

# CONVOLUTIONAL AND RECURRENT NEURAL NETWORKS

---

# Neural networks

- Fully connected networks
  - Neuron
  - Non-linearity
  - Softmax layer
- DNN training
  - Loss function and regularization
  - SGD and backprop
  - Learning rate
  - Overfitting – dropout, batchnorm
- CNN, RNN, LSTM, GRU <- This class

# Dropout

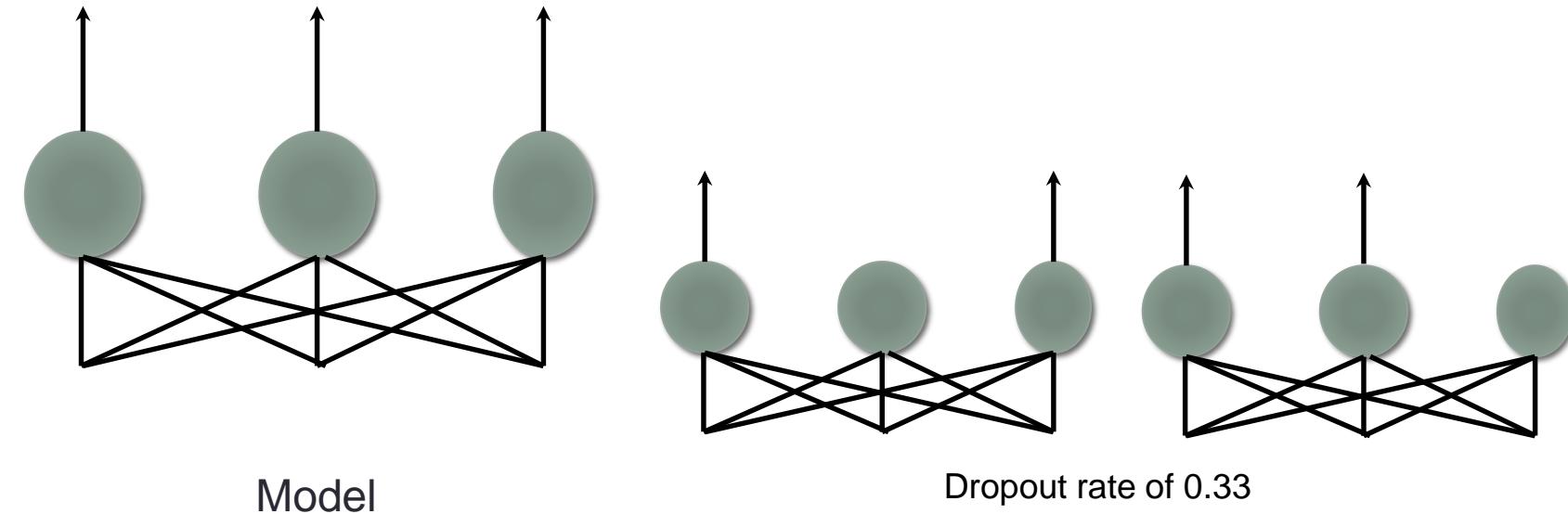
An **implicit regularization** technique for reducing overfitting

Randomly turn off different subset of neurons during training

Network no longer depend on any particular neuron

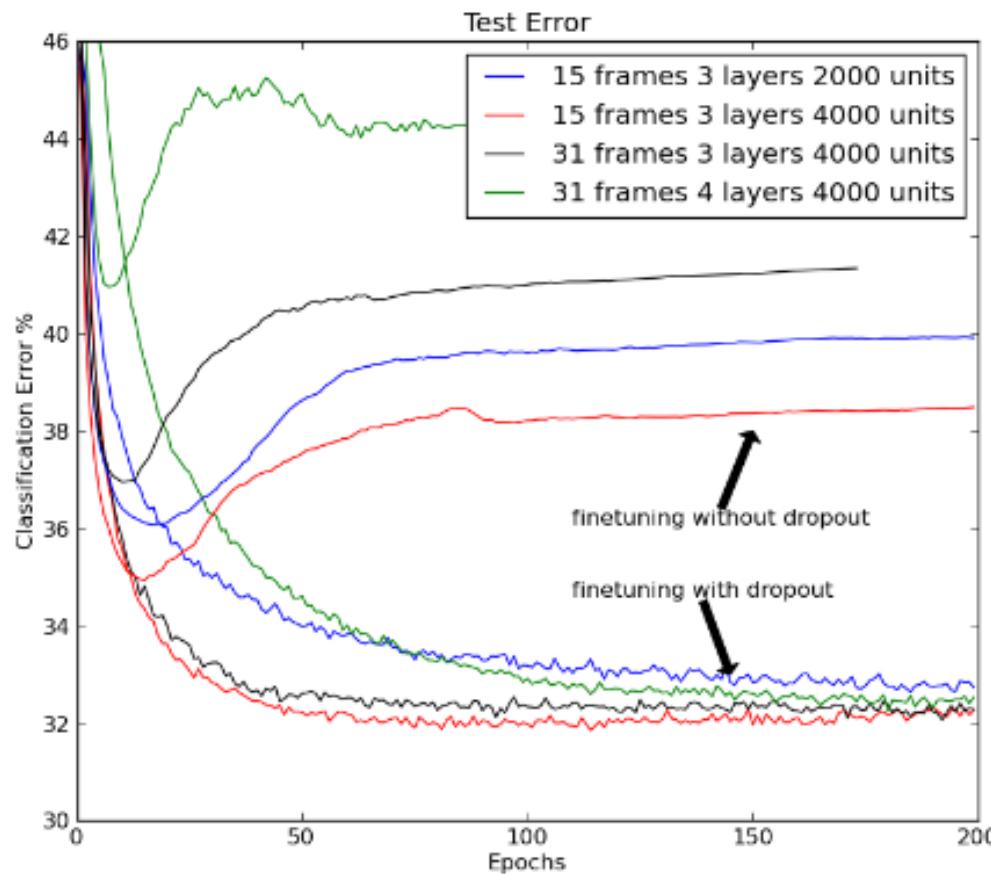
Force the model to have redundancy – robust to any corruption in input data

A form of performing model averaging (ensemble of experts)



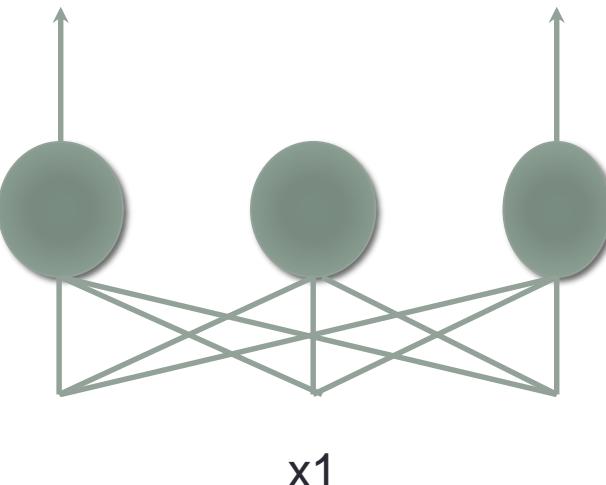
# Dropout on TIMIT

- A phoneme recognition task

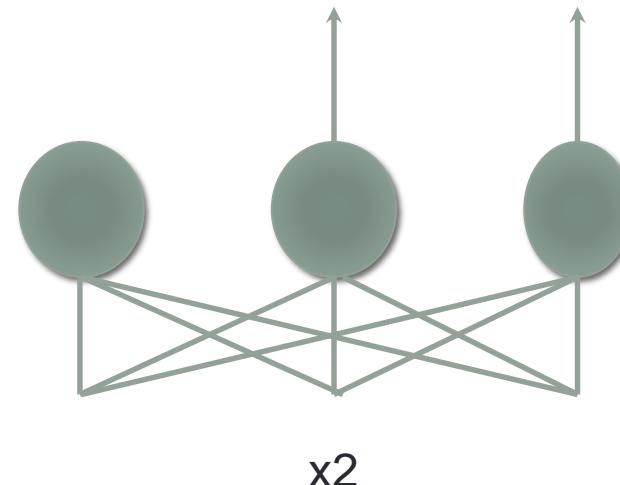


# Dropout training time

Drop out rate 0.33

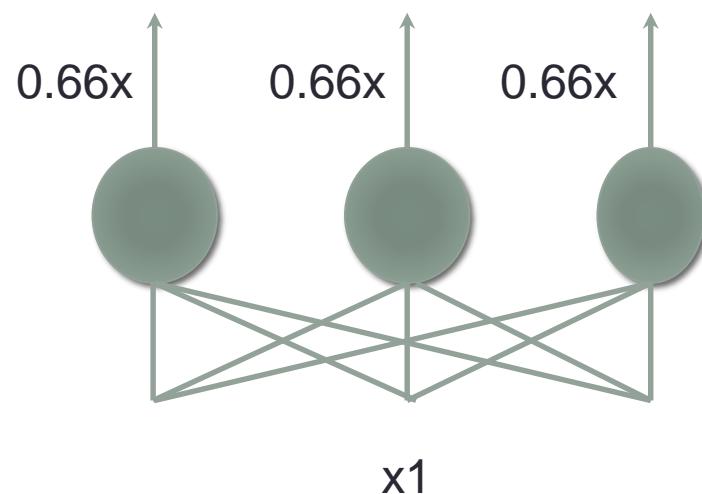


Randomly removed outputs  
for each training sample

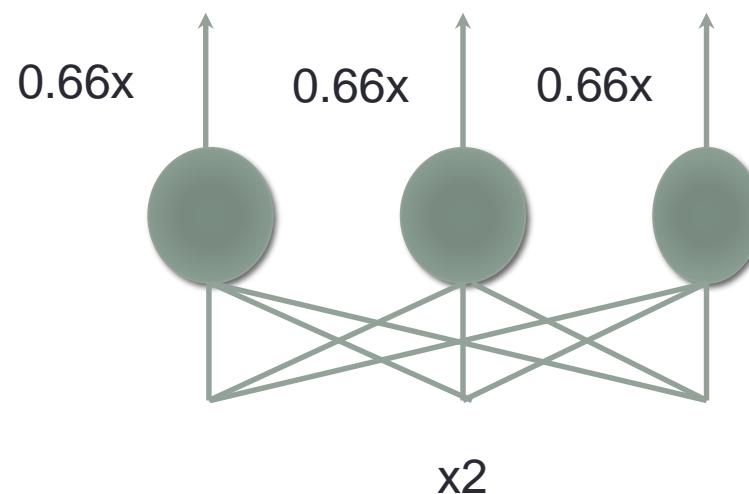


# Dropout test time

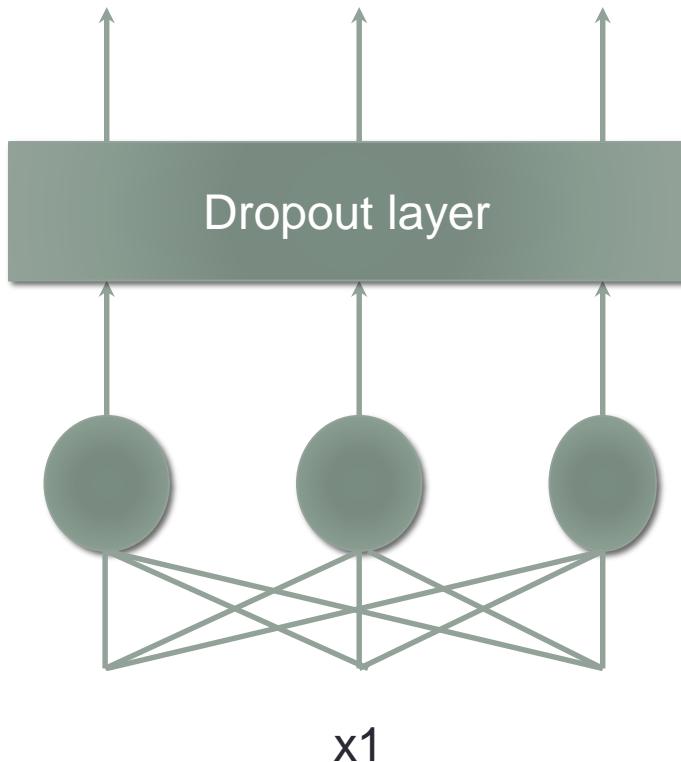
Dropout rate 0.33



Scale outputs so the output contribution is around the same



# Dropout implementation



## Dropout layer

Just another layer that drops inputs

## Inverted dropout

A variant of dropout that scales the dropout at training time, so that you don't have to scale at test time.

# Convolution Neural Networks

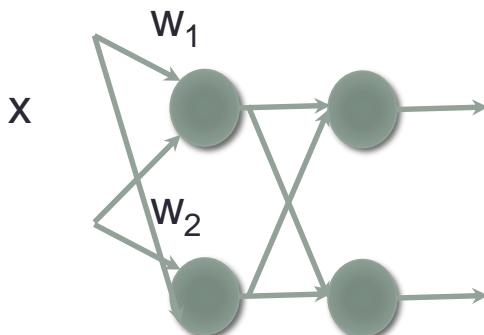
Convolution

Convolution Layers and Pooling Layers

CNN progress

# PCA as a filter

- $W^T x$ 
  - What happens if I have a person that is off-frame?
  - Need another filter that is shifted



$w_1$	$x = y$
$w_2$	

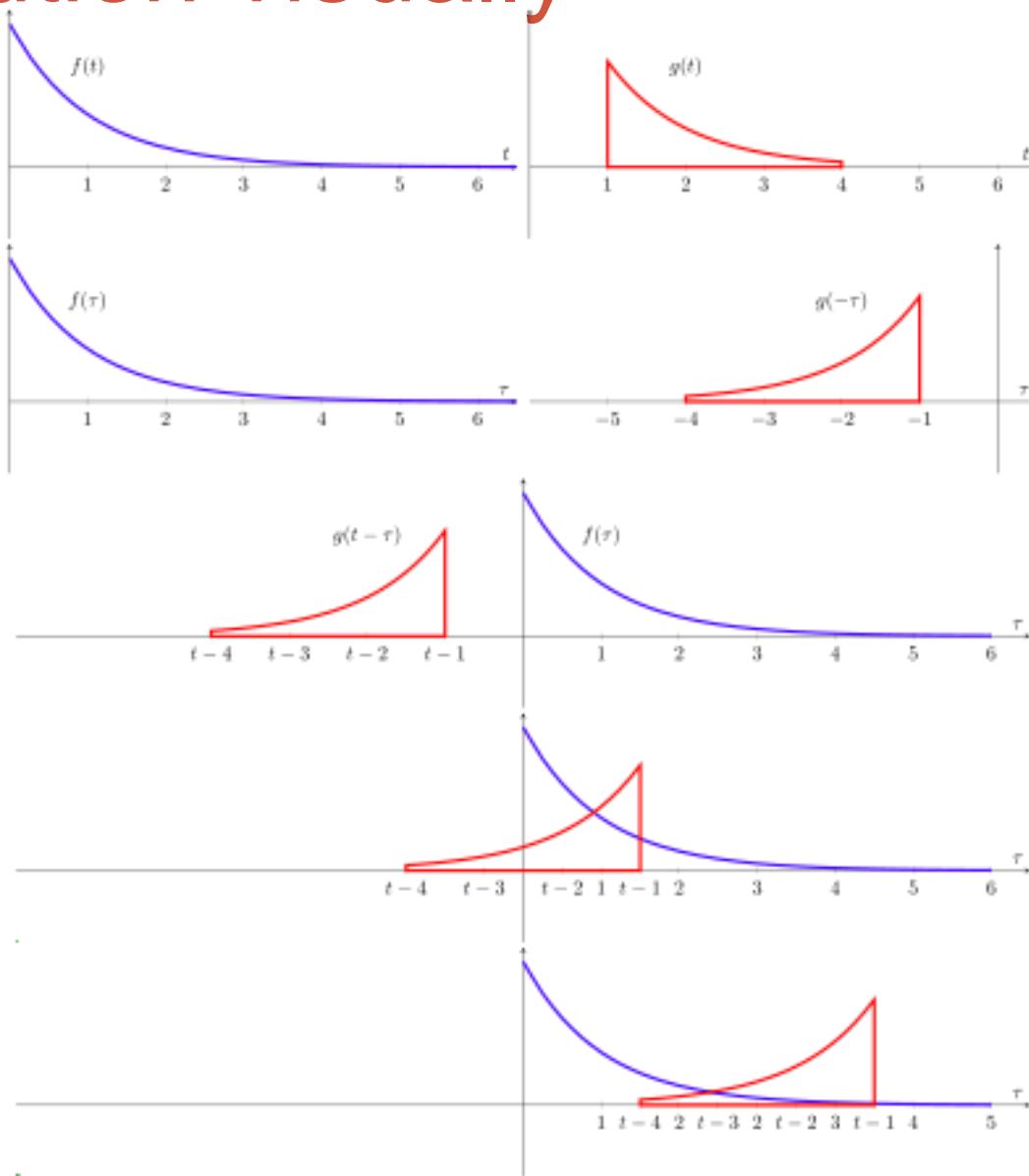
# Convolution

- Continuous convolution

$$\begin{aligned}(f * g)(t) &\stackrel{\text{def}}{=} \int_{-\infty}^{\infty} f(\tau)g(t - \tau) d\tau \\&= \int_{-\infty}^{\infty} f(t - \tau)g(\tau) d\tau.\end{aligned}$$

- Meaning of  $(t-T)$  : Flip then shift

# Convolution visually



# Convolution discrete

- Discrete convolution

$$\begin{aligned}(f * g)[n] &= \sum_{m=-\infty}^{\infty} f[m]g[n-m] \\ &= \sum_{m=-\infty}^{\infty} f[n-m]g[m]\end{aligned}$$

- Continuous convolution

$$\begin{aligned}(f * g)(t) &\stackrel{\text{def}}{=} \int_{-\infty}^{\infty} f(\tau)g(t - \tau) d\tau \\ &= \int_{-\infty}^{\infty} f(t - \tau)g(\tau) d\tau.\end{aligned}$$

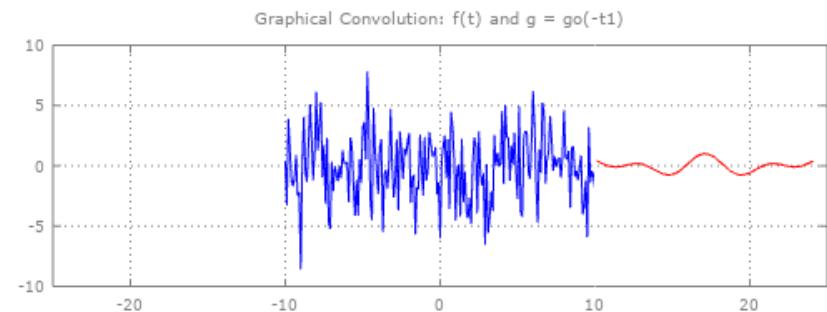
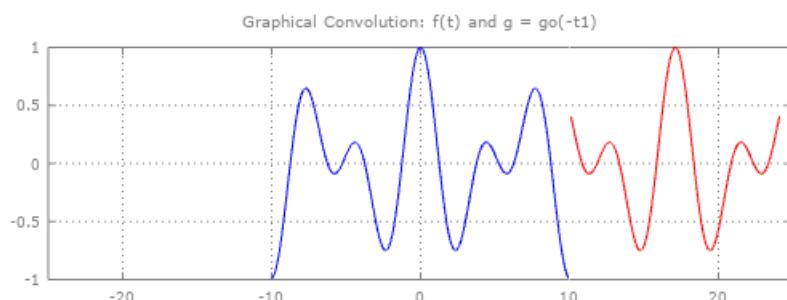
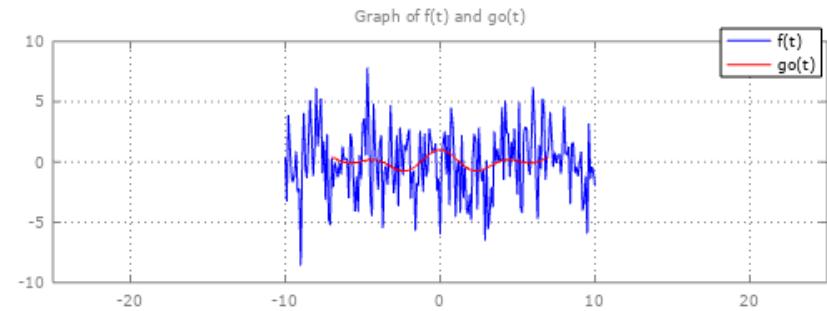
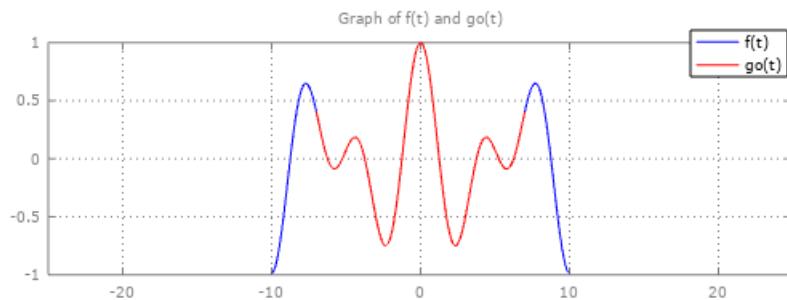
- Same concept as continuous version

# Matched filters

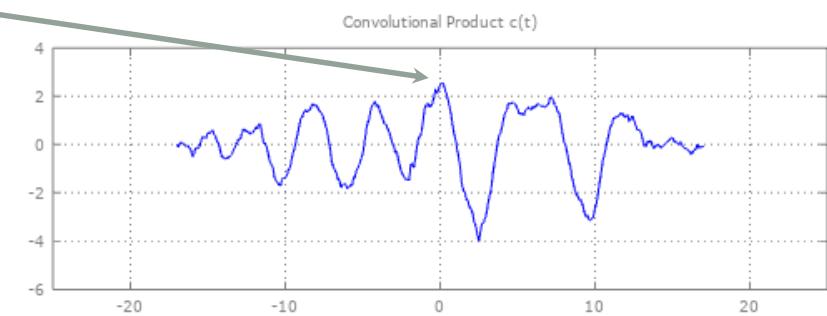
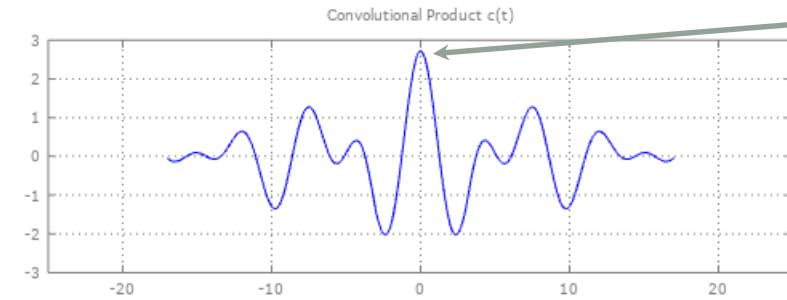
- We can use convolution to detect things that **match** our pattern
- Convolution can be considered as a **filter**
- If the filter detects our pattern, it will show up as a nice peak even if there are noise.

# Matched filters

Red: matched filter  
Blue: signal



Matched peak



# Convolution and Cross-Correlation

- Convolution

$$\begin{aligned}(f * g)(t) &\stackrel{\text{def}}{=} \int_{-\infty}^{\infty} f(\tau)g(t - \tau) d\tau \\ &= \int_{-\infty}^{\infty} f(t - \tau)g(\tau) d\tau.\end{aligned}$$

$$\begin{aligned}(f * g)[n] &= \sum_{m=-\infty}^{\infty} f[m]g[n - m] \\ &= \sum_{m=-\infty}^{\infty} f[n - m]g[m]\end{aligned}$$

- (Cross)-Correlation

$$(f \star g)(\tau) \stackrel{\text{def}}{=} \int_{-\infty}^{\infty} f^*(t) g(t + \tau) dt,$$

$$(f \star g)[n] \stackrel{\text{def}}{=} \sum_{m=-\infty}^{\infty} f^*[m] g[m + n].$$

Convolution and cross-correlation are the same if  $g(t)$  is symmetric (even function). Using convolution or cross-correlation yield the same results, if the filters are learned. However, cross-correlation is simpler to code.  
In image processing, people used convolution then switched to cross-correlation, but kept the same name.

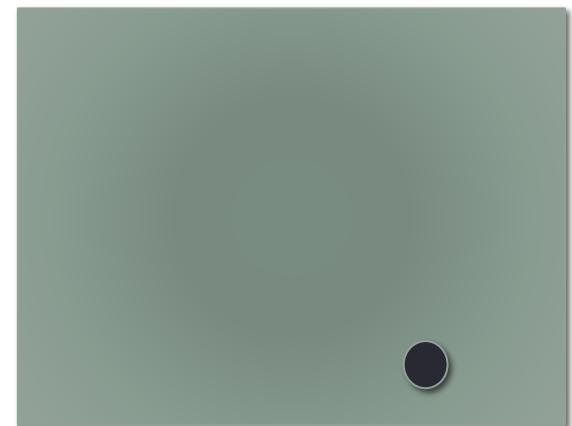
# 2D convolution

- Flip and shifts in 2D



- But, we no longer flips

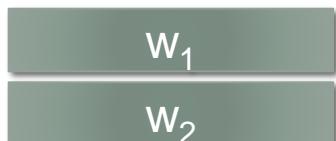
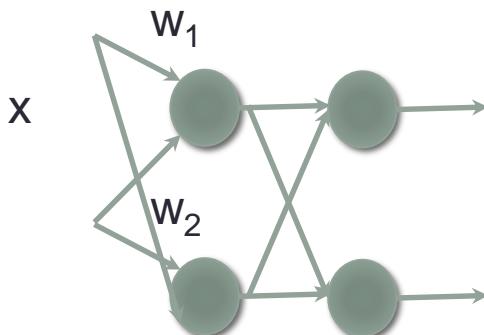
Our match filter



Will get some peak here

# PCA as a filter

- $W^T x$ 
  - What happens if I have a person that is off-frame?
  - Need another filter that is shifted
  - **Ans: Convolution with  $W$  as filter**

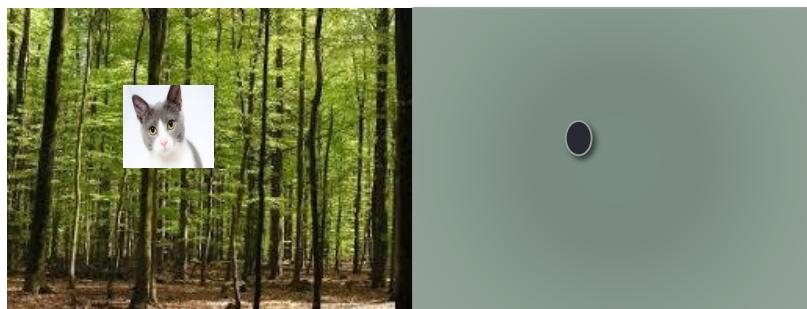


# Convolutional Neural networks

- A neural network with convolutions! (cross-correlation to be precise)
- But we have peaks at different location

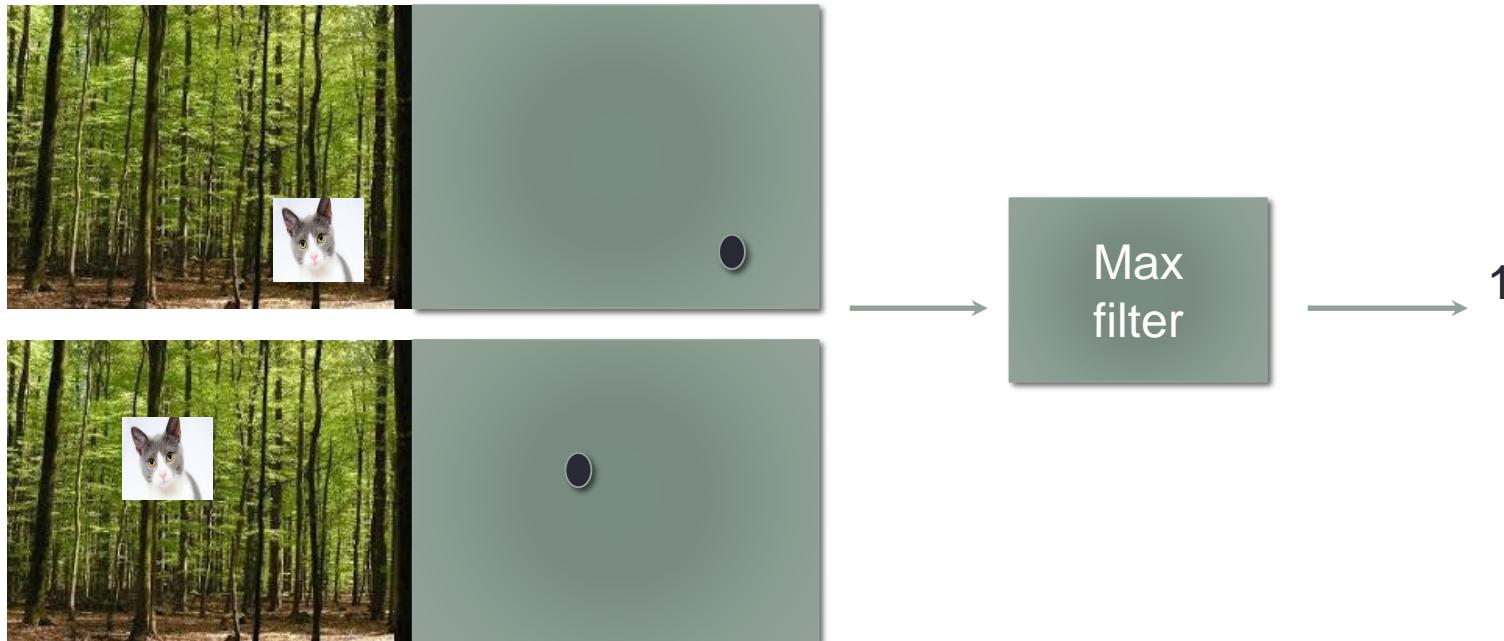


From the point of view of a network,  
these are two different things.

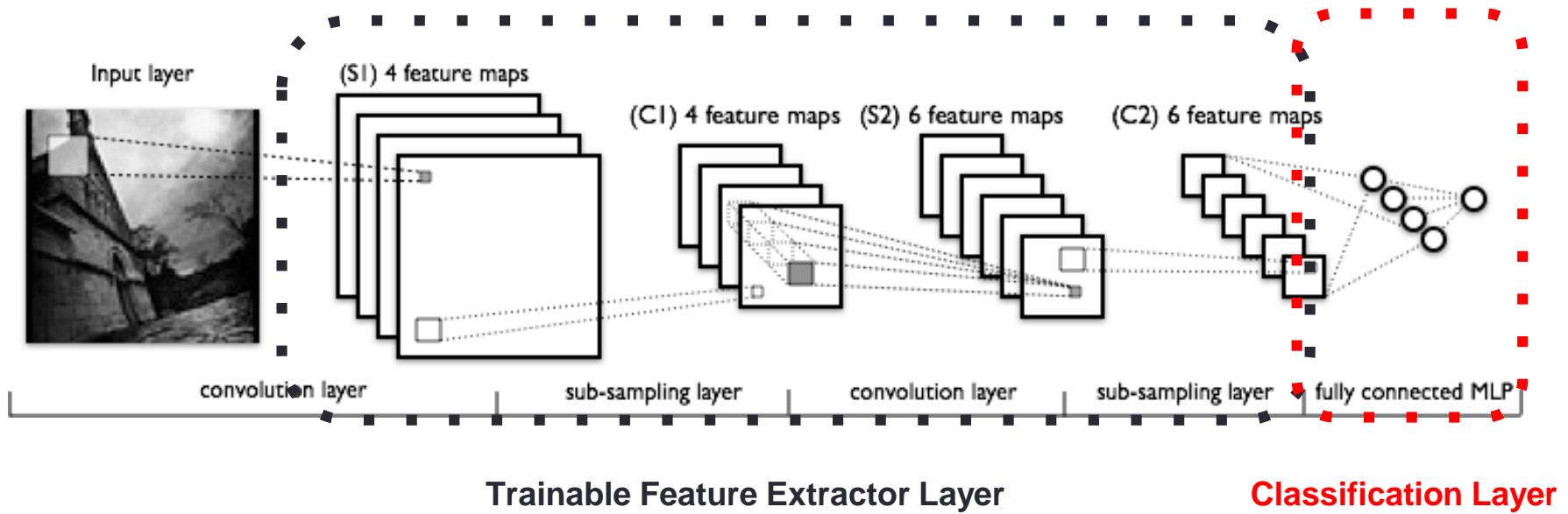


# Pooling layers/Subsampling layers

- Combine different locations into one
- One possible method is to use a max
  - Interpretation: Yes, I found a cat somewhere



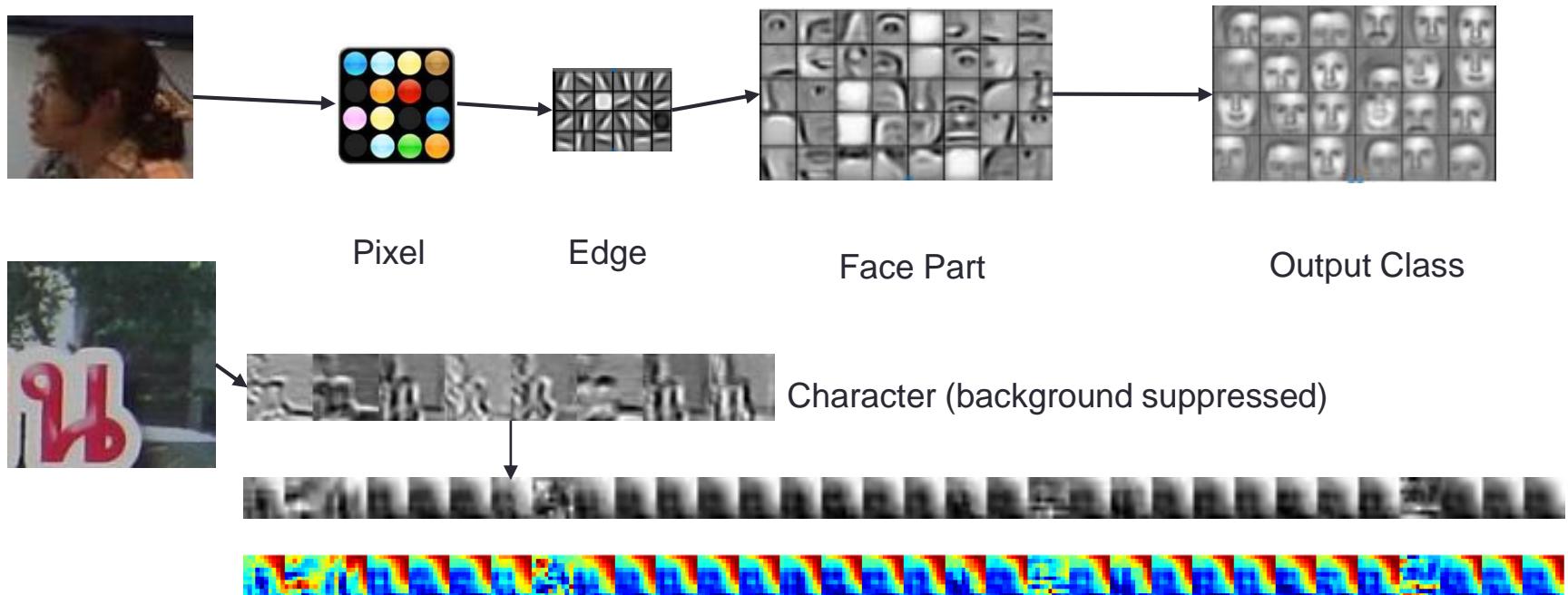
# Convolution Neural Network (CNN)



Convolution layers followed by sub-sampling (pooling) layers  
Output of each layer is called a feature map.

