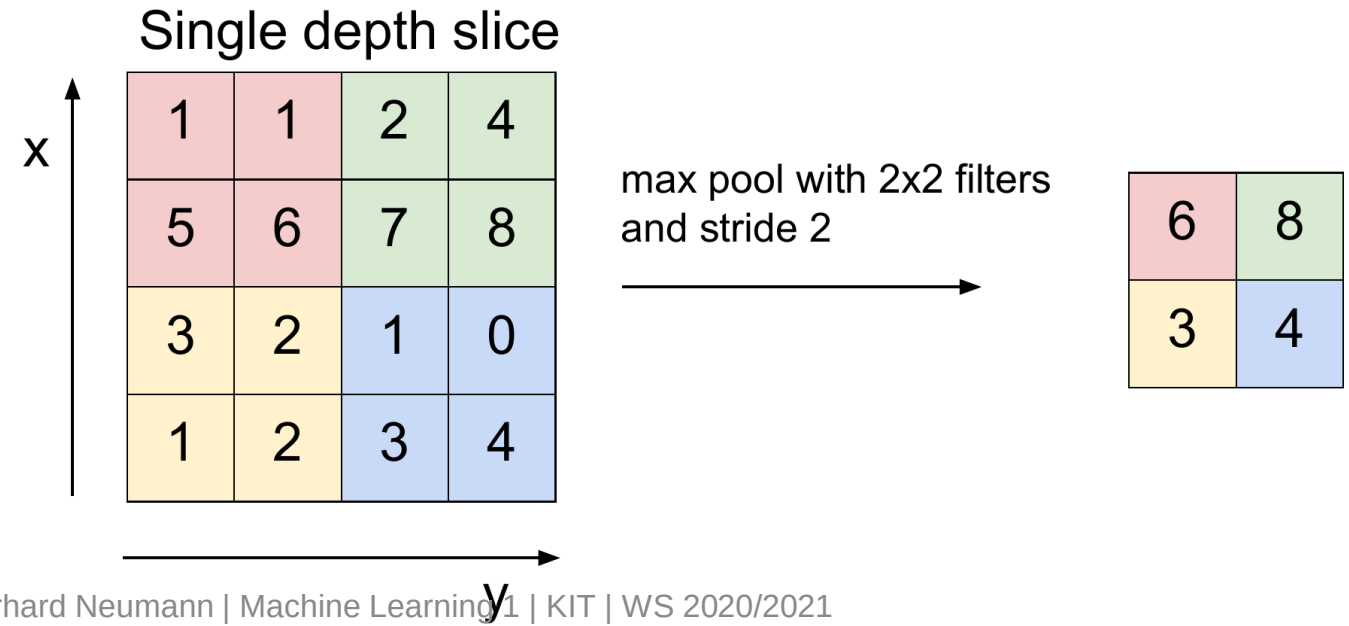
- Pooling layer makes the representations smaller and more manageable
- operates over each activation map independently:



Max-Pooling

The most common pooling is max-pooling:

- For each channel, compute the max over the whole window



Pooling Layer Summary

Accepts a volume of size $W_1 \times H_1 \times D_1$

- Two hyperparameters:
  - Spatial extend / kernel size F
  - Stride S

Produces a volume of size $W_2 \times H_2 \times D_2$ *where*

- $W_2 = (W_1 - F)/S + 1$
- $H_2 = (H_1 - F)/S + 1$
- $D_2 = D_1$

Common settings:

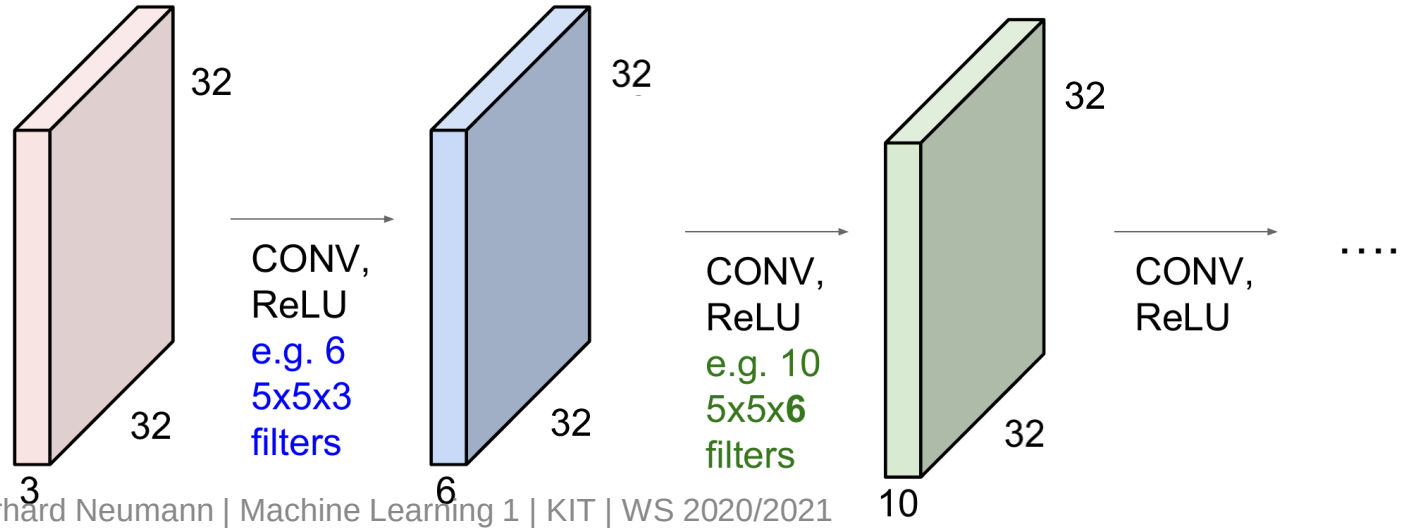
- F = 2, S = 2
- F = 3, S = 2

- Introduces 0 parameters since it computes a fixed function of the input
- Note that typically no zero padding is used for pooling

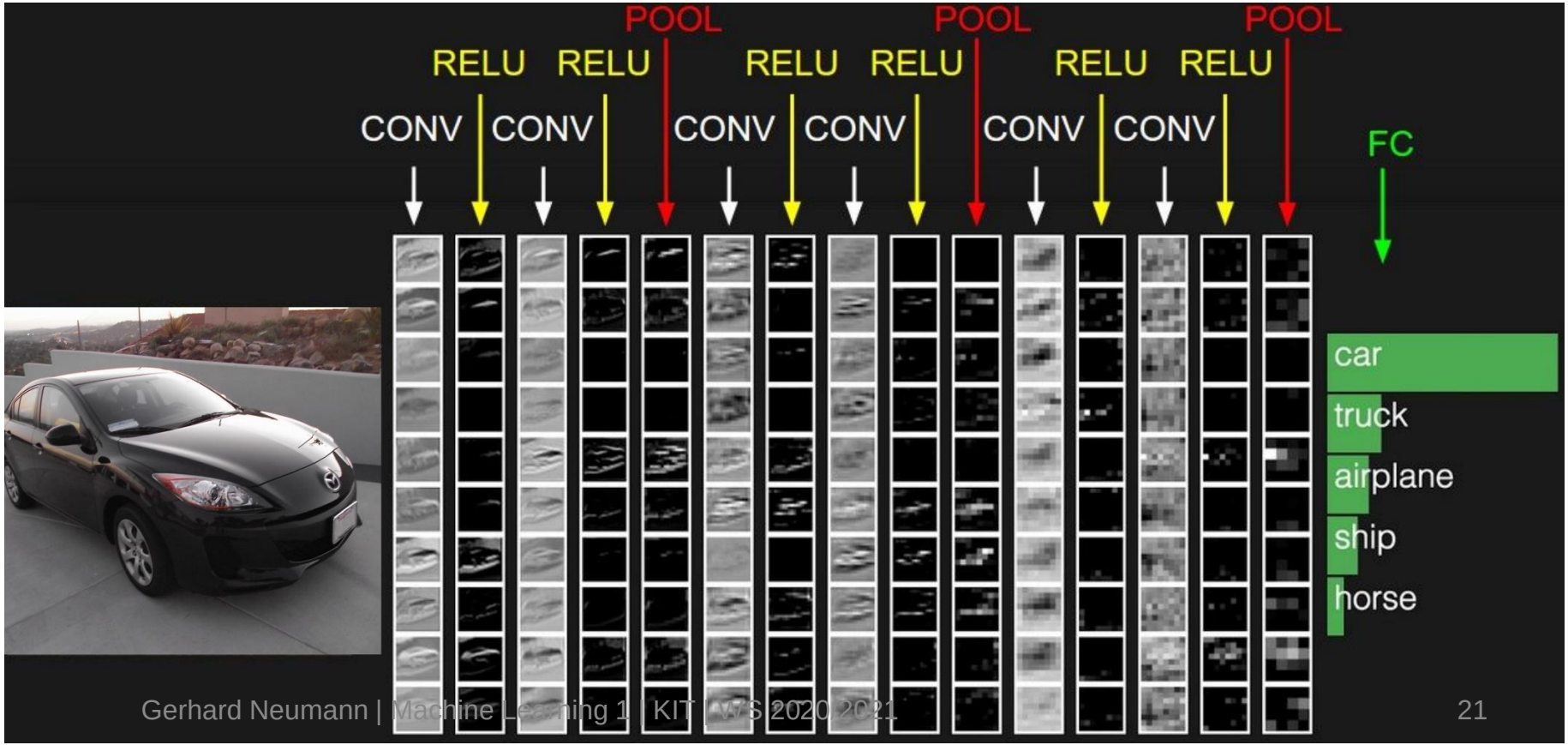
Convolutional Network

A convolutional Network is a sequence of Convolution Layers, interspersed with activation functions and pooling functions…

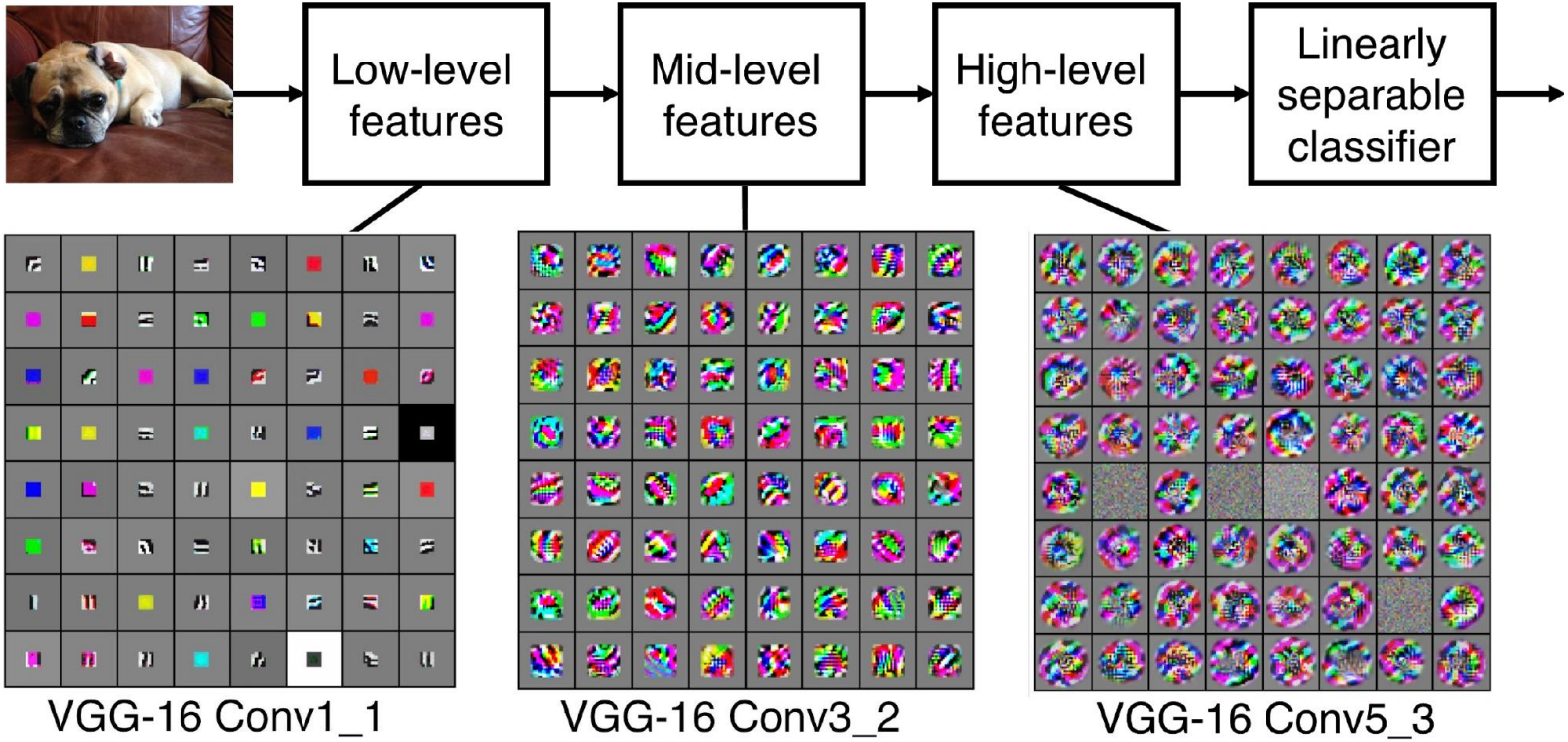
… followed by one or multiple FC layers to compute the output (regression or classification)



Convolutional Network



Visualization of the filters



Visualization of VGG-16 by Lane McIntosh. VGG-16 architecture from [Simonyan and Zisserman 2014].

CNN architectures

Case Study:

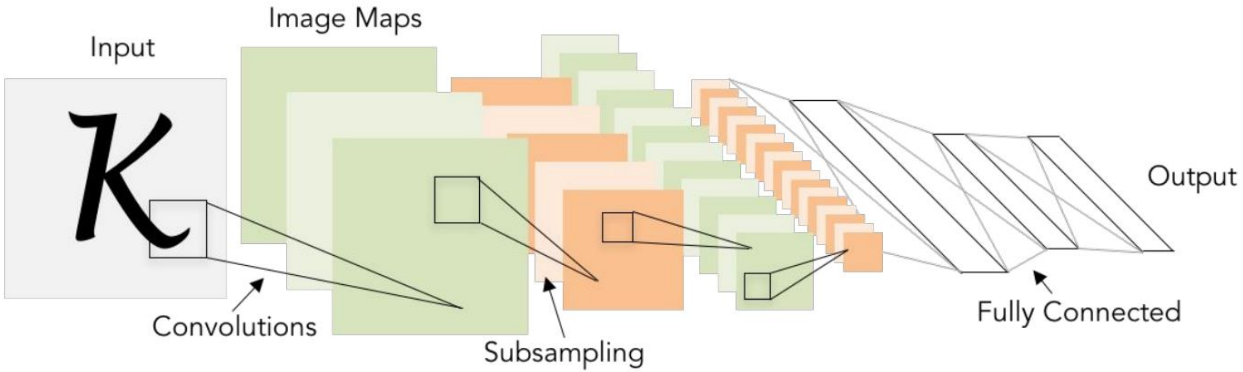
- AlexNet
- VGG
- GoogLeNet
- ResNet

Also....