# Convolution Neural Network (CNN)

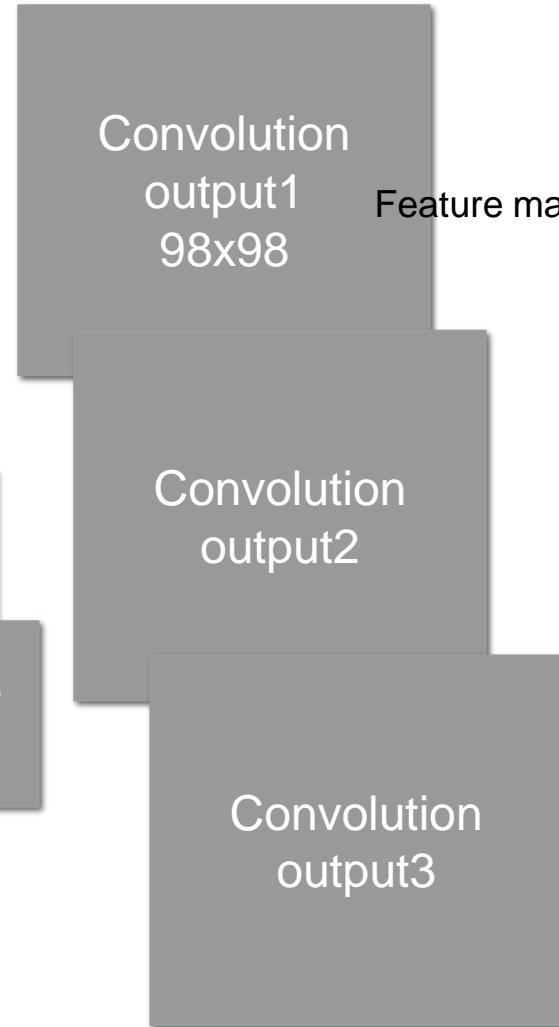
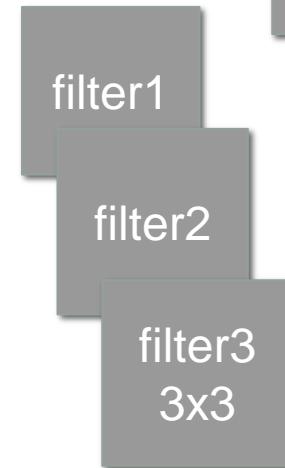
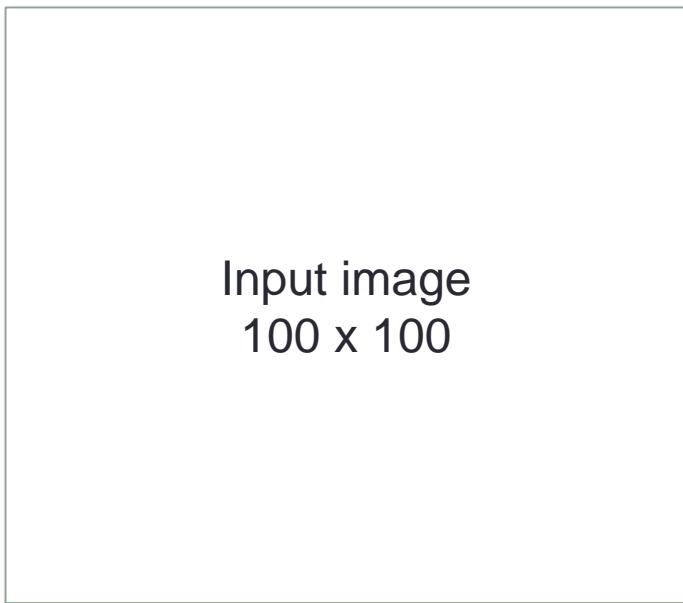
- Hierarchy of representations with increasing level of abstraction
- Each stage is a kind of trainable feature transform
- Image recognition: Pixel → edge → texture → part → object



# Convolution filters

Multiply inputs with filter values

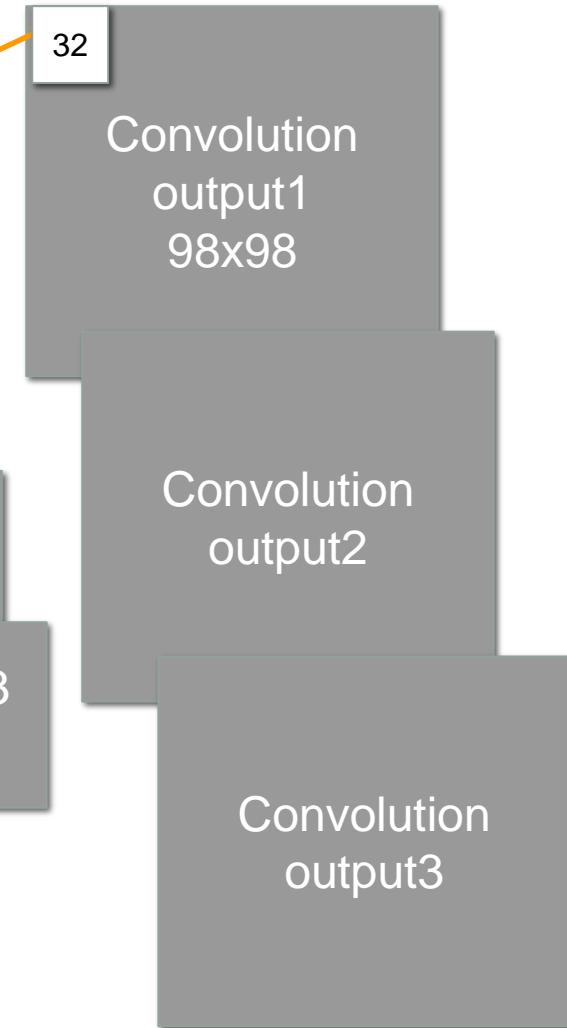
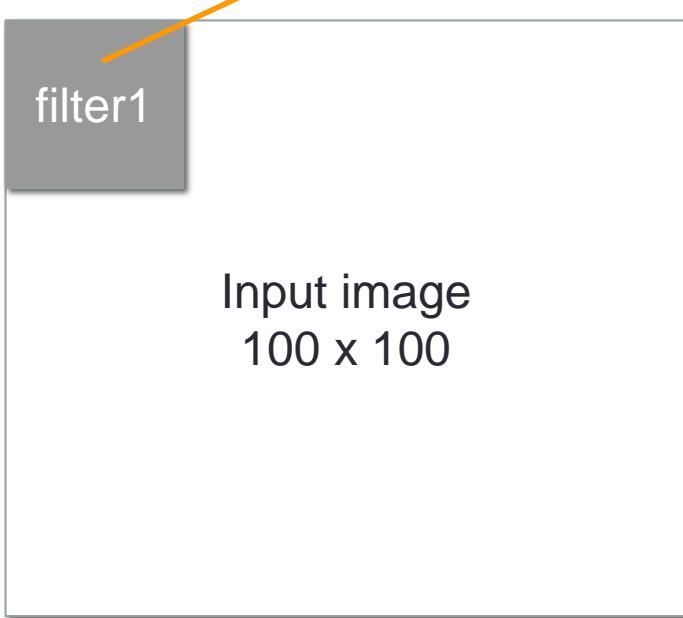
Output one feature map per filter



# Convolution filters

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

$$1*2 + -1*3 + \\ 1*4 + 1*6 + 1*7 + 2*8 = 32$$

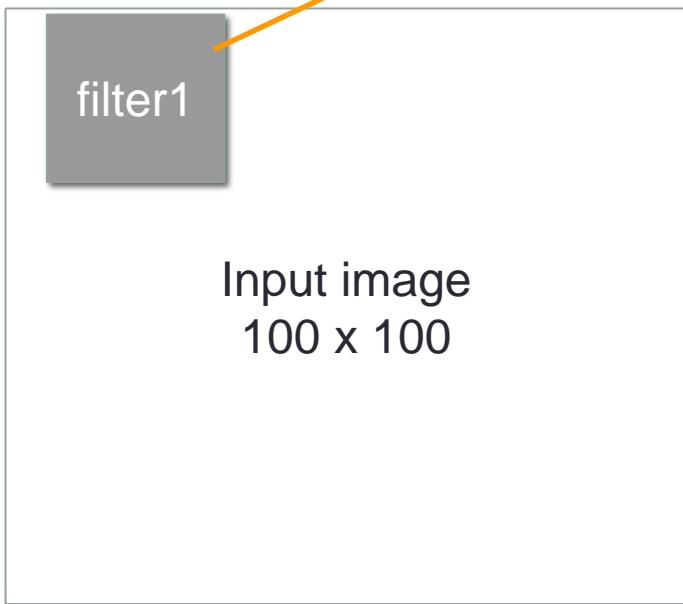


# Convolution filters

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

} Stride of 1

$$1*3 + -1*1 + \\ 1*5 + 1*3 + 1*8 \\ + 2*9 = 36$$



Stride of 1

32 36

Convolution  
output1  
98x98

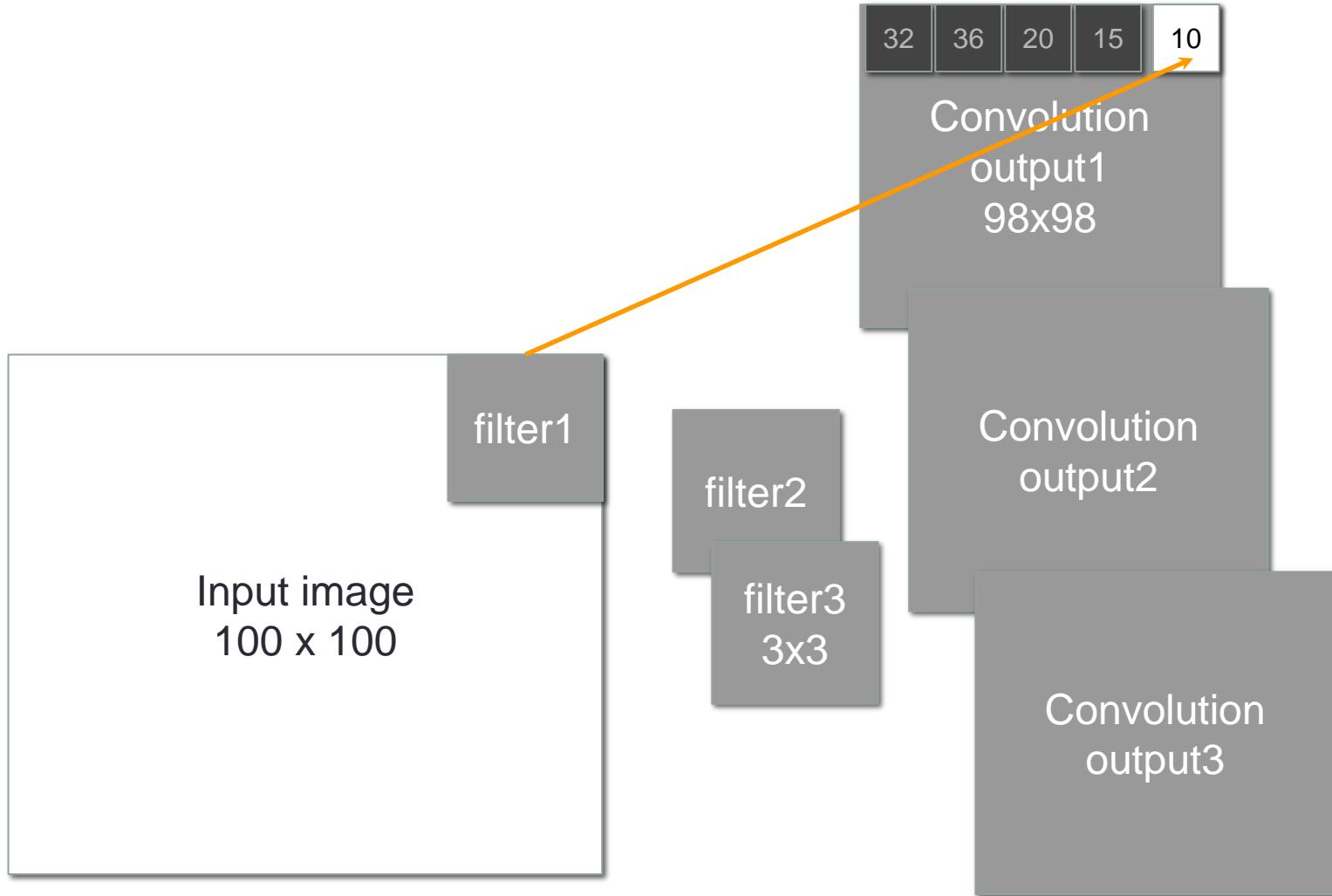
filter2

filter3  
3x3

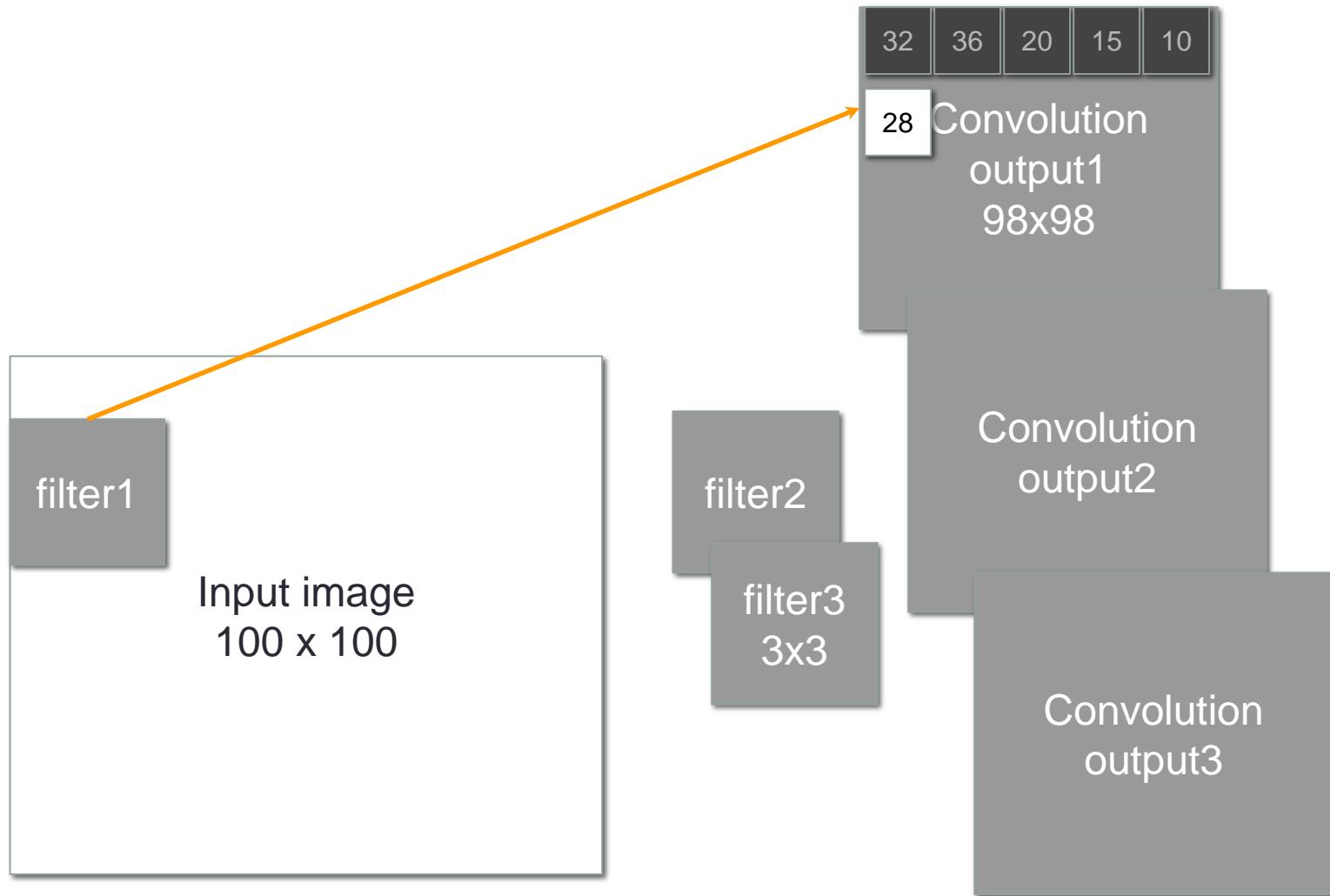
Convolution  
output2

Convolution  
output3

# Convolution filters

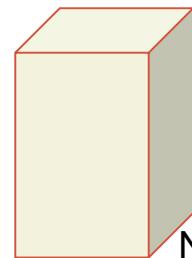


# Convolution filters

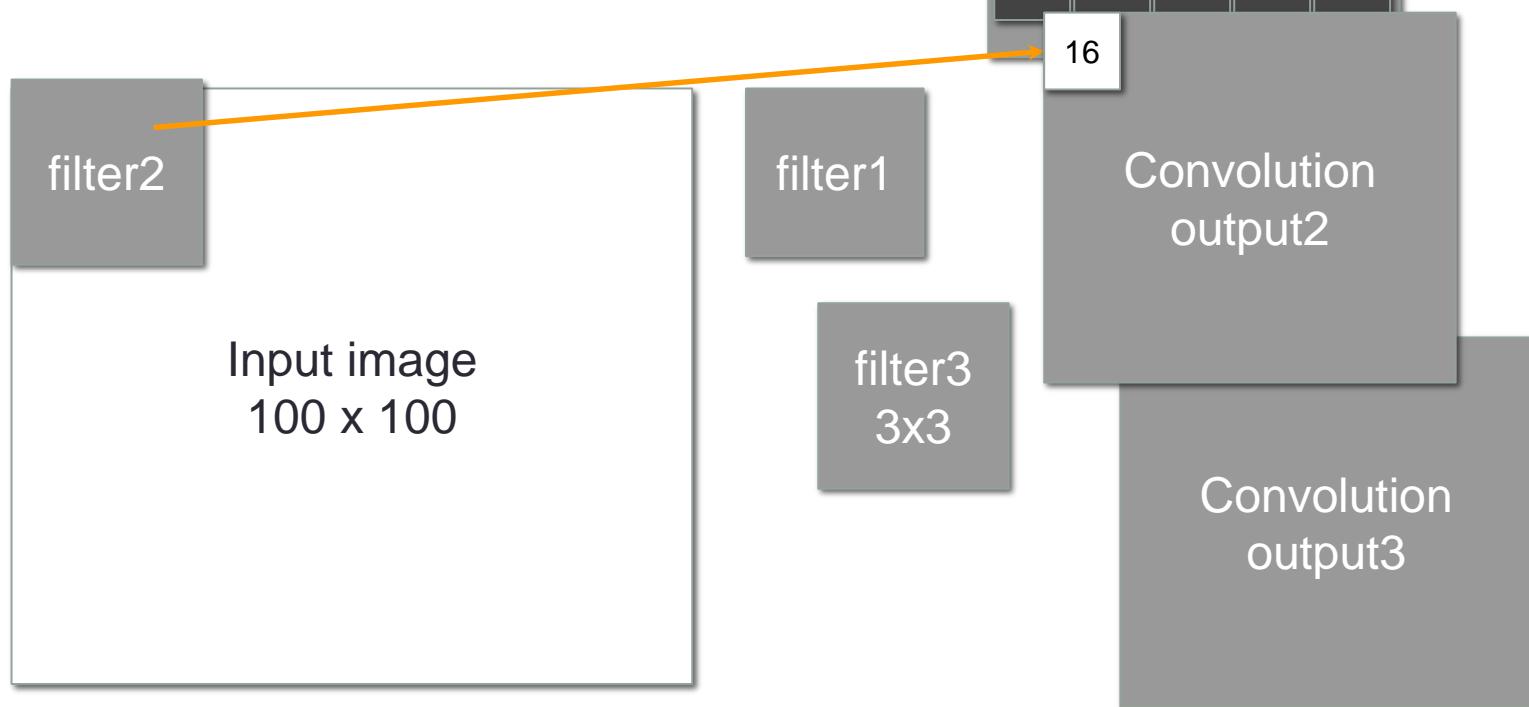


# Convolution filters

N filters means N feature maps  
You get a 3 dimensional output



32	36	20	15	10
28	72	0	12	50
32	36	18	9	2
17	6	2	17	1



# Pooling/subsampling

Reduce dimension of the feature maps



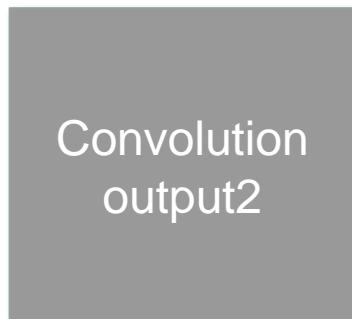
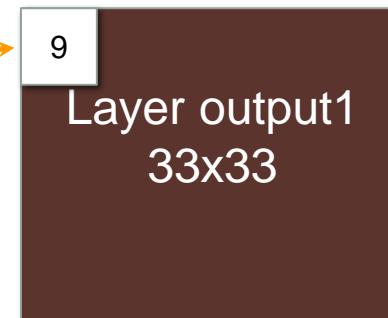
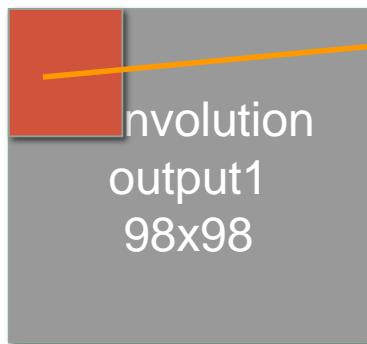
3x3 Max filter  
with no overlap



# Pooling/subsampling

1	2	3
4	5	6
7	8	9

Max = 9

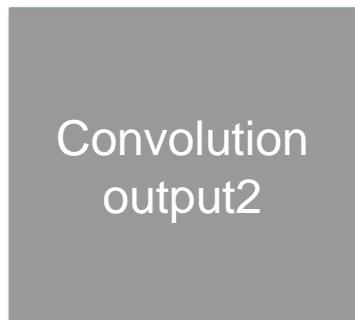
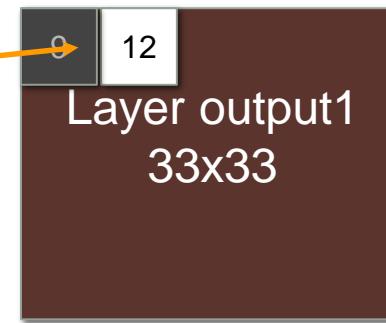
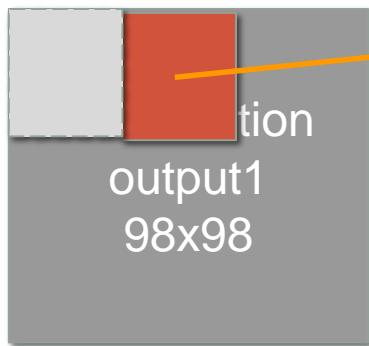


# Pooling/subsampling

5	2	1
5	7	1
9	5	12

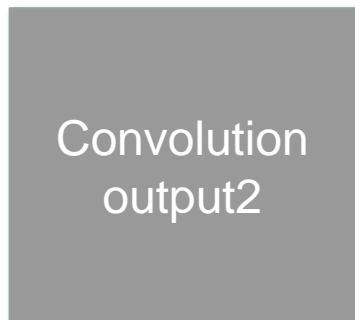
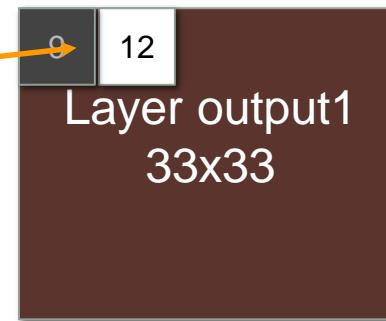
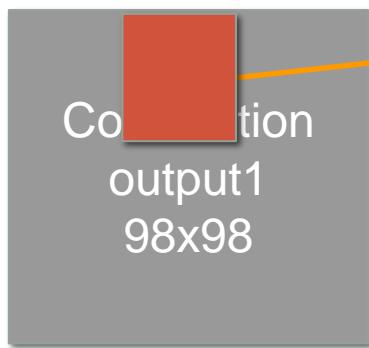
Max = 12

Stride = 3



# Pooling/subsampling

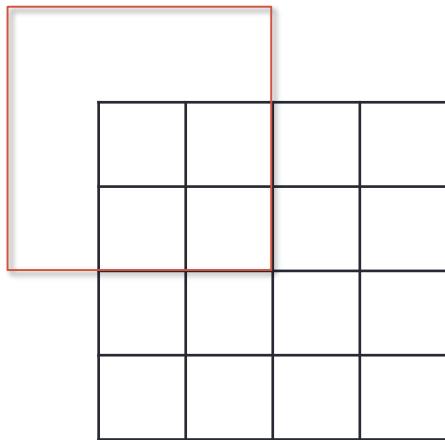
Can use other functions besides max  
Example, average



# Convolution puzzle

5 filters 3x3 filter pad, stride 1, pad 1

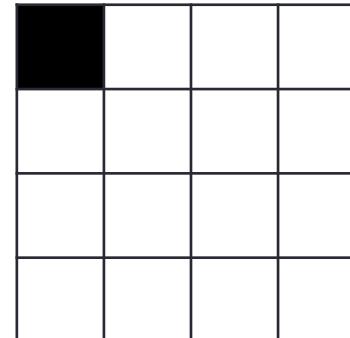
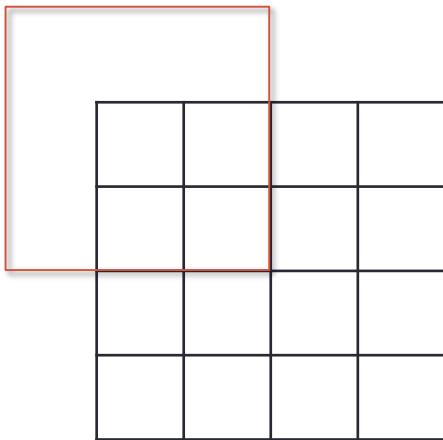
What is the output size?



# Convolution puzzle

5 filters 3x3 filter pad, stride 1, pad 1

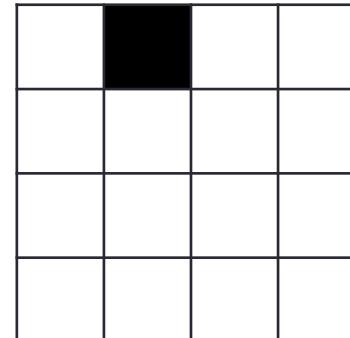
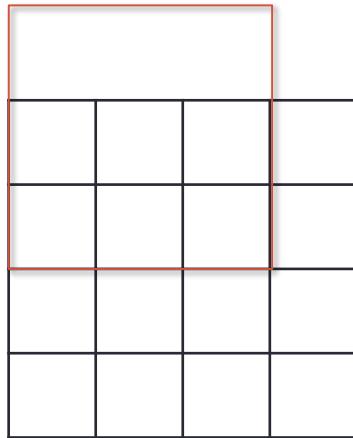
What is the output size?



# Convolution puzzle

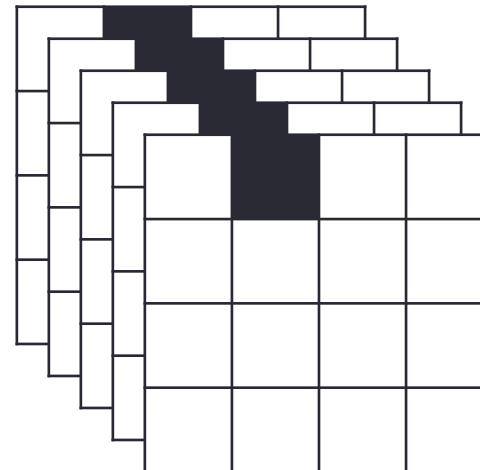
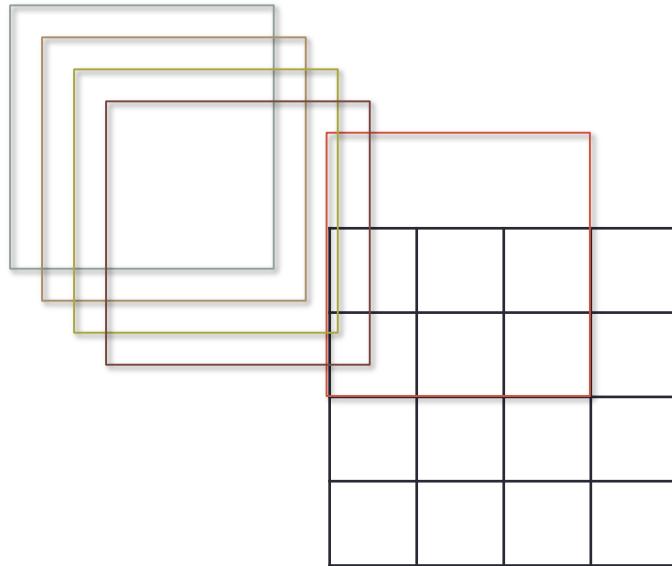
5 filters 3x3 filter pad, stride 1, pad 1

What is the output size?



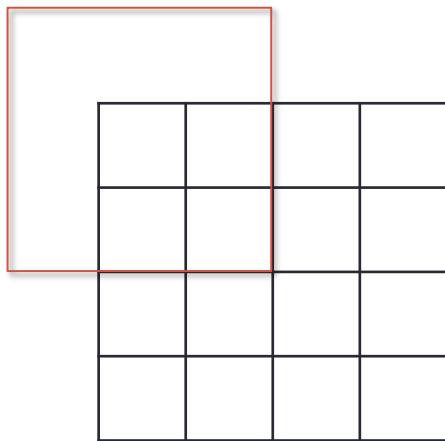
# Convolution puzzle

5 filters 3x3 filter pad, stride 1, pad 1



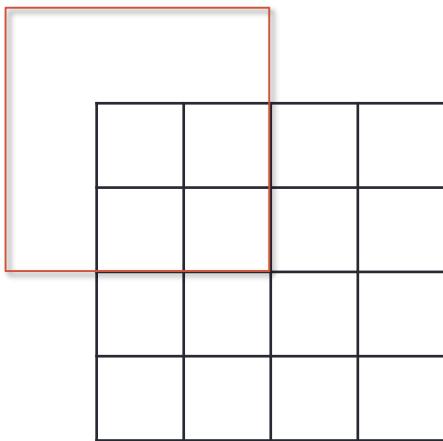
# Convolution puzzle

3x3 filter pad, stride 2, pad 1



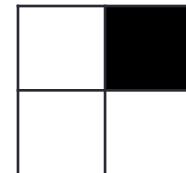
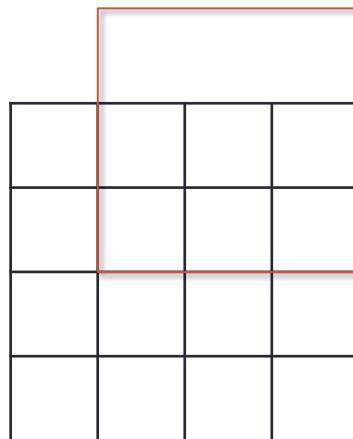
# Convolution puzzle

3x3 filter pad, stride 2, pad 1



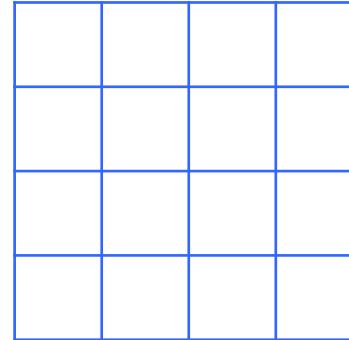
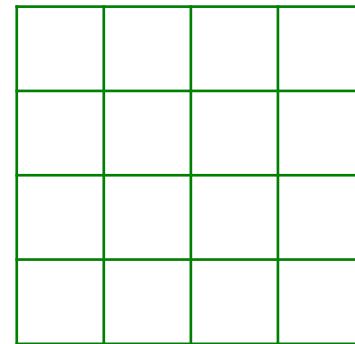
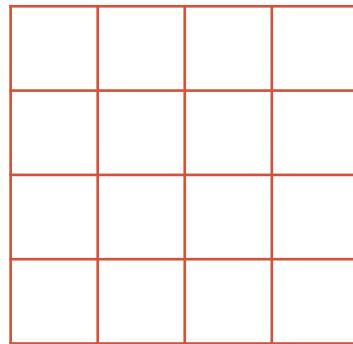
# Convolution puzzle

3x3 filter pad, stride 2, pad 1



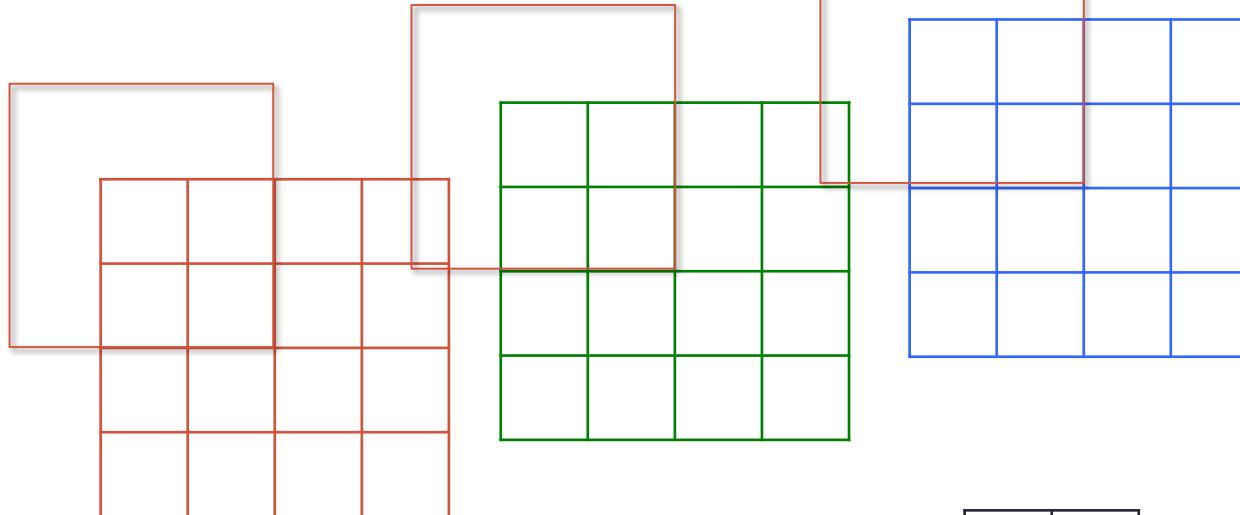
# Convolution puzzle

RGB input (3 channels) 5 filters 3x3 filter pad, stride 2, pad 1

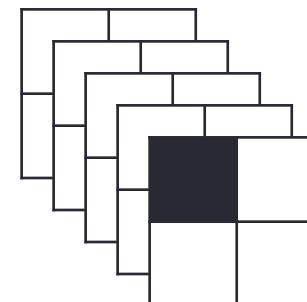


# Convolution puzzle

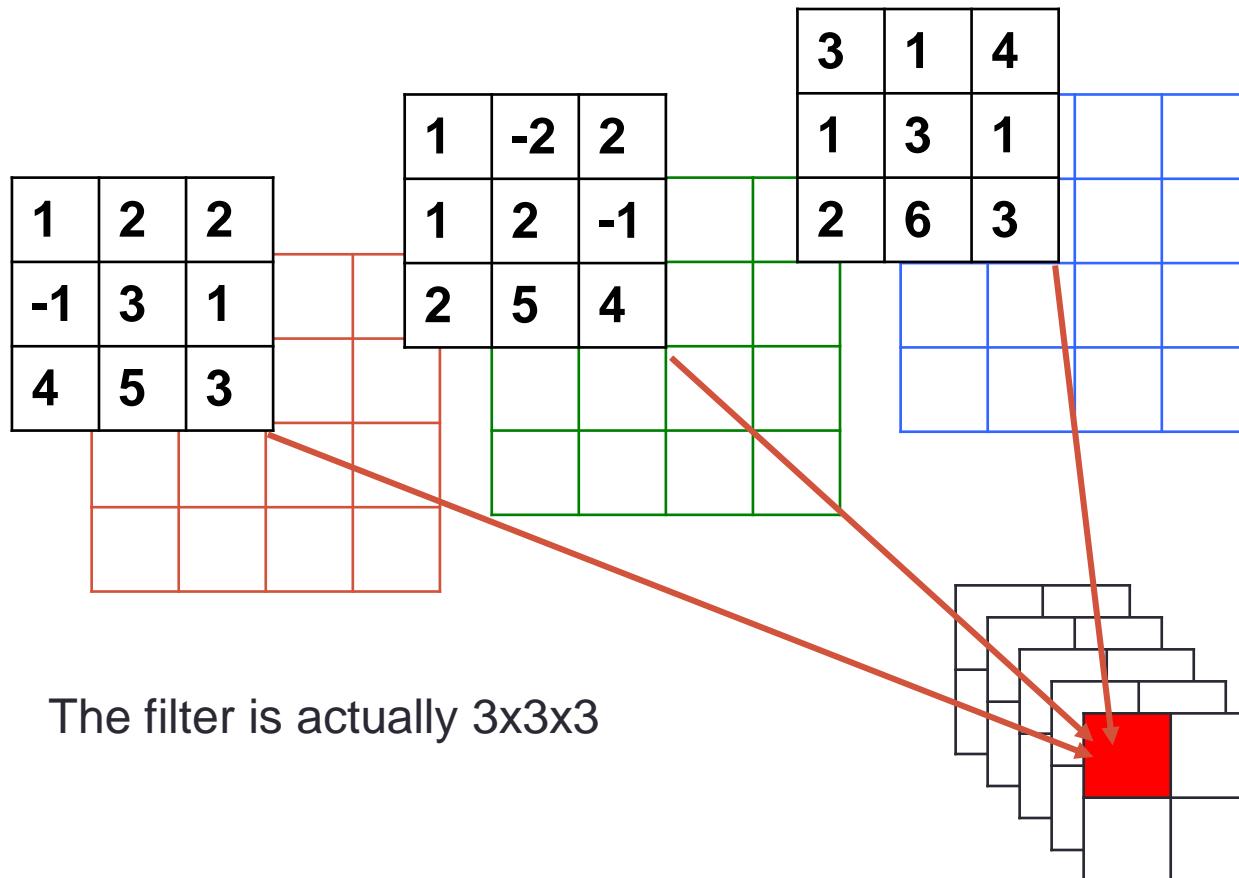
RGB input (3 channels) 5 filters 3x3 filter pad, stride 2, pad 1



The filter is actually 3x3x3

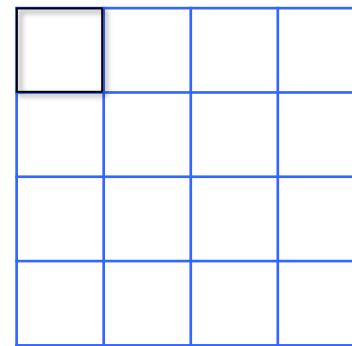
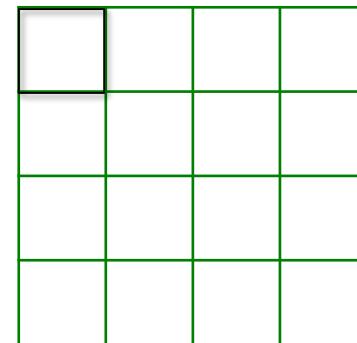
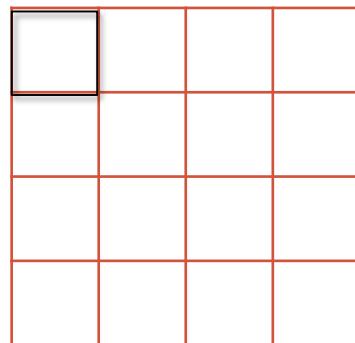


# Convolution puzzle

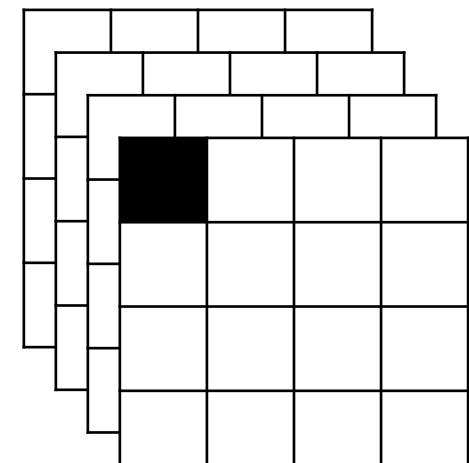
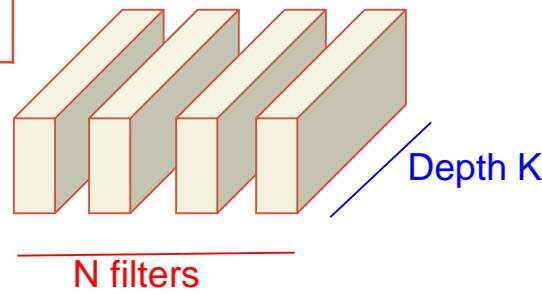


# 1x1 convolution

Reduces the dimension of feature maps

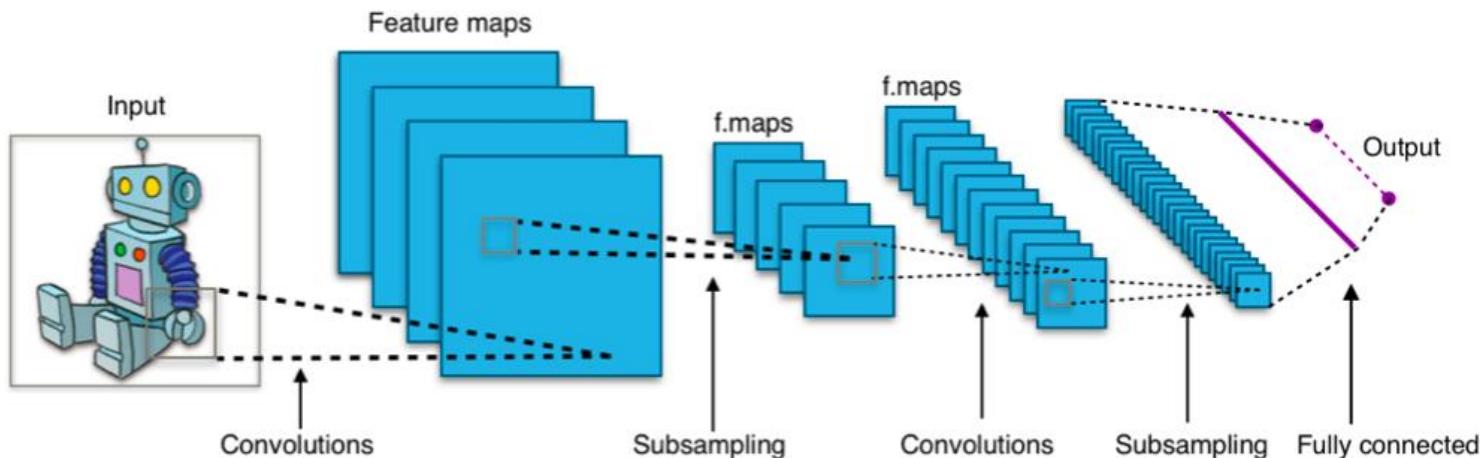


The filter is actually  $1 \times 1 \times K$



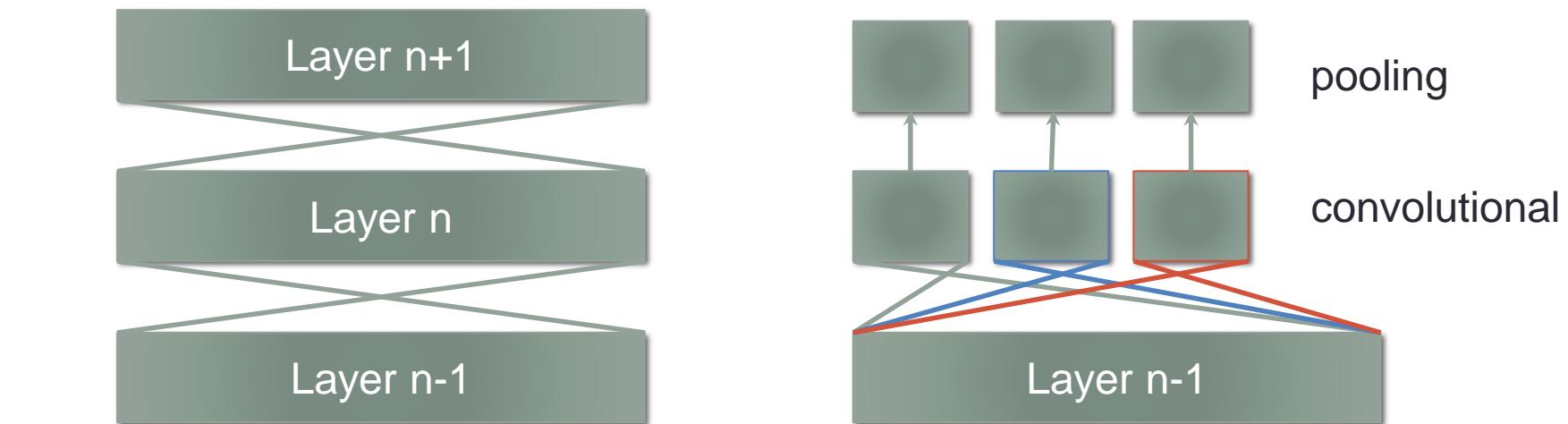
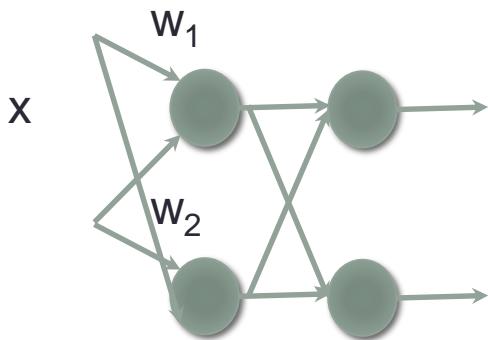
# CNN overview

- Filter size, number of filters, filter shifts, and pooling rate are all parameters
- Usually followed by a fully connected network at the end
  - CNN is good at learning low level features
  - DNN combines the features into high level features and classify



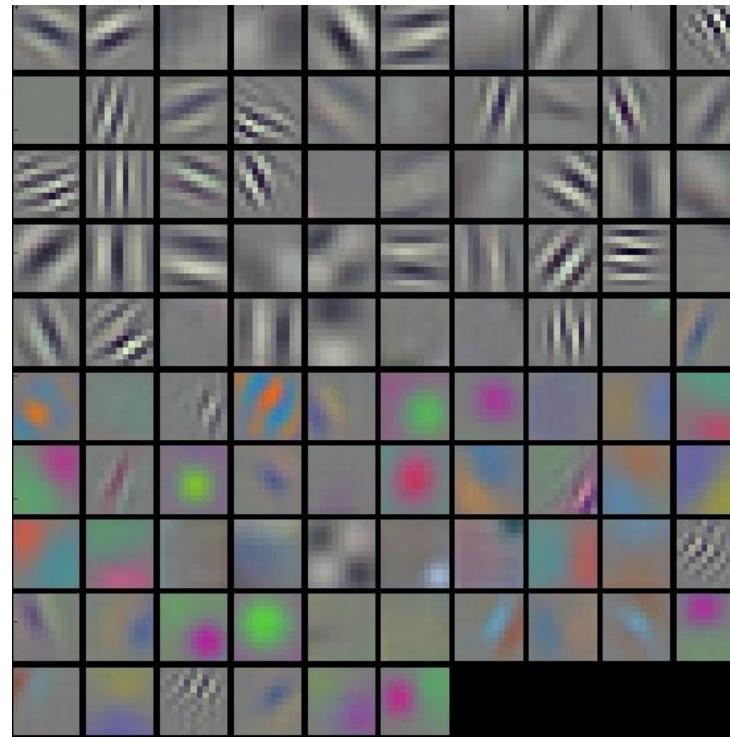
# Inductive bias in convolution neural networks

- $W^T x$ 
  - Cats at different location might need two neurons for different locations in fully connect NNs.
  - CNN shares the parameters in 1 filter
  - The network is no longer fully connected

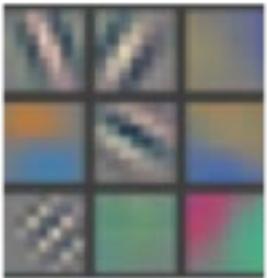


# Visualizing convolutional layers

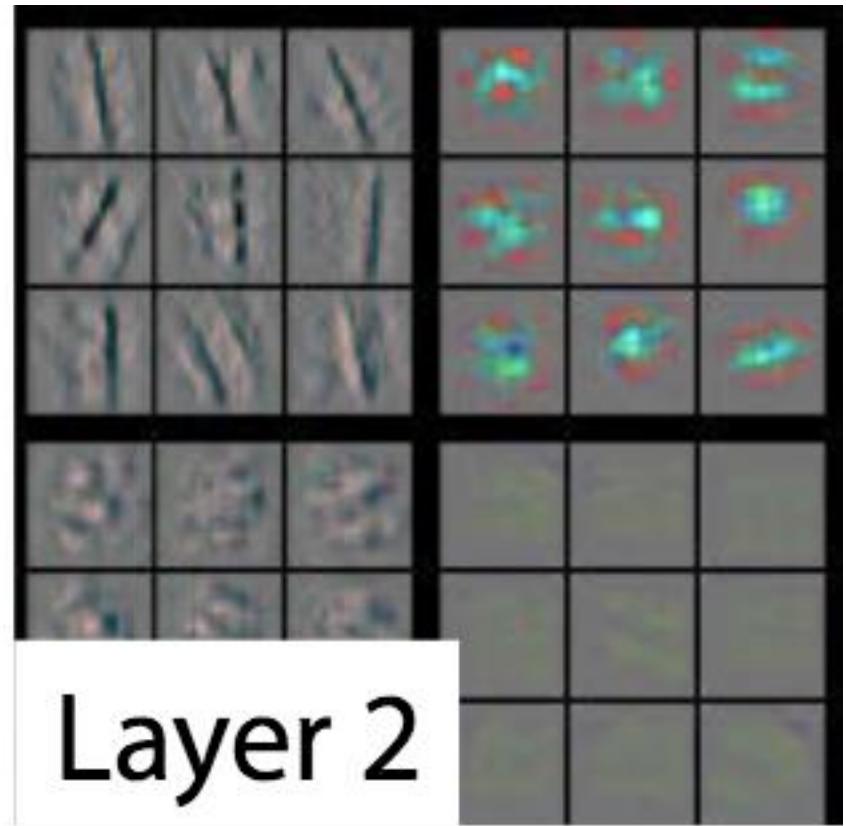
- Just like PCA, we can visualize the weights of a transform
- “Matched filters”



# Higher layer captures higher-level concepts

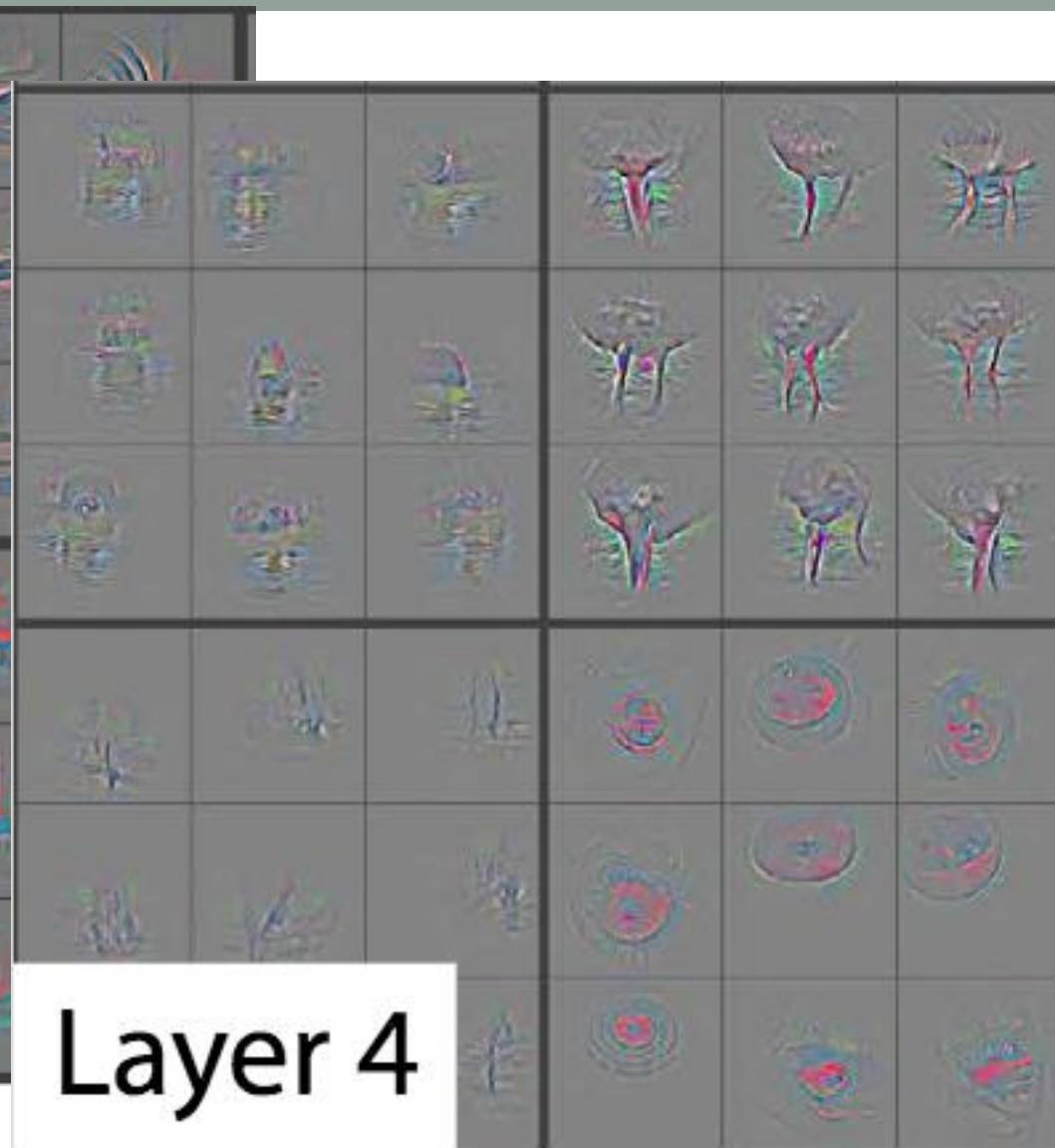


Layer 1





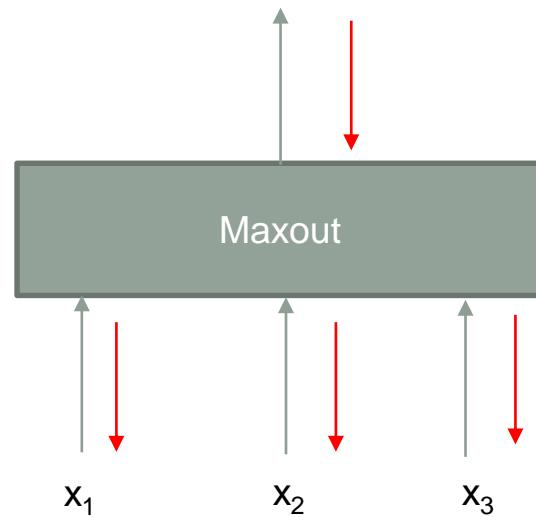
Layer 3



Layer 4

# Pooling/subsampling

- Max filter -> Maxout
  - Backward pass?

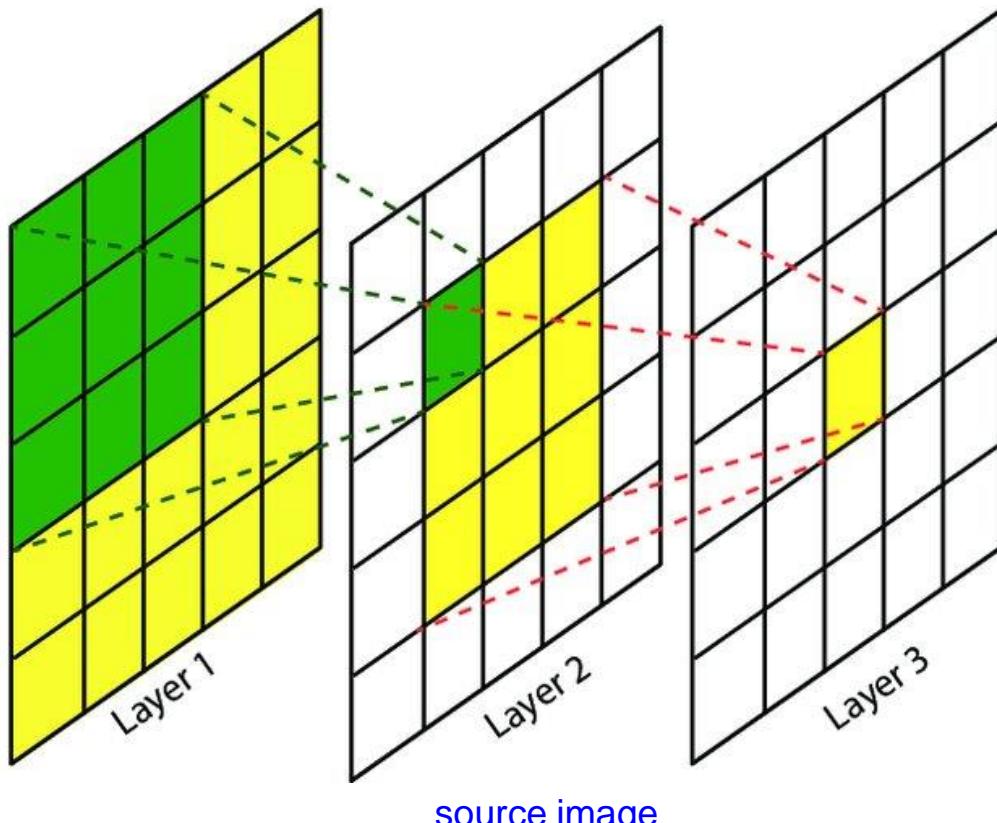


# Common schemes

- INPUT -> [CONV -> RELU -> POOL]<sup>\*N</sup> -> [FC -> RELU]<sup>\*M</sup> -> FC
- INPUT -> [CONV -> RELU -> CONV -> RELU -> POOL]<sup>\*N</sup> -> [FC -> RELU]<sup>\*M</sup> -> FC

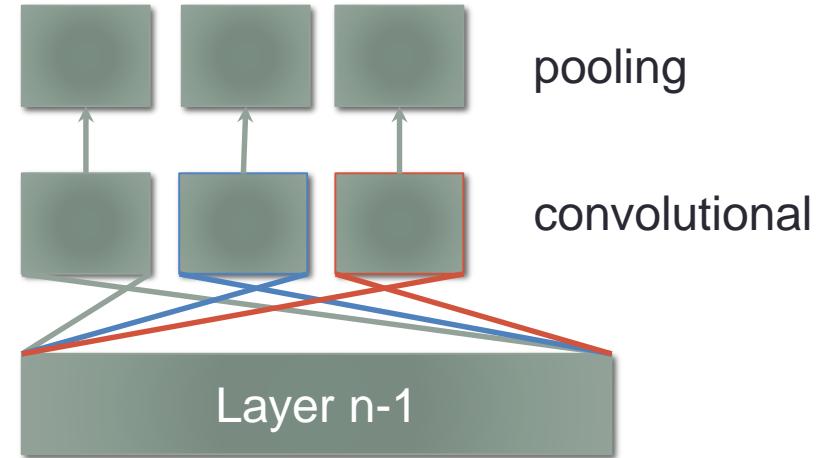
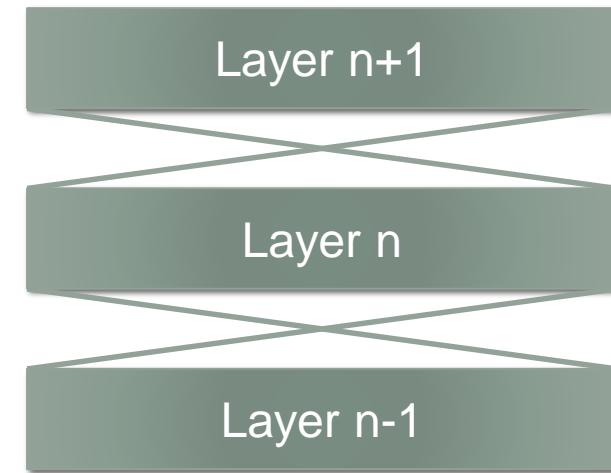
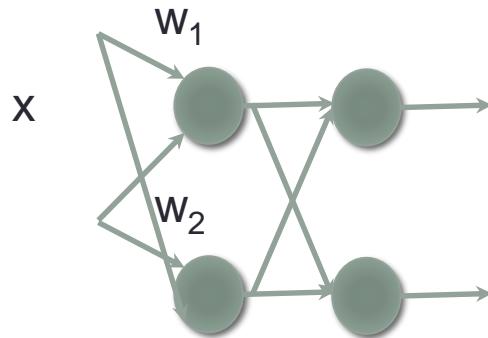
# Receptive field

You might want to consider about how large is your pattern when designing your network



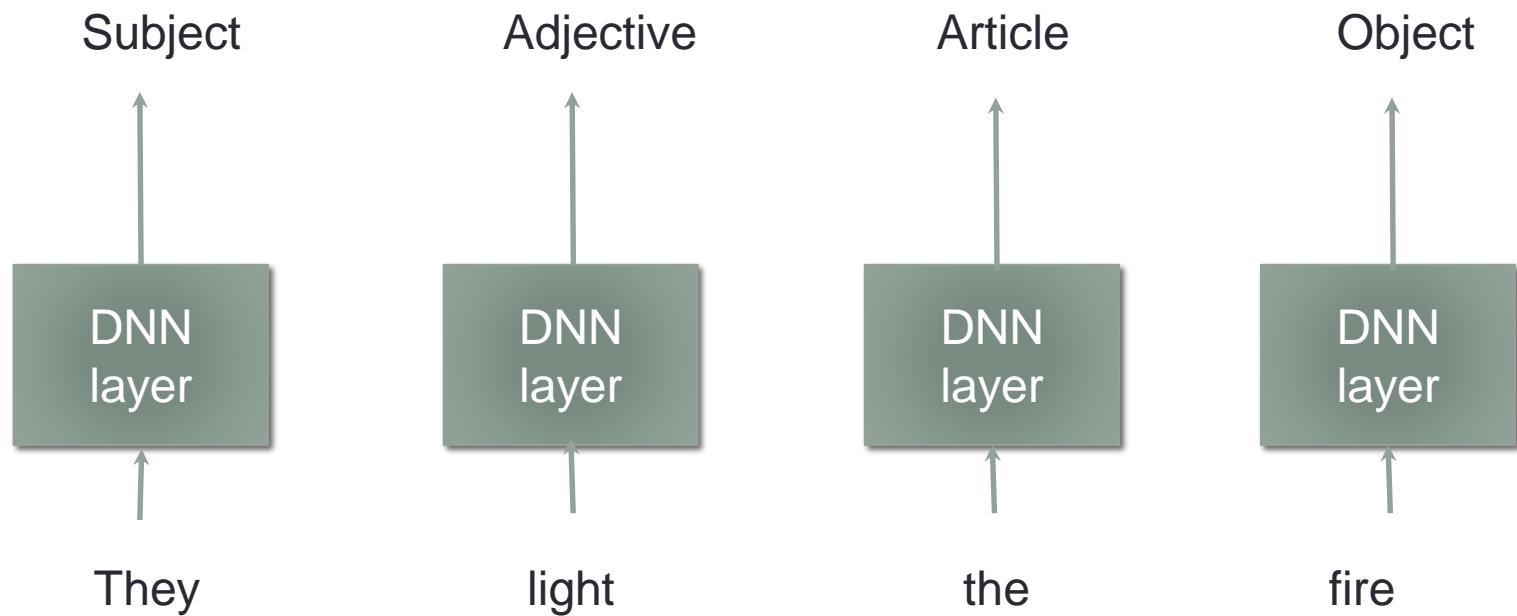
# Inductive bias for time?

- $W^T x$ 
  - CNN shares parameters in space
  - What about sharing parameters in time?



# Recurrent neural network (RNN)

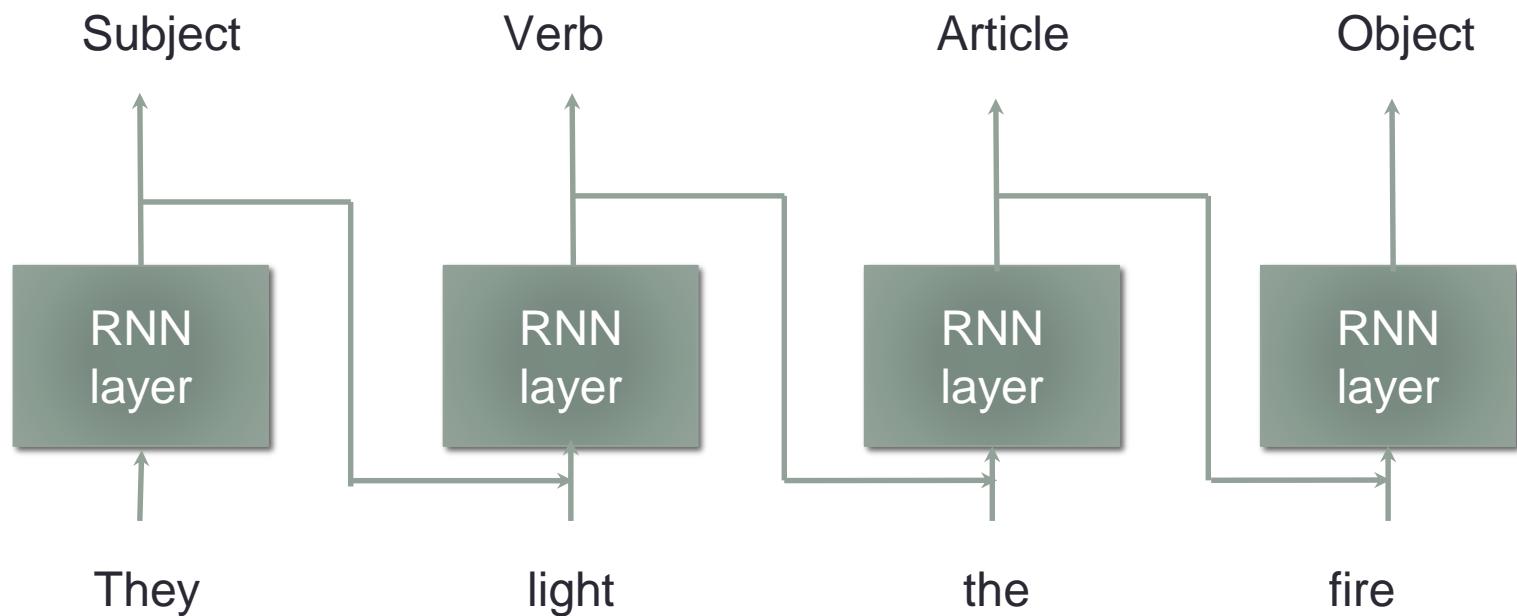
- DNN framework



Problem 1: need a way to remember the past

# Recurrent neural network (RNN)

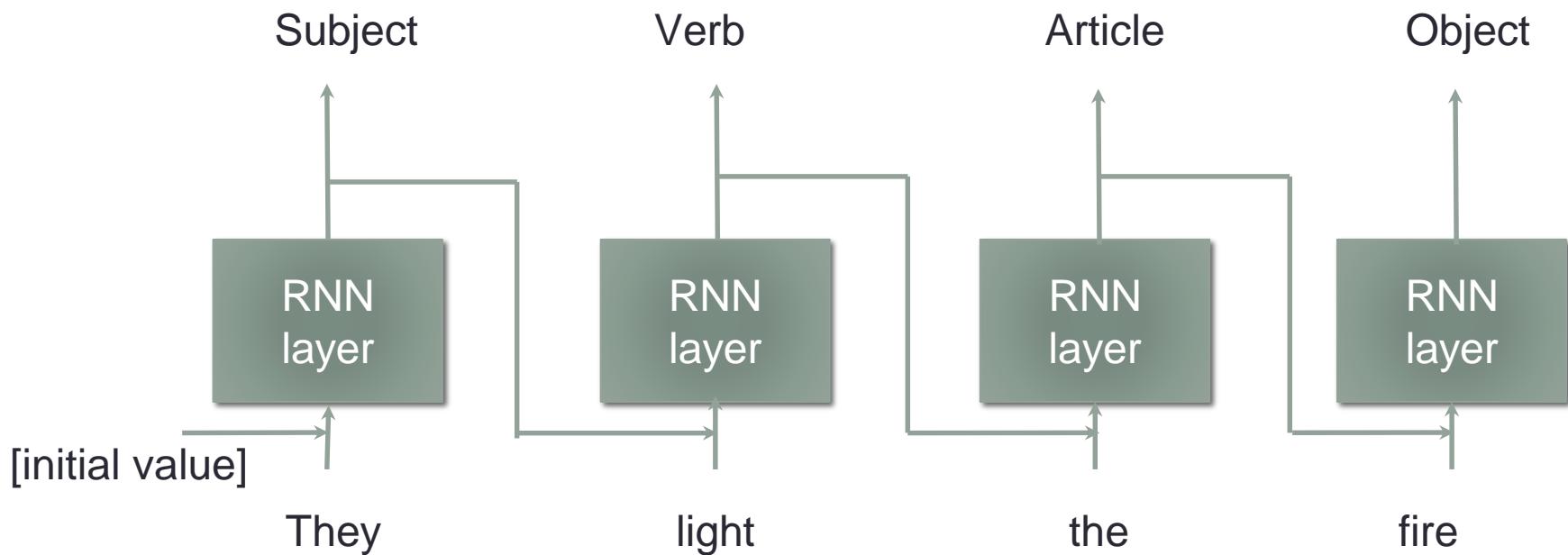
- RNN framework



Output of the layer encodes something meaningful about the past

# Recurrent neural network (RNN)

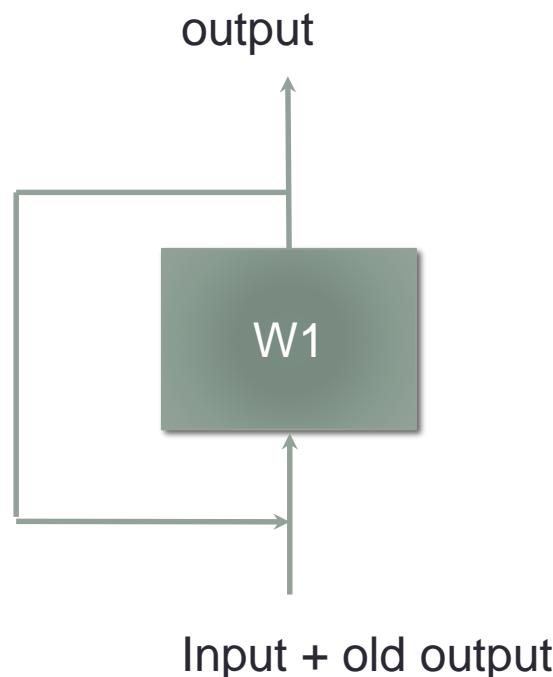
- RNN framework



New input feature = [original input feature, output of the layer at previous time step]

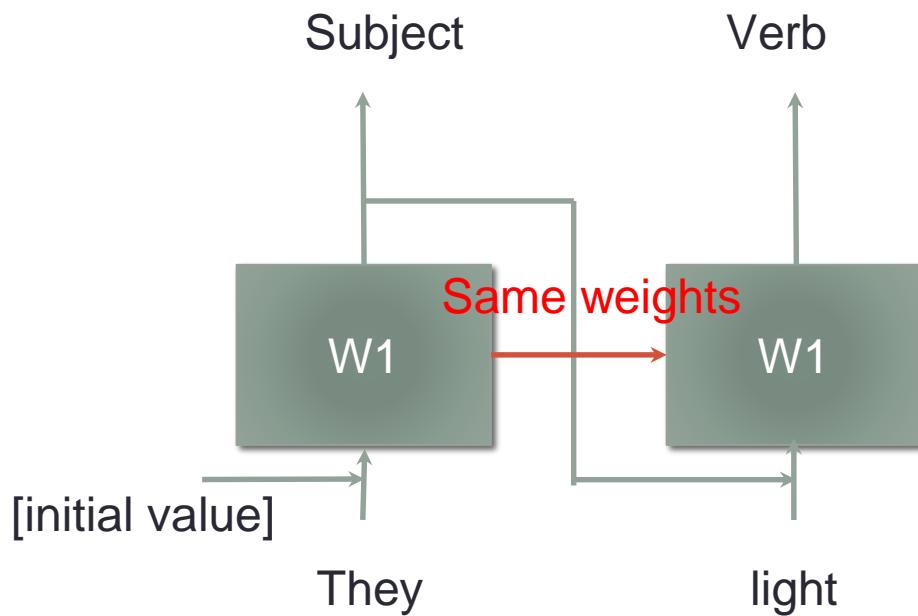
# Recurrent neural network (RNN)

- Unrolling of a recurrent layer.



# Recurrent neural network (RNN)

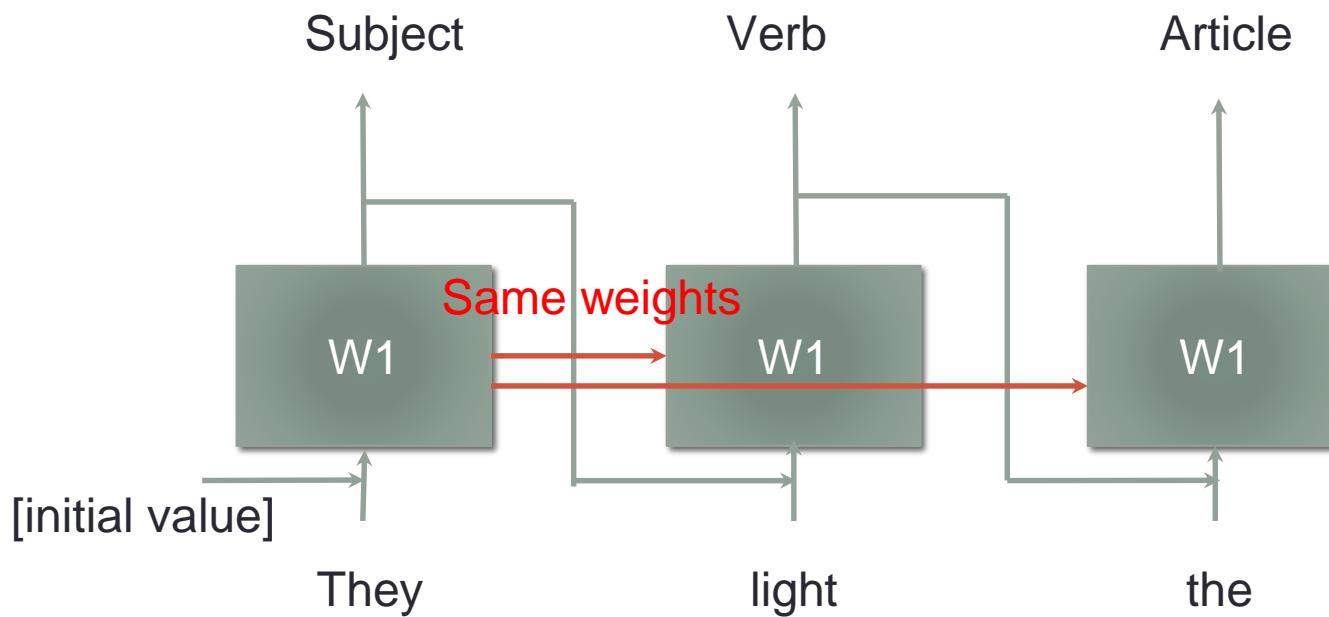
- Unrolling of a recurrent layer.



Parameter sharing across time

# Recurrent neural network (RNN)

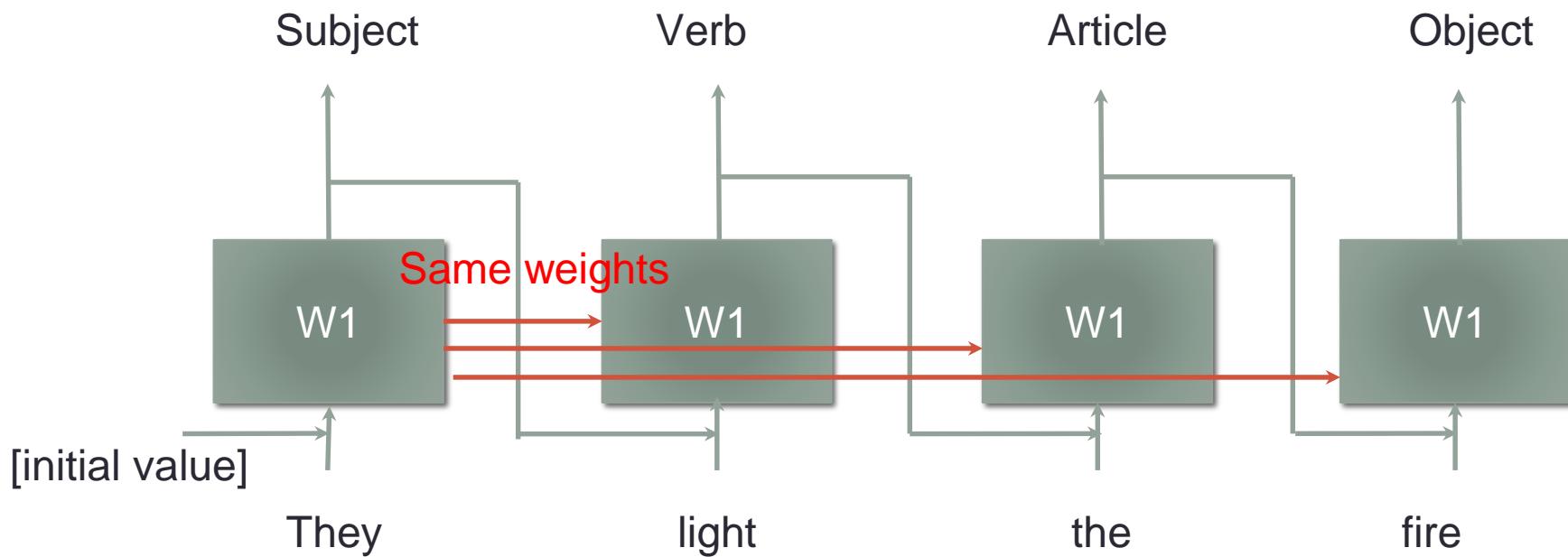
- Unrolling of a recurrent layer.



Parameter sharing across time

# Recurrent neural network (RNN)

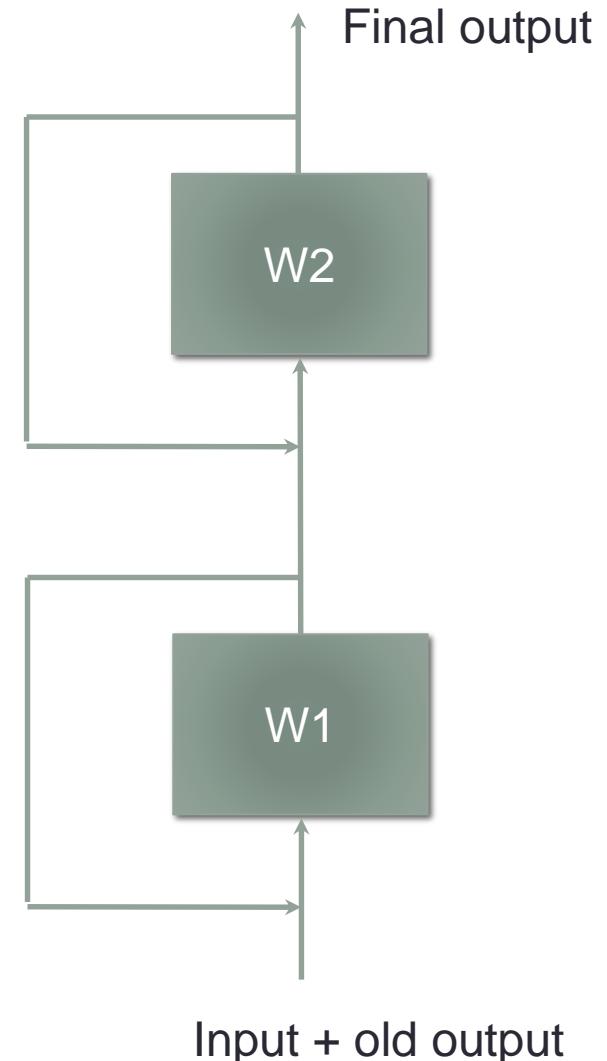
- Unrolling of a recurrent layer.