- SENet
- NiN (Network in Network)
- Wide ResNet
- ResNeXT
- DenseNet
- FractalNet
- MobileNets
- NASNet

A bit of history…

Gradient-based learning applied to document recognition
[LeCun, Bottou, Bengio, Haffner 1998]



LeNet-5

- Conv filters were 5x5, applied at stride 1
- Subsampling (Pooling) layers were 2x2 applied at stride 2
- i.e. architecture is [CONV-POOL-CONV-POOL-FC-FC]

ImageNet

Standard benchmark for vision:

- 1.2 M images
- 1000 classes
- > 500 images per class

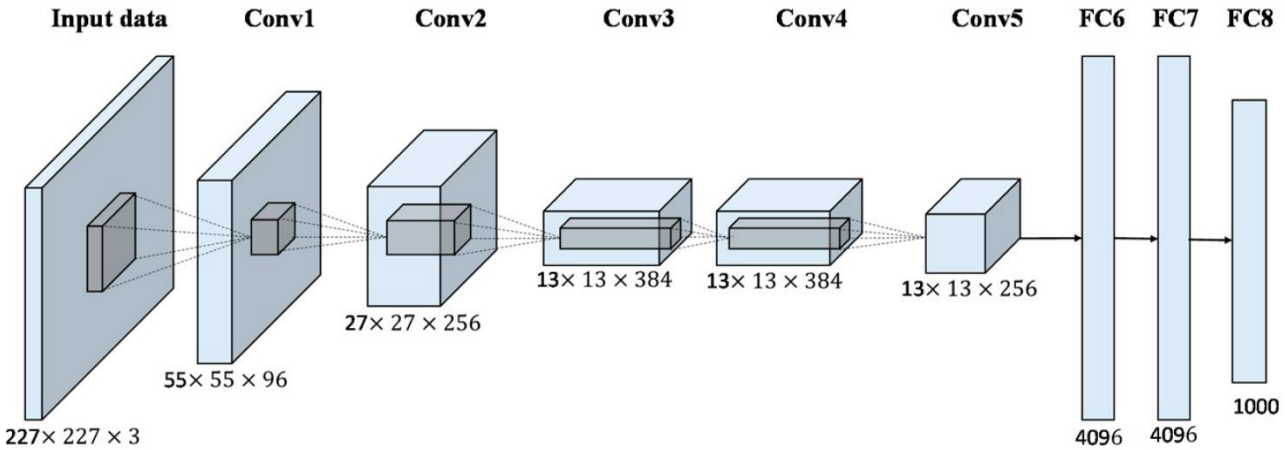
AlexNet

ImageNet Classification with Deep Convolutional Neural Networks

[Krizhevsky, Sutskever, Hinton, 2012]

Architecture:

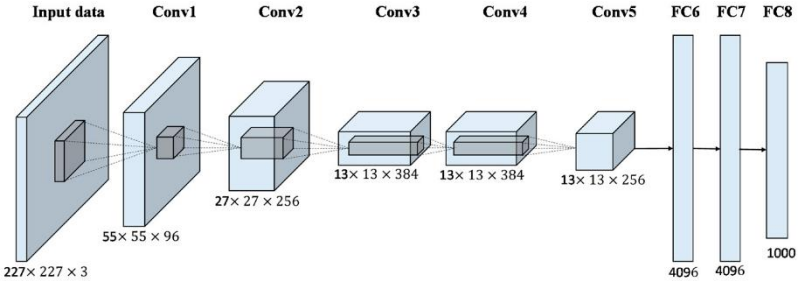
CONV1
MAX POOL1
NORM1
CONV2
MAX POOL2
NORM2
CONV3
CONV4
CONV5
MAX POOL3
FC6
FC7
FC8



AlexNet

ImageNet Classification with Deep Convolutional Neural Networks

[Krizhevsky, Sutskever, Hinton, 2012]



Input: 227x227x3 images

First layer (CONV1): 96 11x11 filters applied at stride 4

Q: what is the output volume size? Hint: (227-11)/4+1 = 55

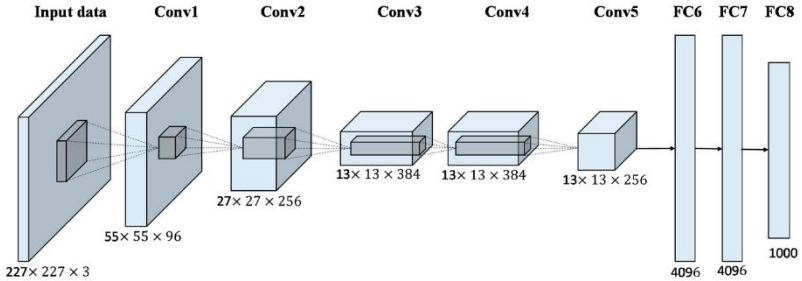
Output volume: [55x55x96]

Parameters: (11*11*3)*96 = 35K

AlexNet

ImageNet Classification with Deep Convolutional Neural Networks

[Krizhevsky, Sutskever, Hinton, 2012]



Input: 227x227x3 images

After CONV1: 55x55x96

Second layer (POOL1): 3x3 filters applied at stride 2

Q: what is the output volume size? Hint: (55-3)/2+1 = 27

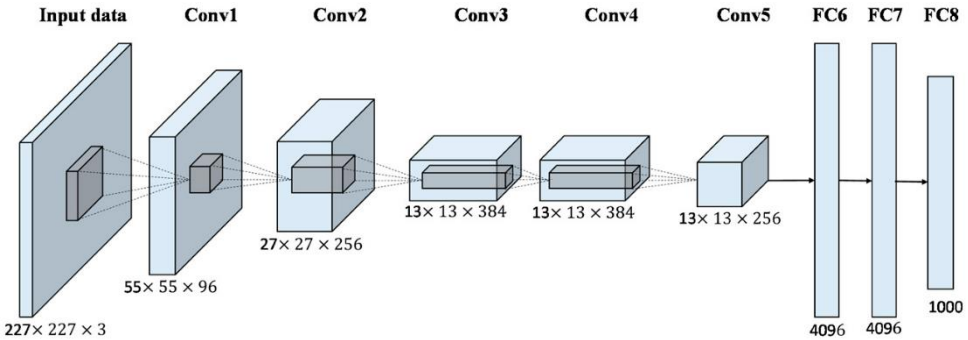
Output volume: 27x27x96

Parameters: 0!

AlexNet

ImageNet Classification with Deep Convolutional Neural Networks
[Krizhevsky, Sutskever, Hinton, 2012]

Full (simplified) AlexNet architecture:
[227x227x3] INPUT
[55x55x96] CONV1: 96 11x11 filters at stride 4, pad 0
[27x27x96] MAX POOL1: 3x3 filters at stride 2
[27x27x96] NORM1: Normalization layer
[27x27x256] CONV2: 256 5x5 filters at stride 1, pad 2
[13x13x256] MAX POOL2: 3x3 filters at stride 2
[13x13x256] NORM2: Normalization layer
[13x13x384] CONV3: 384 3x3 filters at stride 1, pad 1
[13x13x384] CONV4: 384 3x3 filters at stride 1, pad 1
[13x13x256] CONV5: 256 3x3 filters at stride 1, pad 1
[6x6x256] MAX POOL3: 3x3 filters at stride 2
[4096] FC6: 4096 neurons
[4096] FC7: 4096 neurons
[1000] FC8: 1000 neurons (class scores)

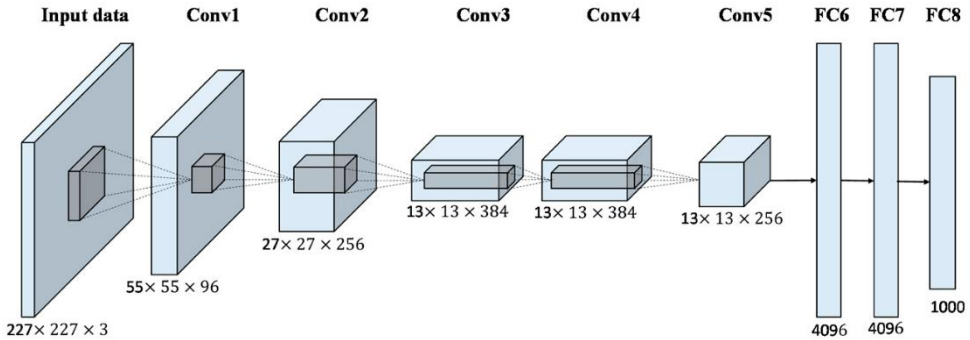


AlexNet

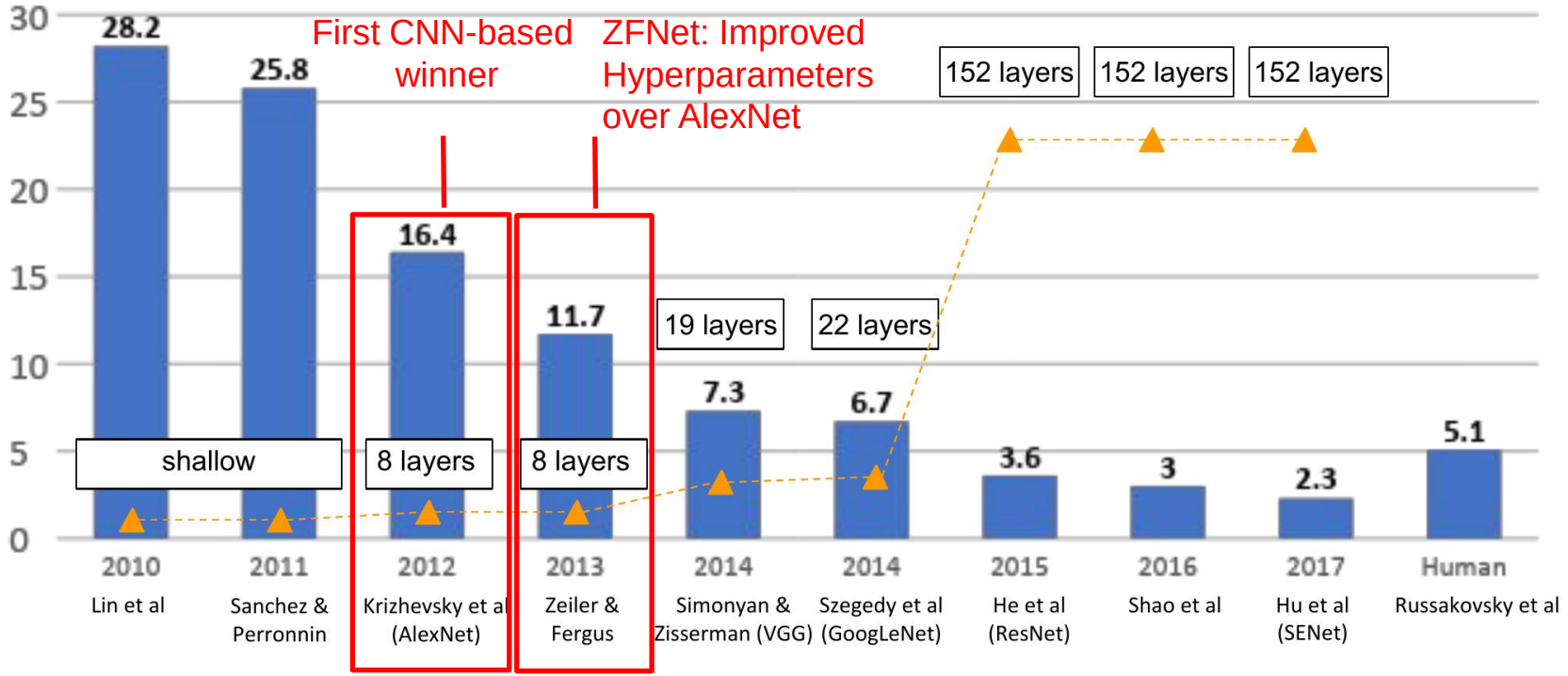
ImageNet Classification with Deep Convolutional Neural Networks
[Krizhevsky, Sutskever, Hinton, 2012]

Details/Retrospectives:

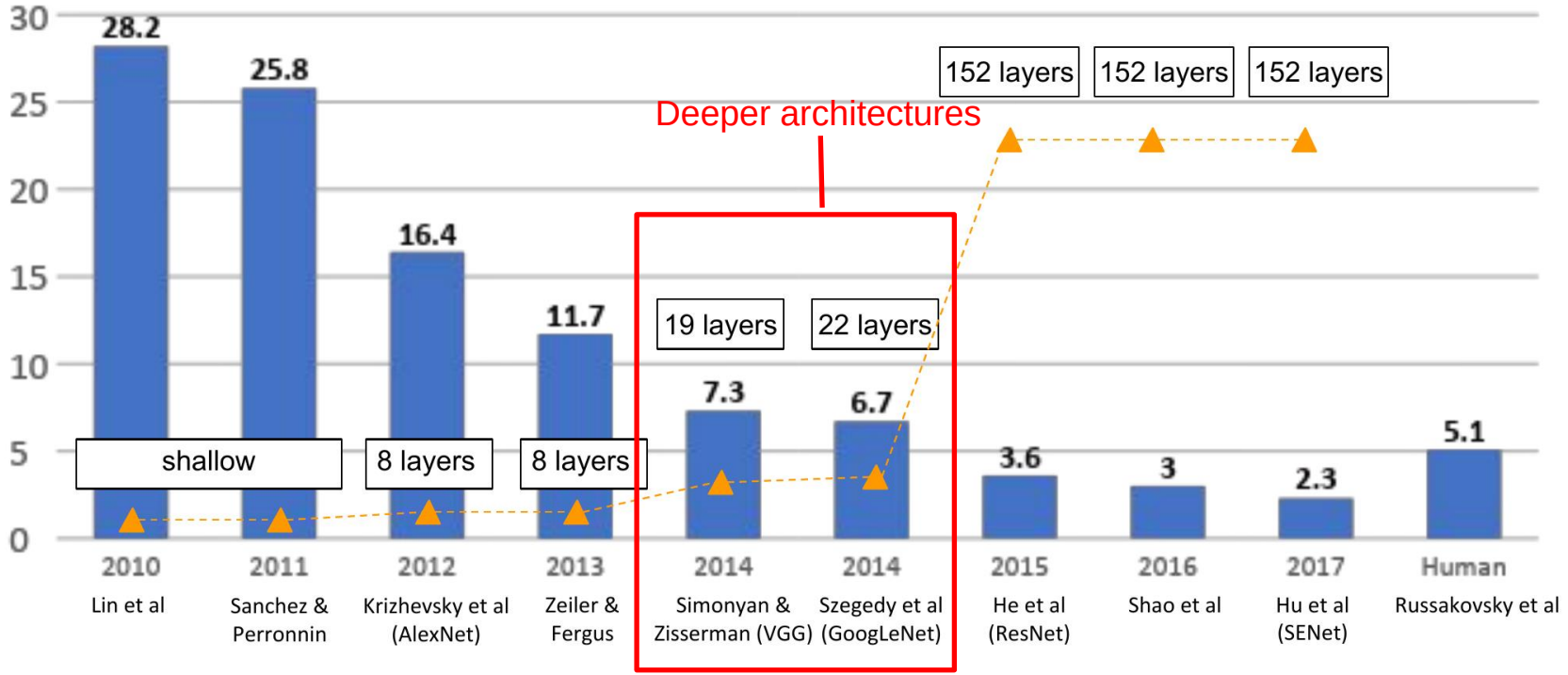
- first use of ReLU
- used Norm layers (not common anymore)
- heavy data augmentation
- dropout 0.5
- batch size 128
- SGD Momentum 0.9
- Learning rate 1e-2, reduced by a factor of 10 manually when val accuracy plateaus
- L2 weight decay 5e-4
- 7 CNN ensemble: 18.2% -> 15.4%



ImageNet challenge (ILSVRC) winners



ImageNet challenge winners



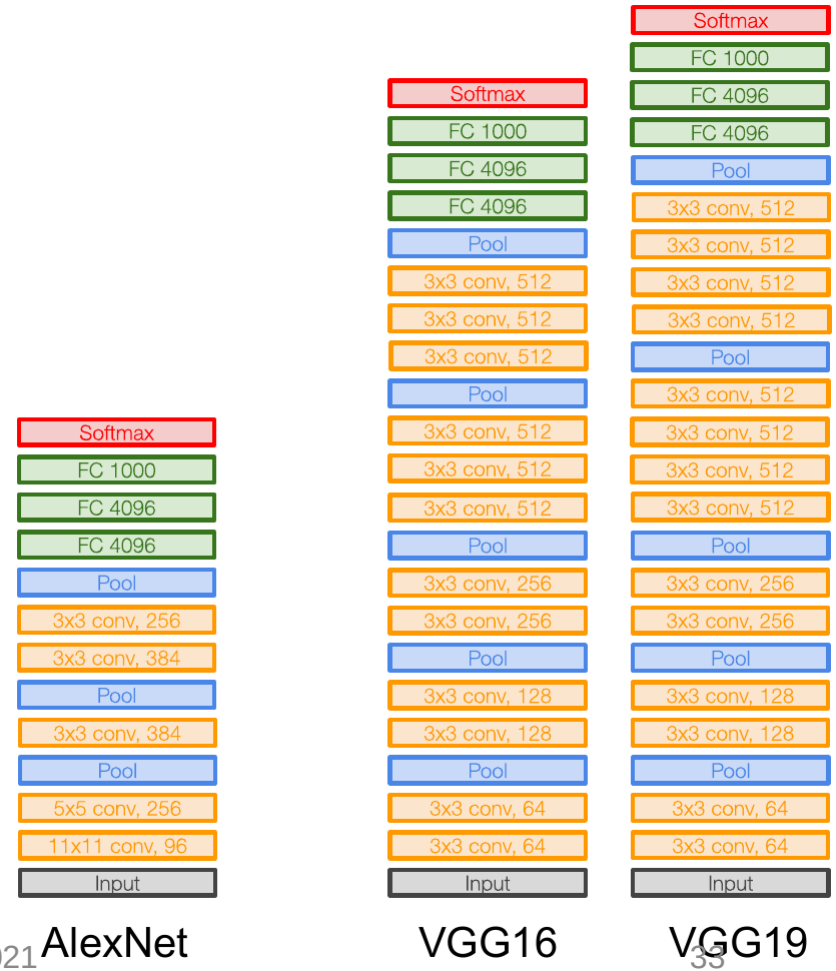
VGG Net

Small filters, Deeper networks

- 8 layers (AlexNet) -> 16 - 19 layers (VGG16Net)
- Only 3x3 CONV stride 1, pad 1
- and 2x2 MAX POOL stride 2
- 11.7% top 5 error in ILSVRC'13 (ZFNet)
- -> 7.3% top 5 error in ILSVRC'14