Parameter sharing across time

# Recurrent neural network (RNN)

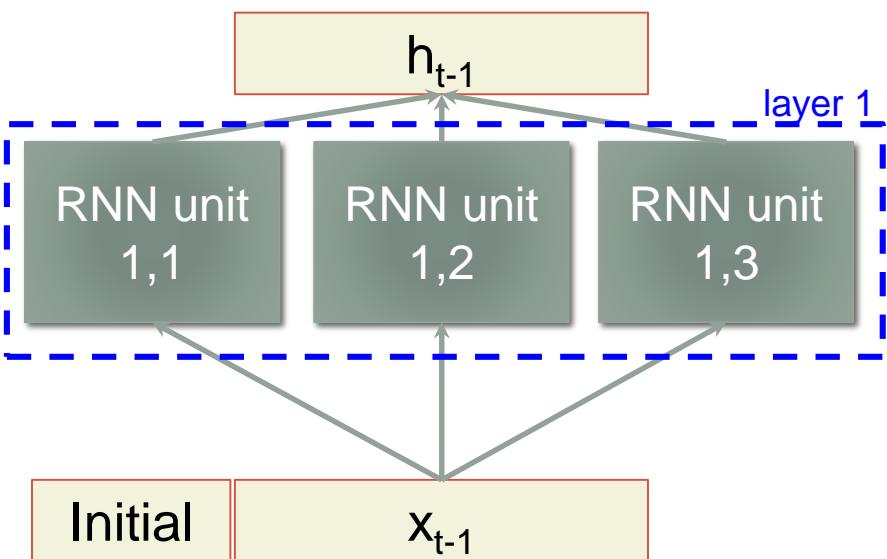
- Stacks of recurrent layer



# RNN layers (expanded in time)

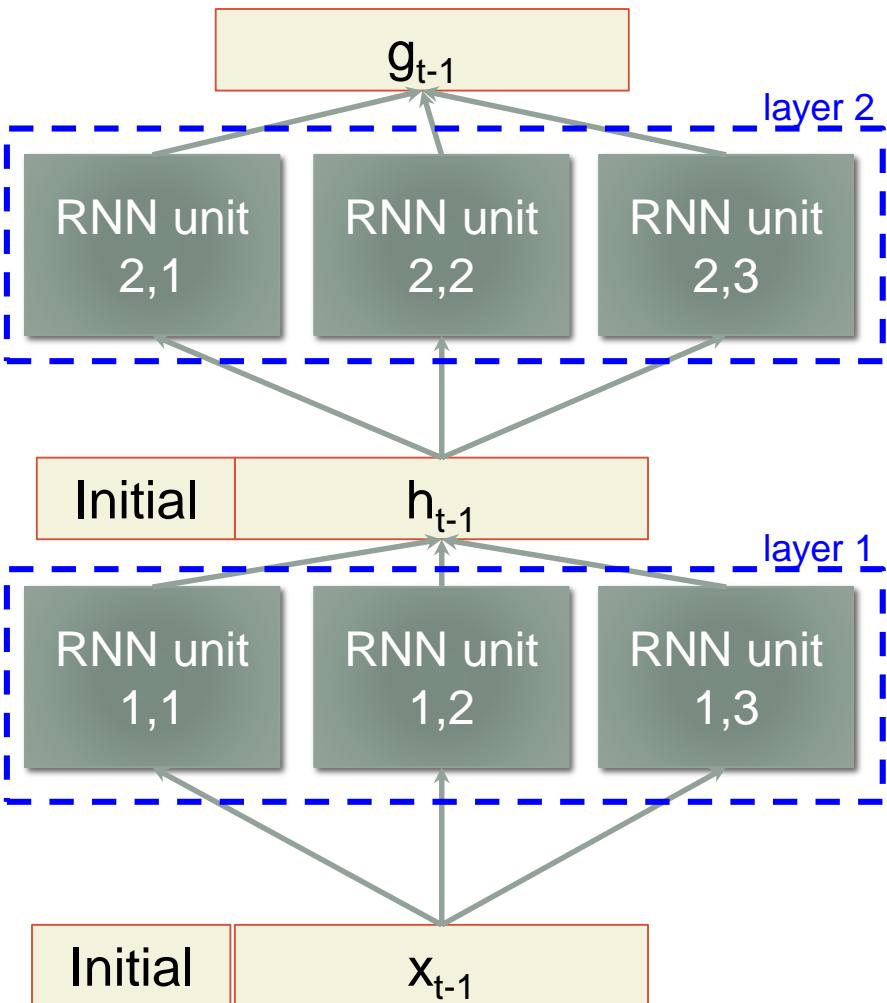
Time step 1

Time step 2



# RNN layers (expanded in time)

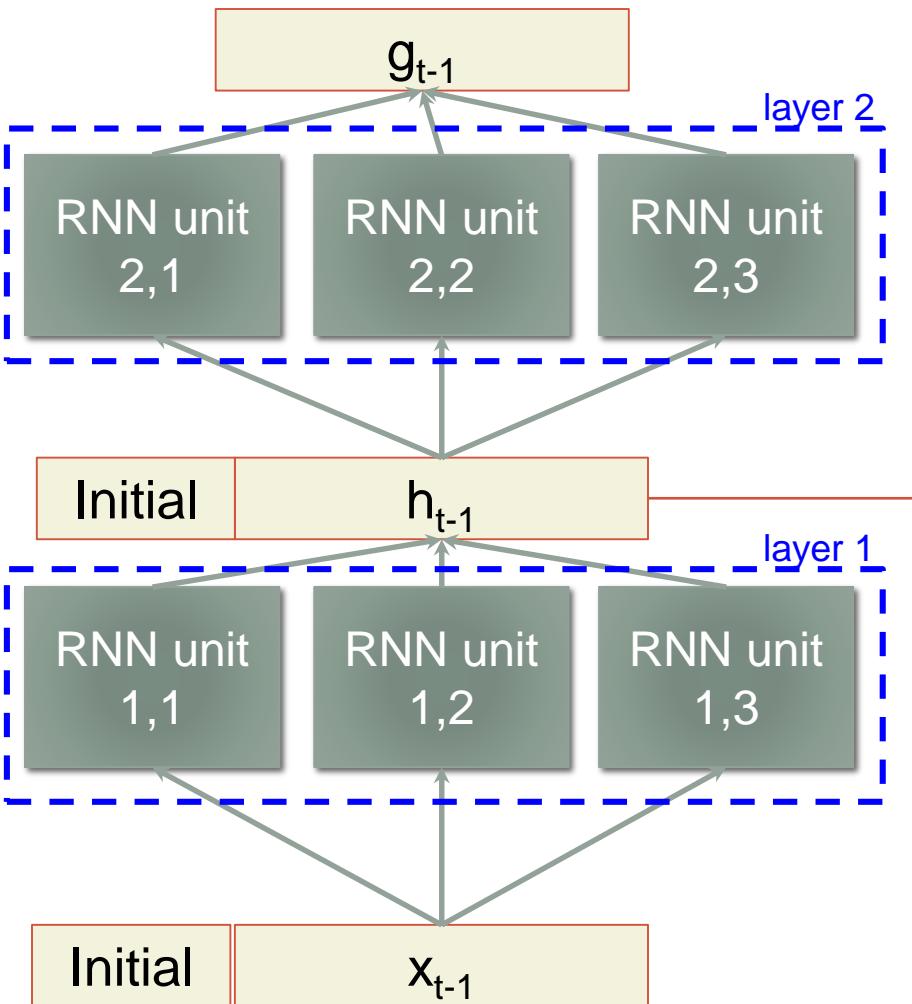
Time step 1



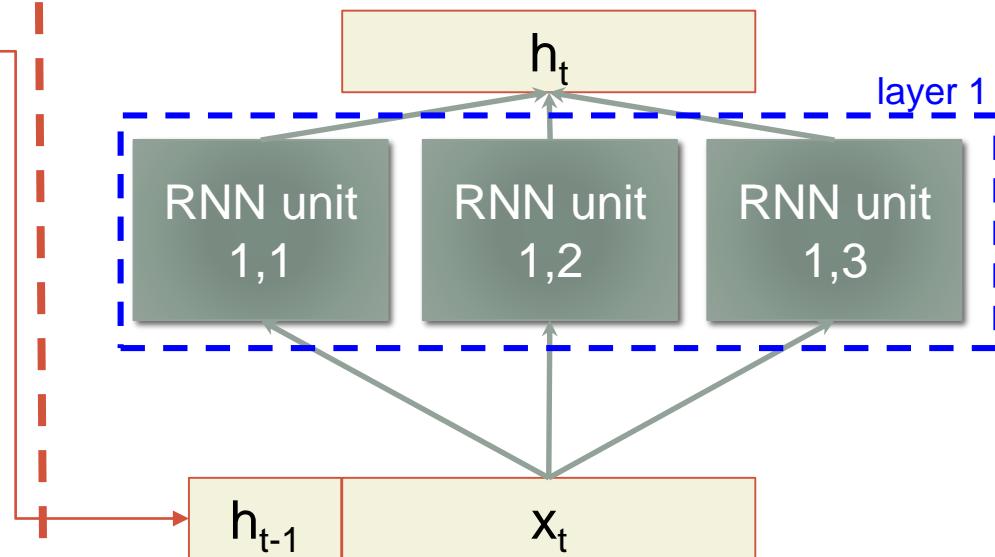
Time step 2

# RNN layers (expanded in time)

Time step 1

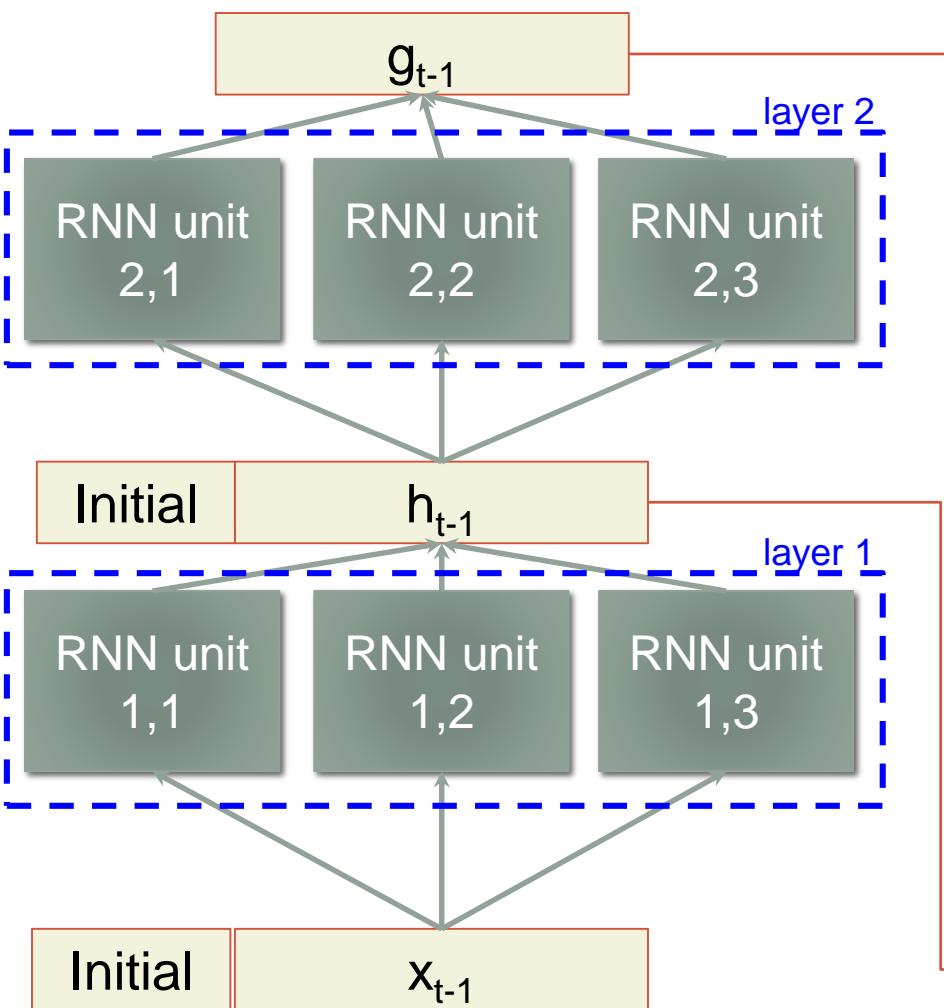


Time step 2

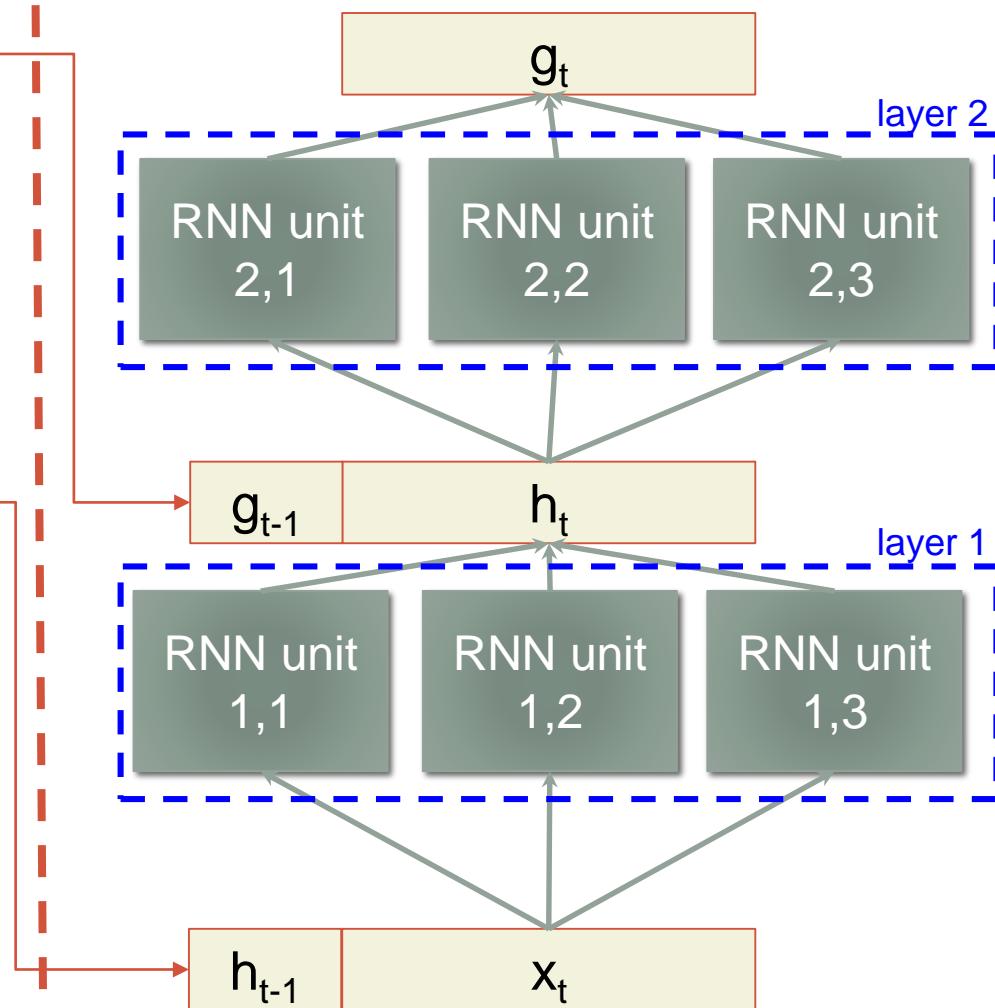


# RNN layers (expanded in time)

Time step 1

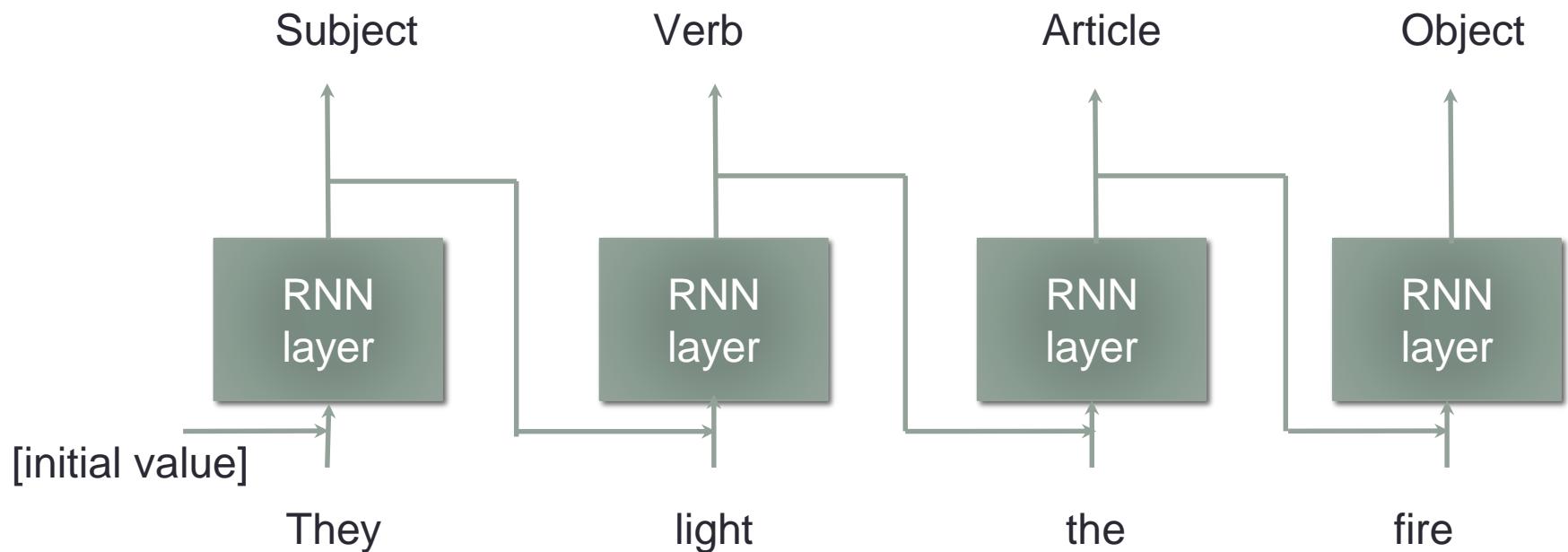


Time step 2



# Training a recurrent neural network

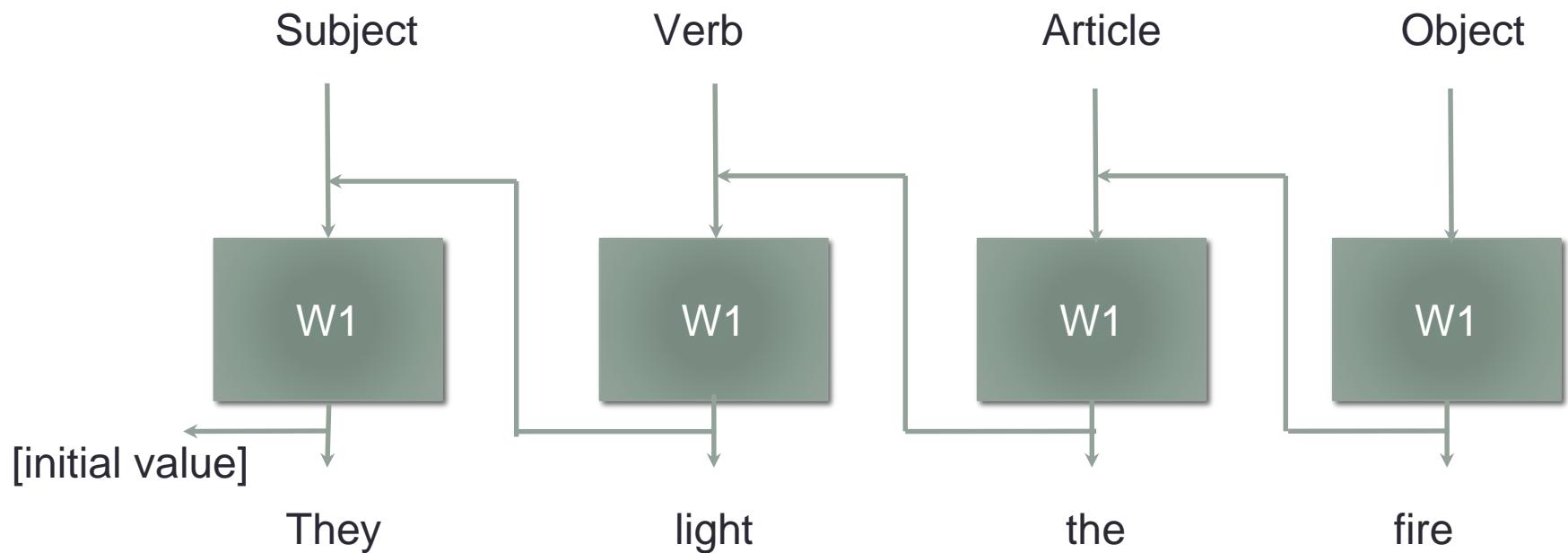
- RNN framework



New input feature = [original input feature, output of the layer at previous time step]

# Training a recurrent neural network

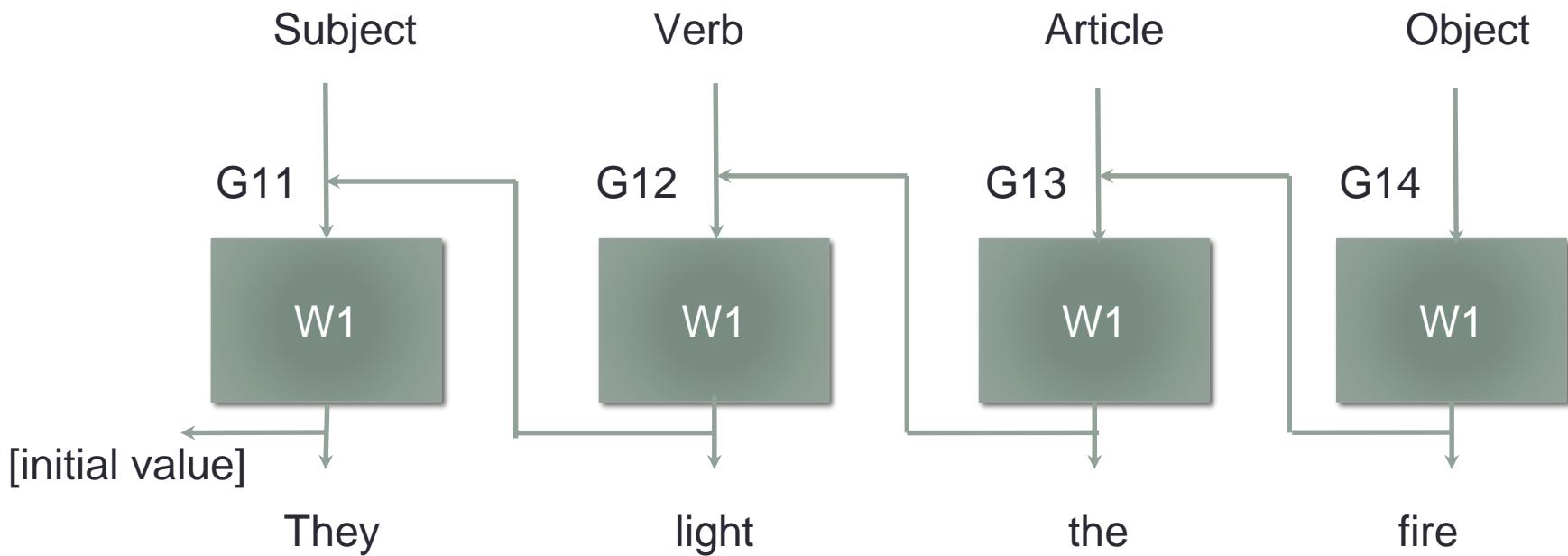
- Backward Computation graph



Backpropagation through time (BPTT)

# BPTT

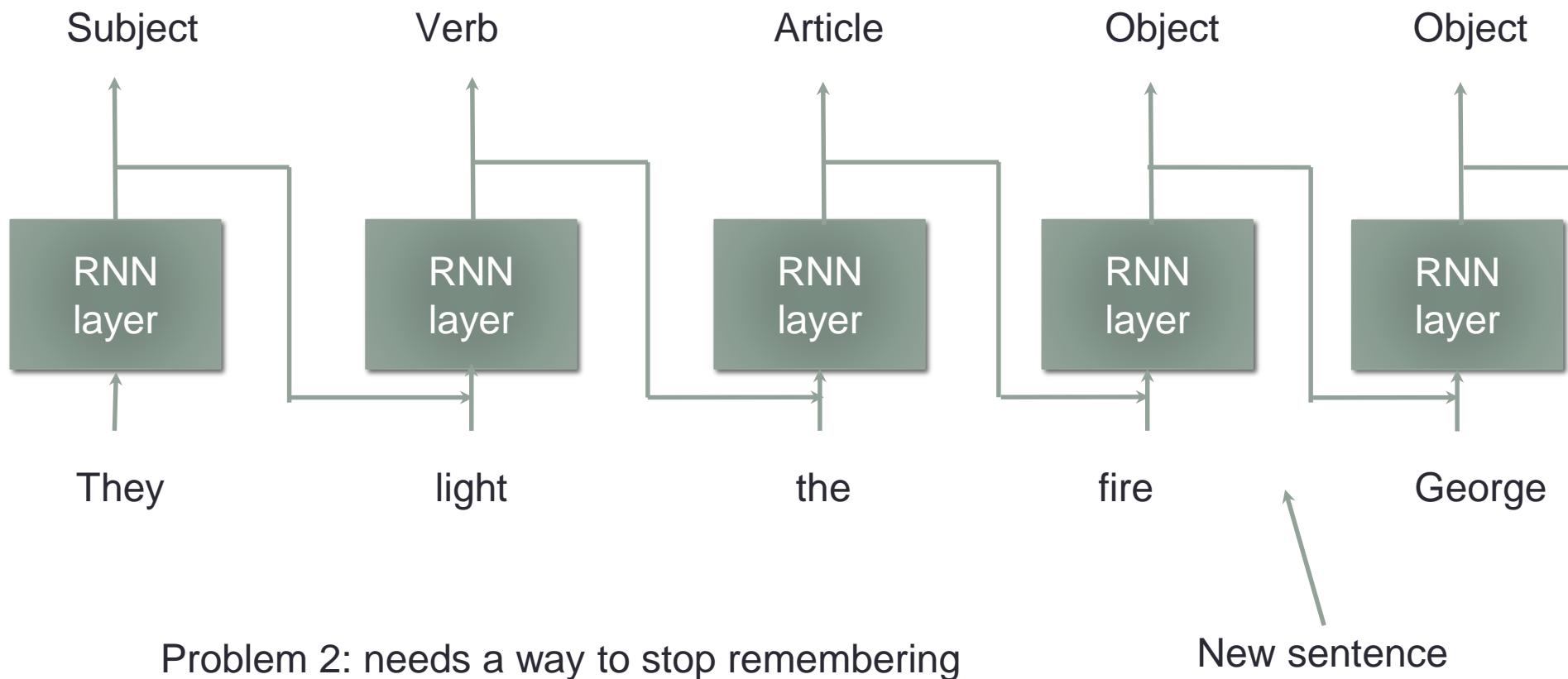
- Backward Computation graph



$$W1 \leftarrow W1 + G11 + G12 + G13 + G14$$

Problem 1: cannot deal with infinitely long recurrent  
Gradient explosion, vanishing gradient

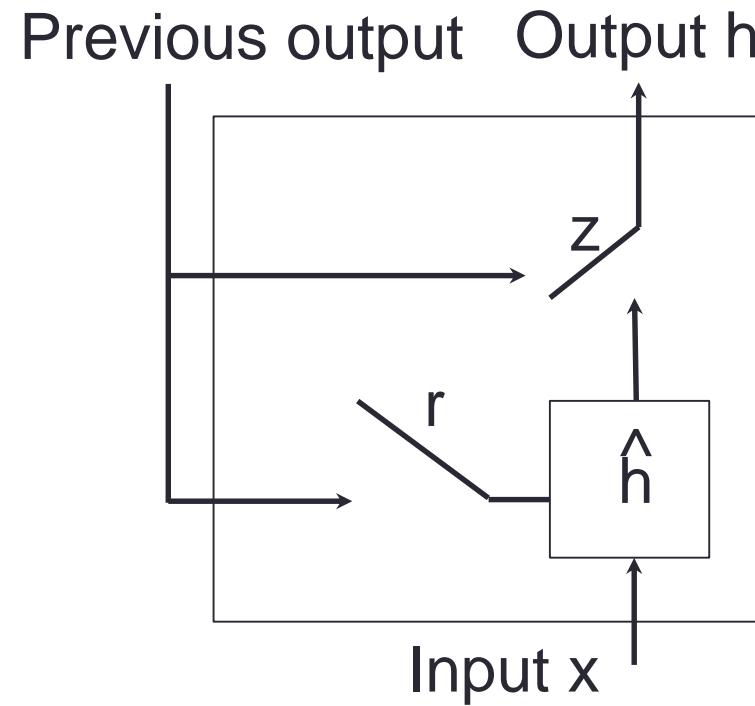
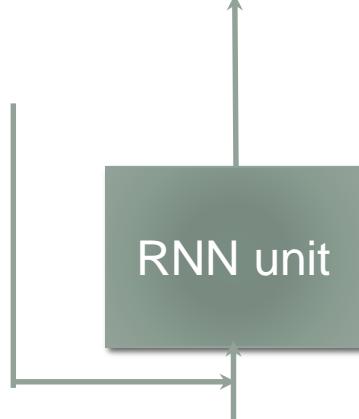
# Recurrent neural network (RNN)



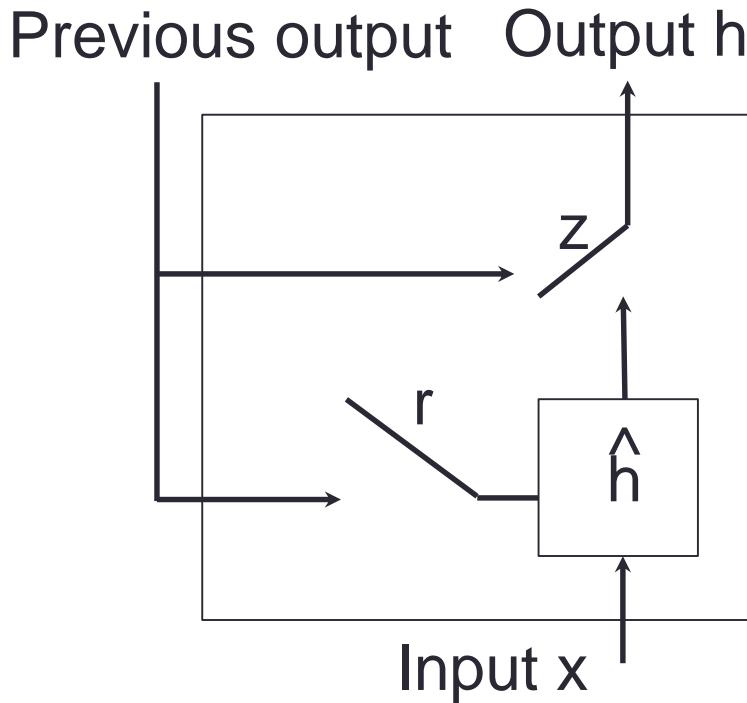
Can the network learn when to start and stop remembering things?

# Gated Recurrent Unit (GRU)

- Forms a Gated Recurrent Neural Networks (GRNN)
- Add gates that can choose to reset ( $r$ ) or update ( $z$ )



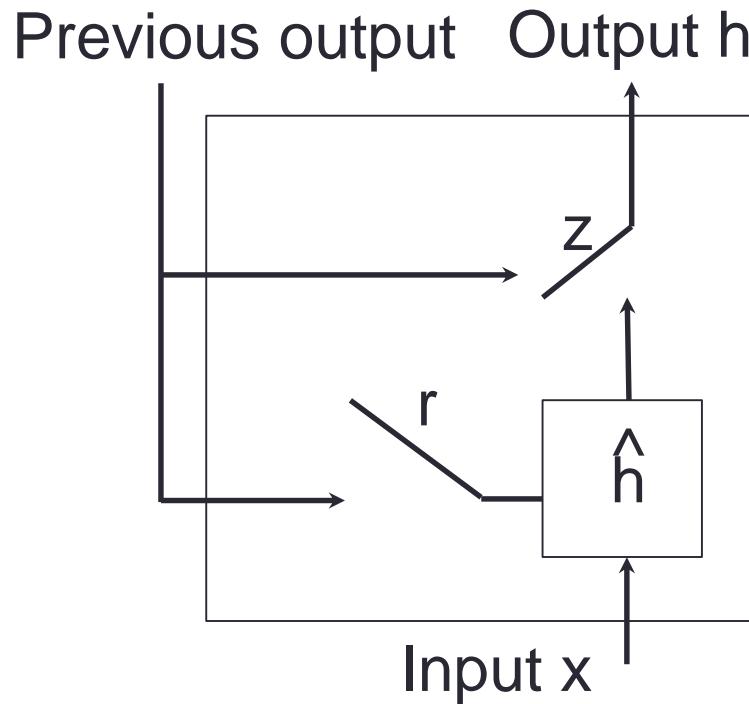
# Gated Recurrent Unit (GRU)



Neuron index  
time index

$$h_t^j = (1 - z_t^j)h_{t-1}^j + z_t^j \hat{h}_t^j$$

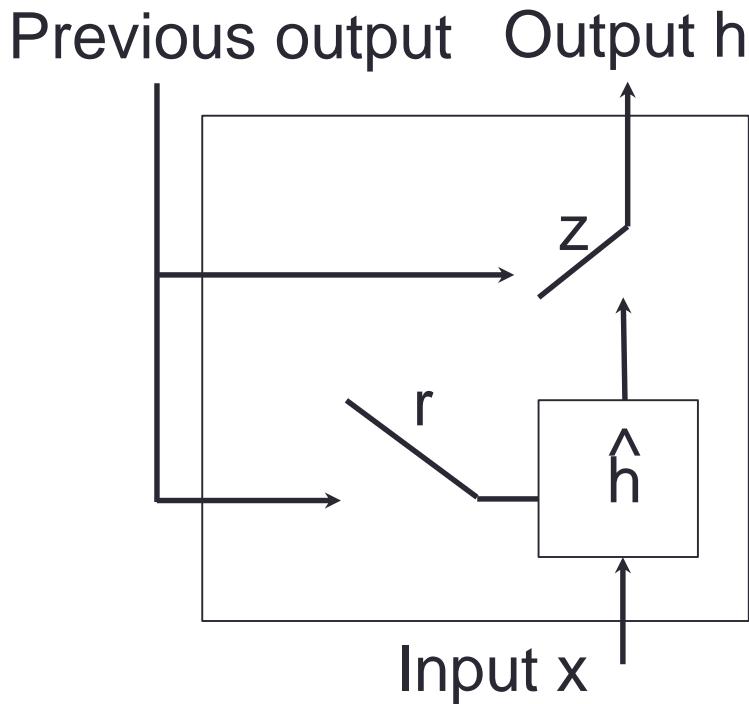
# Gated Recurrent Unit (GRU)



$$h_t^j = (1 - z_t^j)h_{t-1}^j + z_t^j \hat{h}_t^j$$

One GRU neuron output (scalar)

# Gated Recurrent Unit (GRU)



$$h_t^j = (1 - z_t^j)h_{t-1}^j + z_t^j \hat{h}_t^j$$

$$\hat{h}_t^j = \tanh^j (W \boxed{x_t} + U (\boxed{r_t \odot h_{t-1}}))$$

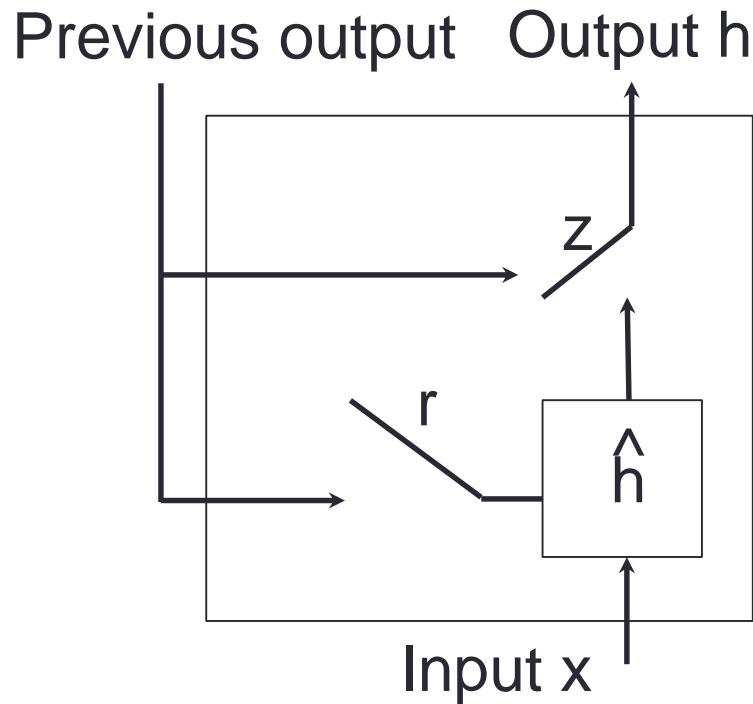
Element-wise product

Linear transform with matrix multiply

Vector (each value from each GRU unit in the previous layer)

$$x_t^j = h_t^j$$

# Gated Recurrent Unit (GRU)

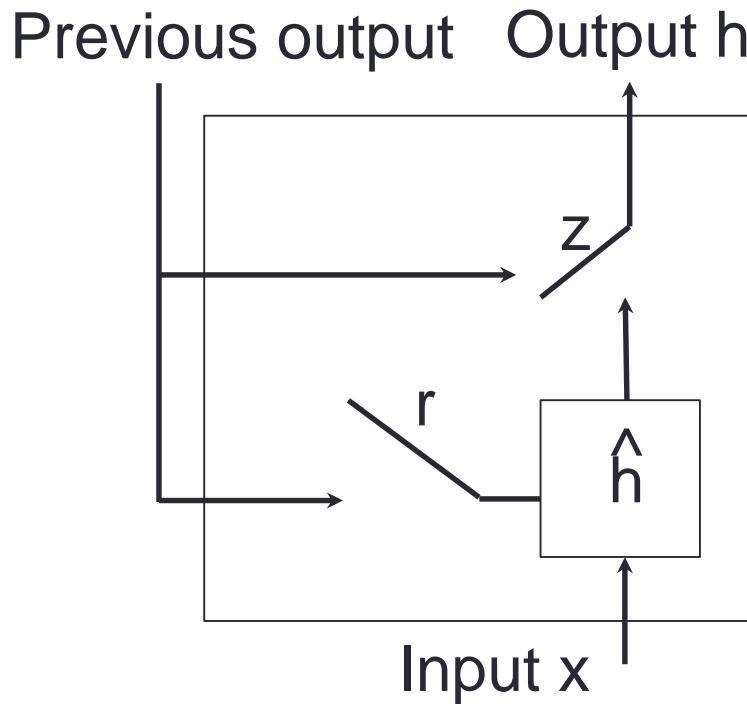


$$h_t^j = (1 - z_t^j)h_{t-1}^j + z_t^j \hat{h}_t^j$$

$$\hat{h}_t^j = \underline{\tanh^j}(W\mathbf{x}_t + U(\mathbf{r}_t \odot \mathbf{h}_{t-1}))$$

Takes the j-th  
element  
Bounds the output

# Gated Recurrent Unit (GRU)



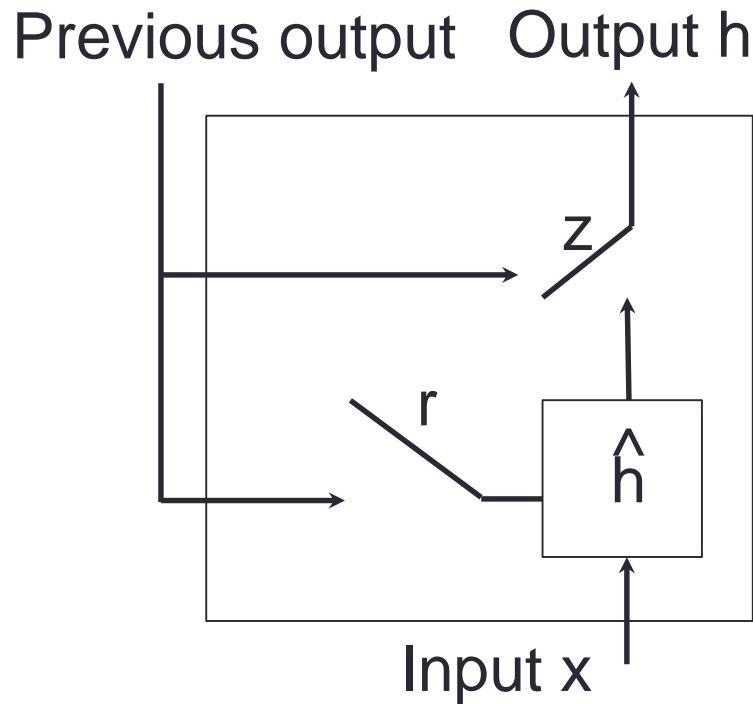
$$h_t^j = (1 - z_t^j)h_{t-1}^j + z_t^j \hat{h}_t^j$$

$$\hat{h}_t^j = \tanh^j(W \mathbf{x}_t + U(\mathbf{r}_t \odot \mathbf{h}_{t-1}))$$

$$z_t^j = \text{sigmoid}^j(W_z \mathbf{x}_t + U_z \mathbf{h}_{t-1})$$

Indicates a different set of weights

# Gated Recurrent Unit (GRU)



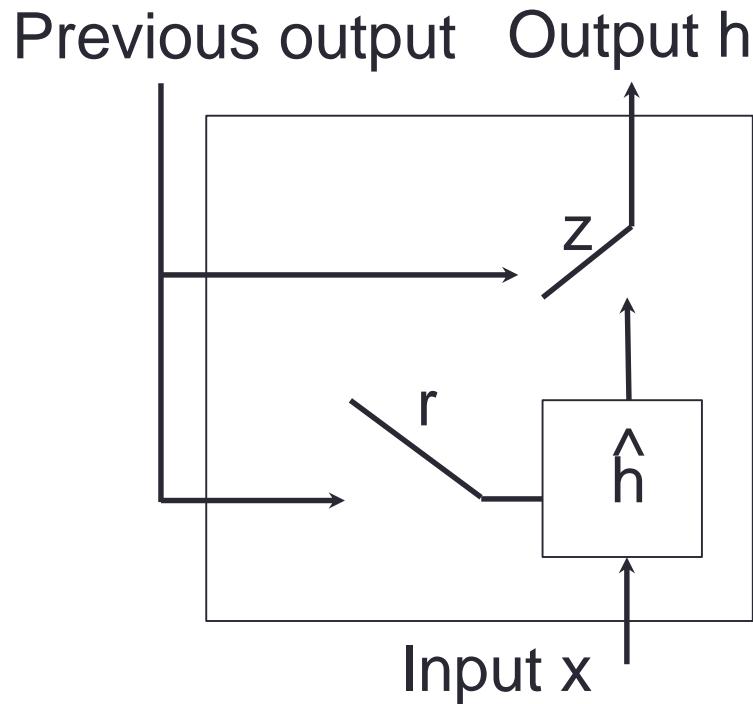
$$h_t^j = (1 - z_t^j)h_{t-1}^j + z_t^j \hat{h}_t^j$$

$$\hat{h}_t^j = \tanh^j(W\mathbf{x}_t + U(\mathbf{r}_t \odot \mathbf{h}_{t-1}))$$

$$z_t^j = \text{sigmoid}^j(W_z \mathbf{x}_t + U_z \mathbf{h}_{t-1})$$

Bounds the output to 0 to 1 for interpolation

# Gated Recurrent Unit (GRU)



$$h_t^j = (1 - z_t^j)h_{t-1}^j + z_t^j \hat{h}_t^j$$

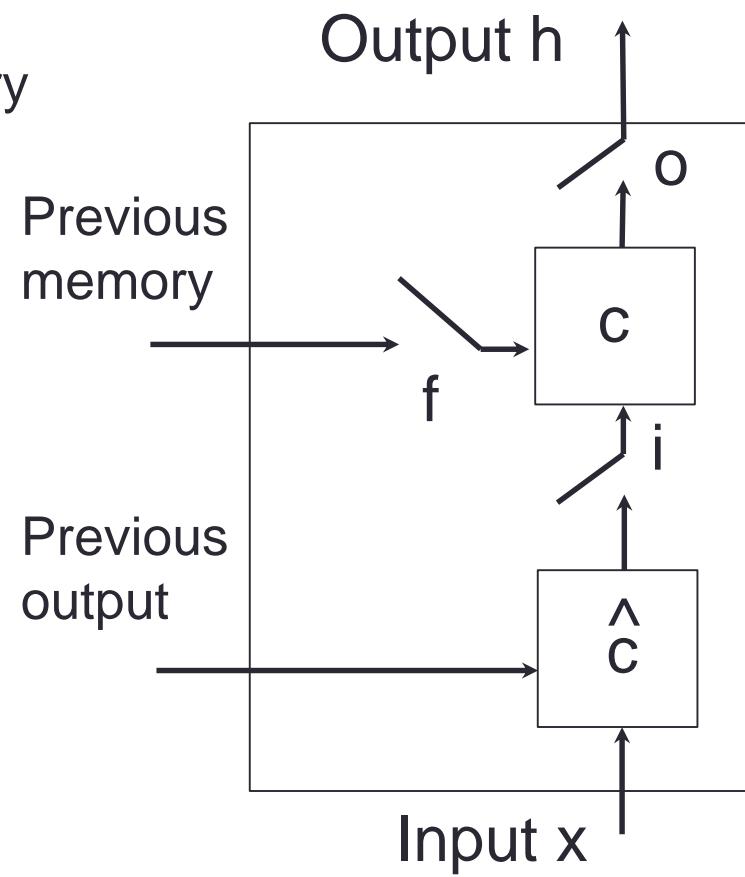
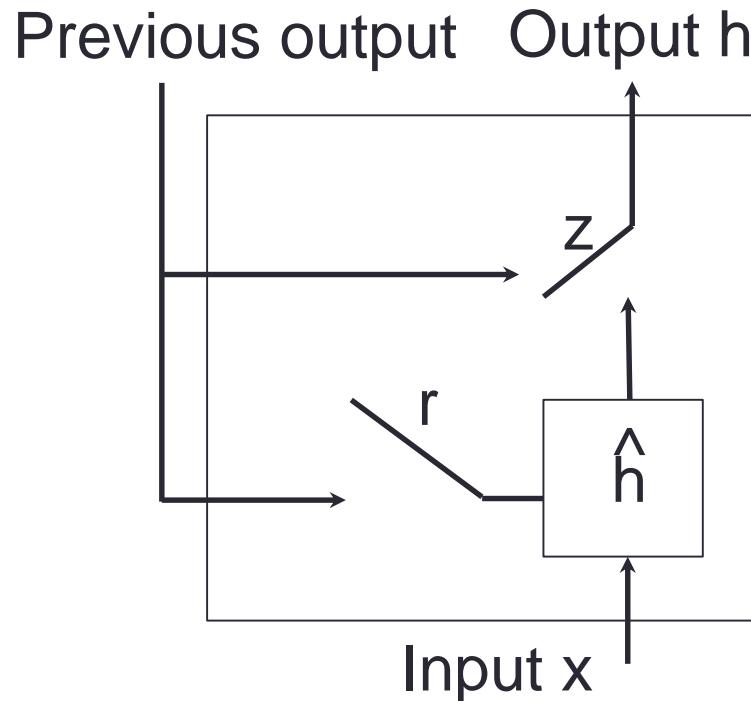
$$\hat{h}_t^j = \tanh^j(W\mathbf{x}_t + U(\mathbf{r}_t \odot \mathbf{h}_{t-1}))$$

$$z_t^j = \text{sigmoid}^j(W_z\mathbf{x}_t + U_z\mathbf{h}_{t-1})$$

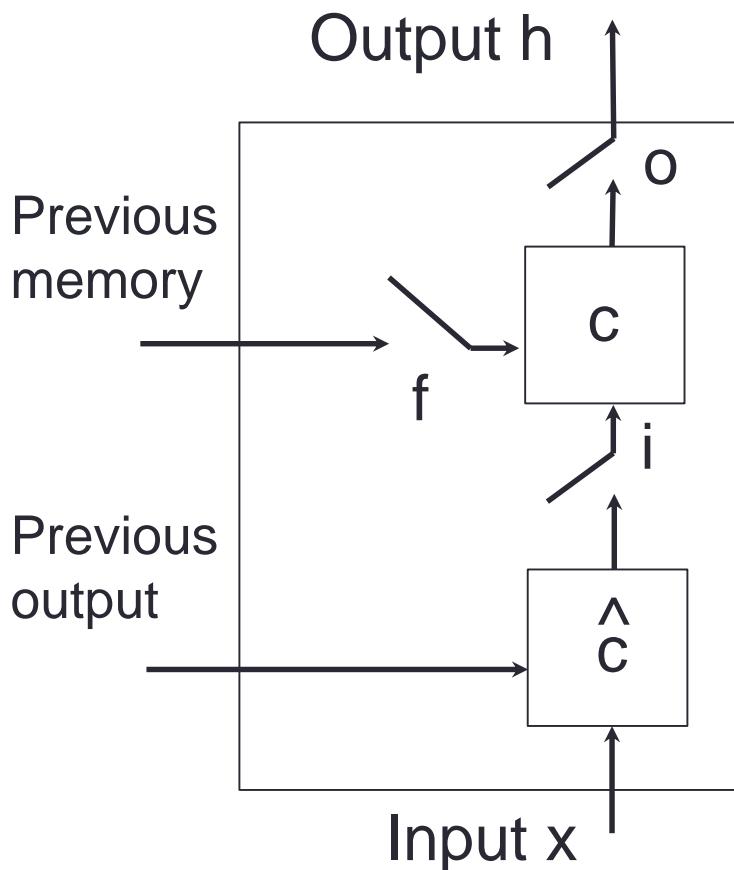
$$r_t^j = \text{sigmoid}^j(W_r\mathbf{x}_t + U_r\mathbf{h}_{t-1})$$

# Long Short-Term Memory (LSTM)

- Have 3 gates, forget ( $f$ ), input ( $i$ ), output ( $o$ )
- Has an **explicit memory cell** ( $c$ )
  - Does not have to output the memory



# Long Short-Term Memory (LSTM)

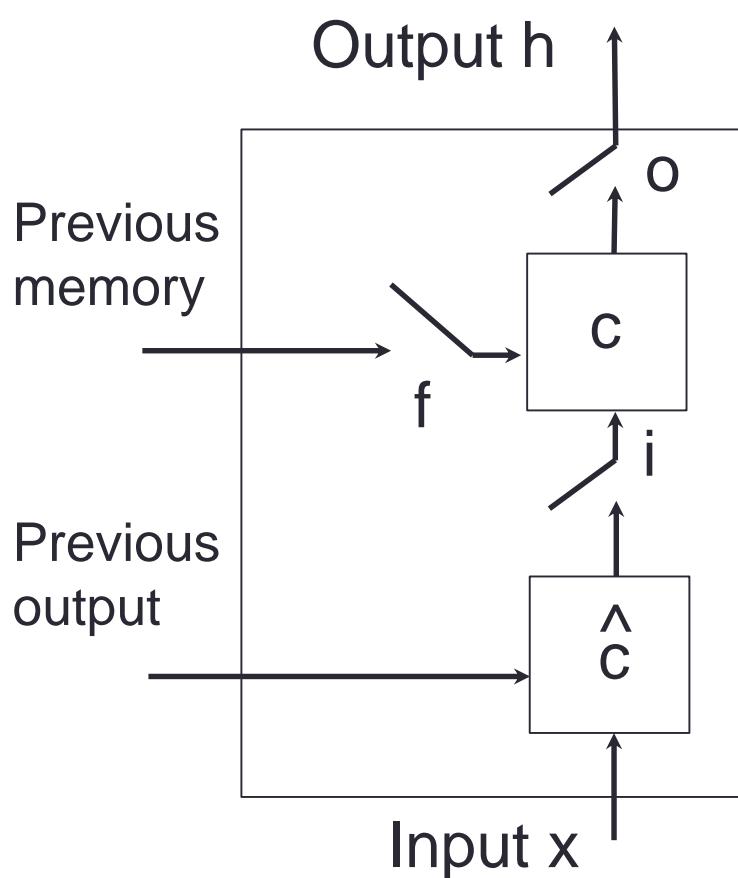


$$i_t^j = F^j(W_i \mathbf{x}_t + U_i \mathbf{h}_{t-1} + V_i \mathbf{c}_{t-1})$$
$$o_t^j = F^j(W_o \mathbf{x}_t + U_o \mathbf{h}_{t-1} + V_o \mathbf{c}_t)$$
$$f_t^j = F^j(W_f \mathbf{x}_t + U_f \mathbf{h}_{t-1} + V_j \mathbf{c}_{t-1})$$

Contribution from memory “Peephole connection”

Vs are diagonal matrices(Each cell can only see its own memory)

# Long Short-Term Memory (LSTM)



$$i_t^j = F^j(W_i \mathbf{x}_t + U_i \mathbf{h}_{t-1} + V_i \mathbf{c}_{t-1})$$

$$o_t^j = F^j(W_o \mathbf{x}_t + U_o \mathbf{h}_{t-1} + V_o \mathbf{c}_t)$$

$$f_t^j = F^j(W_f \mathbf{x}_t + U_f \mathbf{h}_{t-1} + V_f \mathbf{c}_{t-1})$$

$$h_t^j = o_t^j \tanh(c_t^j)$$

$$c_t^j = f_t^j c_{t-1}^j + i_t^j \hat{c}_t^j$$

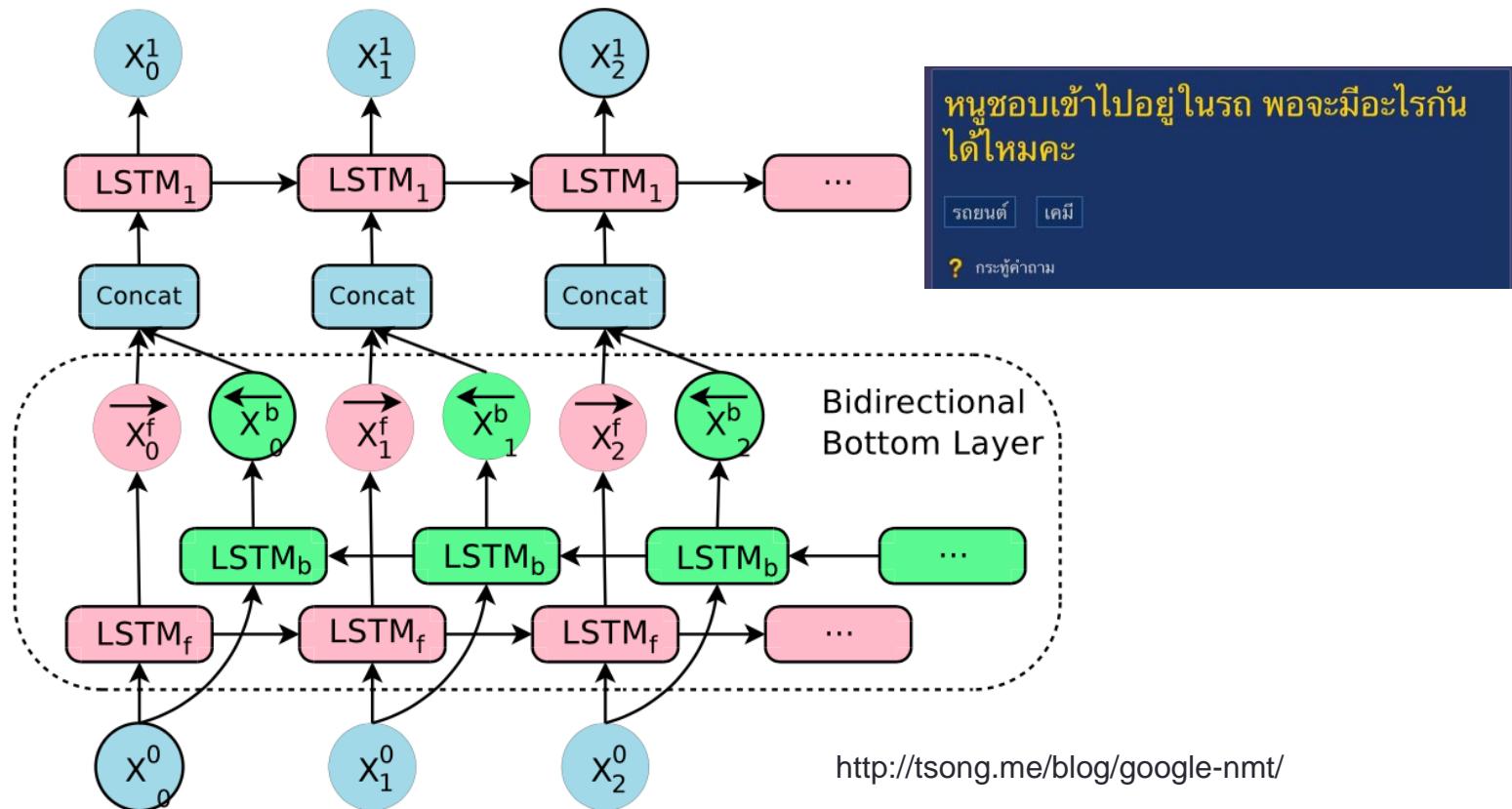
$$\hat{c}_t^j = \tanh^j(W_c \mathbf{x}_t + U_c \mathbf{h}_{t-1})$$

# GRU vs LSTM

- GRU and LSTM offers the same performance with large dataset
  - GRU better for smaller dataset (less parameters)
  - GRU faster to train and faster runtime (smaller model)
- Use GRUs!

# Bi-directional LSTM

- The previous GRU/LSTM only goes backward in time (uni-directional)
- Most of the time information from the future is useful for predicting the current output



# LSTM remembers meaningful things

Cell sensitive to position in line:

The sole importance of the crossing of the Berezina lies in the fact that it plainly and indubitably proved the fallacy of all the plans for cutting off the enemy's retreat and the soundness of the only possible line of action--the one Kutuzov and the general mass of the army demanded--namely, simply to follow the enemy up. The French crowd fled at a continually increasing speed and all its energy was directed to reaching its goal. It fled like a wounded animal and it was impossible to block its path. This was shown not so much by the arrangements it made for crossing as by what took place at the bridges. When the bridges broke down, unarmed soldiers, people from Moscow and women with children who were with the French transport, all--carried on by vis inertiae--pressed forward into boats and into the ice-covered water and did not, surrender.

Cell that turns on inside quotes:

"You mean to imply that I have nothing to eat out of.... On the contrary, I can supply you with everything even if you want to give dinner parties," warmly replied Chichagov, who tried by every word he spoke to prove his own rectitude and therefore imagined Kutuzov to be animated by the same desire.

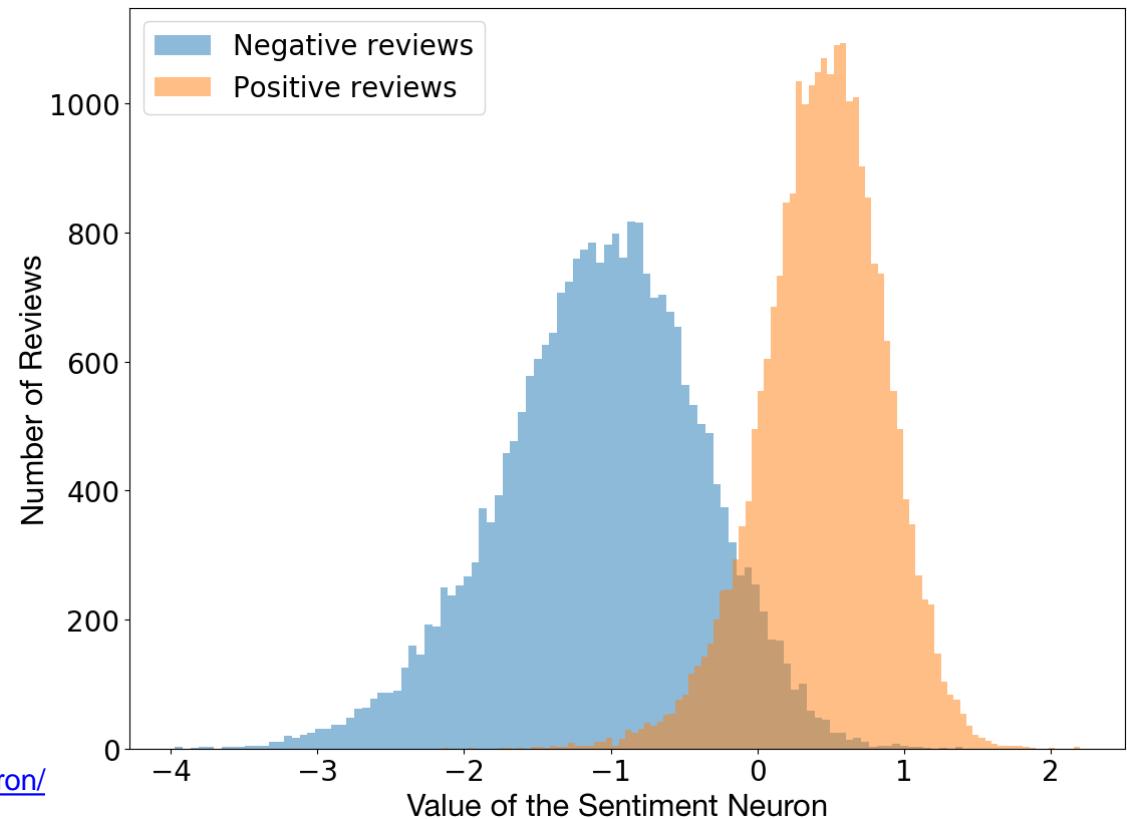
Kutuzov, shrugging his shoulders, replied with his subtle penetrating smile: "I meant merely to say what I said."

# Sentiment Neuron

Trained a LSTM network to predict next character from raw amazon reviews text (82 million reviews)

LSTM with 4096 hidden units

One unit seems to be learning sentiment



# Sentiment neuron

This is one of Crichton's best books. The characters of Karen Ross, Peter Elliot, Munro, and Amy are beautifully developed and their interactions are exciting, complex, and fast-paced throughout this impressive novel. And about 99.8 percent of that got lost in the film. Seriously, the screenplay AND the directing were horrendous and clearly done by people who could not fathom what was good about the novel. I can't fault the actors because frankly, they never had a chance to make this turkey live up to Crichton's original work. I know good novels, especially those with a science fiction edge, are hard to bring to the screen in a way that lives up to the original. But this may be the absolute worst disparity in quality between novel and screen adaptation ever. The book is really, really good. The movie is just dreadful.

# Embeddings

- A way to encode information to a lower dimensional space
  - PCA
  - We learn about this lower dimensional space through data

# One hot encoding

- Categorical representation is usually represented by one hot encoding
- Categorical representations examples:
  - Words in a vocabulary, characters in Thai language

Apple -> 1 -> [1, 0, 0, 0, ...]

Bird -> 2 -> [0, 1, 0, 0, ...]

Cat -> 3 -> [0, 0, 1, 0, ...]

- Sparse representation
  - Sparse means most dimension are zero

# One hot encoding

- Sparse – but lots of dimension
  - Curse of dimensionality
- Does not represent meaning.

Apple -> 1 -> [1, 0, 0, 0, ...]

Bird -> 2 -> [0, 1, 0, 0, ...]

Cat -> 3 -> [0, 0, 1, 0, ...]

$$|\text{Apple} - \text{Bird}| = |\text{Bird} - \text{Cat}|$$

# Getting meaning into the feature vectors

- You can add back meanings by hand-crafted rules
- Old-school NLP is all about feature engineering
- Word segmentation example:
  - Cluster Numbers
  - Cluster letters
- Concatenate them
- $1 = [0 \ 0 \ 0 \ 0 \ 1 \ 0 \ 0 \ 0, \ 1, \ 0]$
- $n = [0 \ 0 \ 0 \ 1 \ 0 \ 0 \ 0 \ 0, \ 0, \ 1]$
- $\gamma = [1 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0, \ 0, \ 2]$
- Which rules to use?
  - Try as many as you can think of, and do feature selection or use models that can do feature selection

# Dense representation

- We can encode sparse representation into a lower dimensional space
  - $F: \mathbb{R}^N \rightarrow \mathbb{R}^M$ , where  $N > M$

Apple -> 1 -> [1, 0, 0, 0, ...] -> [2.3, 1.2]

Bird -> 2 -> [0, 1, 0, 0, ...] -> [-1.0, 2.4]

Cat -> 3 -> [0, 0, 1, 0, ...] -> [-3.0, 4.0]

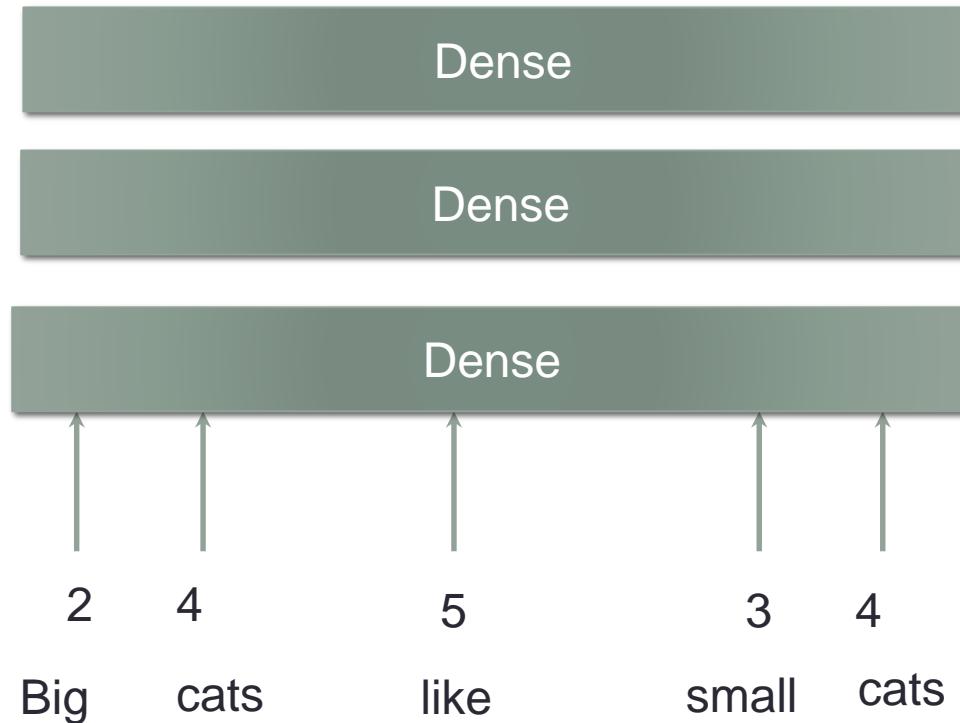
- We can do this by using an embedding layer

# Sentiment analysis with fully connected networks

1 = positive, 0 = negative



Logistic function



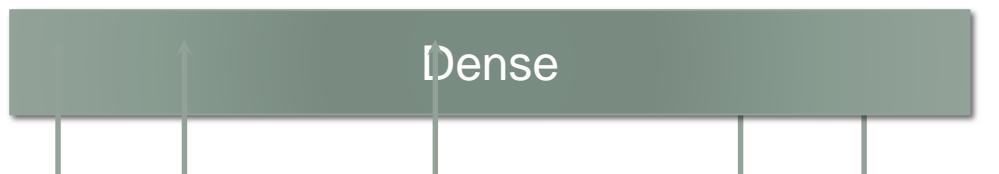
# Adding embedding layer



Embedding layer  
shares the same  
weights



Parameter sharing!



[1, -1] [3, -2] [5.3, -2.1] [1, 1] [3, -2]



More on embeddings  
in the next two  
lectures!

Big cats like small cats

# Embedding Projector

## DATA

5 tensors found

Word2Vec 10K

Label by

word

Color by

No color map

Edit by

word

Tag selection as

Load

Publish

Download

Label

Sphereize data [?](#)

Checkpoint: Demo datasets

Metadata: oss\_data/word2vec\_10000\_200d

UMAP

T-SNE

PCA

CUSTOM

Dimension

2D



3D

Neighbors [?](#)



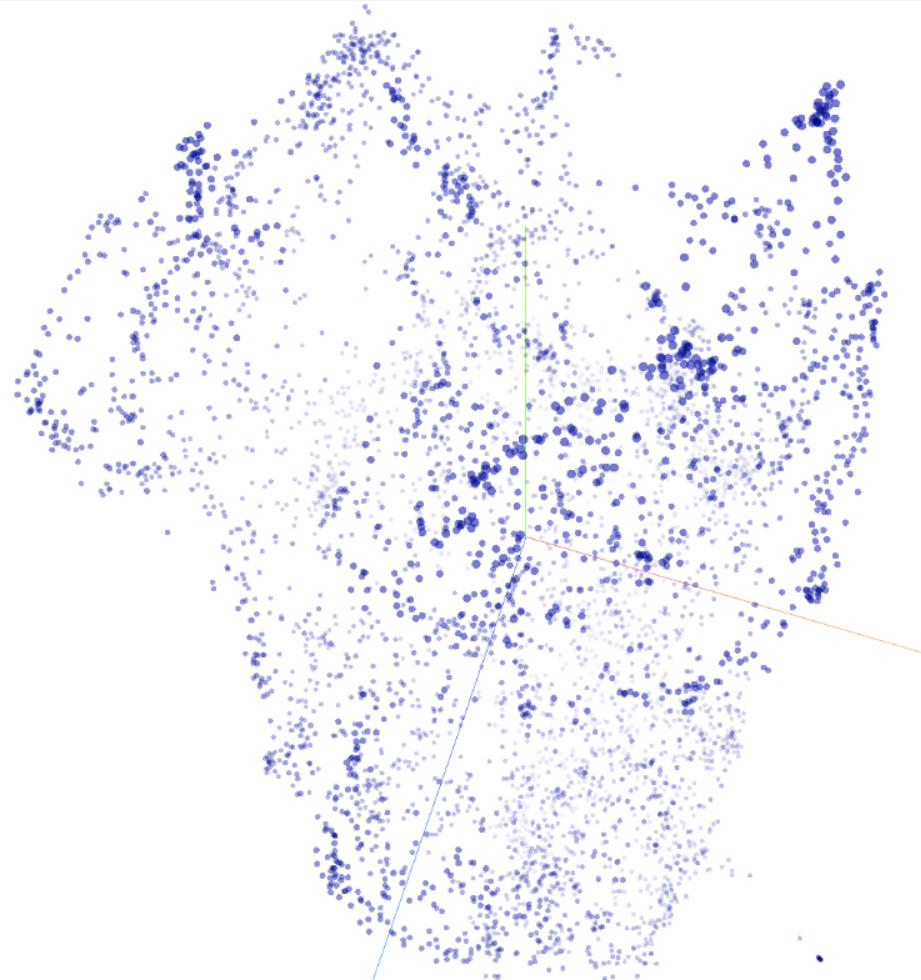
15

Run

For faster results, the data will be sampled down to 5,000 points.

 [Learn more about UMAP.](#)

 | Points: 10000 | Dimension: 200

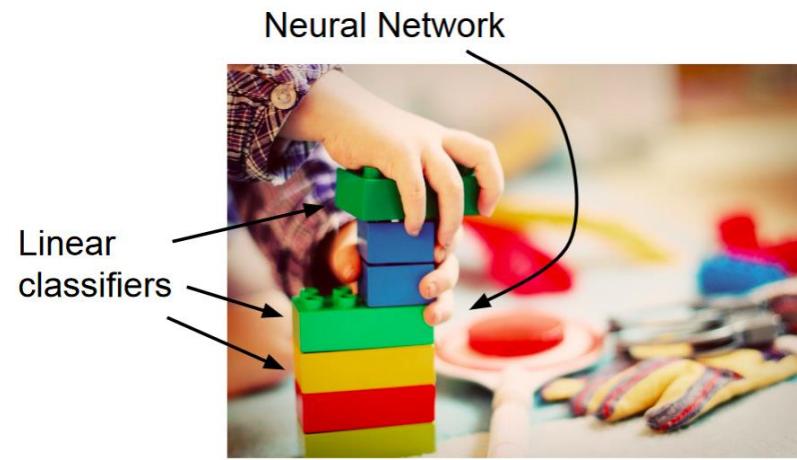


# Embedding vs PCA

- PCA – unsupervised method
- Embedding – can be trained supervised
  - Embedding should be superior with the task
- PCA – linear
- Embedding – potentially non-linear
  - Should be more powerful
- You can learn an embedding on one task (with lots of data) and use it on another task
  - Embedding learns meaningful feature representations

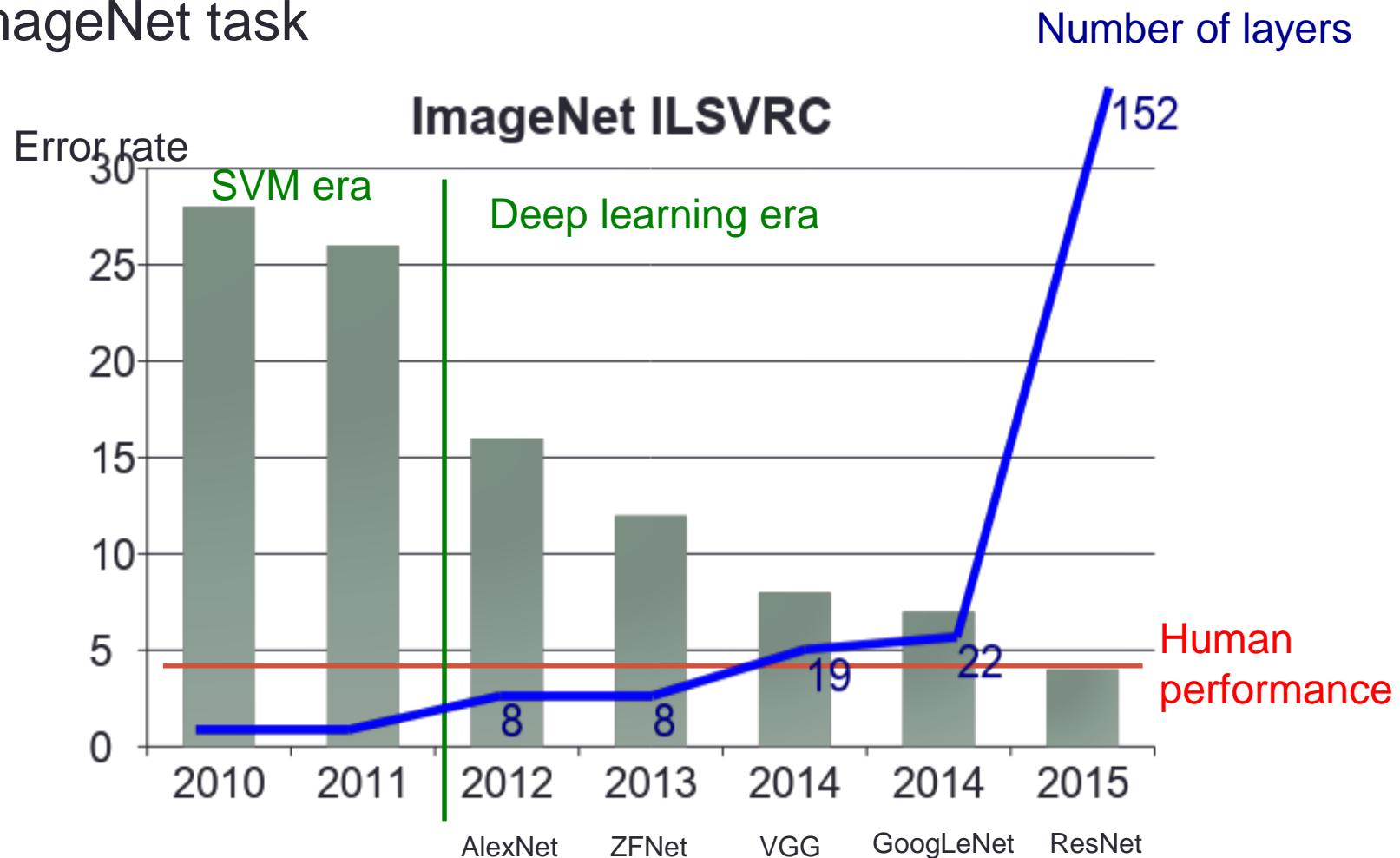
# DNN Legos

- Typical models now consists of all 3 types
  - CNN: local structure in the feature. Used for feature learning.
  - LSTM: remembering longer term structure or across time
  - DNN: Good for mapping features for classification. Usually used in final layers
- DNN structures encode inductive bias



# A brief history of imagenet architectures

- ImageNet task

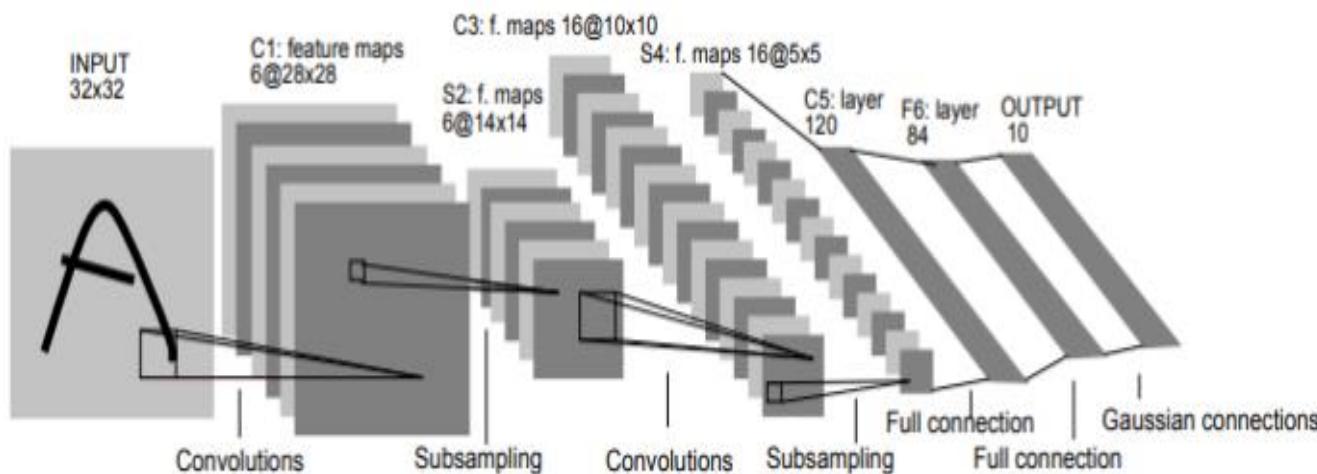


# LeNet

Convolutions and poolings followed by fully connected layers

Tanh activations

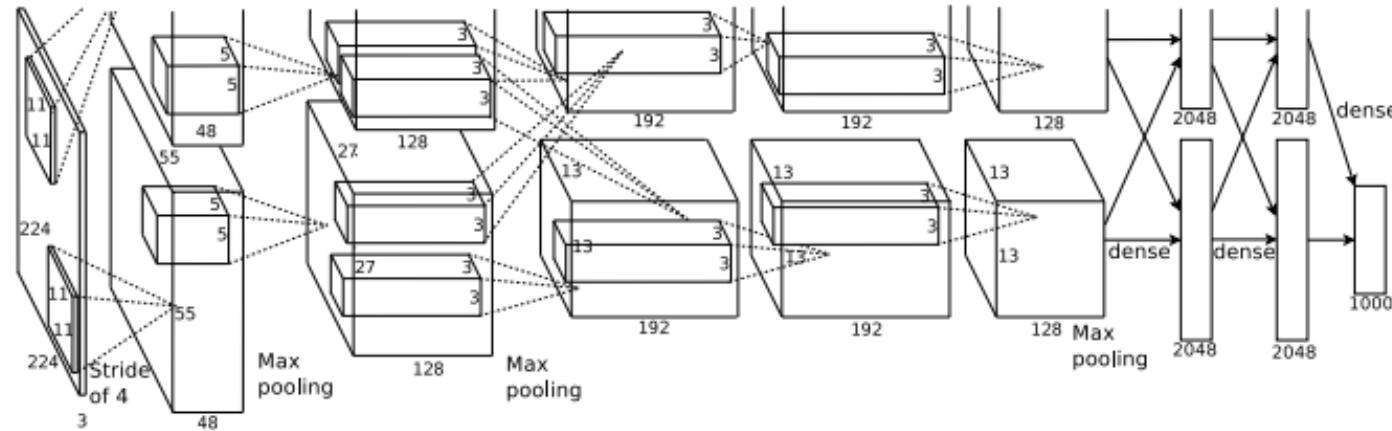
Ability to handle larger images limited by compute



Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. "Gradient-based learning applied to document recognition," 1998.