Why use smaller filters? (3x3 conv)

- Stack of three 3x3 conv (stride 1) layers has same effective receptive field as one 7x7 conv layer
- But deeper, more non-linearities
- And fewer parameters:
  $3 * (3^2C^2)$ vs. 7^2C^2 for C channels per layer

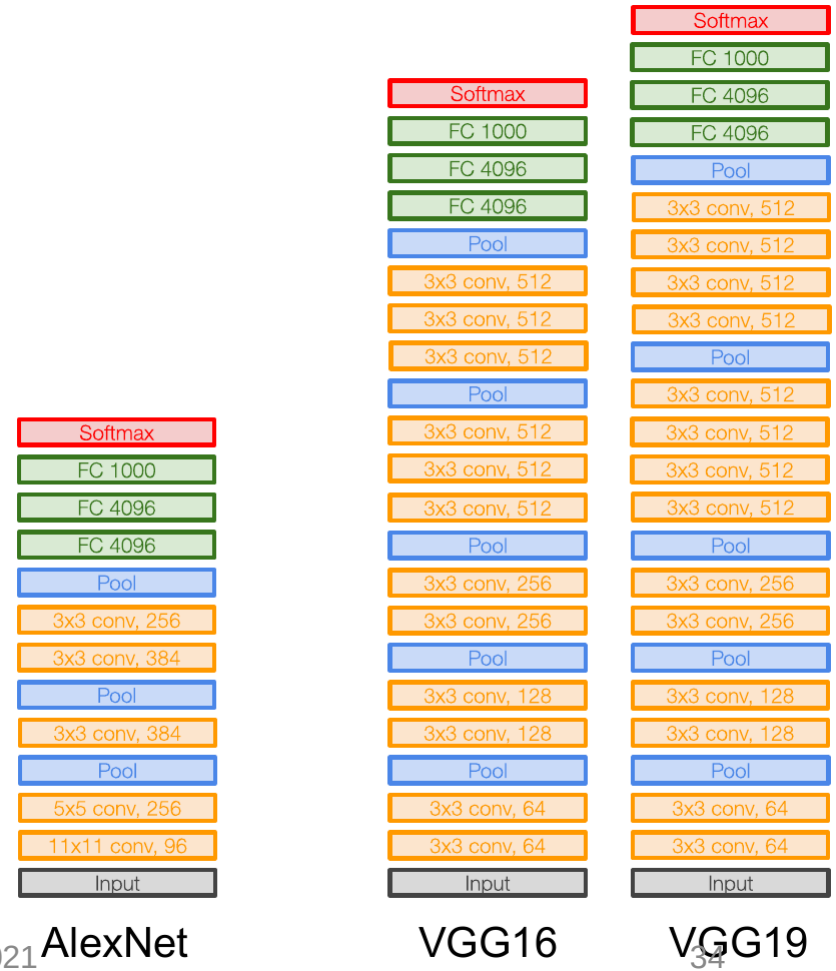


[Simonyan and Zisserman, 2014]

VGG Net

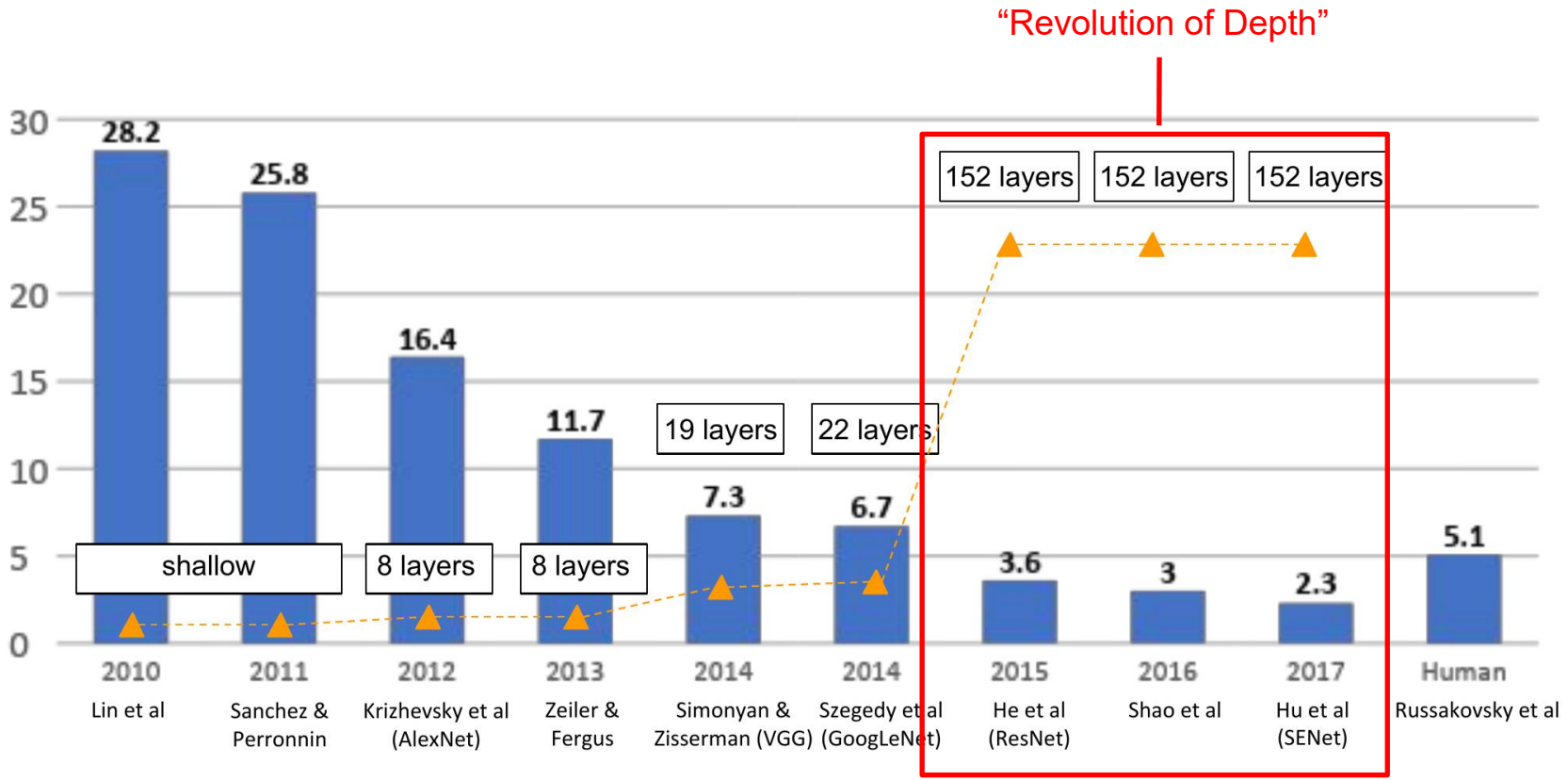
Details:

- ILSVRC'14 2nd in classification, 1st in localizatior
- Similar training procedure as AlexNet
- No Local Response Normalisation (LRN)
- Use VGG16 or VGG19 (VGG19 only slightly better, more memory)
- Use ensembles for best results
- FC7 features generalize well to other tasks



[Simonyan and Zisserman, 2014]

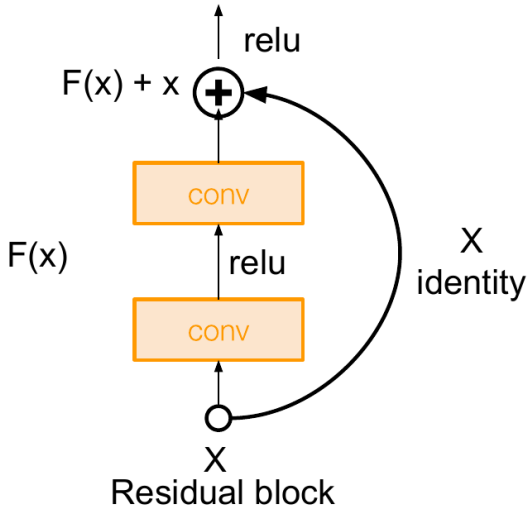
ImageNet challenge winners

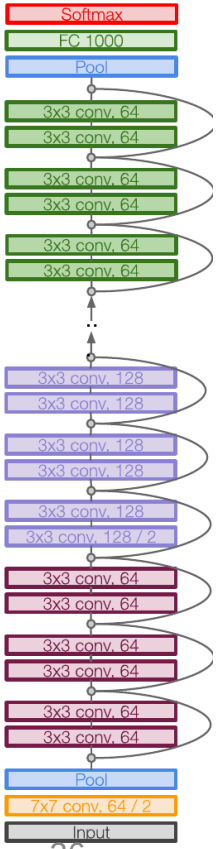


ResNet

Very deep networks using residual connections

- 152-layer model for ImageNet
- ILSVRC'15 classification winner (3.57% top 5 error)
- Swept all classification and detection competitions in ILSVRC'15 and COCO'15!

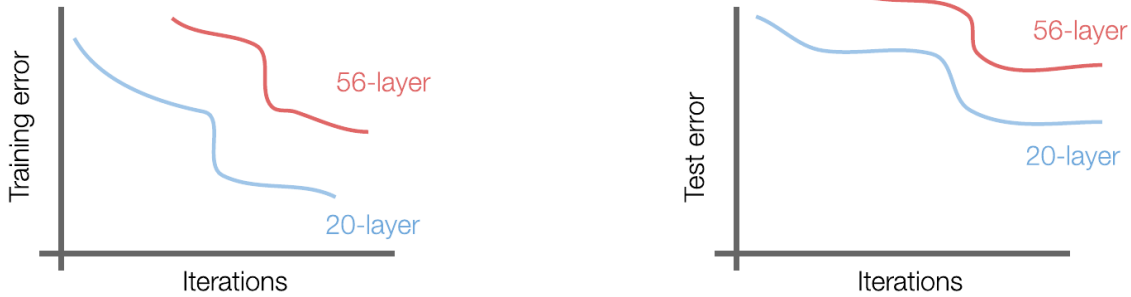




A new level of Depth…

How can we train such deep networks?

Deeper models should be able to perform at least as well as the shallower model. However, this is typically not the case



56-layer model performs worse on both training and test error

-> The deeper model performs worse, but it's not caused by overfitting!

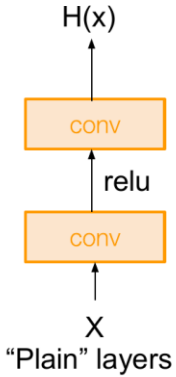
Hypothesis: the problem is an optimization problem, deeper models are harder to optimize.

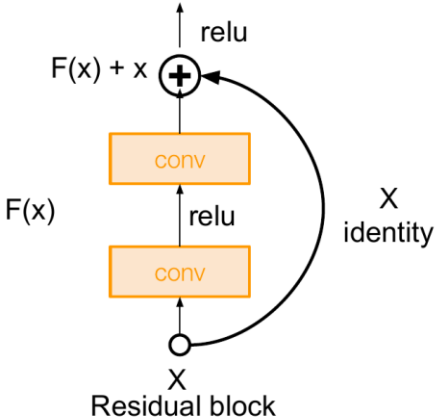
Residual Layers

How can we train such deep networks?

Solution: Use network layers to fit a residual mapping instead of directly trying to fit a desired underlying mapping

- Use layers to fit residual F(x) = H(x) – x instead of H(x) directly
- Initially, F(x) is set to 0, so the layer just computes the identity
- I.e. adding more layers does not harm

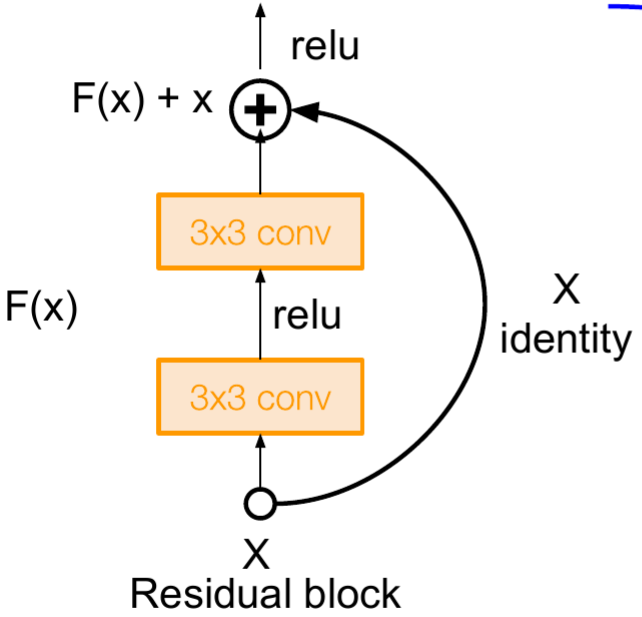


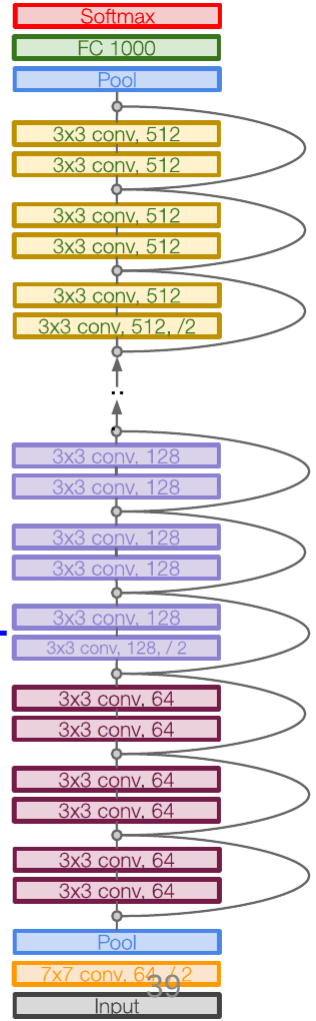


ResNet

Full ResNet architecture:

- Stack residual blocks
- Every residual block has two 3x3 conv layers

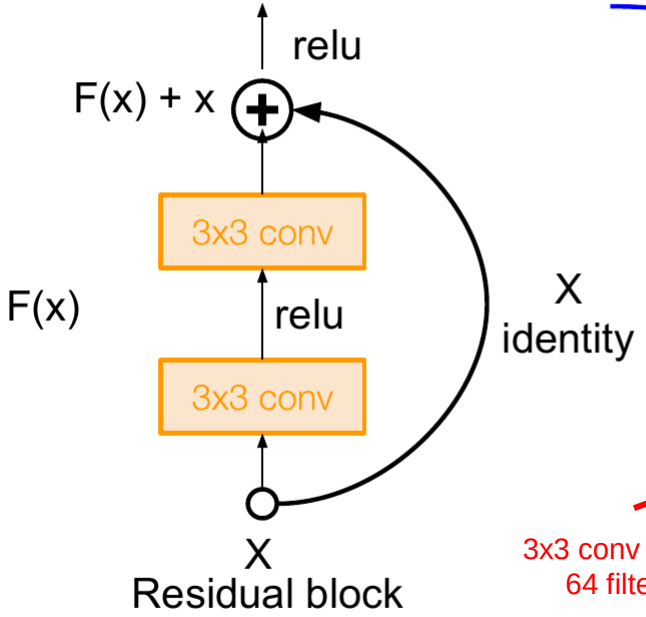


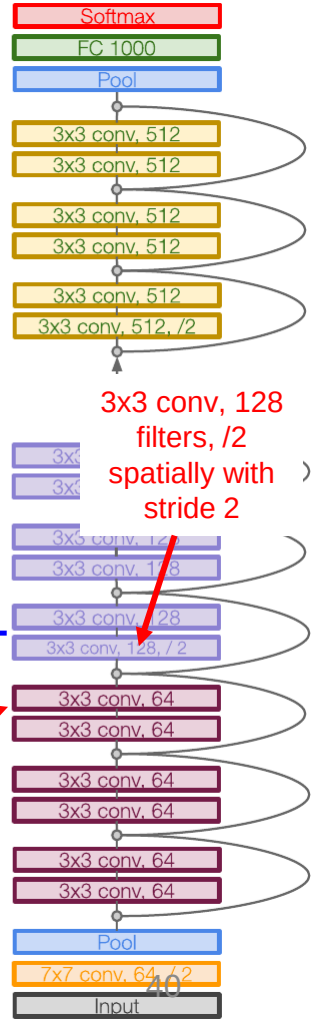


ResNet

Full ResNet architecture:

- Stack residual blocks
- Every residual block has two 3x3 conv layers
- Periodically, double # of filters and downsample spatially using stride 2 (/2 in each dimension)