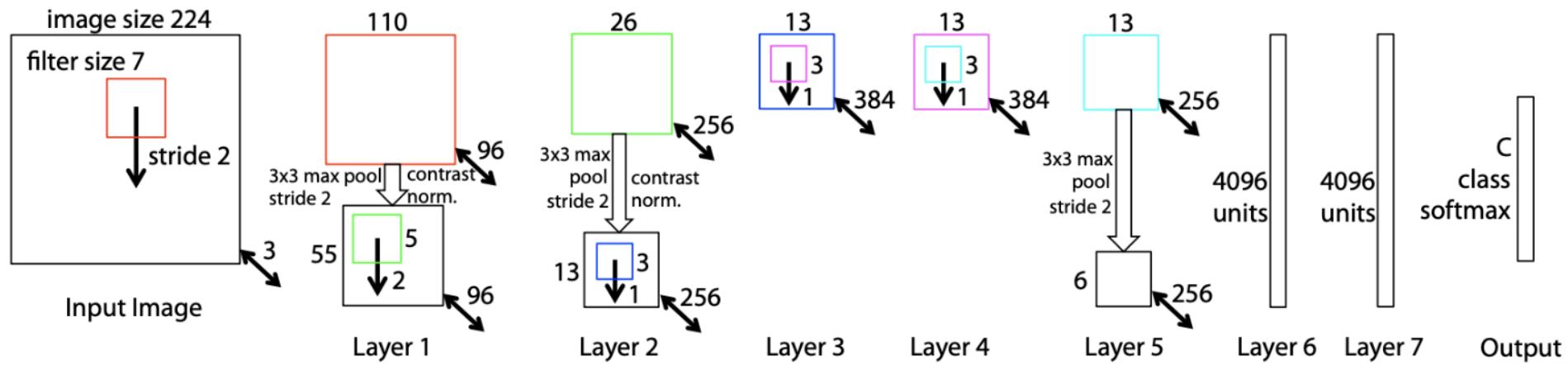
# AlexNet

Convolutions, max pooling, dropout, data augmentation, ReLU activations, SGD with momentum  
Two pipelines to fit into two GPUs



# ZFNet

## Tweaking hyperparameters



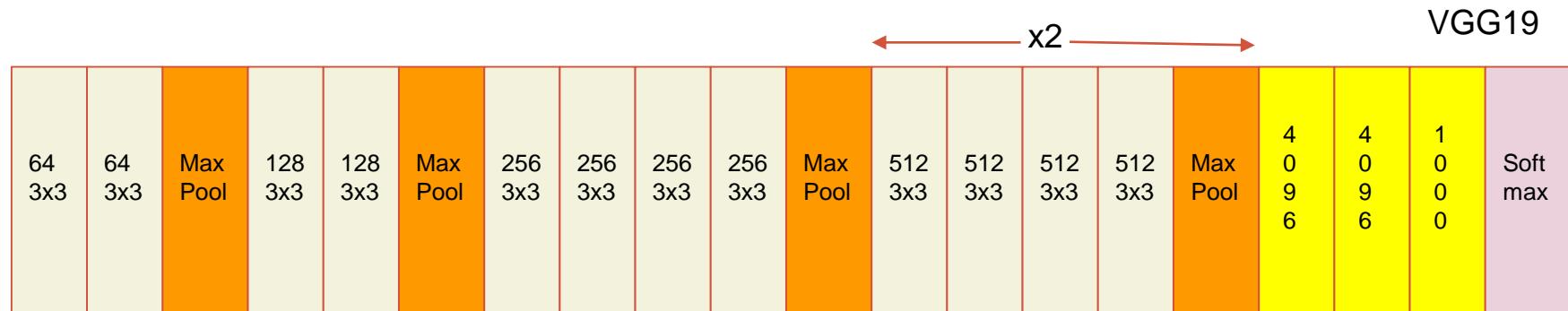
Matthew D. Zeiler, Rob Fergus, "Visualizing and Understanding Convolutional Networks," 2013

# VGG

Uniform 3x3 convolutional filters

19 layers! Pushing the limits of conventional wisdom at that time.

Used by many since pre-train weights are publically available

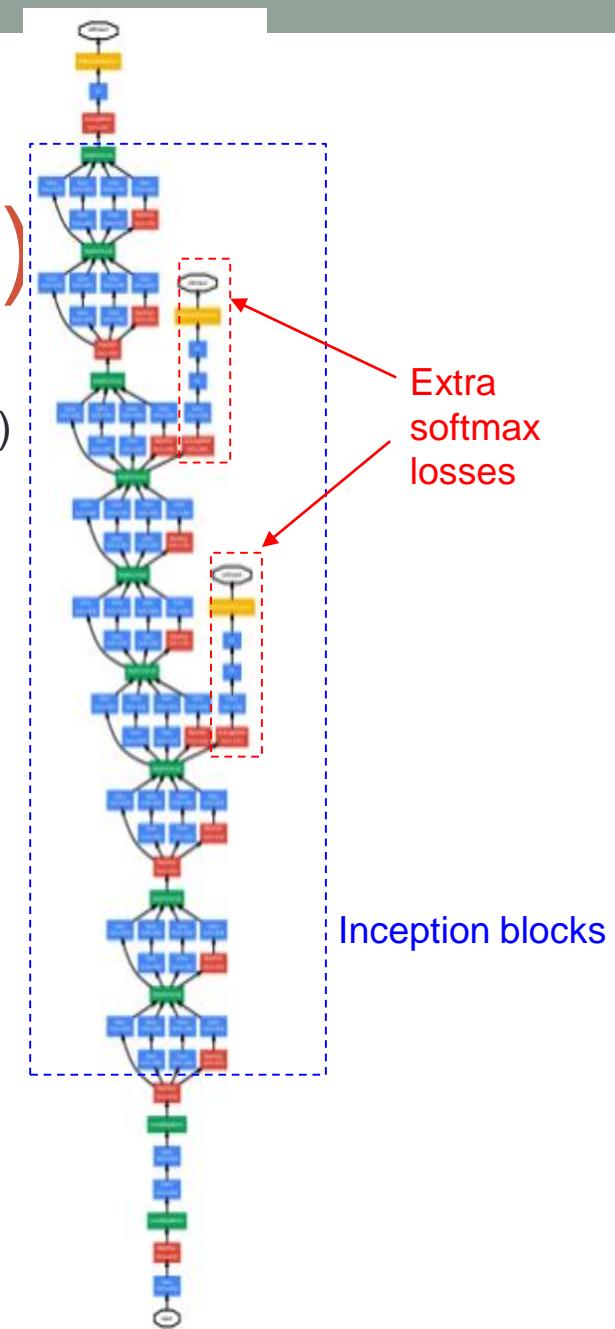
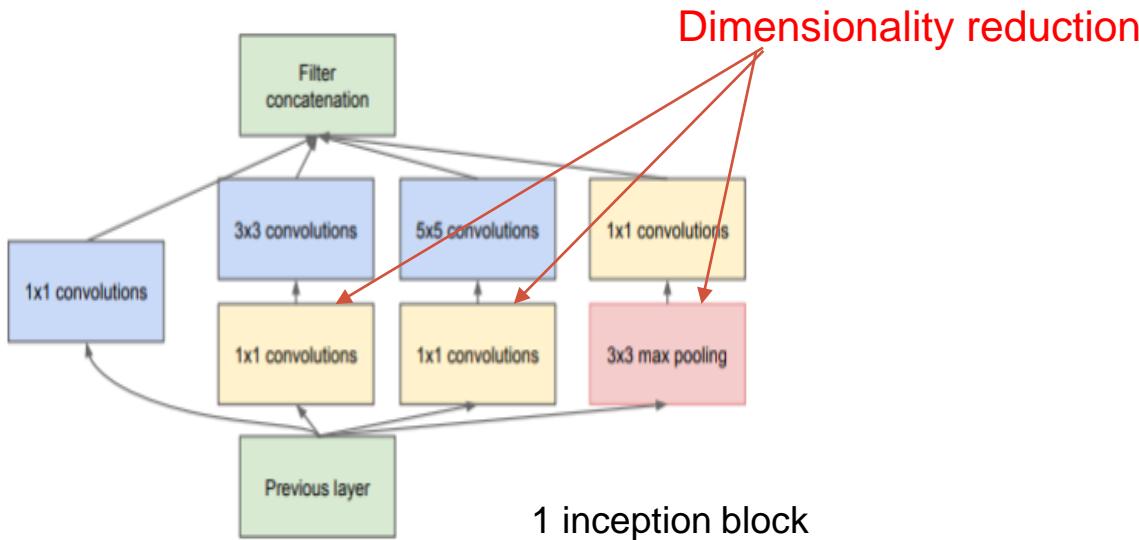


# GoogLeNet (Inception v1)

Multiple filter sizes per layer (objects come in different scales)

Dimensionality reduction via 1x1 convolution

Multiple softmax losses to help the gradient problem

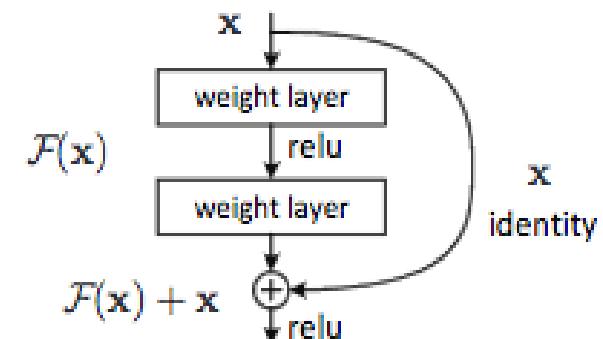


# Vanishing/Exploding gradient

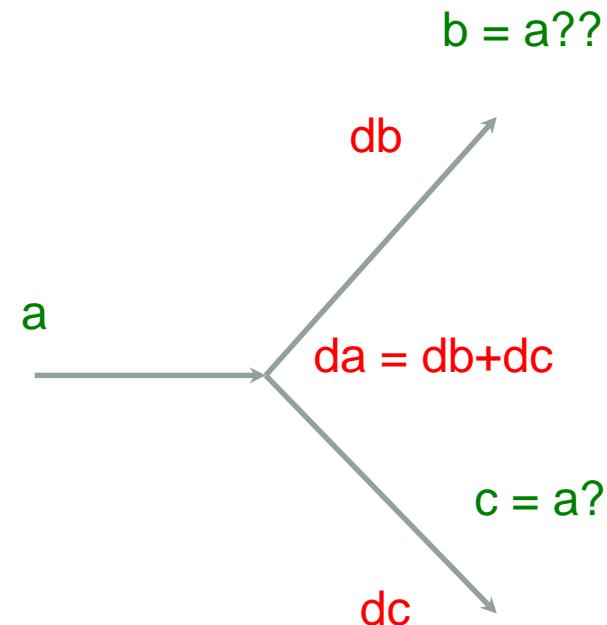
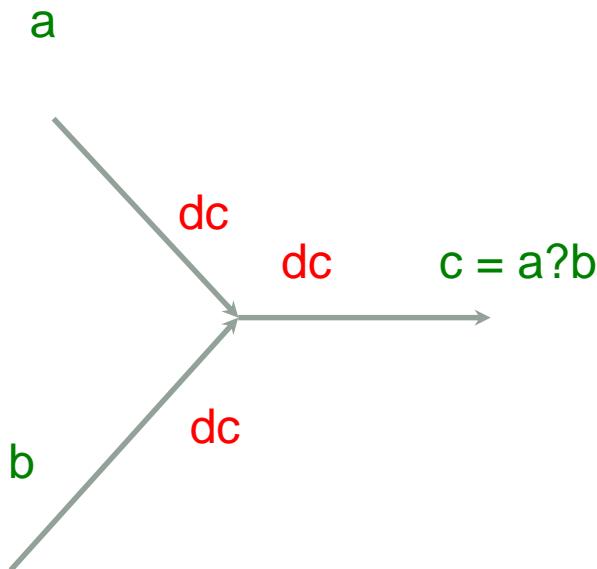
- Backprop introduces many multiplications down chain
- The gradient value gets smaller and smaller
  - The deeper the network the smaller the gradient in the lower layers
  - Lower layers changes too slowly (or not at all)
  - Hard to train very deep networks (>6 layers)
- The opposite can also be true. The gradient explodes from repeated multiplication
  - Put a maximum value for the gradient (Gradient clipping)

- How to deal with this?
  - Residual connection

<https://arxiv.org/abs/1512.03385>



# Gradient flow at forks

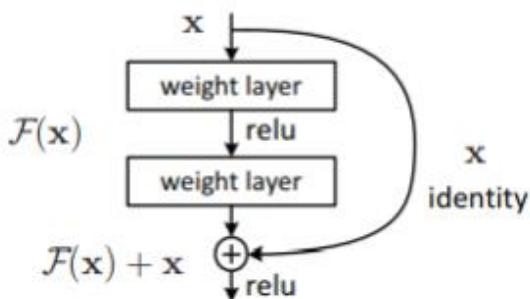


Forward and backward pass acts differently at forks

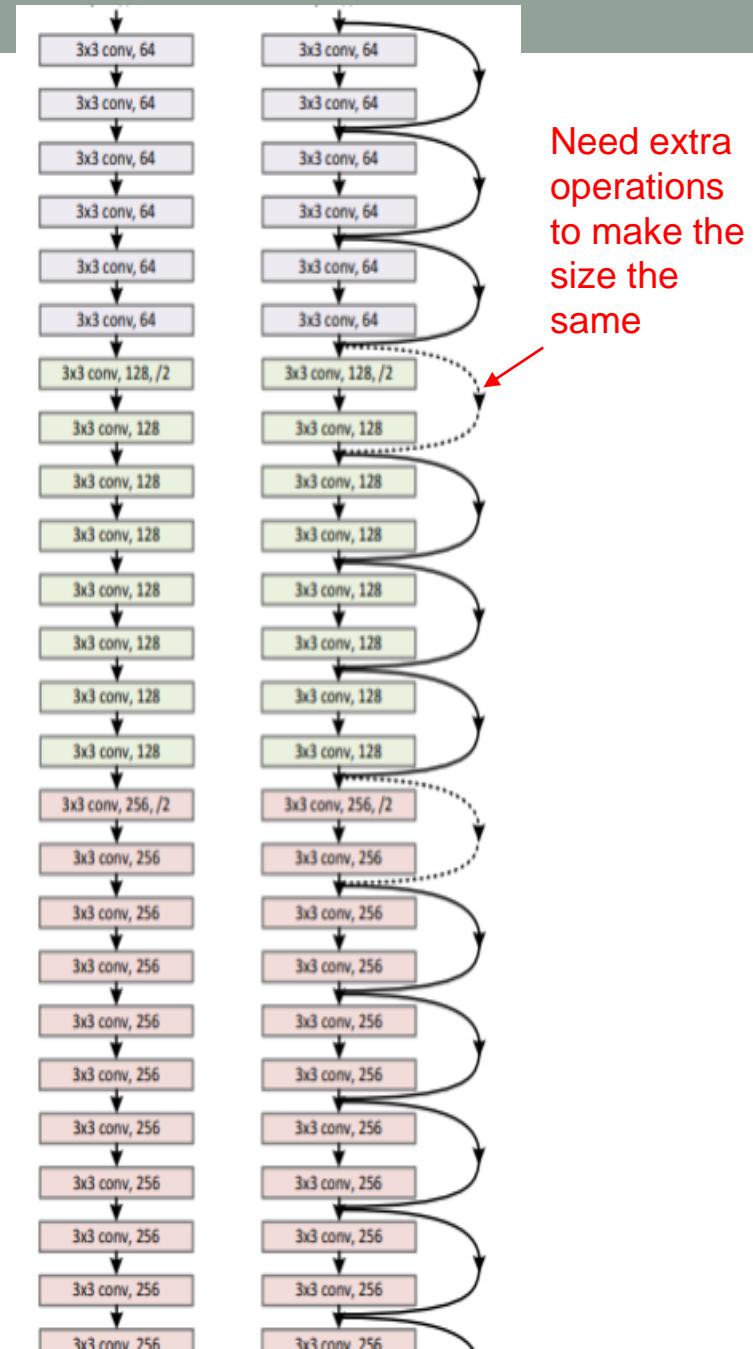
# ResNet (Residual Network)

Batch norm

Extra “skip connections” to reduce the vanishing gradient problem

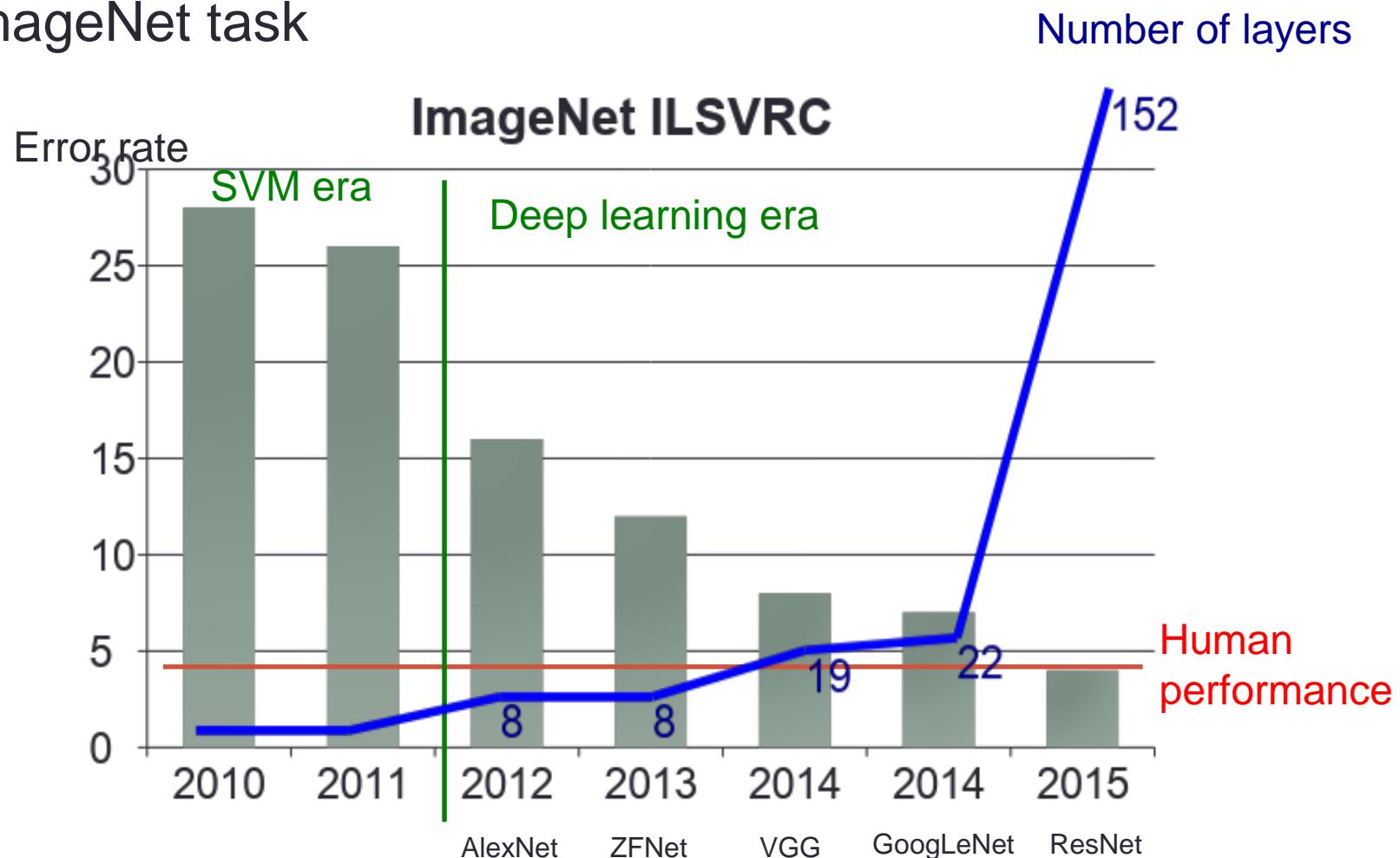


Kaiming He, et al. “Deep Residual Learning for Image Recognition” 2015



# A brief history of imagenet architectures

- ImageNet task



# Inception v2+v3

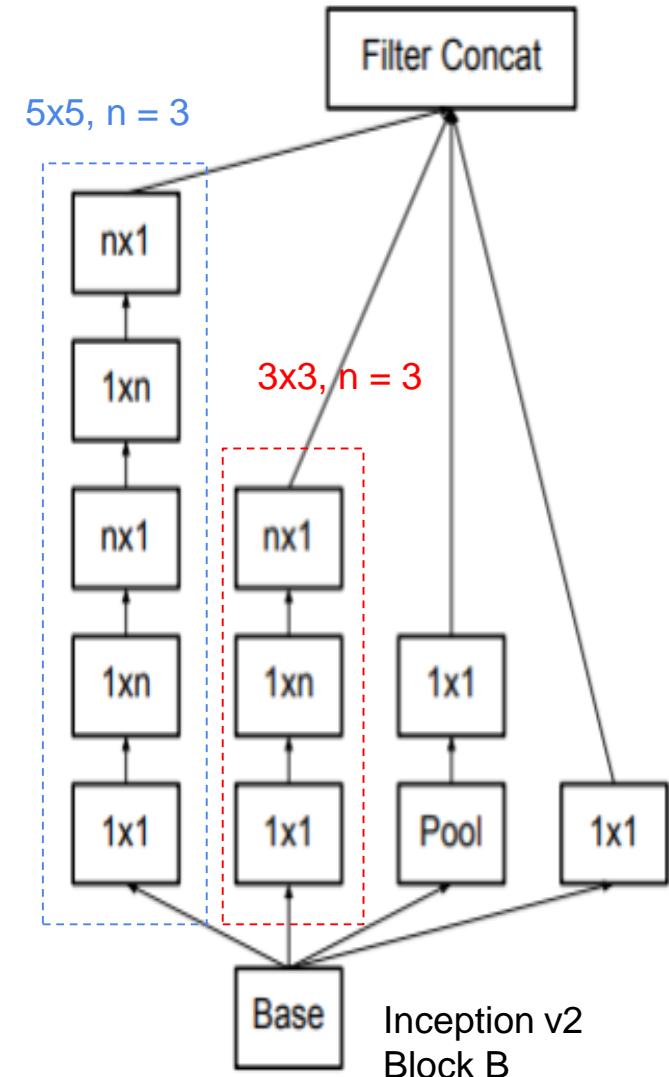
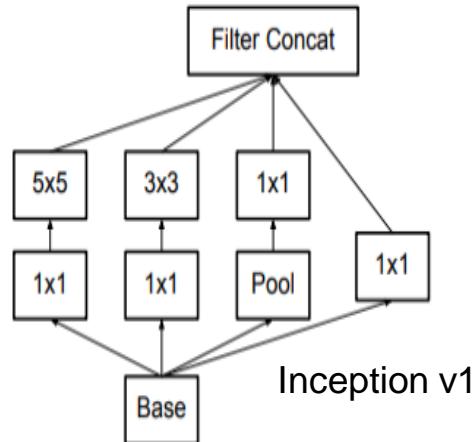
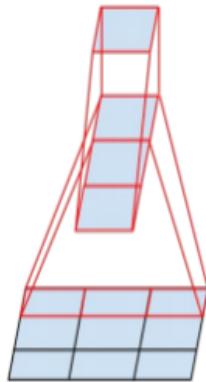
Implement 5x5 with two 3x3s

Factorized convolution

3x3 -> 3x1 and 1x3

3 types of inception blocks

RMSprop, Batch norm, label smoothing



# Batch normalization

- Recent technique for (implicit) regularization
- **Normalize every mini-batch** at various batch norm layers to standard Gaussian (different from global normalization of the inputs)
- Place batch norm layers before (or after?) non-linearities
- Faster training (put numbers in expected range – can use larger learning rates) and better generalizations

For each mini-batch that goes through  
batch norm

1. Normalize by the mean and variance of the mini-batch for each dimension
2. Shift and scale by learnable parameters

Replaces dropout in some networks  
<https://arxiv.org/abs/1502.03167>

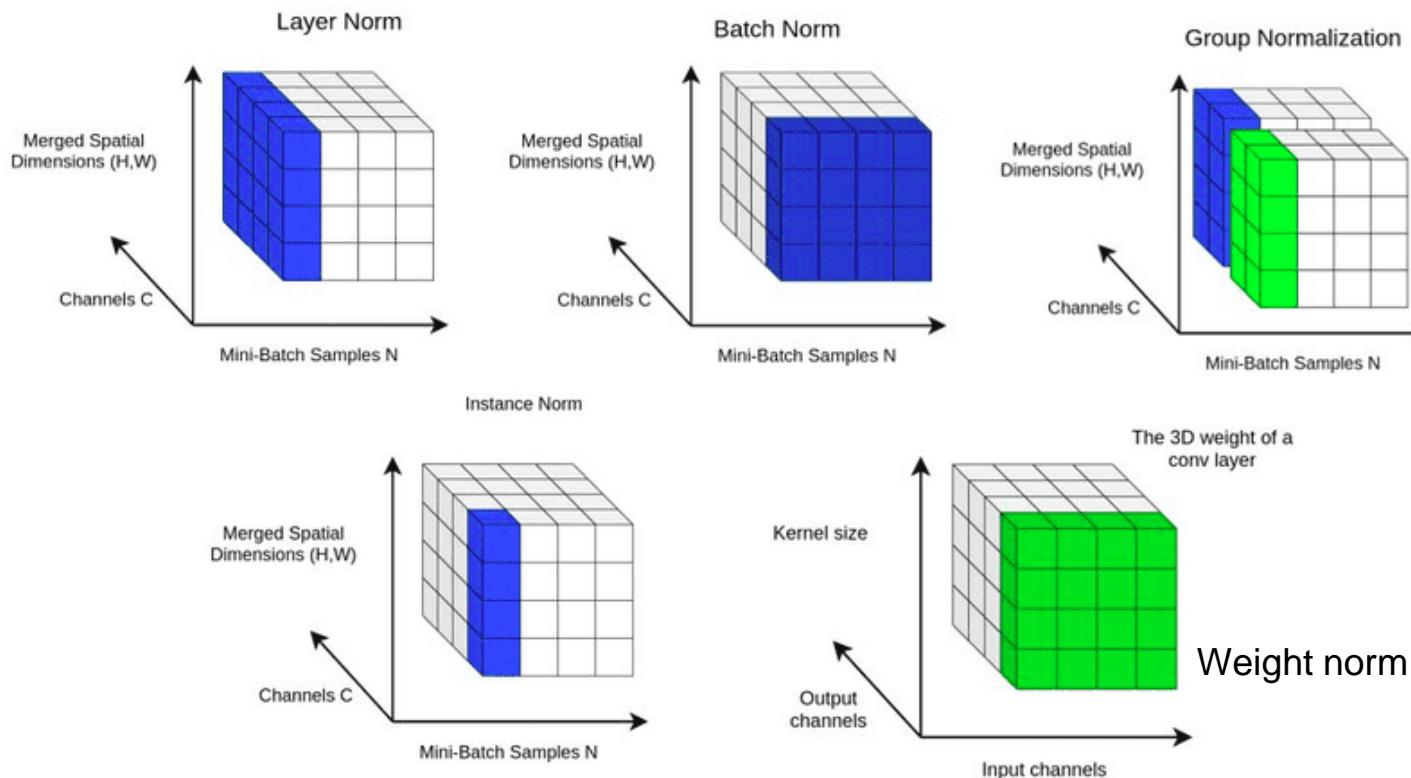
$$\hat{x} = \frac{x - \mu_b}{\sigma_b}$$

$$y = \alpha \hat{x} + \beta$$

Mean changes every minibatch – a form of noise (regularization)

# Other normalizations

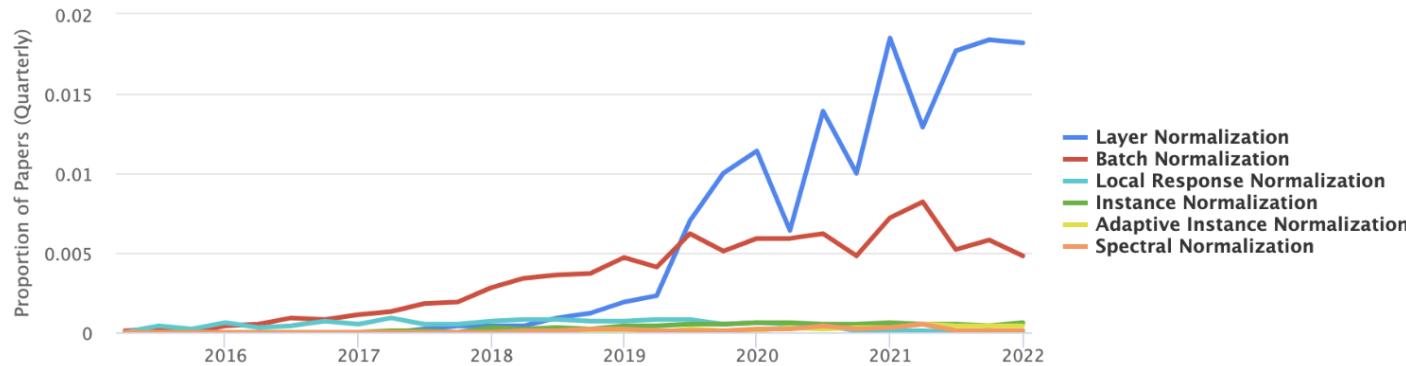
- Other normalizations are out there



# Other normalizations

Trend of normalization used in papers

## Usage Over Time



⚠ This feature is experimental; we are continuously improving our matching algorithm.

NLP – layer norm

Image – Batch norm/layer norm

Small batches - Group norm (batch norm is not reliable in small batches)

# Training and inference of batch norm

## BATCHNORM2D

```
CLASS torch.nn.BatchNorm2d(num_features, eps=1e-05, momentum=0.1, affine=True,  
    track_running_stats=True, device=None, dtype=None) [SOURCE]
```

Applies Batch Normalization over a 4D input (a mini-batch of 2D inputs with additional channel dimension) as described in the paper [Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift](#).

$$y = \frac{x - \mathbb{E}[x]}{\sqrt{\text{Var}[x] + \epsilon}} * \gamma + \beta$$

The mean and standard-deviation are calculated per-dimension over the mini-batches and  $\gamma$  and  $\beta$  are learnable parameter vectors of size C (where C is the input size). By default, the elements of  $\gamma$  are set to 1 and the elements of  $\beta$  are set to 0. The standard-deviation is calculated via the biased estimator, equivalent to `torch.var(input, unbiased=False)`.

Also by default, during training this layer keeps running estimates of its computed mean and variance, which are then used for normalization during evaluation. The running estimates are kept with a default `momentum` of 0.1.

If `track_running_stats` is set to `False`, this layer then does not keep running estimates, and batch statistics are instead used during evaluation time as well.

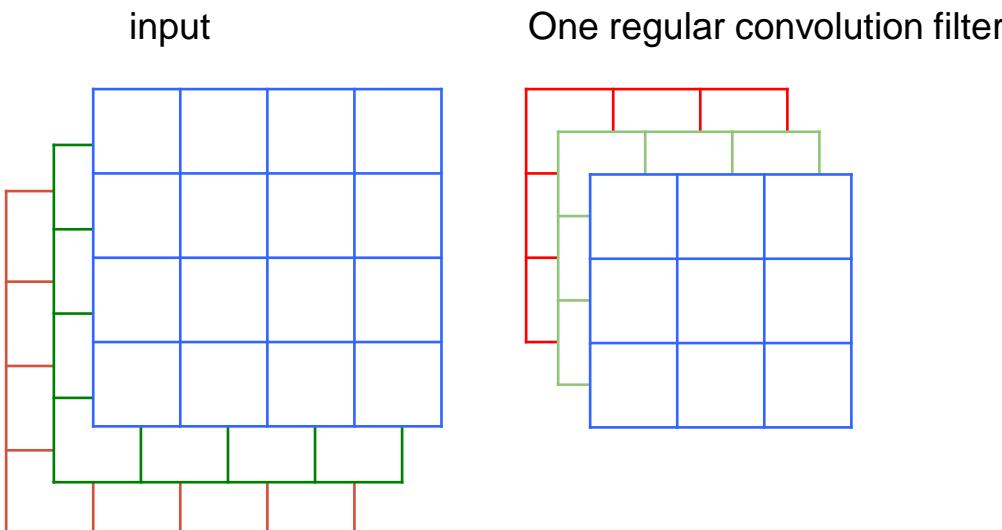
# Xception

## Depthwise separable convolution: two-step convolution

1. Depthwise convolution
2. 1x1 convolution

Typical convolution 3x3 filter is  
3x3xinput channel

Depthwise convolution 3x3 filter is  
3x3x1  
1 filter per input channel



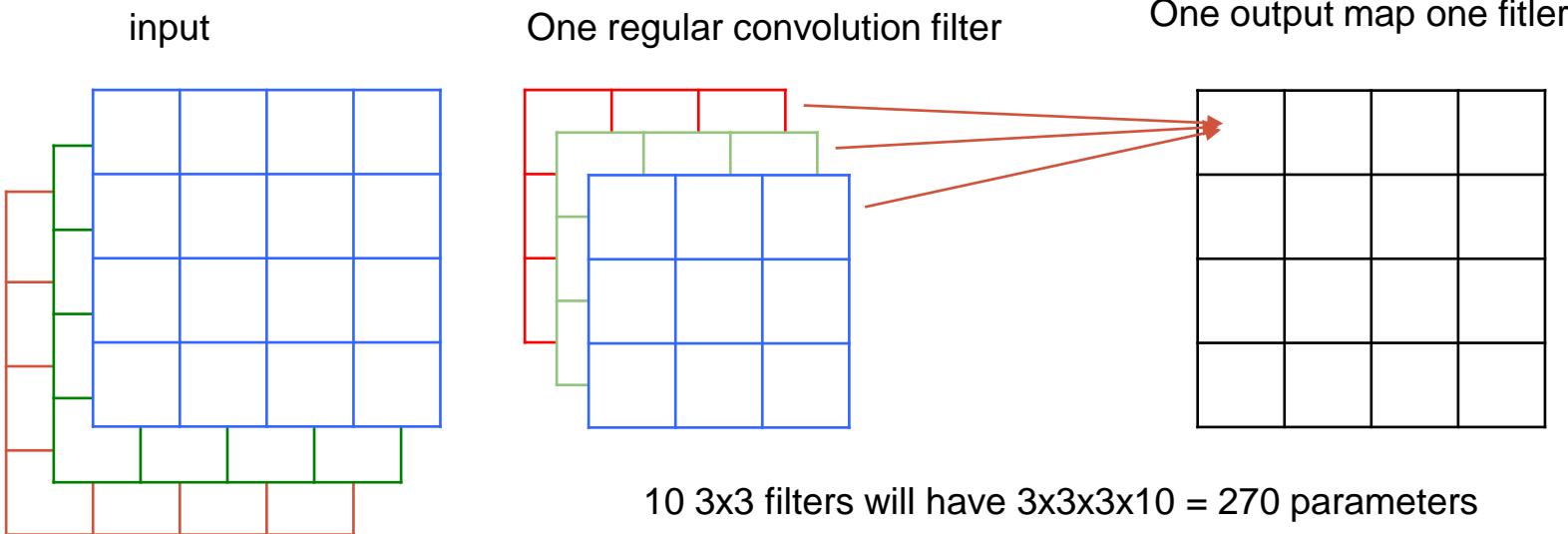
# Xception

## Depthwise separable convolution: two-step convolution

1. Depthwise convolution
2. 1x1 convolution

Typical convolution 3x3 filter is  
3x3xinput channel

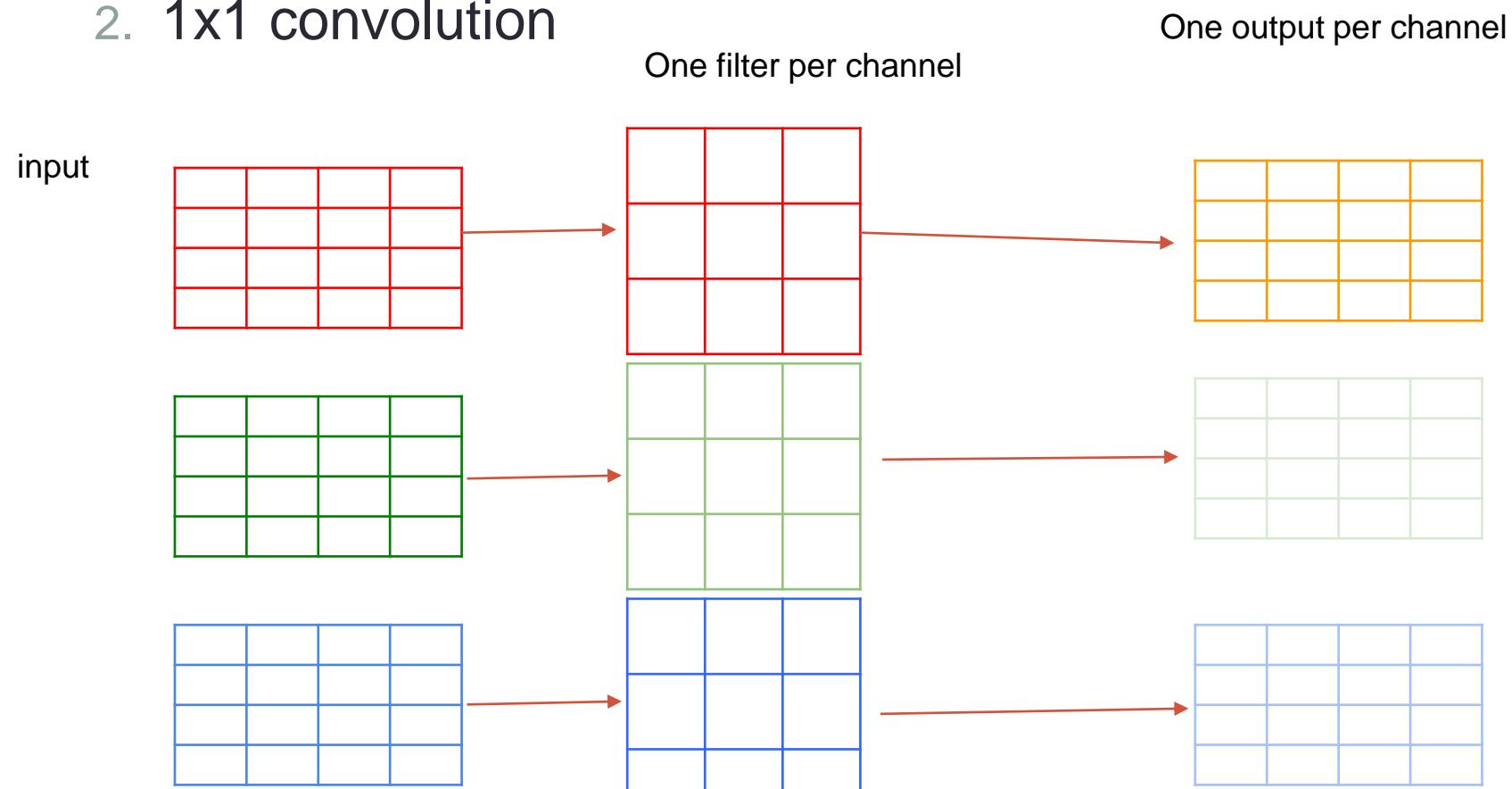
Depthwise convolution 3x3 filter is  
3x3x1  
1 filter per input channel



# Xception

Depthwise separable convolution: two-step convolution

1. Depthwise convolution (learn from each channel, spatial info)
2. 1x1 convolution

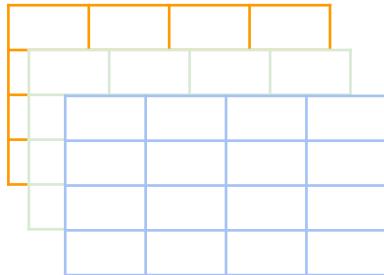


# Xception

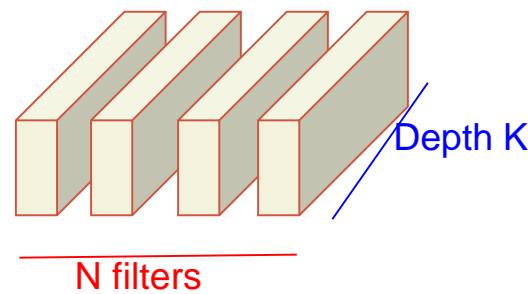
Depthwise separable convolution: two-step convolution

1. Depthwise convolution
2. 1x1 convolution (combine each channel, feature info)

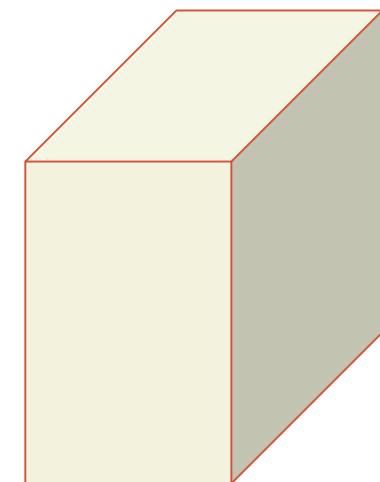
Output from depthwise convolution



1x1 filters



Final output



10 3x3 depthwise separable convolution will have  $3 \times 3 \times 3 + 3 \times 10 = 57$  parameters

# Xception

Replace convolutions in inception with depthwise separable convolutions

Smaller model

Faster compute

Comparable with Inception v3 while much faster

Used in MobileNet and other models

François Chollet, “Xception: Deep Learning with Depthwise Separable Convolutions” 2016.

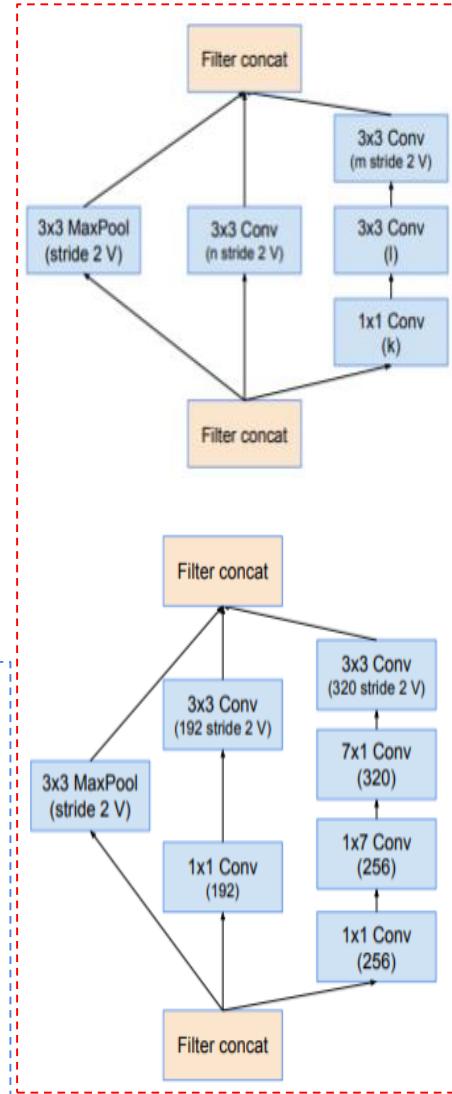
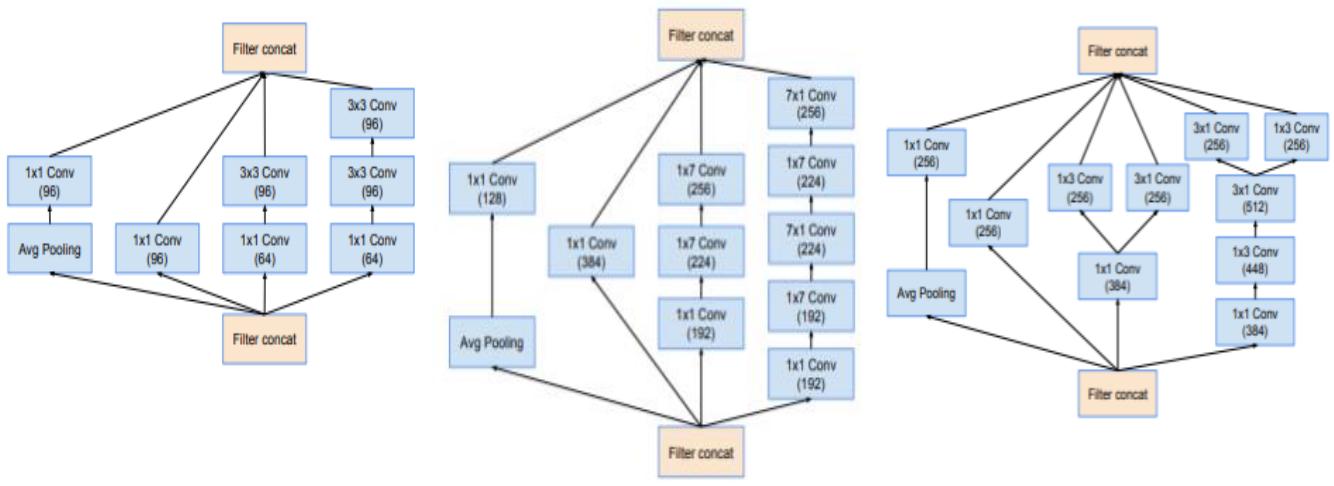
Andrew G. Howard, et al., “MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications” 2017.

# Inception v4

Same three types of inception blocks from v3

Add two types of **reduction blocks** for reducing the size of the grid (super pooling blocks)

Inception blocks



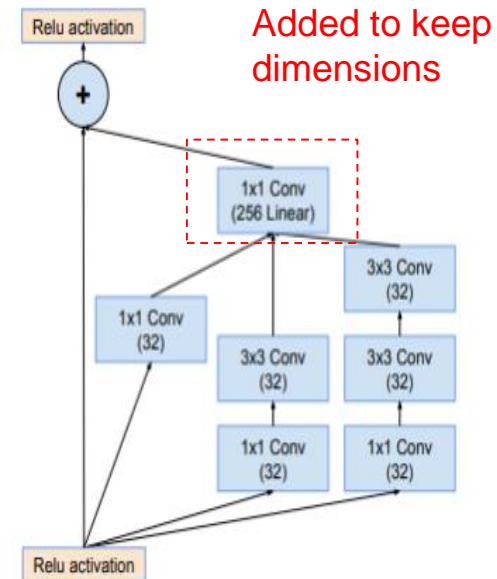
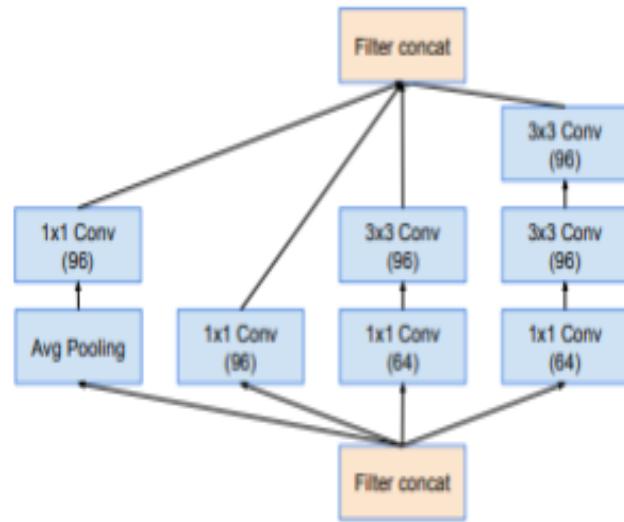
Reduction blocks

# Inception-ResNet v1-v2

Introduce residual connections into inception blocks

Poolings changed to additions, 1x1 added to keep dimensions for the residual

Similar idea to ResNeXt



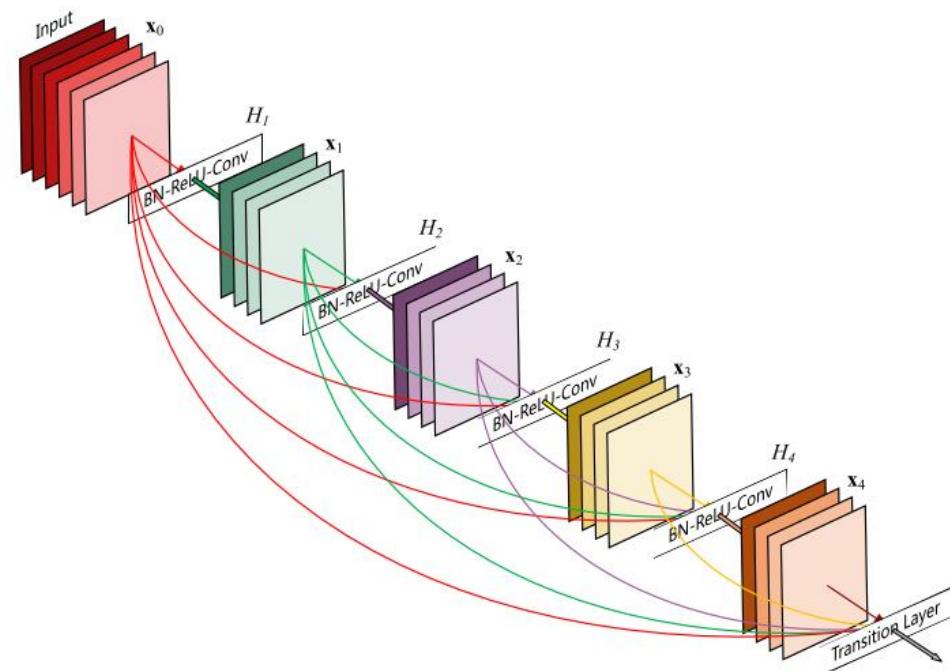
# DenseNet

Instead of adding the residuals, concatenates the feature maps from previous layers

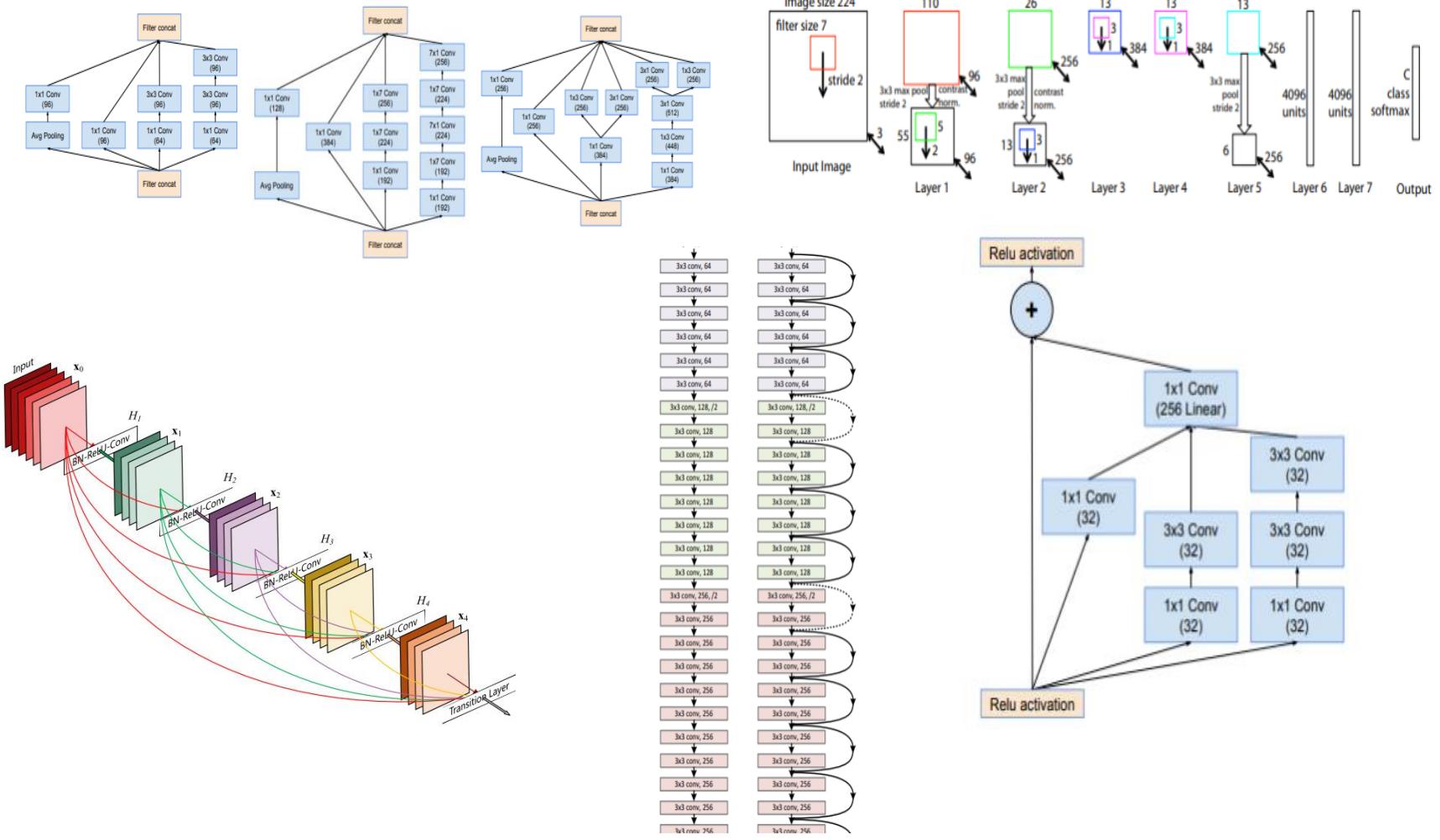
Densely connect multiple layers

**Multi-scale** feature maps

Smaller model due to smaller number of channels



# Do people keep tweaking network architectures until the end of time?

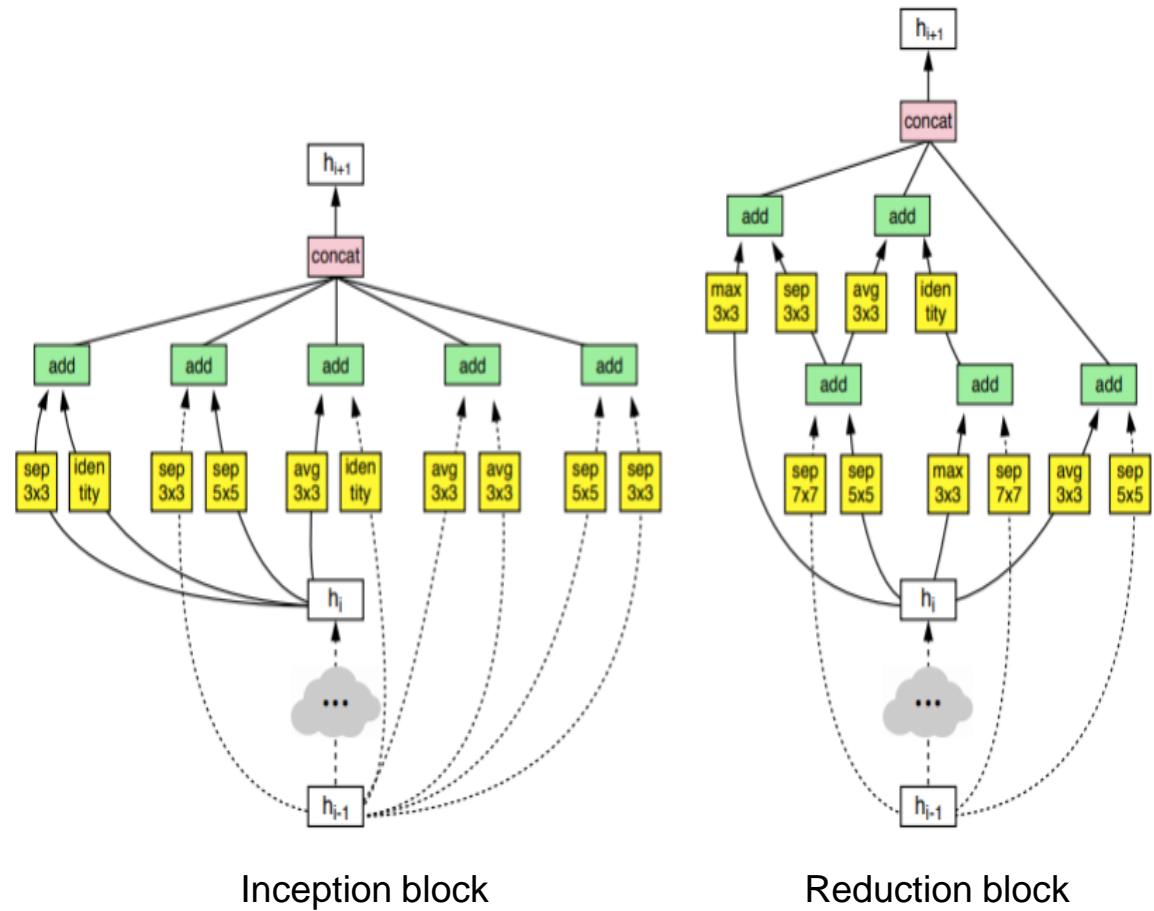


# NasNet

## Neural Architecture Search Network

Use reinforcement learning to search for the best network configuration

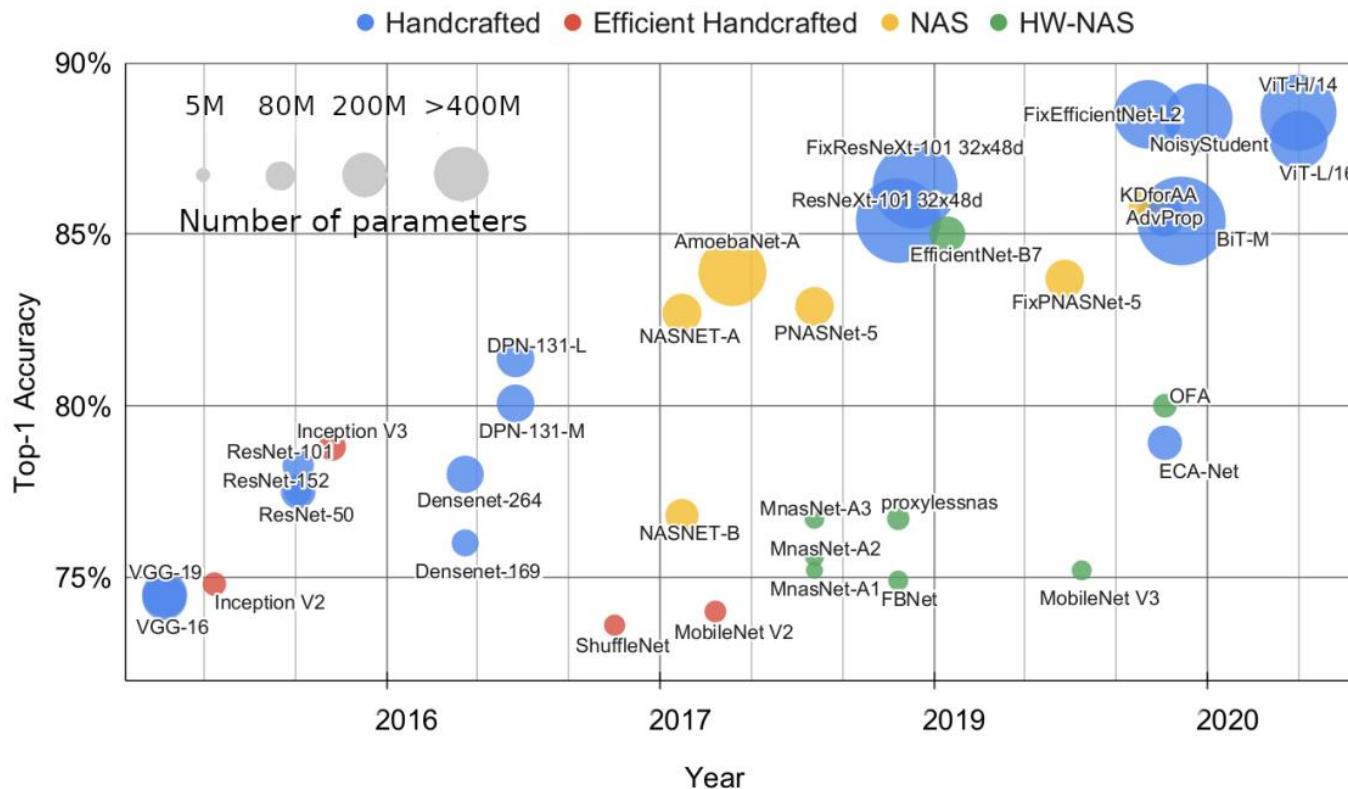
Lowers compute (FLOPs) while maintaining high accuracy



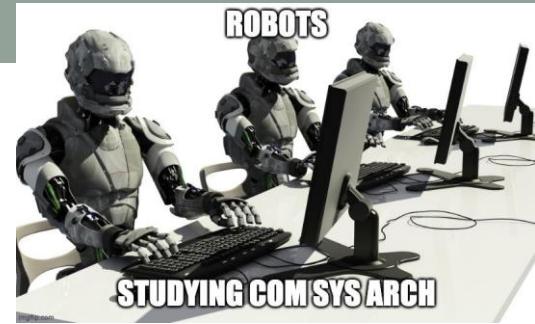
# NAS doesn't know com sys arch

- ★ Response Time
  - Elapsed Time - Wall clock
  - CPU Time - Only CPU time (without the wait time) ... we will revisit this later.
  - From O.S. class, we use: user time, real time, sys time.
- ★ Throughput
  - Tasks per time unit
- ★ Units for commercialization (Not a real performance)
  - MIPS
    - Millions of instructions per second
  - FLOPS
    - Floating Point Operations per second

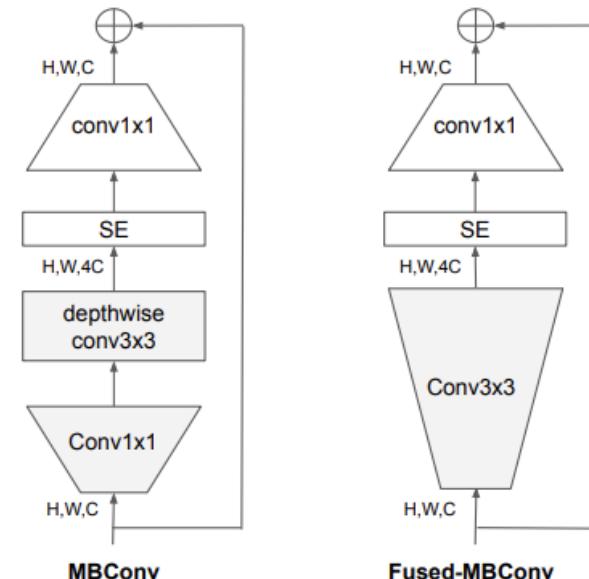
- FLOPs is not a good measure of performance



# EfficientNetv2



- Architecture/hardware aware NAS + handcrafted building blocks (from mobilenetV2)
- Tradeoff between height, width, depth, resolution



<https://arxiv.org/abs/1905.11946>  
<https://arxiv.org/abs/2104.00298>

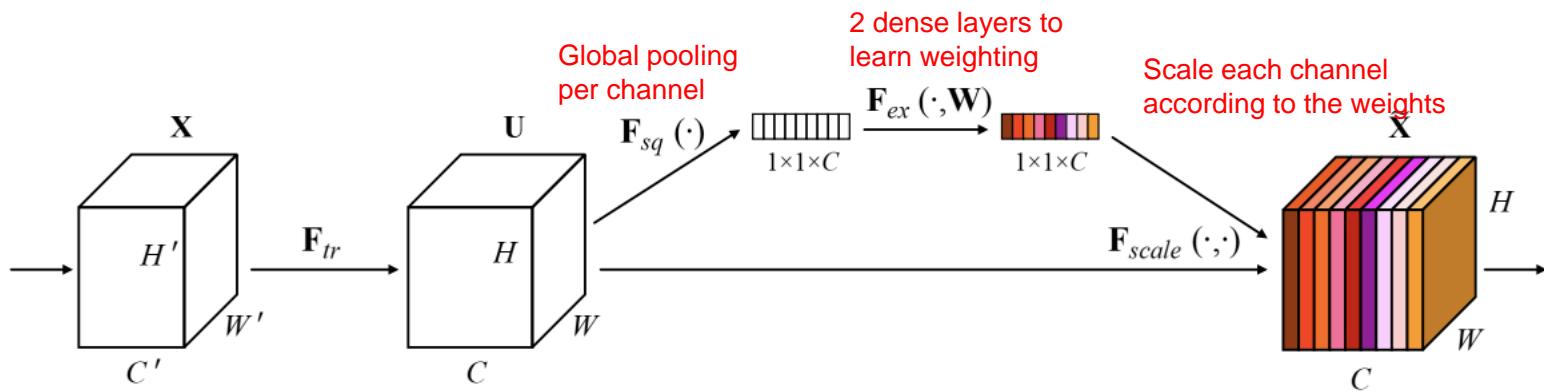
Figure 2. Structure of MBConv and Fused-MBConv.

# Modern convolutional blocks

- Squeeze and excitation
- Mobile inverted bottleneck

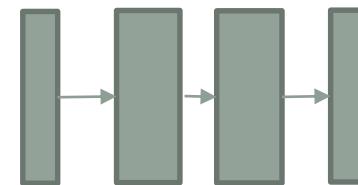
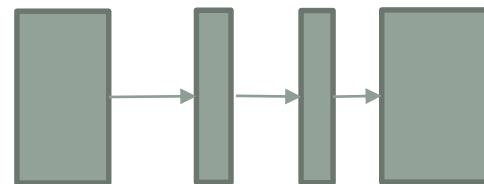
# Squeeze and excitation

- Maybe each channel should be weighted differently?
  - A picture with wave patterns should look for fish features
  - A picture with grass patterns should look for cow features
- “Attention” on channel info



# Inverted bottleneck

- A bottleneck block
  - reduce channels using  $1 \times 1$
  - Regular conv on fewer channels
  - Increase channel using  $1 \times 1$
- Inverted bottleneck
  - Increase channels using  $1 \times 1$
  - Regular conv on more channels
  - Reduce channels using  $1 \times 1$
- Bottleneck combines information between channels
- With residual connections, inverted bottleneck uses less runtime memory during inference (DAG computation graph)



# Mobile inverted Bottleneck

- Use depthwise convo instead of conv (MobileNetv2)
- Add SE (EfficientNetv1)
- Can fused 1x1 with depthwise (EfficientNetv2)

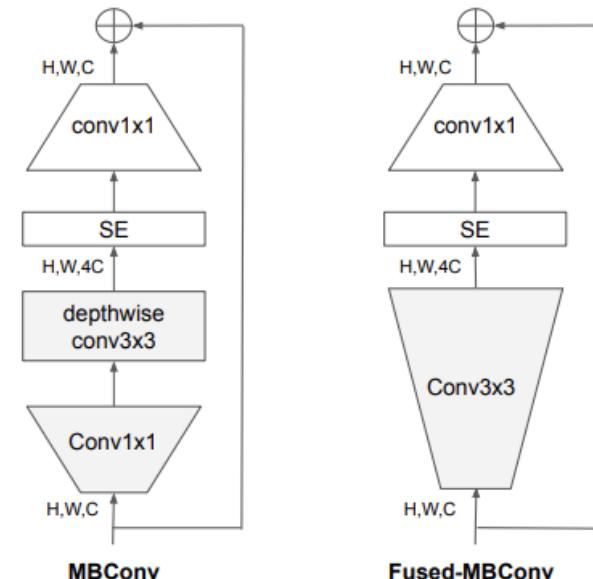
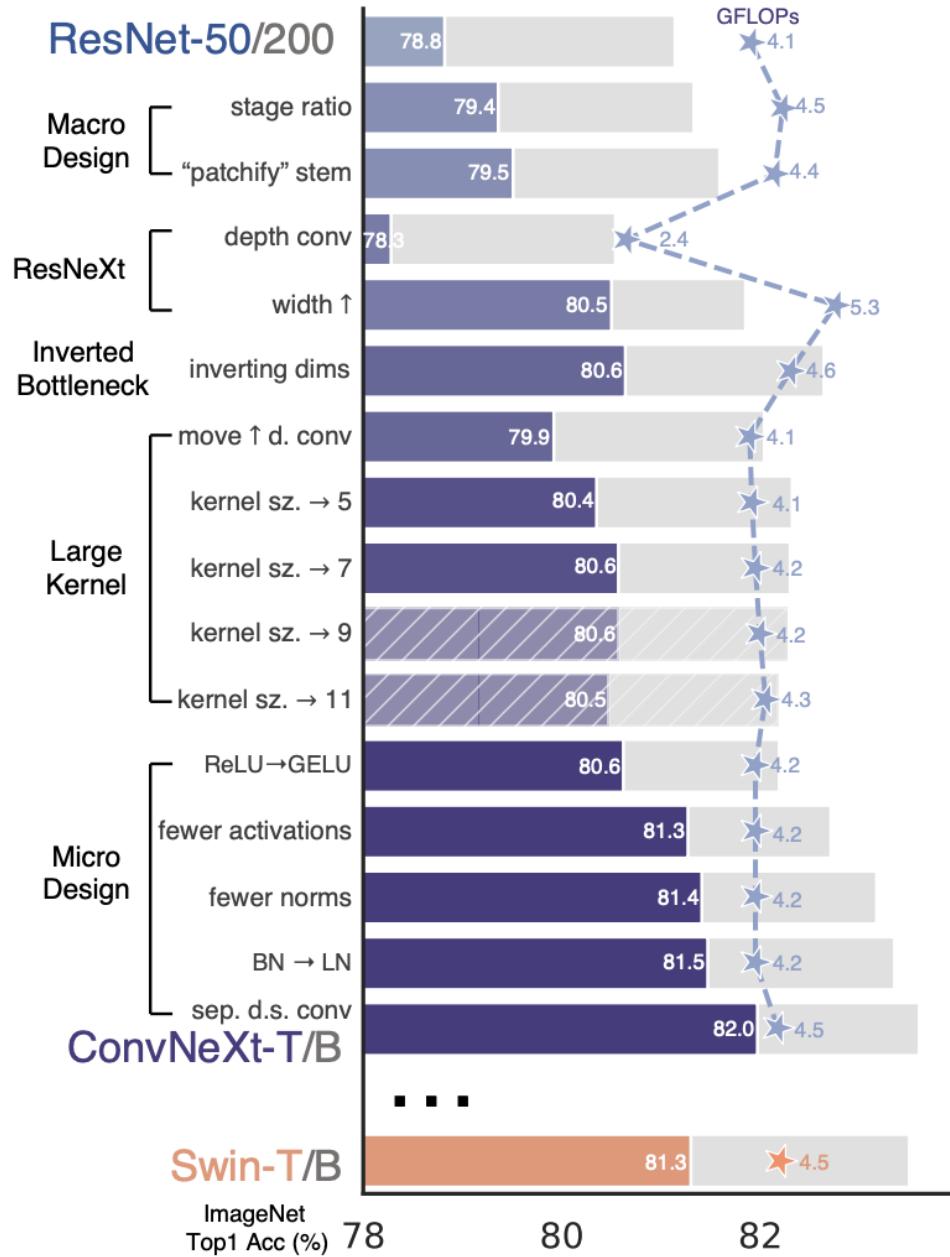


Figure 2. Structure of MBConv and Fused-MBConv.

# ConvNext

- To get the most recent hype see ConvNext
- Bag of tricks++
- Highly recommended as a review paper

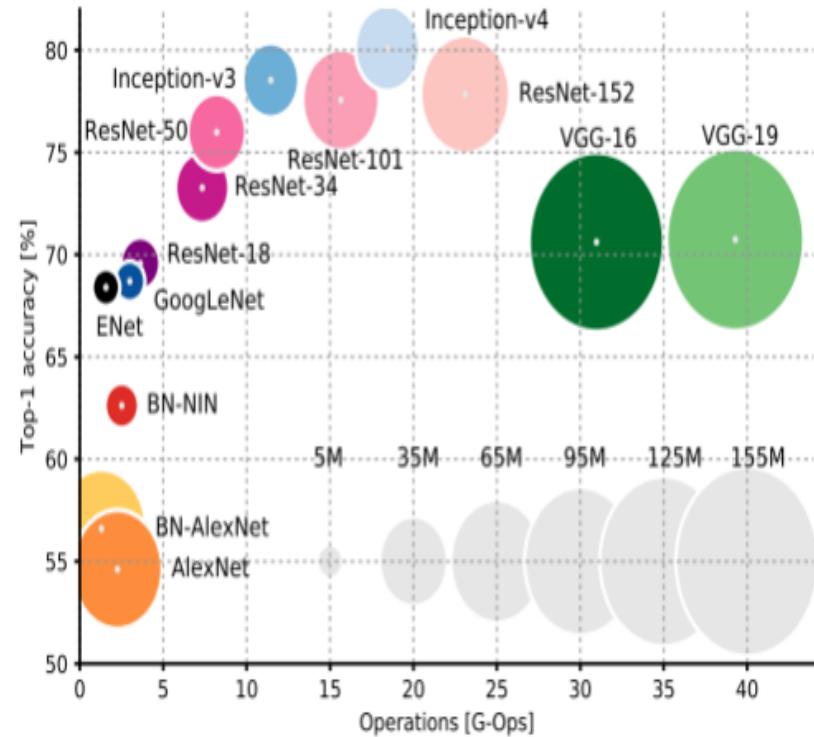


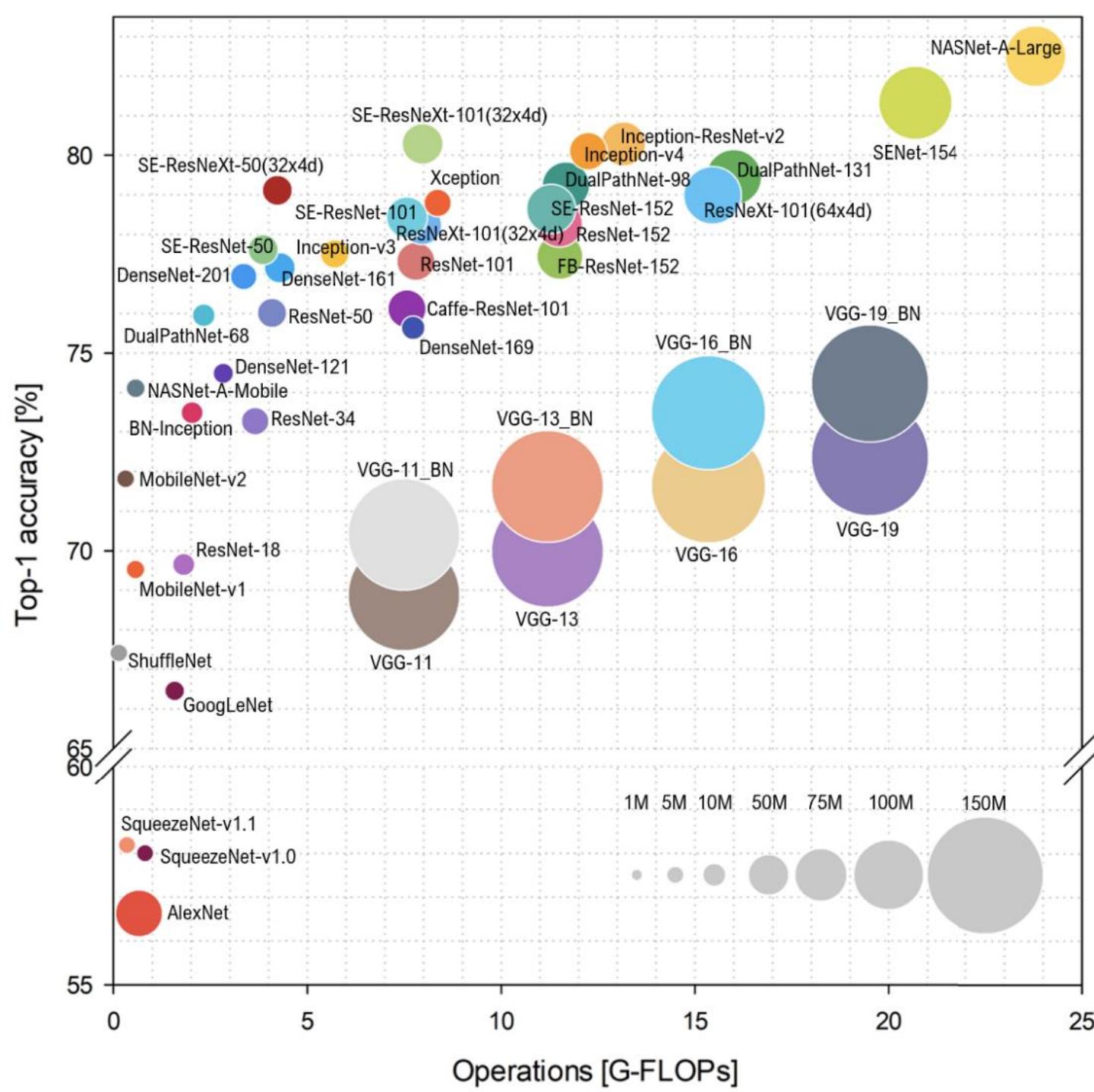
# Object classifiers summary

Most successful tricks:  
Dropout, residual, batch  
norms, multi-path, data  
augmentation

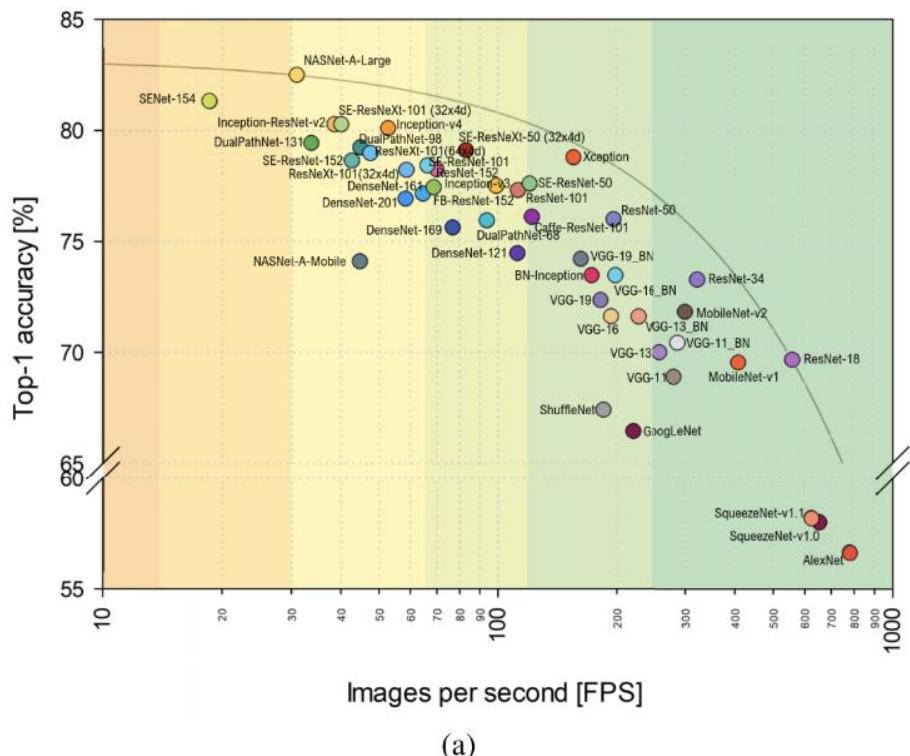
These object classifiers are  
often called **backbones** and  
used by other models

Comparing accuracy, model  
size, transferability and  
compute.