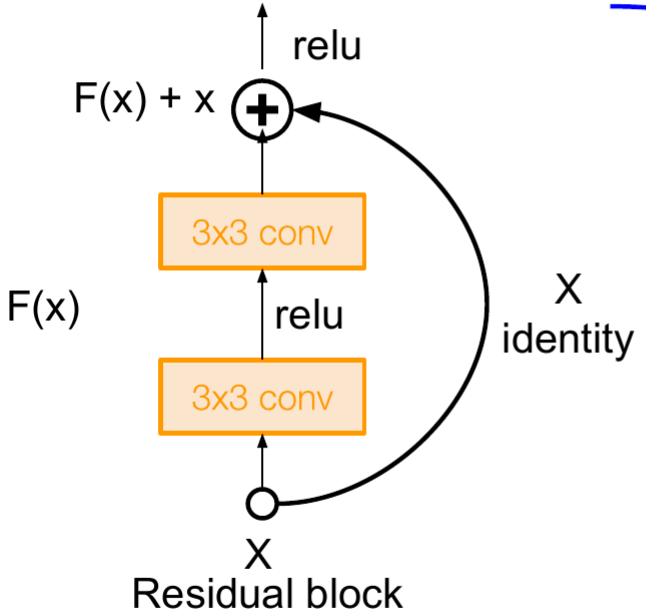


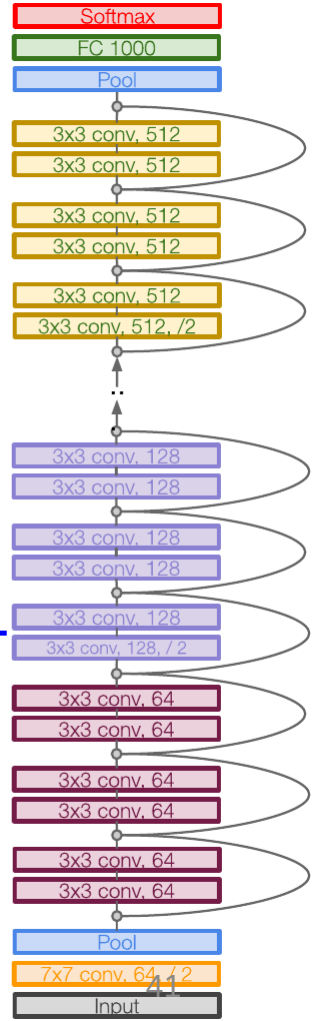


ResNet

Full ResNet architecture:

- Stack residual blocks
- Every residual block has two 3x3 conv layers
- Periodically, double # of filters and downsample spatially using stride 2 (/2 in each dimension)
- Additional conv layer at the beginning



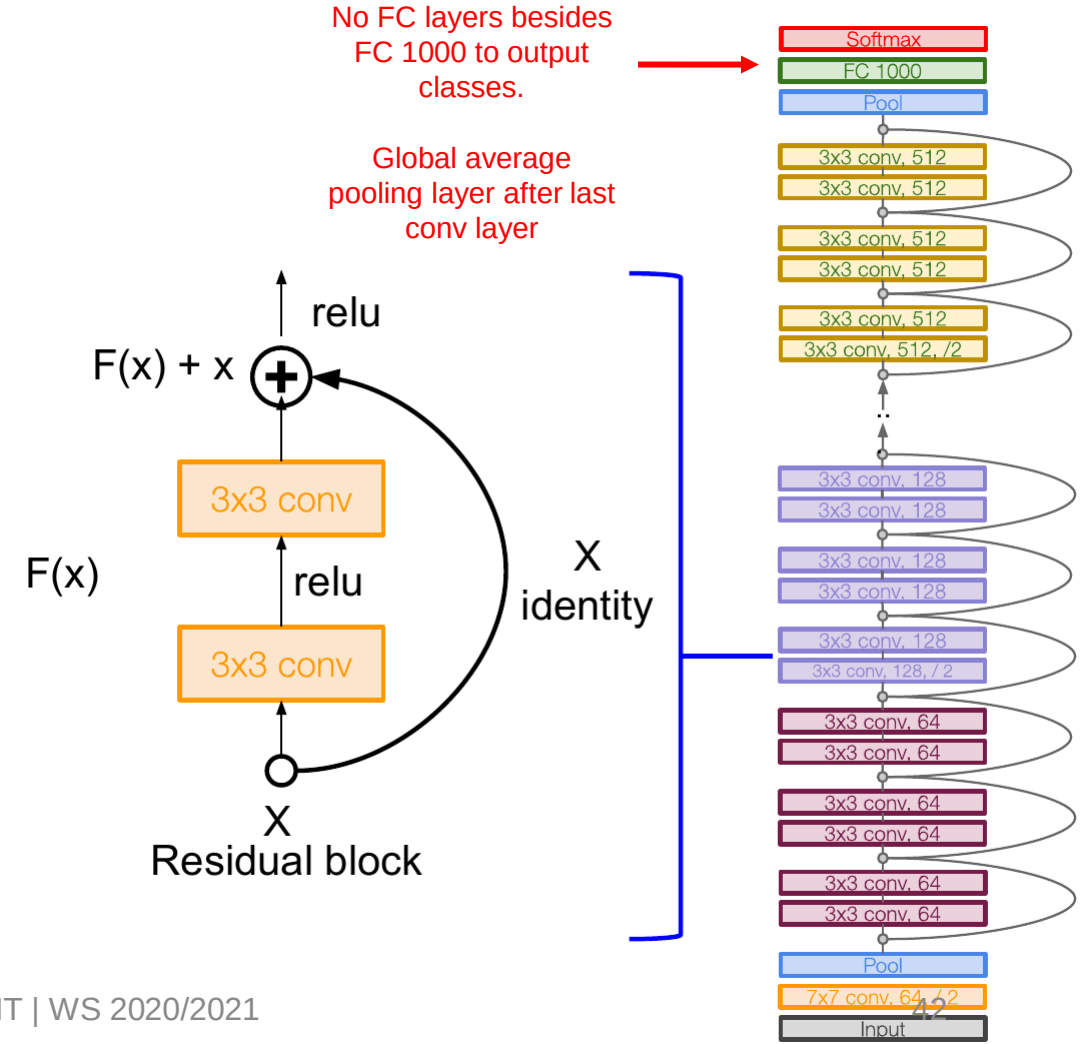


7x7 conv layer at the beginning

ResNet

Full ResNet architecture:

- Stack residual blocks
- Every residual block has two 3x3 conv layers
- Periodically, double # of filters and downsample spatially using stride 2 (/2 in each dimension)
- Additional conv layer at the beginning
- No FC layers at the end (only FC 1000 to output classes)

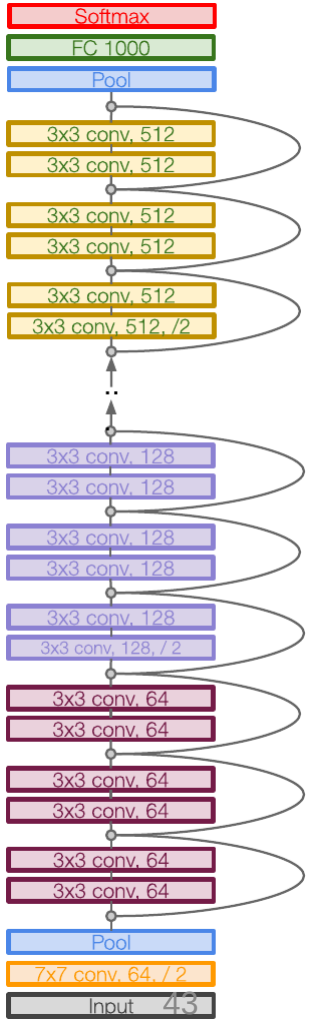


ResNet

Full ResNet architecture:

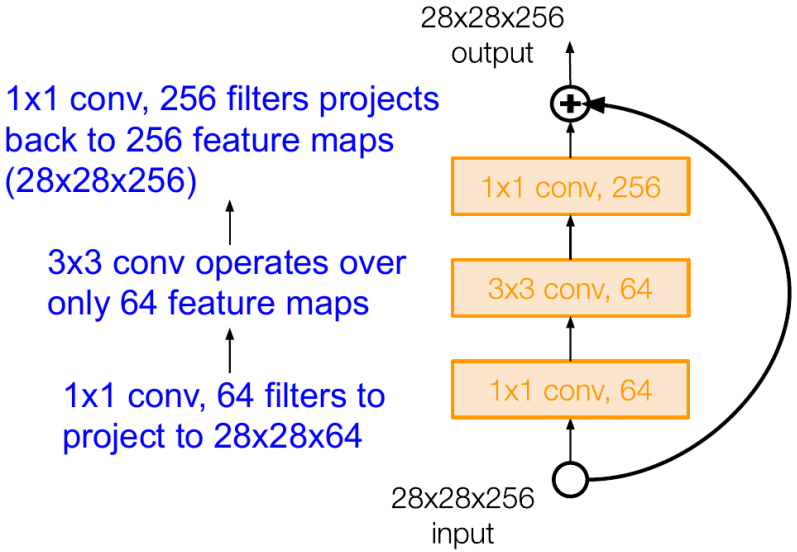
- Stack residual blocks
- Every residual block has two 3x3 conv layers
- Periodically, double # of filters and downsample spatially using stride 2 (/2 in each dimension)
- Additional conv layer at the beginning
- No FC layers at the end (only FC 1000 to output classes)

Total depths of 34, 50, 101, or 152 layers for ImageNet



Case Study: ResNet

- For deeper networks (ResNet-50+), use “bottleneck” layer to improve efficiency



Case Study: ResNet

Training ResNet in practice:

- Batch Normalization after every CONV layer (not covered)
- Xavier 2/ initialization from He et al.
- SGD + Momentum (0.9)
- Learning rate: 0.1, divided by 10 when validation error plateaus
- Mini-batch size 256
- Weight decay of 1e-5
- No dropout used

Experimental Results

- Able to train very deep networks without degrading (152 layers on ImageNet, 1202 on Cifar)
- Deeper networks now achieve lower training error as expected
- Swept 1st place in all ILSVRC and COCO 2015 competitions
- ILSVRC 2015 classification winner (3.6% top 5 error) -- better than “human performance”! (Russakovsky 2014)

Beyond millions of samples

Very impressive results… but ImageNet has 1.2 million images!

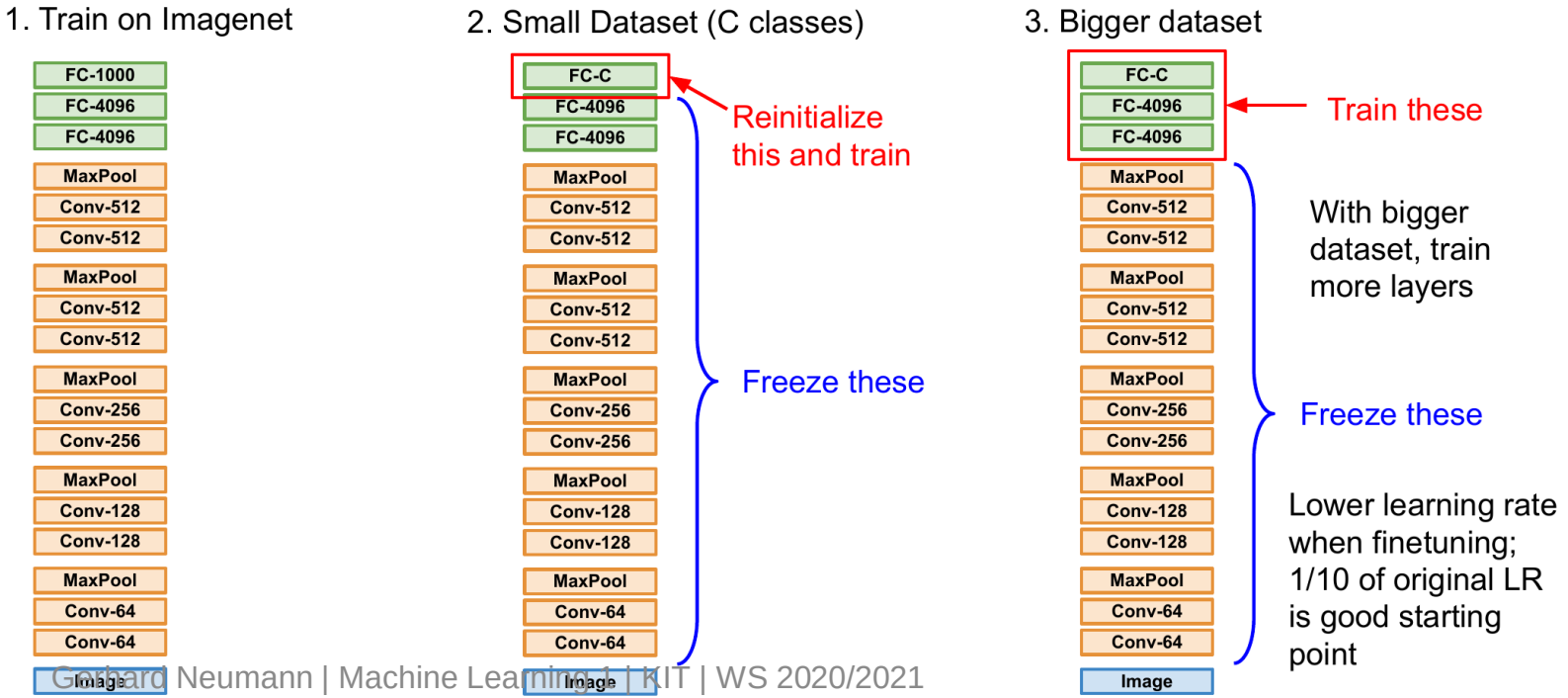
- Typically, we do not have that many!
- Can we also use these methods with less images?

Yes, with transfer learning!

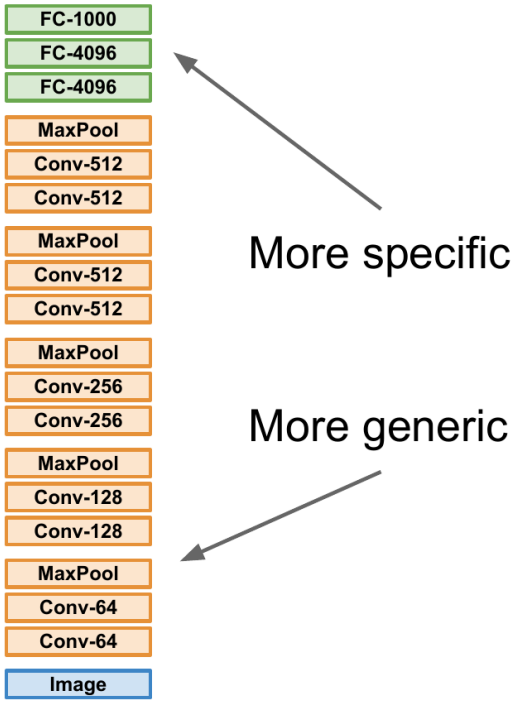
- Features (conv layers) are generic and can be reused!

Transfer learning

- Train on huge data-set (e.g. Imagenet)
- Freeze layers and adapt only last (FC) layers



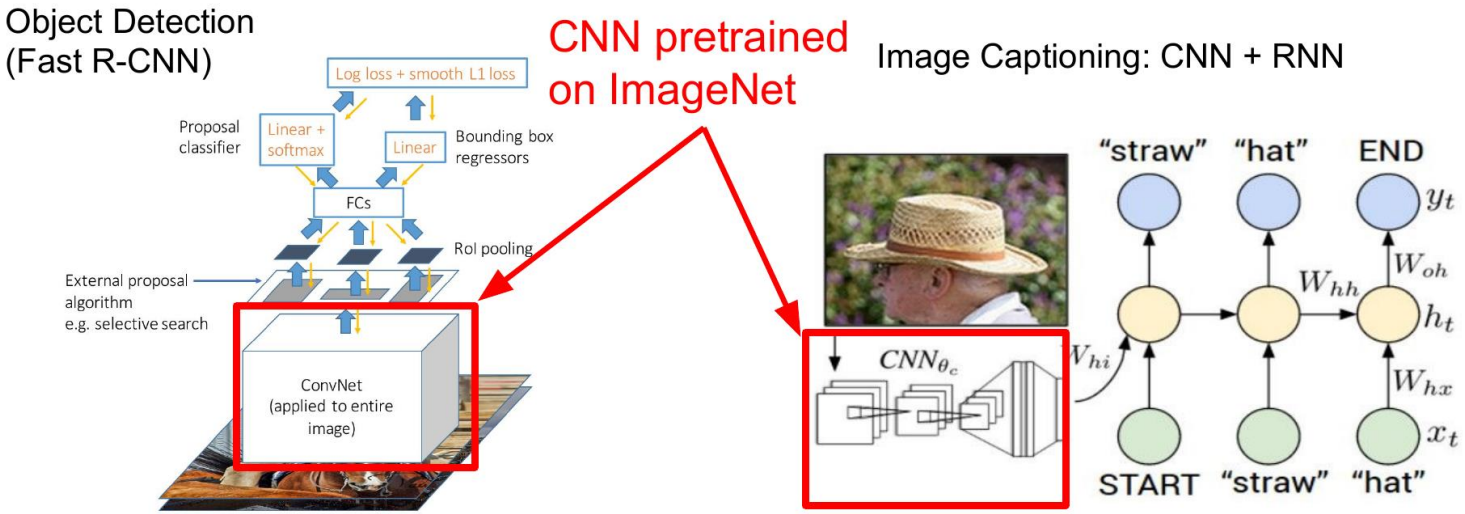
Transfer Learning



- **Very little data, very similar dataset:**
  - Use Linear classifier on top layer
- **Very little data, very different dataset:**
  - You're in trouble... Try linear classifier from different stages and pray
- **A lot of data, very similar dataset:**
  - Finetune a few layers
- **A lot of data, very different dataset:**
  - Finetune a larger number of layers

Transfer learning

Transfer learning with CNNs is the norm, not the exception…



Girshick, "Fast R-CNN", ICCV 2015
Figure copyright Ross Girshick, 2015. Reproduced with permission.

Karpathy and Fei-Fei, "Deep Visual-Semantic Alignments for Generating Image Descriptions", CVPR 2015
Figure copyright IEEE, 2015. Reproduced for educational purposes.

Transfer learning

Deep learning frameworks provide a “Model Zoo” of pretrained models so you don’t need to train your own

- TensorFlow: https://github.com/tensorflow/models
- PyTorch: https://github.com/pytorch/vision

Summary: CNNs

What have we learned today?

- CNNs employ convolutions to exploit spatial structure of images
- Can also be used for other modalities (audio, etc..)

A CNN consists of:

- Conv layers
- ReLU activation units
- Pooling

Many popular architectures available in model zoos

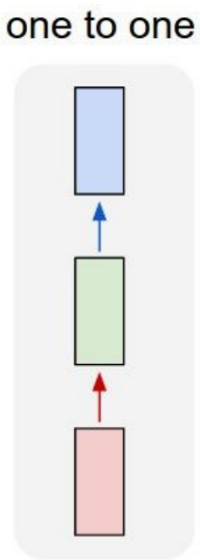
- ResNet and SENet (not covered) currently good defaults to use
- Networks have gotten increasingly deep over time
- Pre-trained models can be fine-tuned if amount of training data is not sufficient

Self-test questions

- Why are fully connected networks for images a bad idea and why do we need images?
- What are the key components of a CNN?
- What hyper-parameters can we set for a convolutional layer and what is their meaning?
- What hyper-parameters can we set for a pooling layer and what is their meaning?
- How can we compute the dimensionality of the output of a convolutional layer
- Describe basic properties of AlexNet and VCG
- What is the main idea of ResNet to make it very deep?

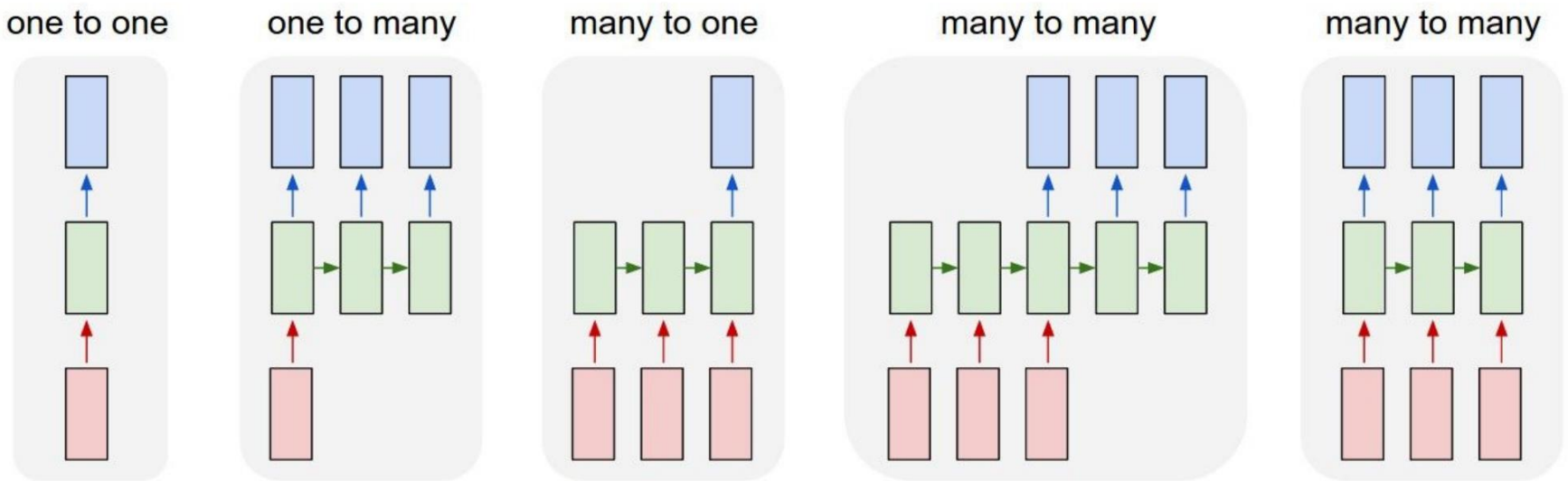
Recurrent Neural Networks

Vanilla Neural Networks



Vanilla Neural Networks

Recurrent Neural Networks: Process Sequences



Vanilla Neural Networks

e.g. **Image Captioning**
Image -> Seq. of words

e.g. **Sentiment Classification**
sequence of words -> sentiment

e.g. **Machine Translation**
seq of words -> seq of words

e.g. Video classification
on frame level

Recurrent Neural Networks

We can process a **sequence of vectors** x by applying a **recurrence formula** at every time step:

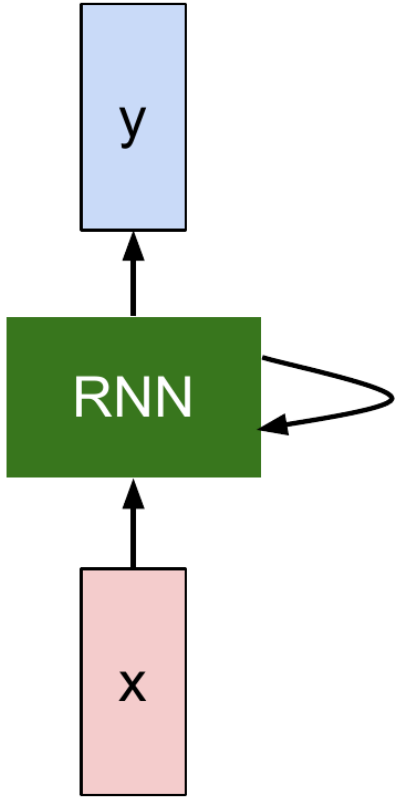
$$\boldsymbol{h}_t = f_{\boldsymbol{W}}(\boldsymbol{h}_{t-1}, \boldsymbol{x}_t)$$

Old state

New state

Some function with parameters **W**

Input



- **Note:** same function and same parameters are used for every time step

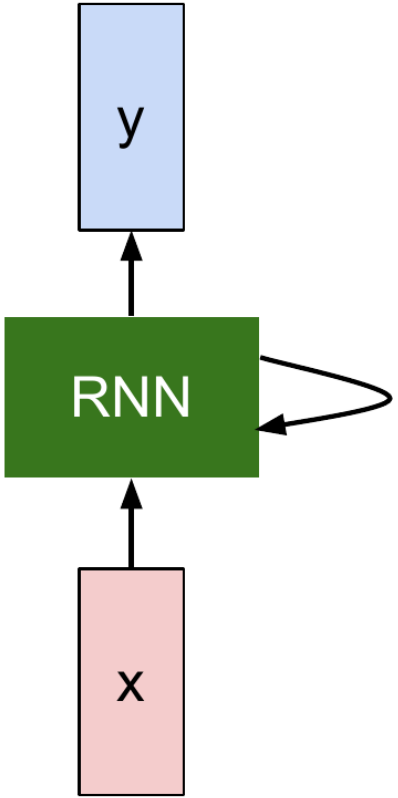
“Vanilla” Recurrent Neural Networks

The state consists of a single “hidden” vector h:

$$\boldsymbol{h}_t = f_{\boldsymbol{W}}(\boldsymbol{h}_{t-1}, \boldsymbol{x}_t)$$

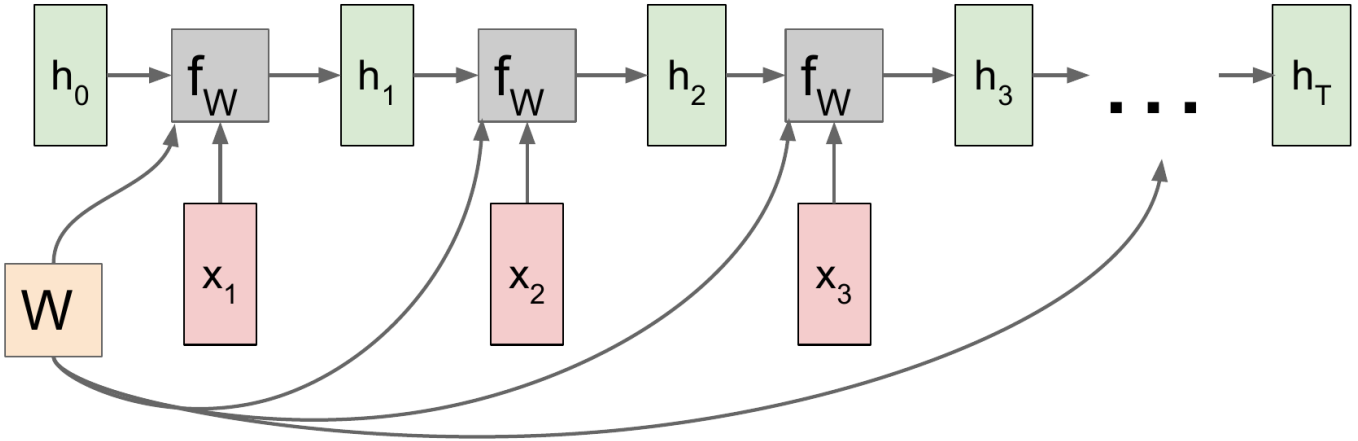
$$\boldsymbol{h}_t = \tanh(\boldsymbol{W}_{hh}\boldsymbol{h}_{t-1} + \boldsymbol{W}_{xh}\boldsymbol{x}_t)$$

$$\boldsymbol{y}_t = \boldsymbol{W}_{hy}\boldsymbol{h}_t$$



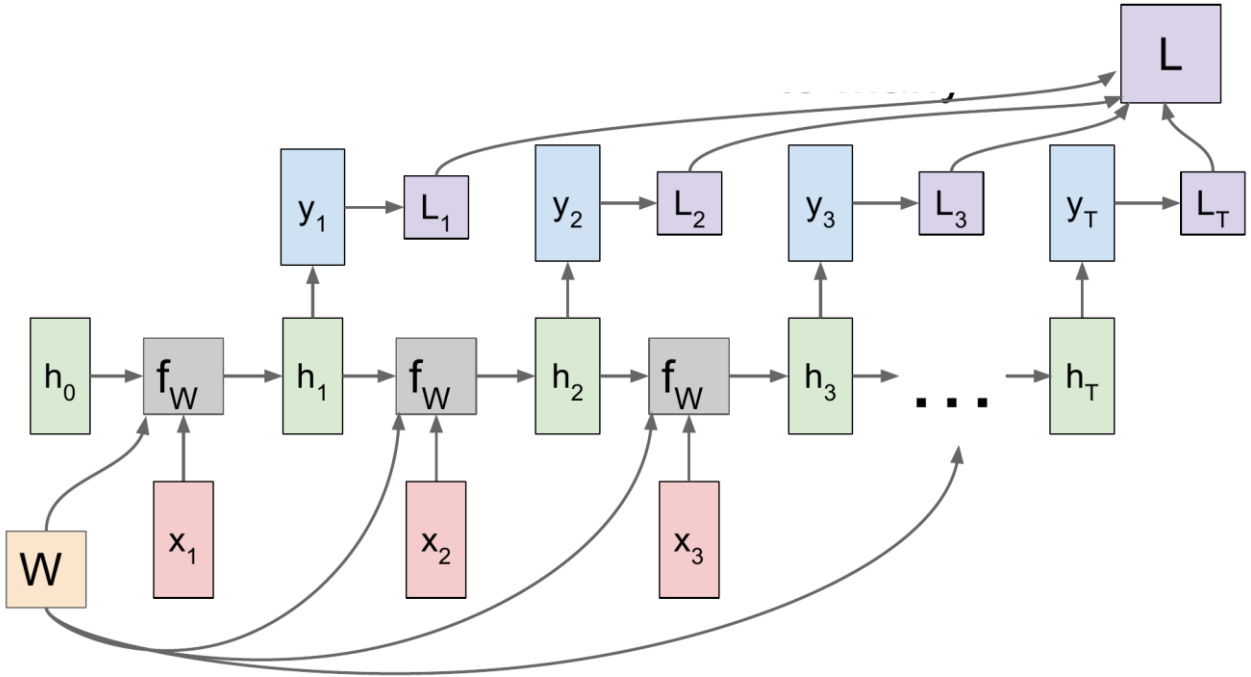
RNN Computational Graph

- Unroll the time steps to get network of depth T
- Re-use the same weight matrix at every time-step



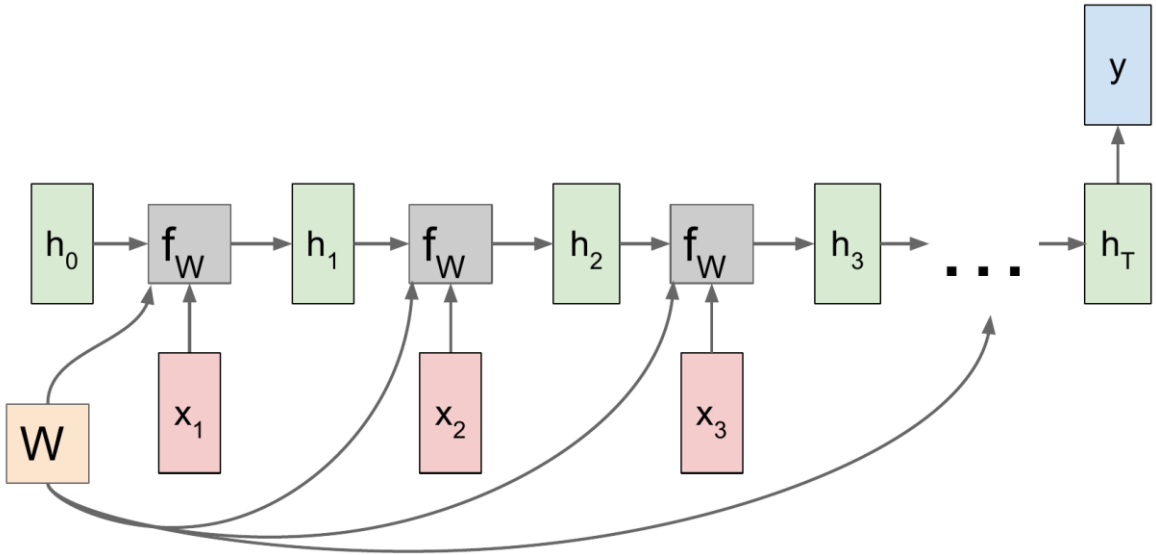
RNN Computational Graph

Many to many computation graph**:**



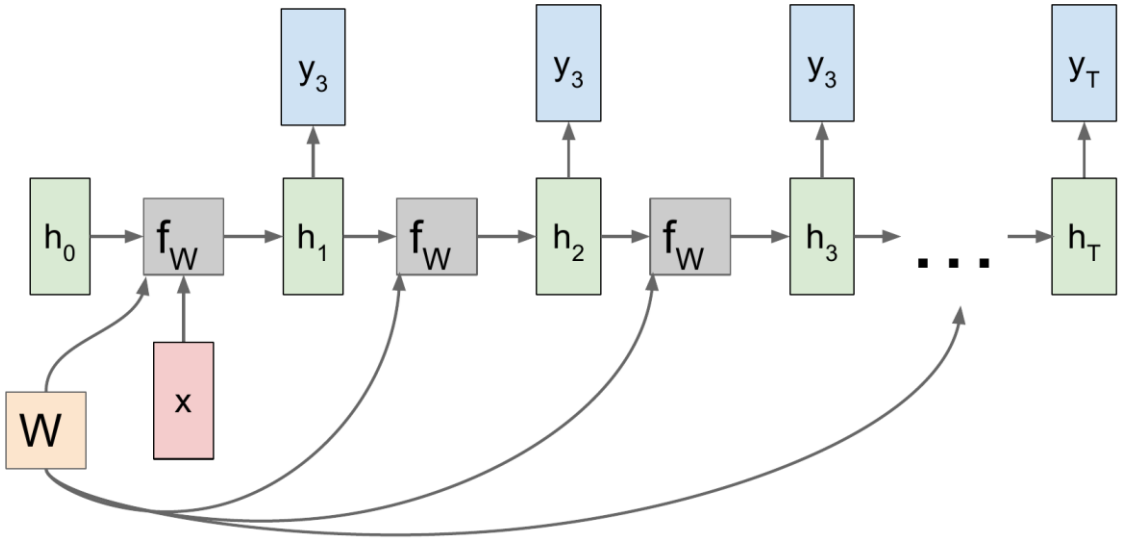
RNN Computational Graph

Many to one computation graph**:**



RNN Computational Graph

One to many computation graph**:**

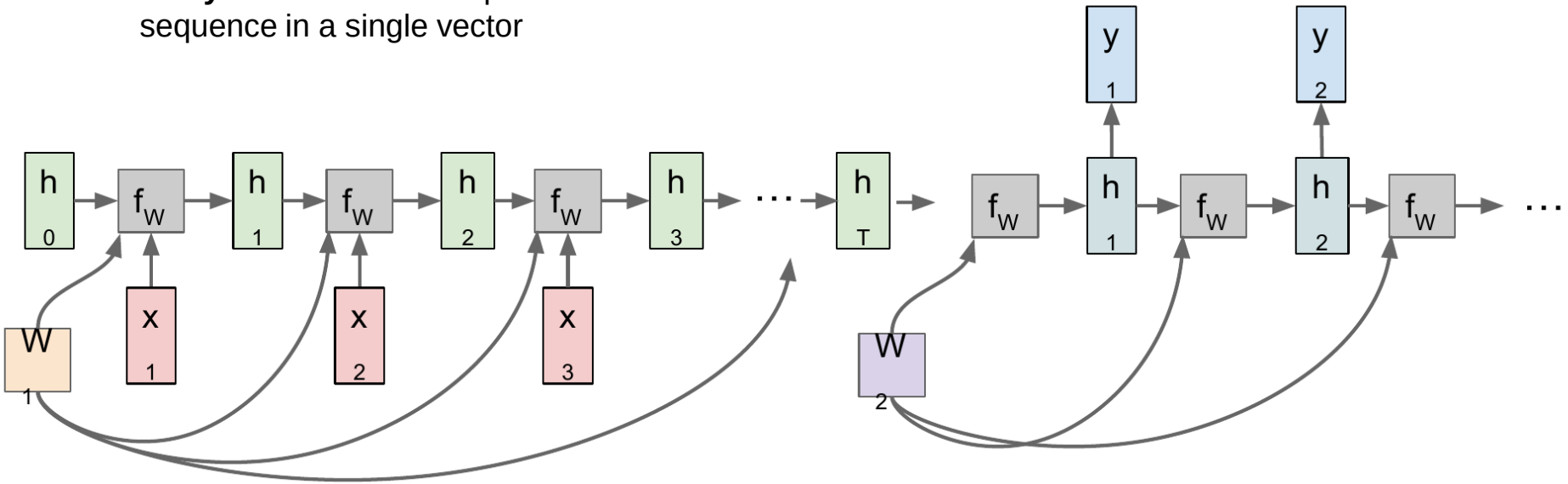


RNN Computational Graph

Sequence to Sequence: Many-to-one + One-to-many

Many to one: Encode input sequence in a single vector

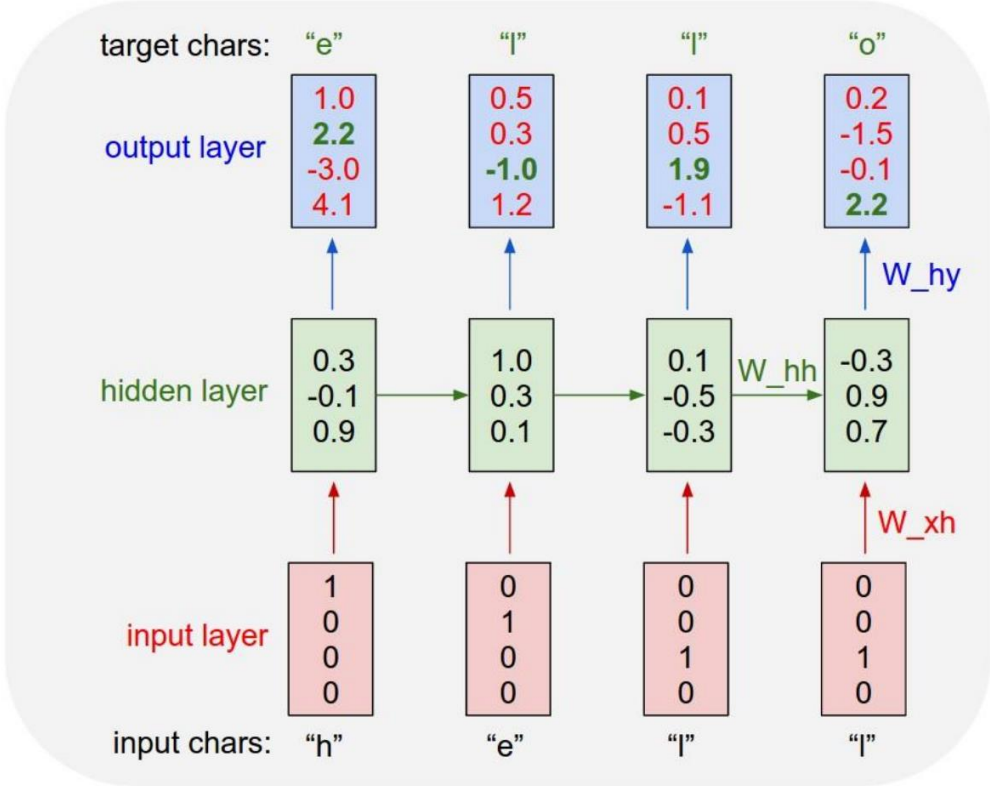
One to many: Produce output sequence from single input vector



Example: Character-level Language Model

Predict next letter:

- Vocabulary: [h,e,l,o]
- Example training sequence: “hello”

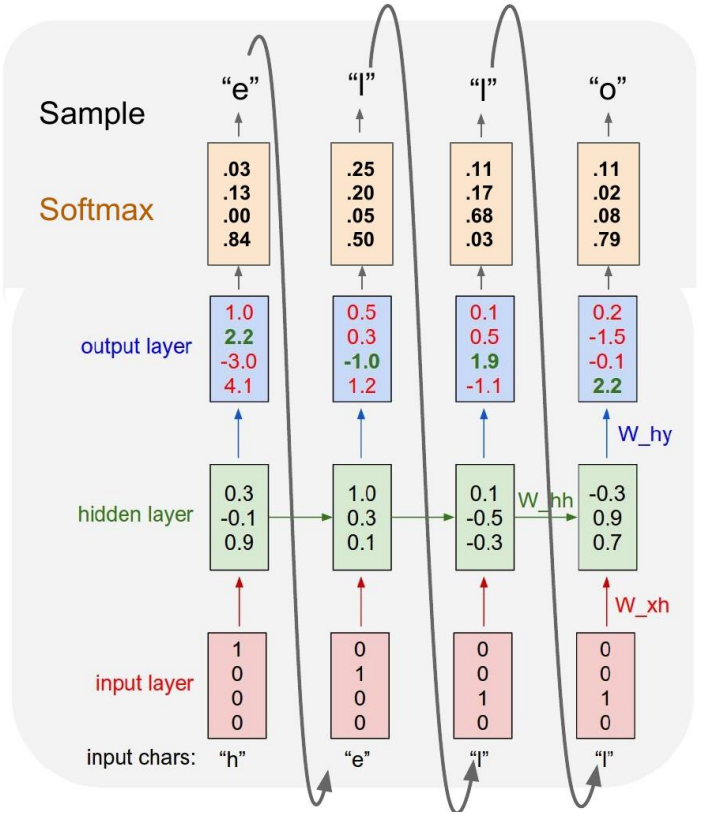


Example: Character-level Language Model

Predict next letter:

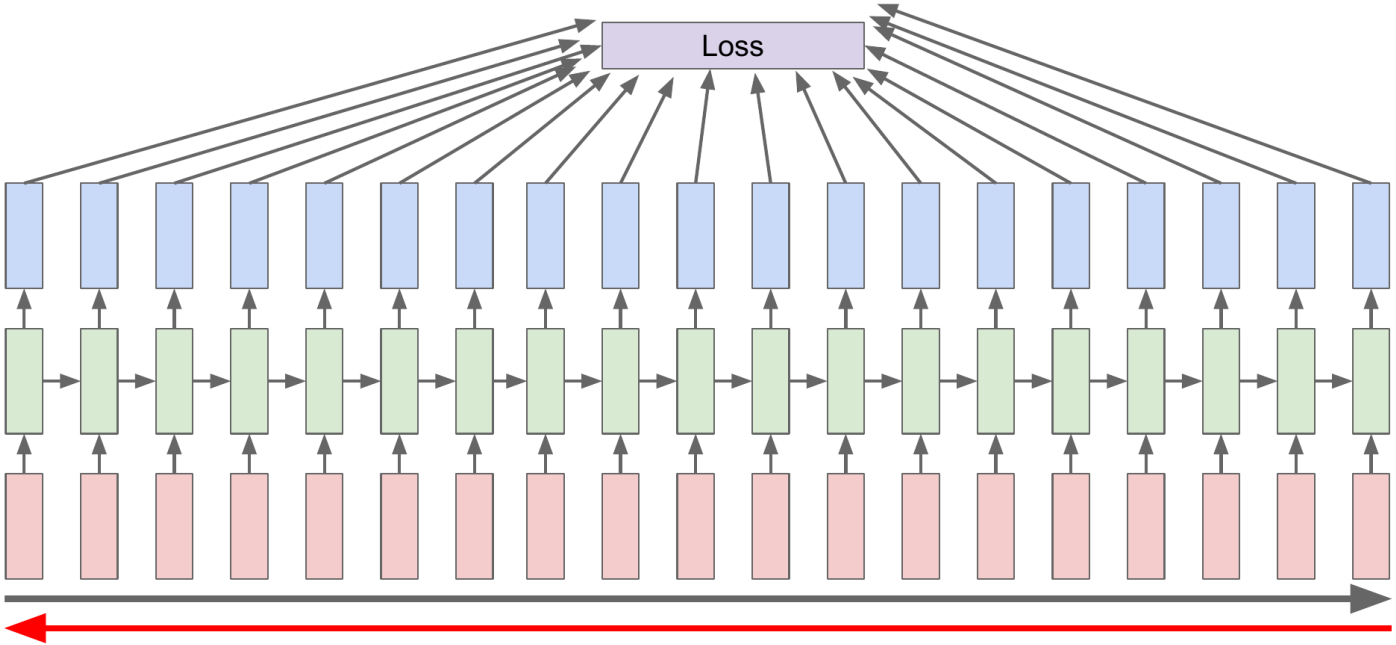
- Vocabulary: [h,e,l,o]
- Example training sequence: “hello”

At test-time **sample characters one at a time, feed back** to model

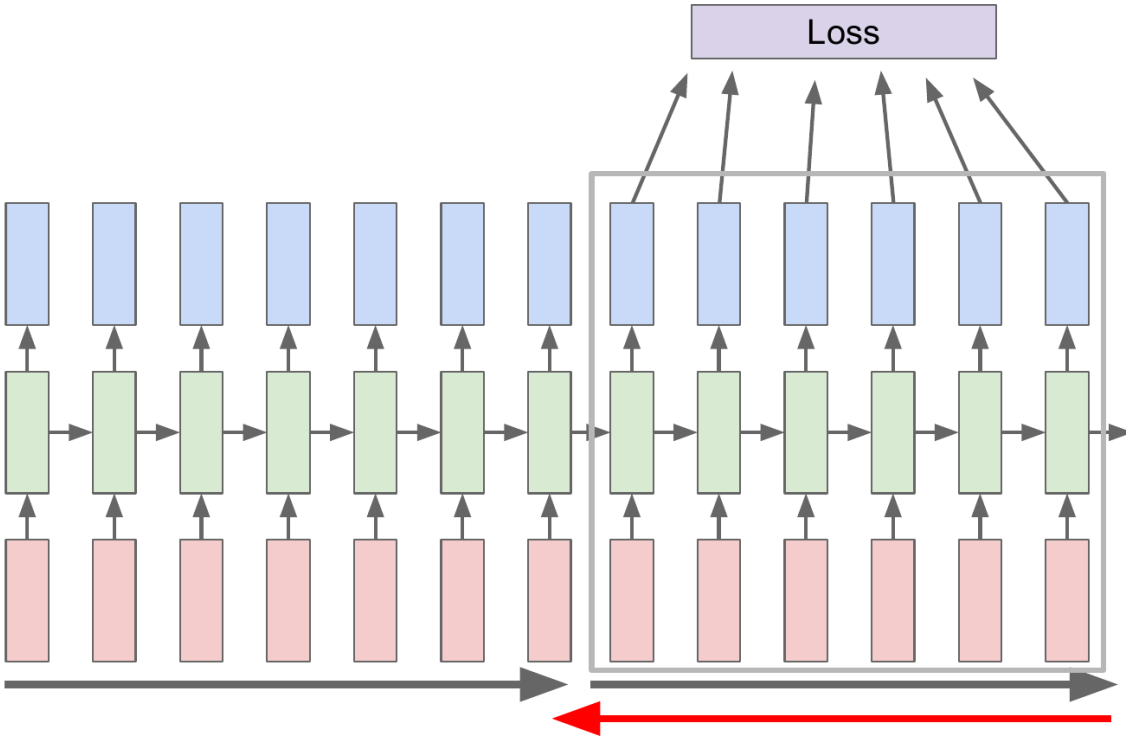


Backpropagation through time (BPTT)

- Forward through entire sequence to compute loss, then
- Backward through entire sequence to compute gradient



Truncated backpropagation through time



Carry hidden states forward in time forever, but only backpropagate for some smaller number of steps

- Computationally more efficient
- While hidden states are “more realistic”

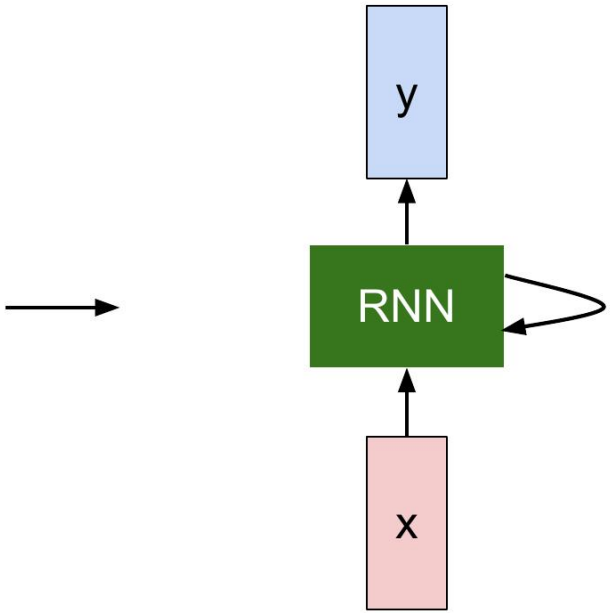
Example: Learn to be a poet

THE SONNETS

by William Shakespeare

From fairest creatures we desire increase,
That thereby beauty's rose might never die,
But as the riper should by time decease,
His tender heir might bear his memory:
But thou, contracted to thine own bright eyes,
Feed'st thy light's flame with self-substantial fuel,
Making a famine where abundance lies,
Thyself thy foe, to thy sweet self too cruel:
Thou that art now the world's fresh ornament,
And only herald to the gaudy spring,
Within thine own bud buriest thy content,
And tender churl mak'st waste in niggarding:
Pity the world, or else this glutton be,
To eat the world's due, by the grave and thee.

When forty winters shall besiege thy brow,
And dig deep trenches in thy beauty's field,
Thy youth's proud livery so gazed on now,
Will be a tatter'd weed of small worth held:
Then being asked, where all thy beauty lies,
Where all the treasure of thy lusty days;
To say, within thine own deep sunken eyes,
Were an all-eating shame, and thriftless praise.
How much more praise deserv'd thy beauty's use,
If thou couldst answer 'This fair child of mine
Shall sum my count, and make my old excuse,'
Proving his beauty by succession thine!
This were to be new made when thou art old,
And see thy blood warm when thou feel'st it cold.



Example: Learn to be a poet

at first:

```
tyntd-iafhatawiaoihrdemot  lytdws  e ,tfti, astai f ogoh eoase rrranbyne 'nhthnee e
plia tklrgd t o idoe ns,smtt   h ne etie h,hregtrs nigtike,aoaenns lng
```

train more

```
"Tmont thithey" fomesscerliund
Keushey. Thom here
sheulke, anmerenith ol sivh I lalterthend Bleipile shuwy fil on aseterlome
coaniogennc Phe lism thond hon at. MeiDimorotion in ther thize."
```

train more

```
Aftair fall unsuch that the hall for Prince Velzonski's that me of
her hearly, and behs to so arwage fiving were to it beloge, pavu say falling misfort
how, and Gogition is so overelical and ofter.
```

train more

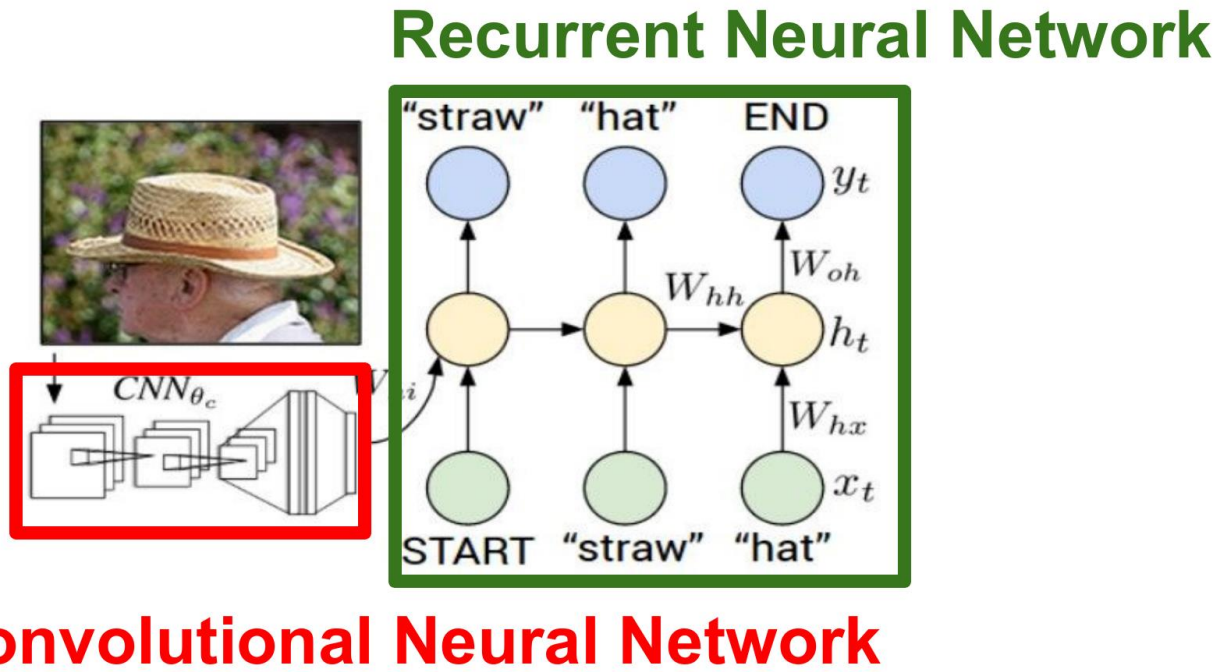
```
"Why do what that day," replied Natasha, and wishing to himself the fact the
princess, Princess Mary was easier, fed in had oftened him.
Pierre aking his soul came to the packs and drove up his father-in-law women.
```

Example: Generate C-Code

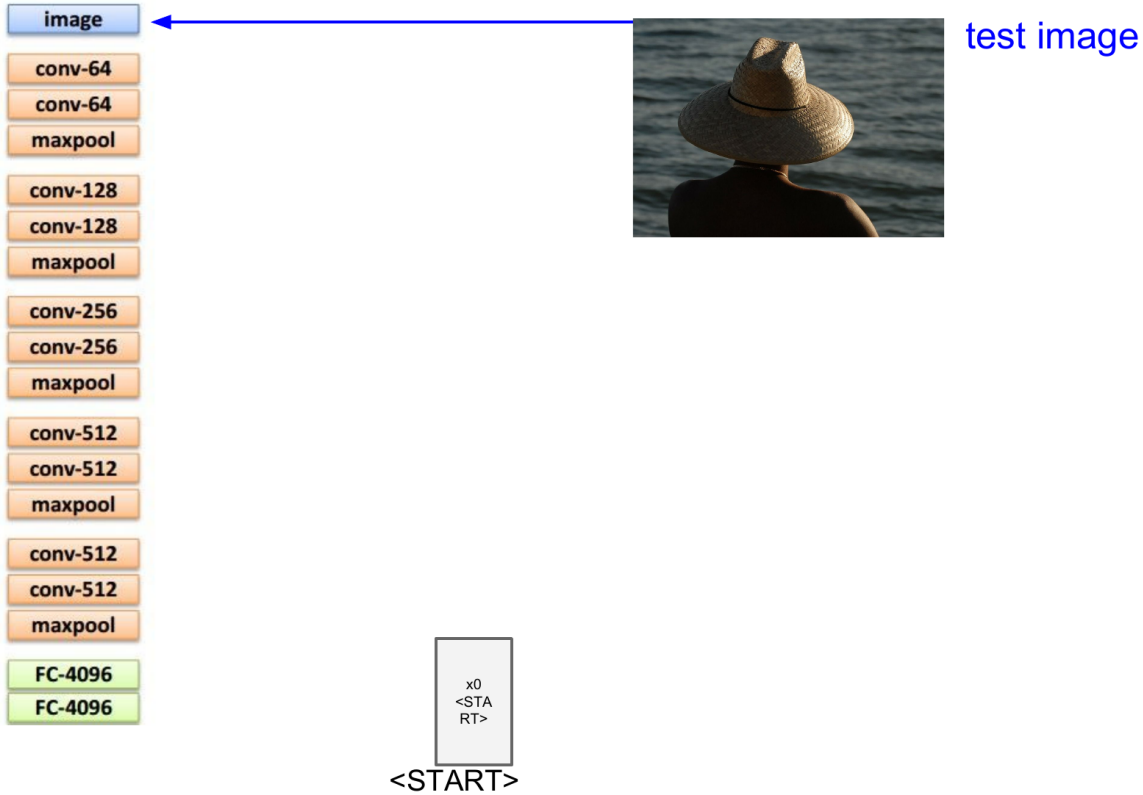
Synthized code snipped

```
static void do_command(struct seq_file *m, void *v)
{
  int column = 32 << (cmd[2] & 0x80);
  if (state)
    cmd = (int)(int_state ^ (in_8(&ch->ch_flags) & Cmd) ? 2 : 1);
  else
    seq = 1;
  for (i = 0; i < 16; i++) {
    if (k & (1 << 1))
      pipe = (in_use & UMXTHREAD_UNCCA) +
        ((count & 0x00000000ffffff8) & 0x000000f) << 8;
    if (count == 0)
      sub(pid, ppc_md.kexec_handle, 0x20000000);
    pipe_set_bytes(i, 0);
  }
  /* Free our user pages pointer to place camera if all dash */
  subsystem_info = &of_changes[PAGE_SIZE];
  rek_controls(offset, idx, &soffset);
  /* Now we want to deliberately put it to device */
  control_check_polarity(&context, val, 0);
  for (i = 0; i < COUNTER; i++)
    seq_puts(s, "policy ");
}
```

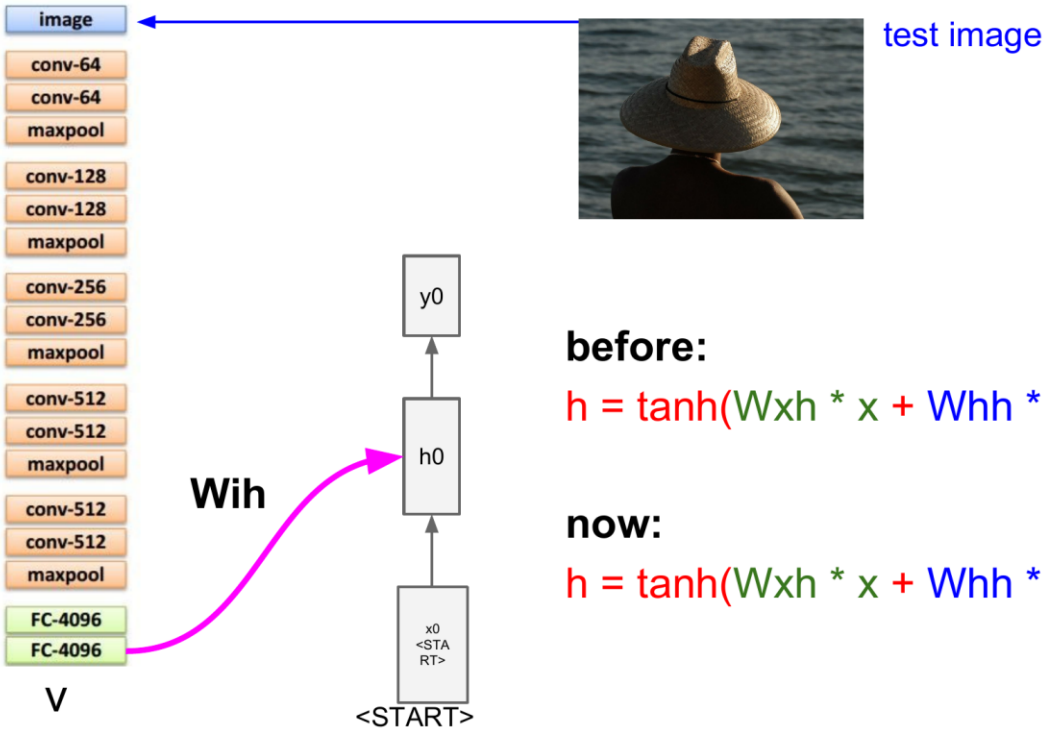
Example: Image Captioning



Example: Image Captioning



Example: Image Captioning



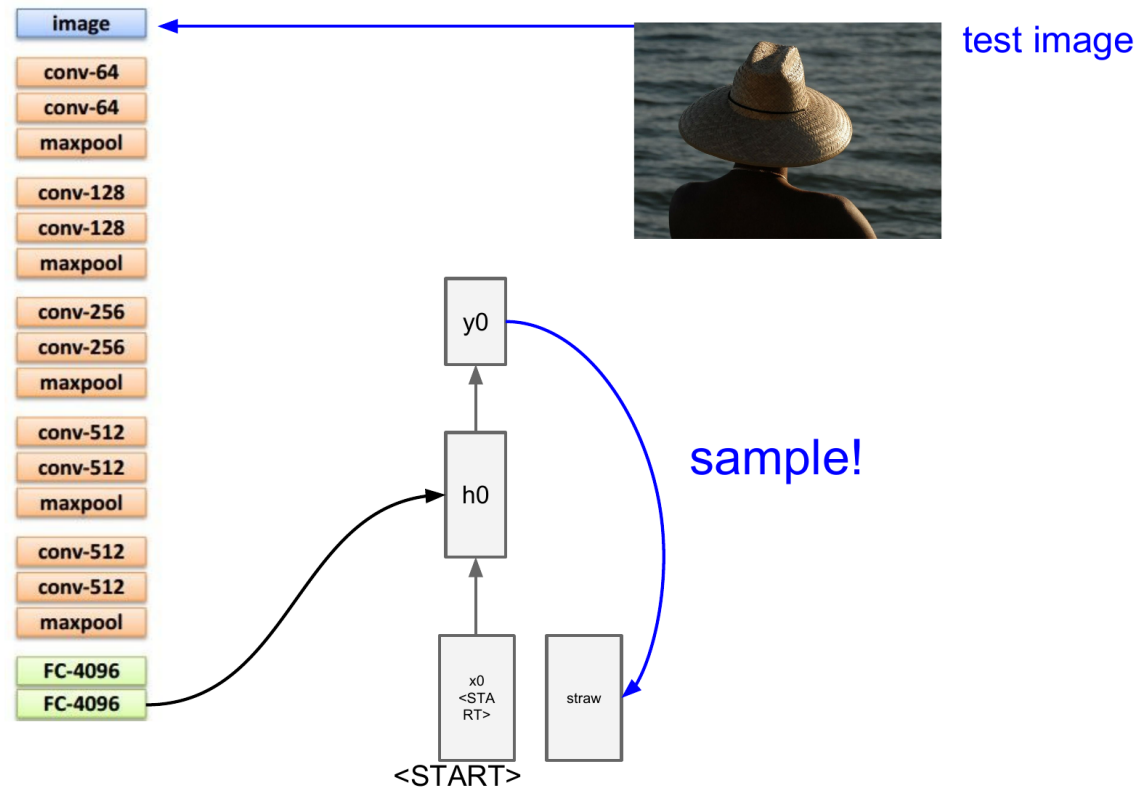
before:

h = tanh(Wxh * x + Whh * h)

now:

h = tanh(Wxh * x + Whh * h + **Wih * v**)

Example: Image Captioning



Example: Image Captioning

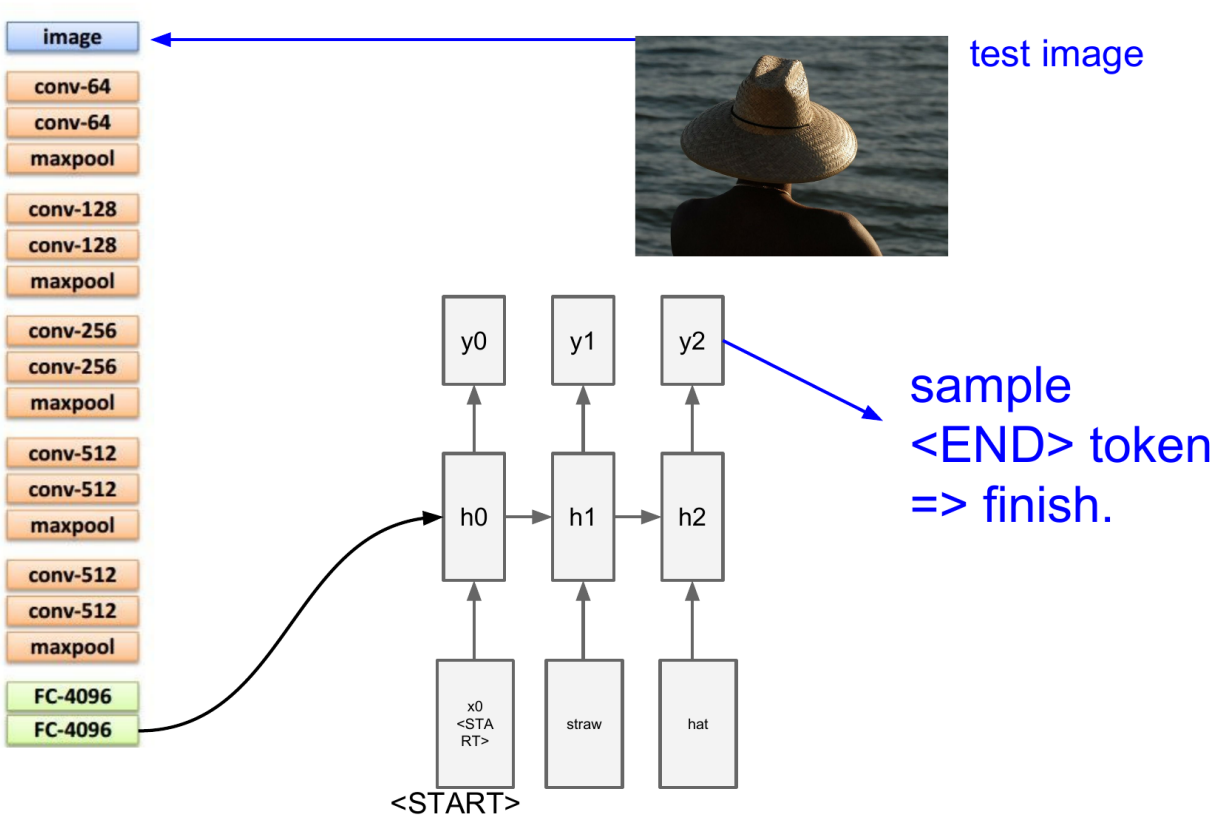


Image Captioning: Results

A cat sitting on a suitcase on the floor

A cat is sitting on a tree branch

A dog is running in the grass with a frisbee

A white teddy bear sitting in the grass

Two people walking on the beach with surfboards

A tennis player in action on the court

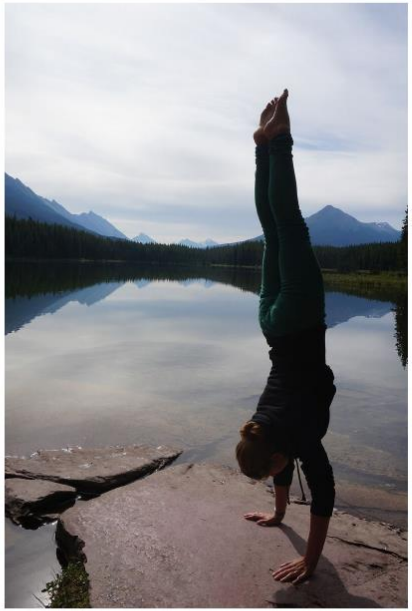
Two giraffes standing in a grassy field

A man riding a dirt bike on a dirt track

Image Captioning: Results

A woman is holding a cat in her hand

A person holding a computer mouse on a desk



A woman standing on a beach holding a surfboard

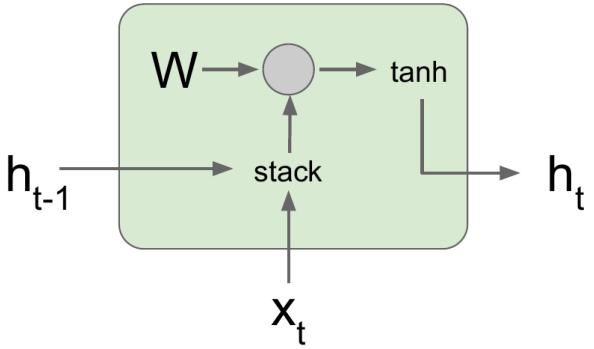
A bird is perched on a tree branch

A man in a baseball uniform throwing a ball

Gradient Flow in Vanilla RNNs

Vanilla RNN unit:

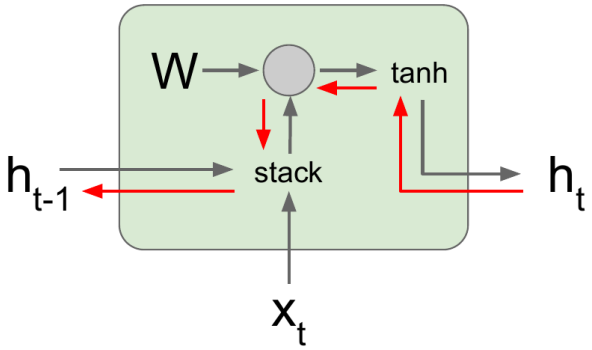
$$\begin{aligned} \boldsymbol{h}_t &= \tanh(\boldsymbol{W}_{hh}\boldsymbol{h}_{t-1} + \boldsymbol{W}_{xh}\boldsymbol{x}_t) \\ &= \tanh\left(\boldsymbol{W}\begin{bmatrix}\boldsymbol{h}_{t-1} \\ \boldsymbol{x}_t\end{bmatrix}\right) \end{aligned}$$



- Gradient flow backwards in time

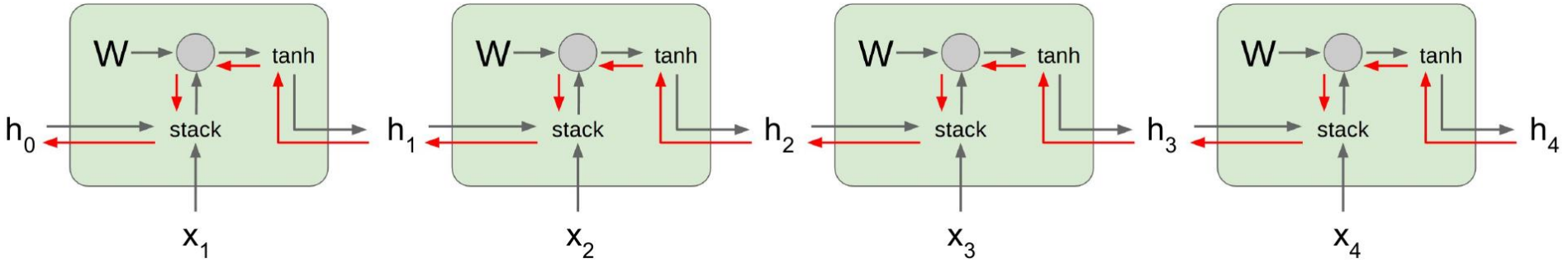
$$\frac{\partial \boldsymbol{h}_t}{\partial \boldsymbol{h}_{t-1}} = \text{diag}\left(\tanh'\left(\boldsymbol{W}\begin{bmatrix}\boldsymbol{h}_{t-1} \\ \boldsymbol{x}_t\end{bmatrix}\right)\right)\boldsymbol{W}_{hh}^T$$

Backpropagation from h_t to h_{t-1} multiplies by W (actually W_{hh}^T)



Bengio et al, "Learning long-term dependencies with gradient descent is difficult", IEEE Transactions on Neural Networks, 1994
Pascanu et al, "On the difficulty of training recurrent neural networks", ICML 2013

Gradient Flow in Vanilla RNNs



Computing gradient of h involves many factors of W (and repeated tanh)

- Largest singular value > 1: Exploding gradients
  - **Gradient Clipping:** Scale gradients if the norm is too big
- Largest singular value < 1: Vanishing gradients
  - We need a **different RNN** architecture!

Long-term short-term memory (LSTM)

Contribution of old memory state and current input is gated

- **f: Forget gate**,
  - whether to erase cell

$$\boldsymbol{f}_t = \sigma\left(\boldsymbol{W}_f \begin{bmatrix} \boldsymbol{h}_{t-1} \\ \boldsymbol{x}_t \end{bmatrix}\right)$$

- **i: Input gate**,
  - whether to write to cell

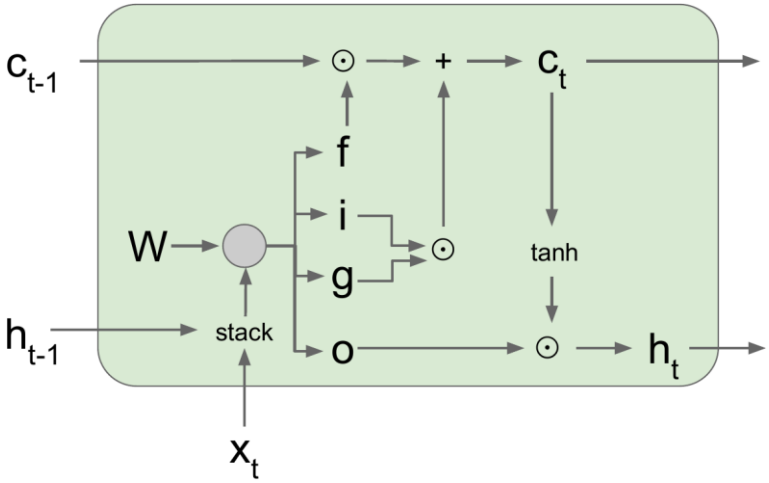
$$\boldsymbol{i}_t = \sigma\left(\boldsymbol{W}_i \begin{bmatrix} \boldsymbol{h}_{t-1} \\ \boldsymbol{x}_t \end{bmatrix}\right)$$

- **g: Gate gate** (?)
  - How much to write to cell

$$\boldsymbol{g}_t = \tanh\left(\boldsymbol{W}_g \begin{bmatrix} \boldsymbol{h}_{t-1} \\ \boldsymbol{x}_t \end{bmatrix}\right)$$

- **o: Output gate**
  - How much to reveal cell

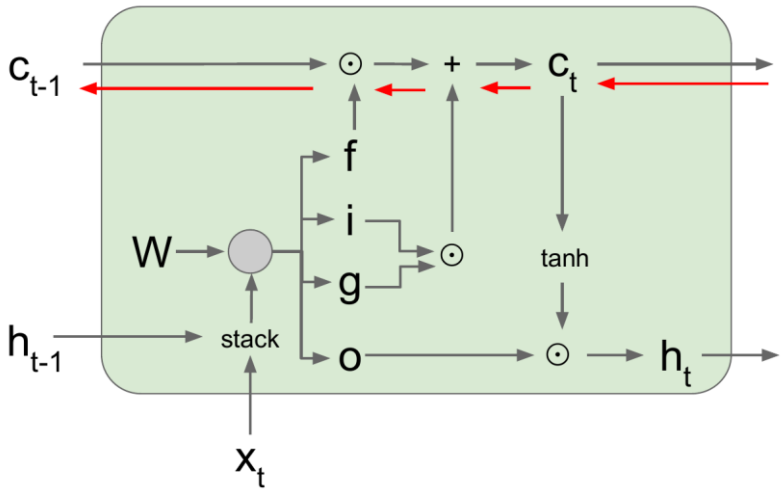
$$\boldsymbol{o}_t = \sigma\left(\boldsymbol{W}_o \begin{bmatrix} \boldsymbol{h}_{t-1} \\ \boldsymbol{x}_t \end{bmatrix}\right)$$



Next memory state:

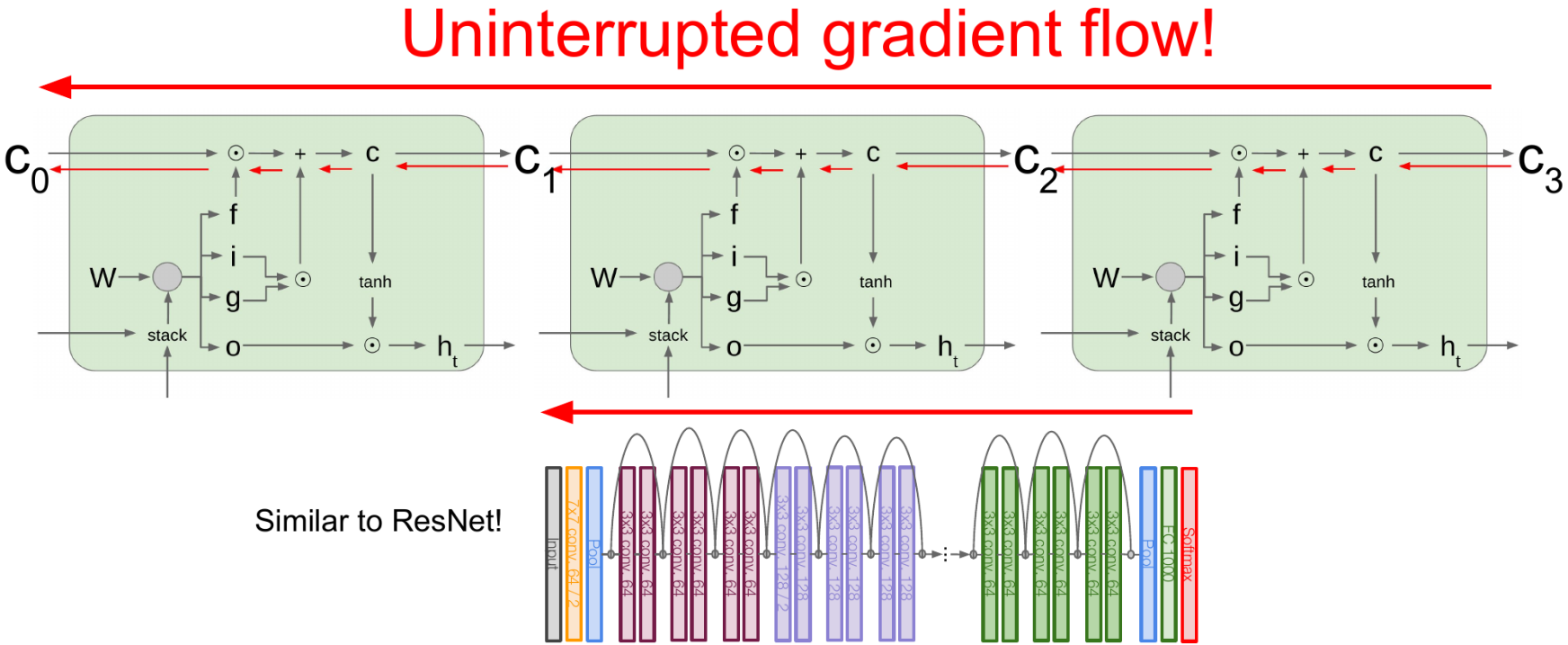
$$\boldsymbol{c}_t = \boldsymbol{f}_t \circ \boldsymbol{c}_{t-1} + \boldsymbol{i}_t \circ \boldsymbol{g}_t, \quad \boldsymbol{h}_t = \boldsymbol{o}_t \circ \tanh(\boldsymbol{c}_t)$$

LSTM: Gradient Flow



Backpropagation from c_t to c_{t-1} only elementwise multiplication by f, no matrix multiply by W

LSTM: Gradient Flow



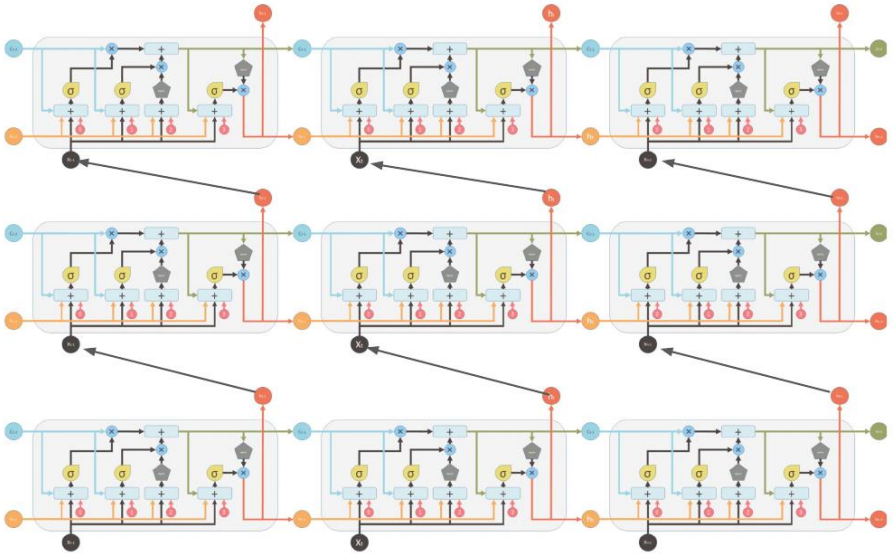
Deep LSTMs

Stacking multiple LSTM layers vertically

- Output sequence of one layer forming the input sequence of the next
  - in addition to recurrent connections within the same layer
- Increases the number of parameters - but given sufficient data, performs significantly better than single-layer LSTMs (Graves et al. 2013)
- Dropout usually applied only to non-recurrent edges, including between layers

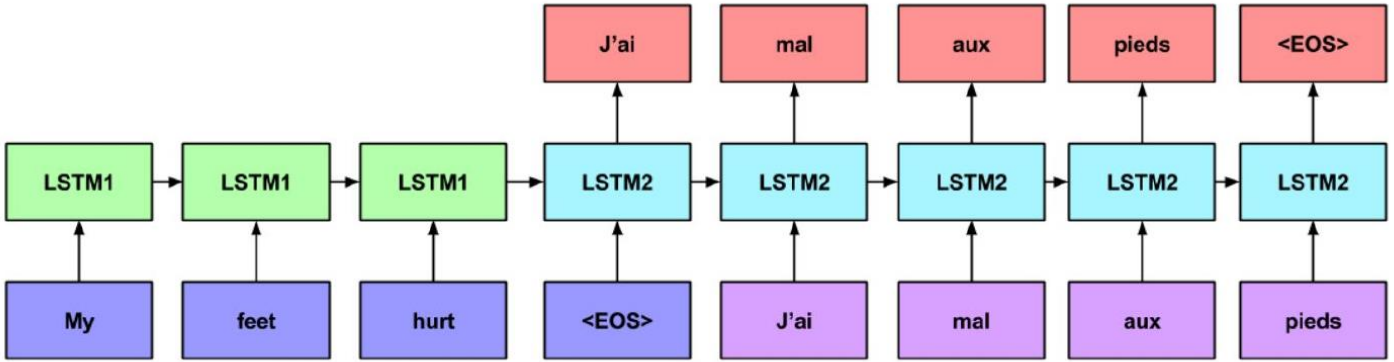
Other alternatives:

- Encoder architecture, encode input x with deep NN



LSTM for Machine Translation

Sutskever et al. 2014:



State of the art now: Transformer networks (no recurrency but attention)

Demos

- **Handwriting generation demo:**
  - http://www.cs.toronto.edu/~graves/handwriting.html
- **Music composition:**
  - http://www.hexahedria.com/2015/08/03/composing-music-with-recurrent-neural-networks/
- **Image captioning and other stuff:**
  - http://karpathy.github.io/2015/05/21/rnn-effectiveness/
- **And many more...**
  - https://www.dlology.com/blog/top-10-deep-learning-experiences-run-on-your-browser/

Other RNN Variants

Gated Recurrent Units (GRU):

$$
\begin{aligned}
r_t &= \sigma(W_{xr} x_t + W_{hr} h_{t-1} + b_r) \\
z_t &= \sigma(W_{xz} x_t + W_{hz} h_{t-1} + b_z) \\
\tilde{h}_t &= \tanh(W_{xh} x_t + W_{hh}(r_t \odot h_{t-1}) + b_h) \\
h_t &= z_t \odot h_{t-1} + (1 - z_t) \odot \tilde{h}_t
\end{aligned}
$$

- No explicit forget gate
- Less parameters
- Often similar performance

[Learning phrase representations using rnn encoder-decoder for statistical machine translation, Cho et al. 2014]

Wrap-Up: RNNs

- RNNs allow a lot of flexibility in architecture design
- Vanilla RNNs are simple but don’t work very well
- Common to use LSTM or GRU: their additive interactions improve gradient flow
- Backward flow of gradients in RNN can explode or vanish.
- Exploding is controlled with gradient clipping. Vanishing is controlled with additive interactions (LSTM)
- Better/simpler architectures are a hot topic of current research
- Better understanding (both theoretical and empirical) is needed.