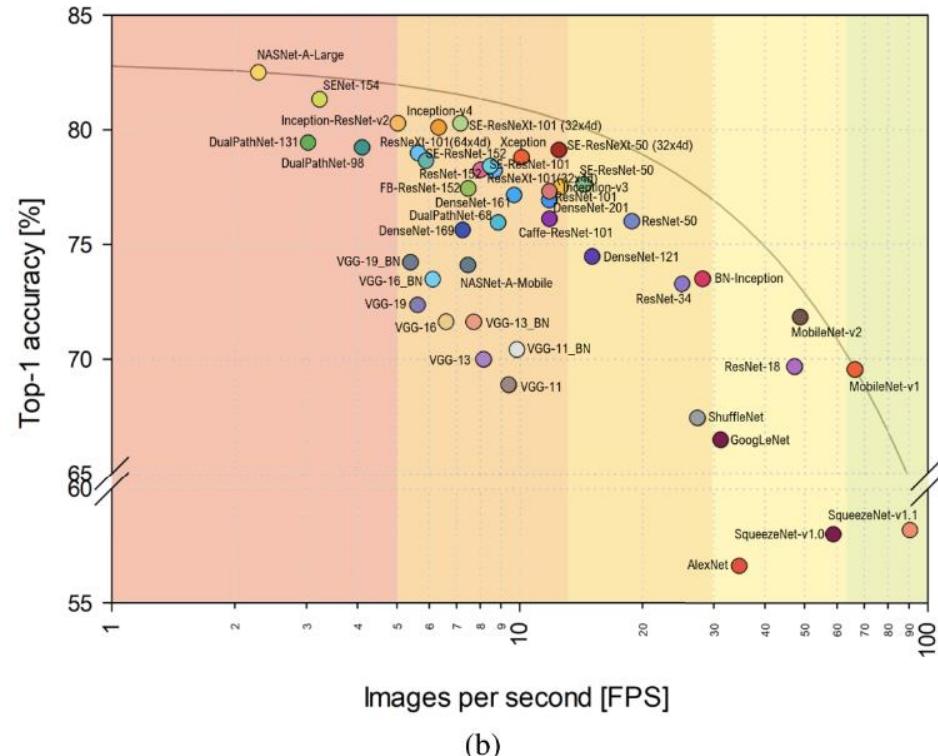




Simone Bianco, et al.,  
Benchmark Analysis of  
Representative Deep  
Neural Network  
Architectures, 2019



(a)



(b)

**FIGURE 3.** Top-1 accuracy vs. number of images processed per second (with batch size 1) using the Titan Xp (a) and Jetson TX1 (b).

Simone Bianco, et al.,  
Benchmark Analysis of  
Representative Deep  
Neural Network  
Architectures, 2019

# Other stuff to look for

## Layers

### Dilated convolution

<https://towardsdatascience.com/understanding-2d-dilated-convolution-operation-with-examples-in-numpy-and-tensorflow-with-d376b3972b25>

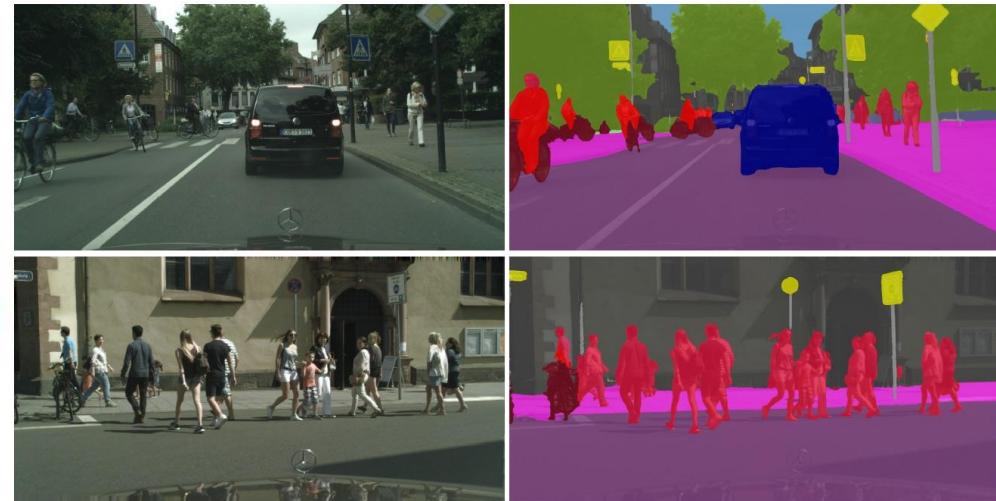
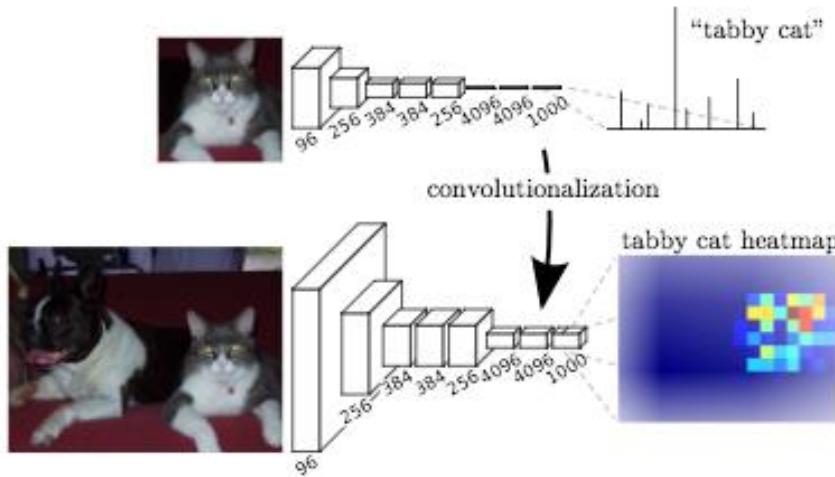
## Necks

### Feature Pyramid Networks (multi-scale backbone)

<https://arxiv.org/abs/1612.03144>

# Sometimes you want to increase the size of your feature map

## Image segmentation



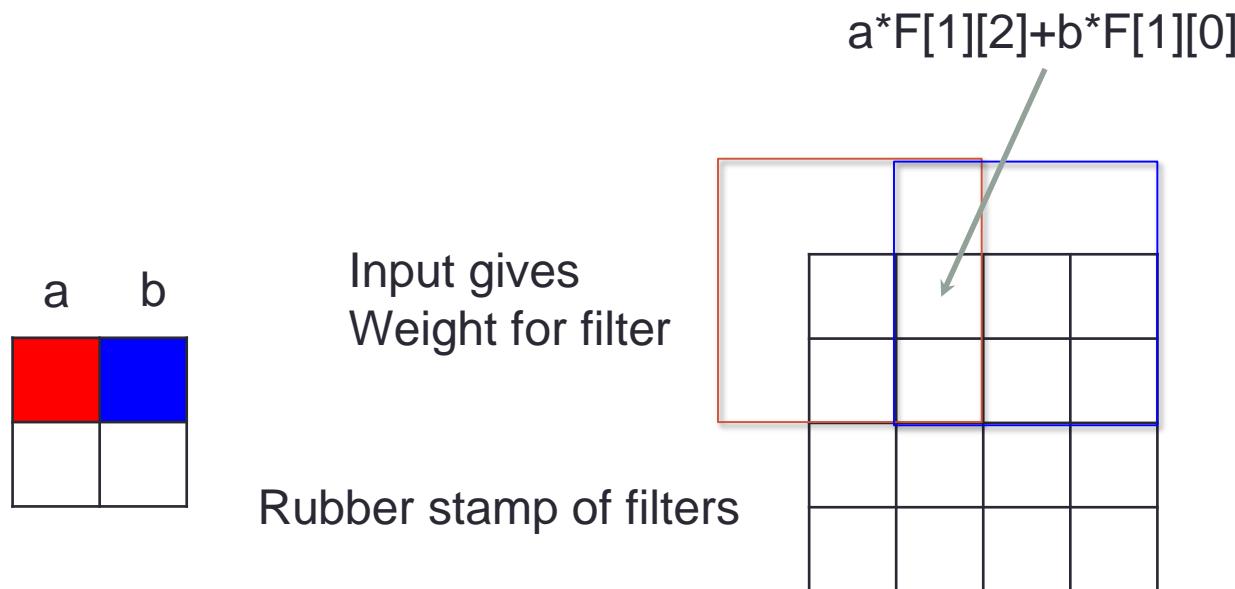
2 main approaches to upsample  
De-convolution  
resize (unpooling) + convolution

[https://people.eecs.berkeley.edu/~jonlong/long\\_shelhamer\\_fcn.pdf](https://people.eecs.berkeley.edu/~jonlong/long_shelhamer_fcn.pdf)

<http://vladlen.info/publications/feature-space-optimization-for-semantic-video-segmentation/>

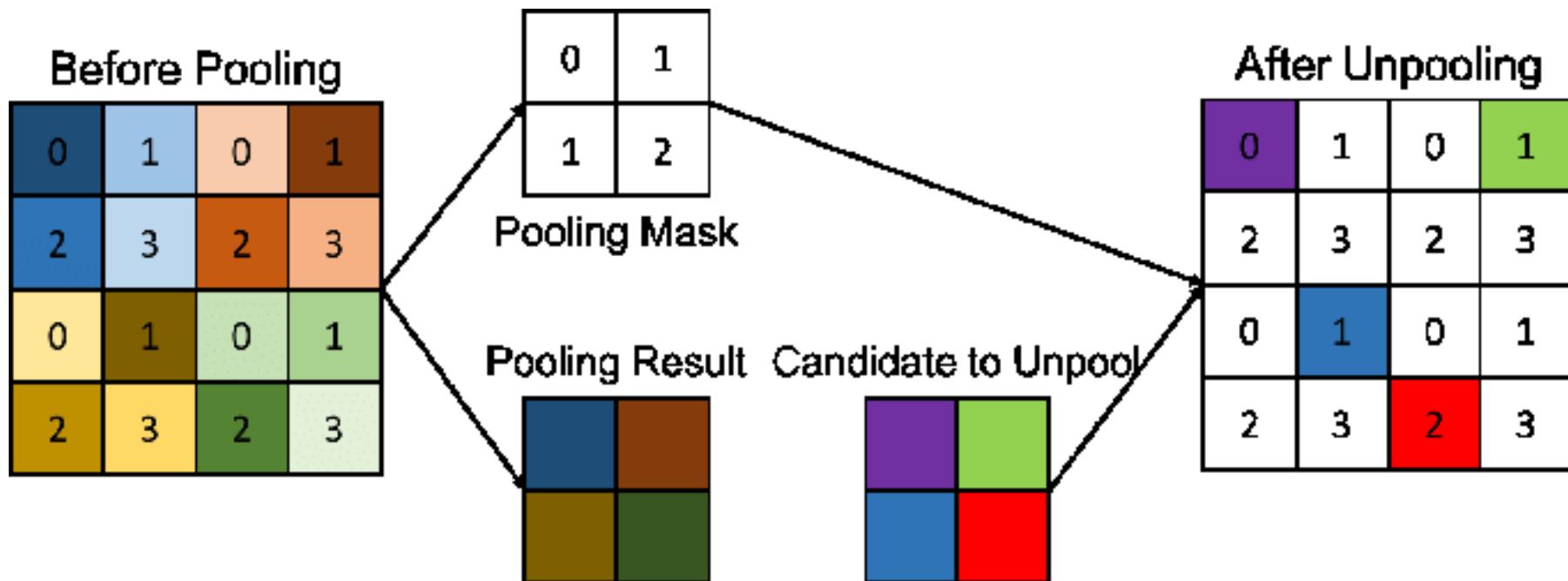
# De-convolution (Upsampling)

- 3x3 de-convolution filter, stride 2, pad 1



Other names because this name sucks (for me)  
- Convolution transpose, upconvolution, backward strided convolution

# Unpooling + conv



Remember the location of the max value

Put the value to unpool at that location

Fill the rest with zeroes

Follow by regular convolution to smooth the image

# Upsampling notes

Deconvolution filter size should be a multiple of the stride to avoid **checkerboard** artifacts

Unpooling can be replaced with regular image resizing techniques (interpolation)

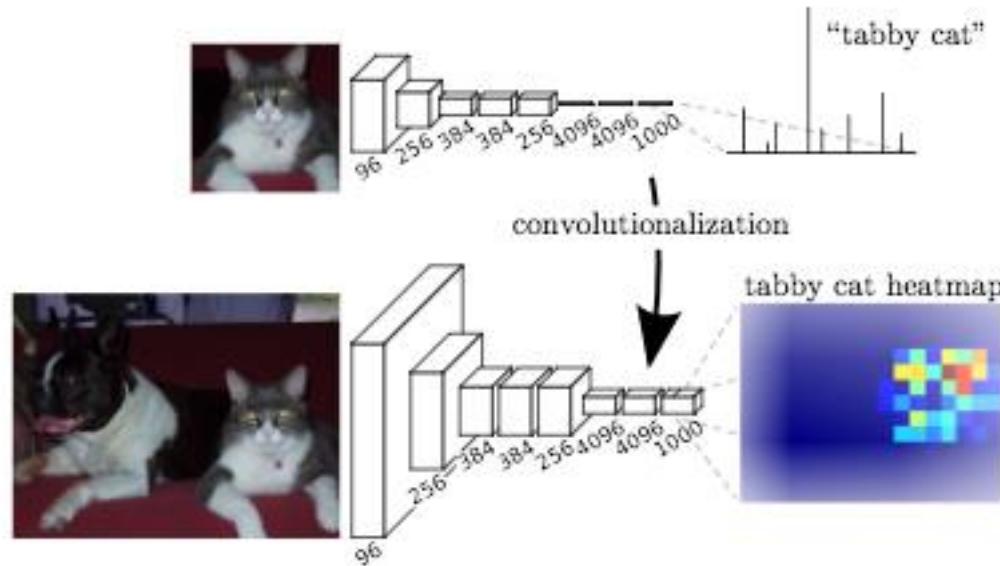


Image using deconv

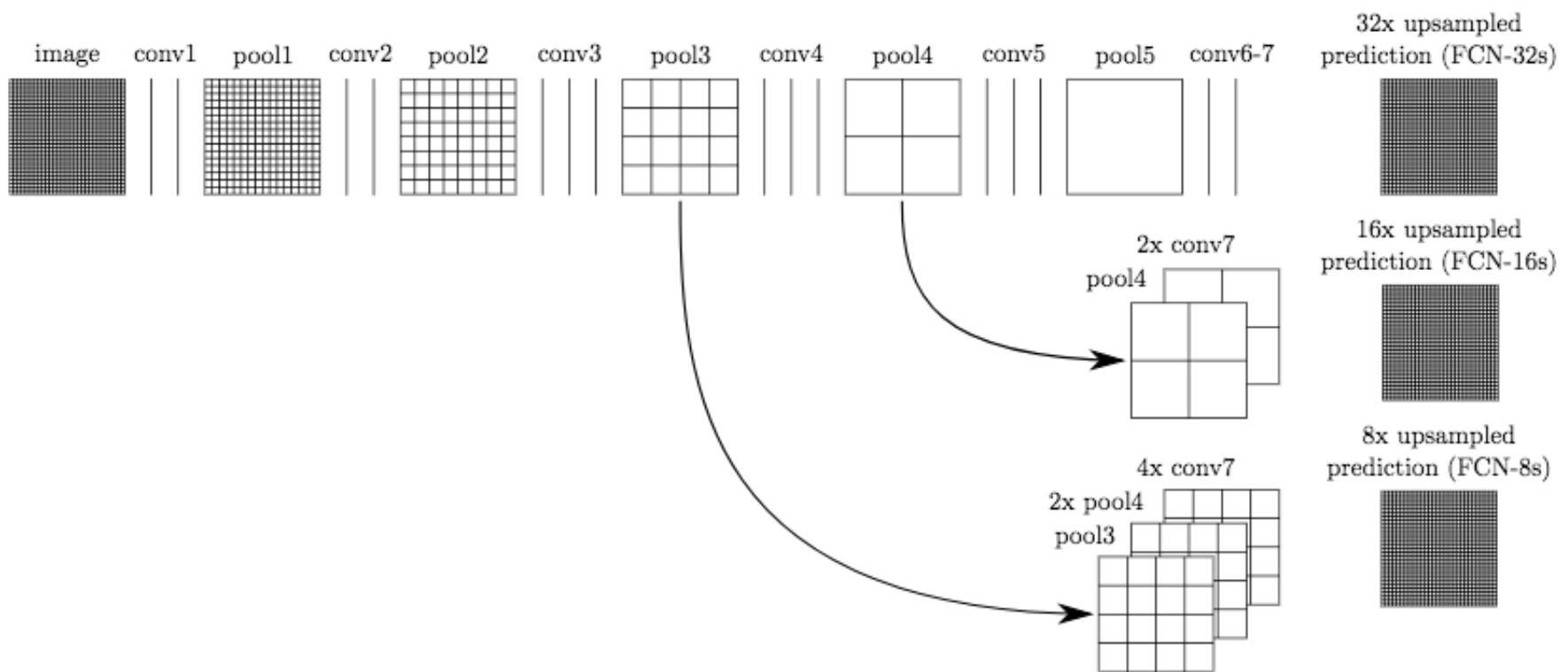


Image using resize upsampling

# De-convolution for segmentation



# De-convolution for segmentation



# De-convolution for segmentation

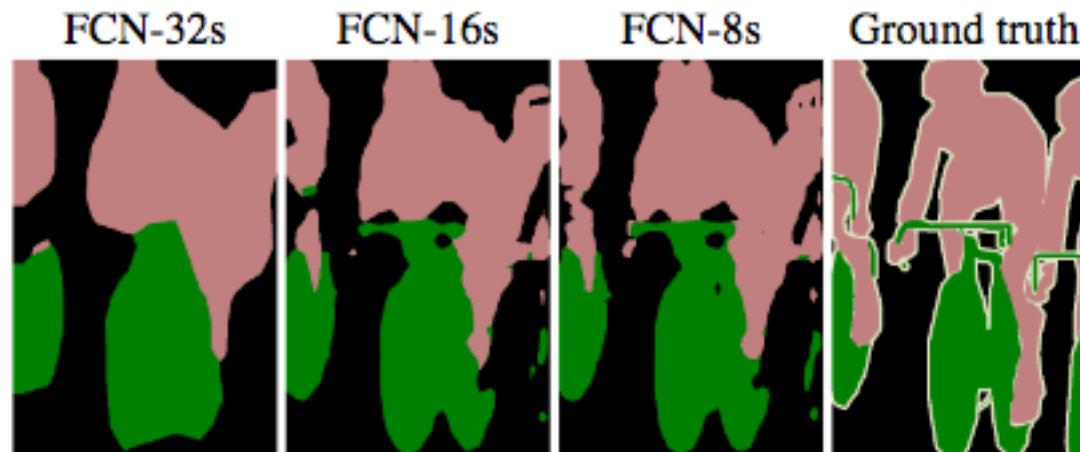


Figure 4. Refining fully convolutional nets by fusing information from layers with different strides improves segmentation detail. The first three images show the output from our 32, 16, and 8 pixel stride nets (see Figure 3).

# CNN Summary

Building blocks

Matched filters and pooling

Filter factorization

$5 \times 5 \rightarrow$  two  $3 \times 3$

depthwise vs matrix factorization

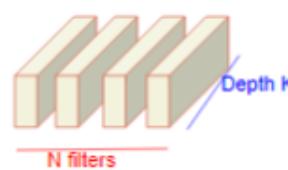
$1 \times 1$  convolution

Deconvolution and upsampling

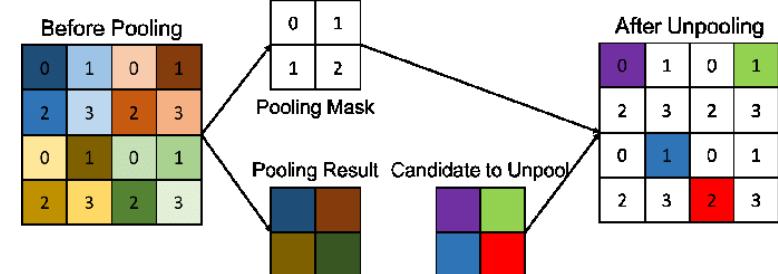
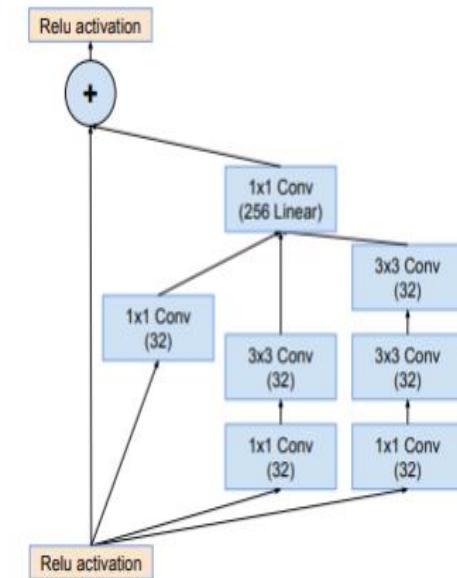
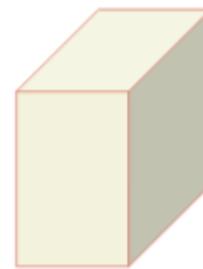
Output from depthwise convolution



$1 \times 1$  filters

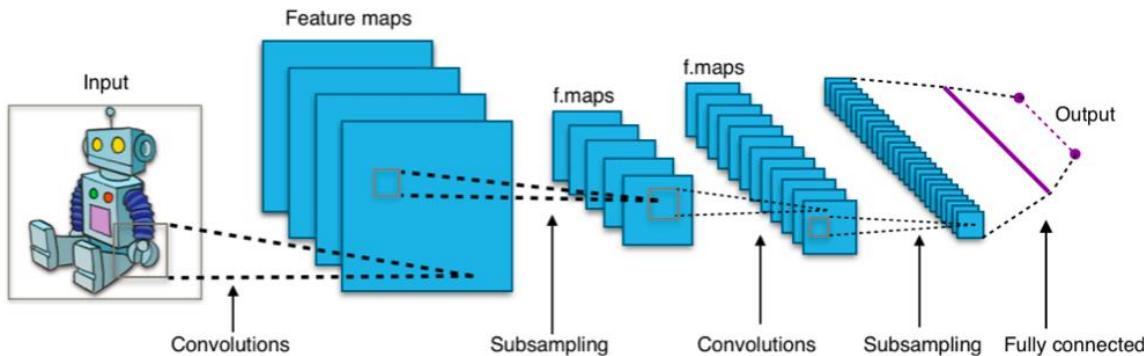
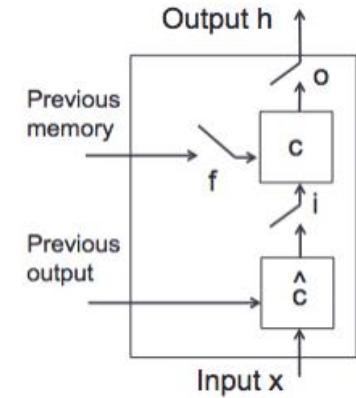
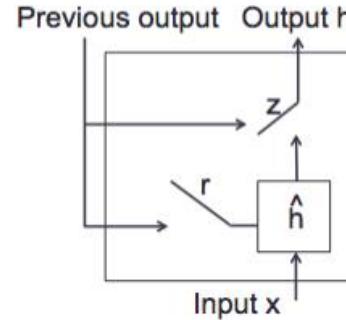


Final output



# Neural networks

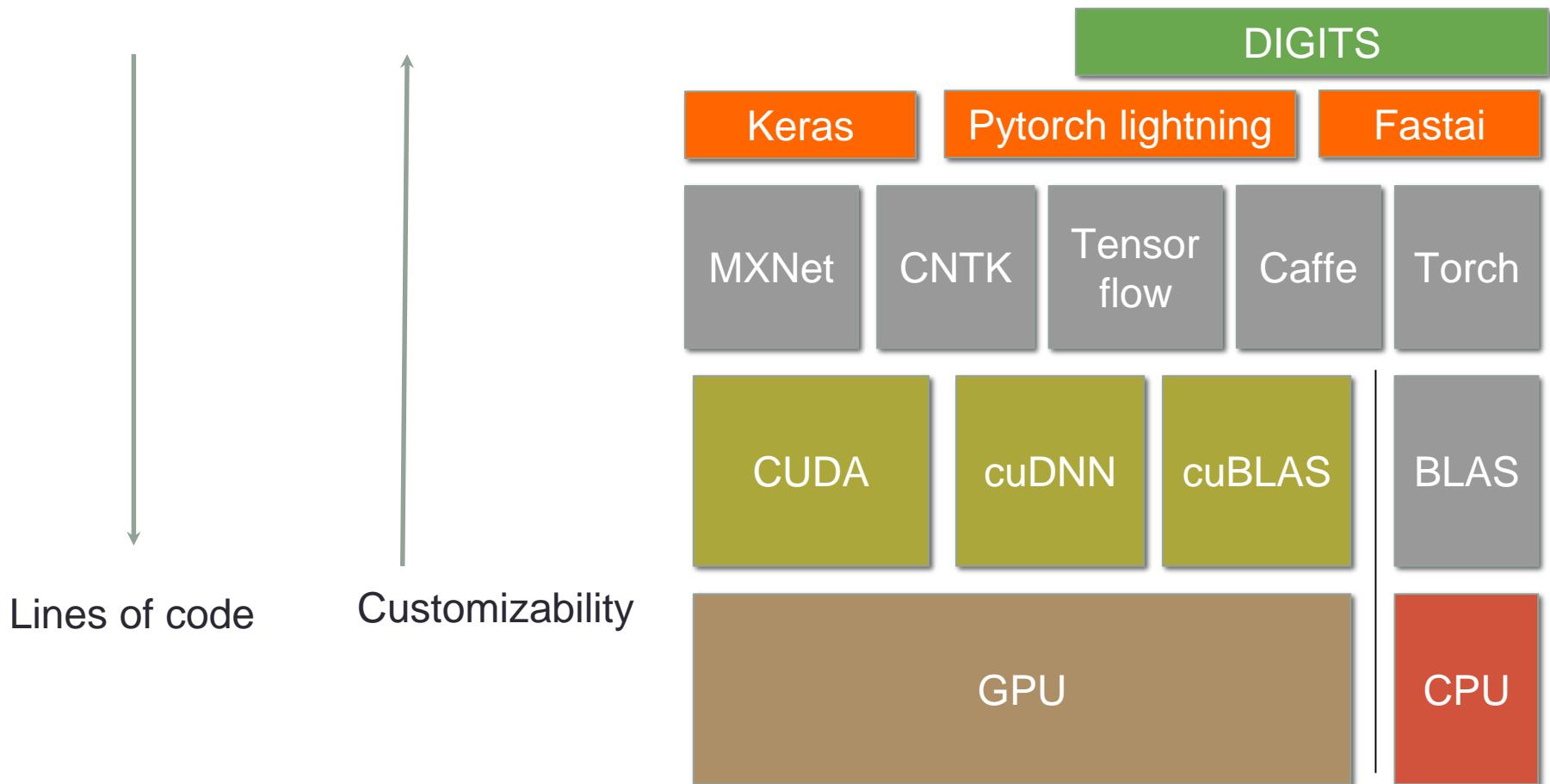
- Fully connected networks
  - SGD, backprop
- CNN
- RNN, LSTM, GRU



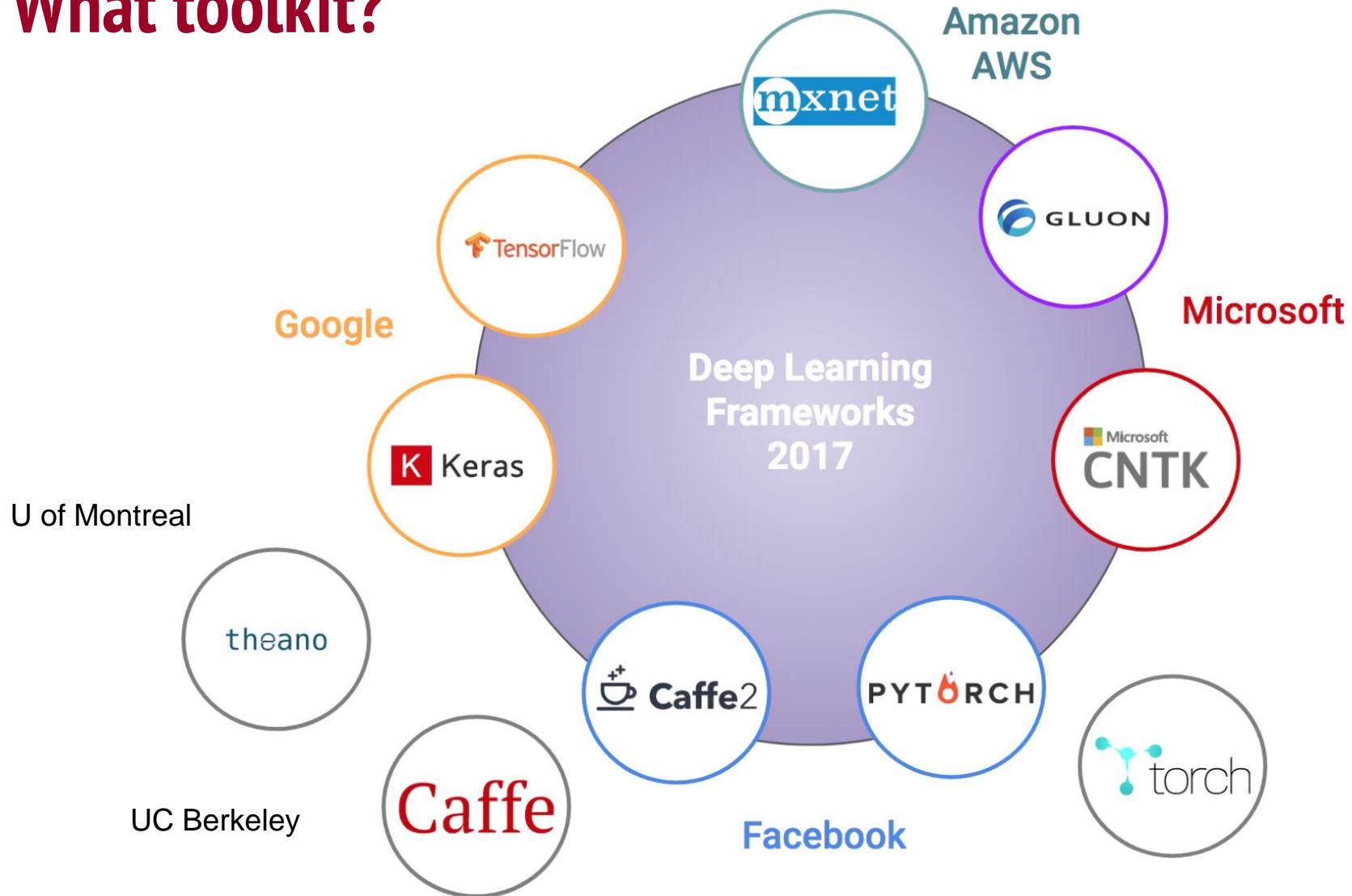
Generation ← Next lecture

# What toolkit

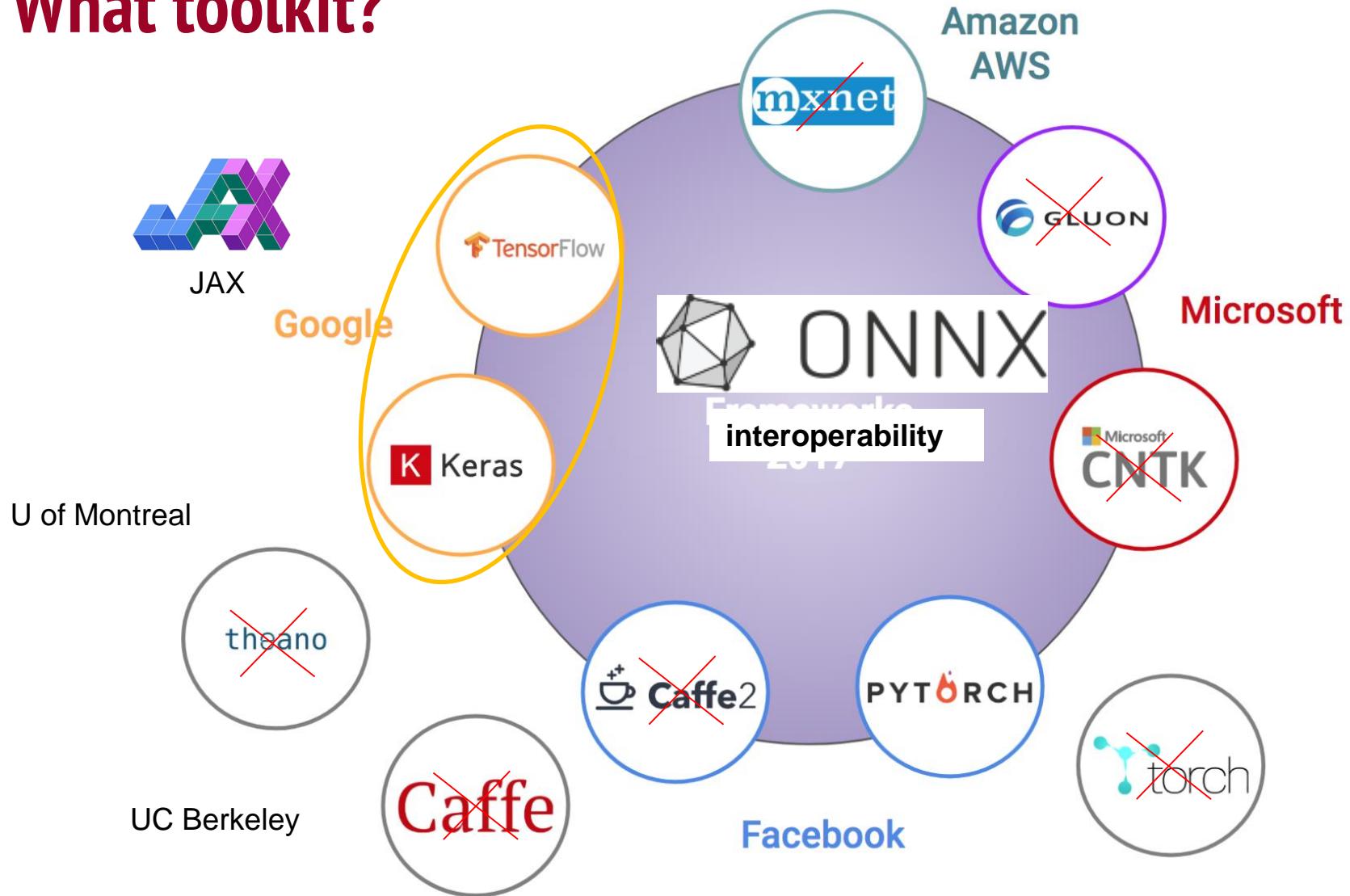
# Tradeoff between customizability and ease of use



# What toolkit?

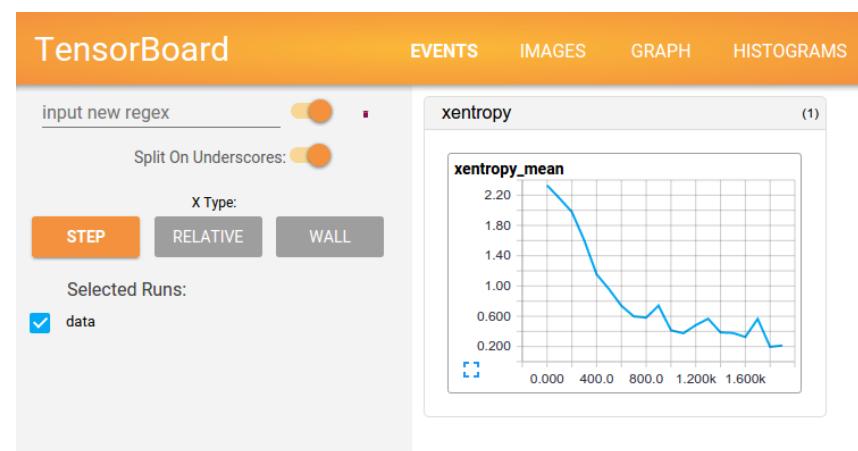
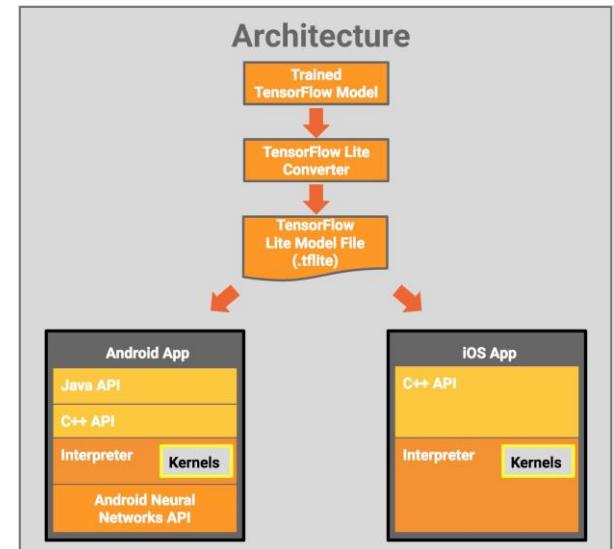


# What toolkit?



# Which?

- Easiest to use and play with deep learning: Keras
- Easiest to use and tweak: pytorch
- Easiest to deploy: tensorflow
  - Tensorflow lite for mobile
  - TensorRT support
  - Javaruntime support
- Best tools: increasingly pytorch
- Community: increasingly pytorch
- In the end you should know some tf and pytorch



# Pytorch steps

- Setting up dataloader
  - Gives minibatch
- Define a network
  - Init weights
  - Define computation graph
- Setup optimization method
  - Pick LR scheduler
  - Pick optimizer
- Training loop
  - Forward (compute Loss)
  - Backward (compute gradient and apply gradient)
- Let's demo



HW4

I've written deep  
learning model from scratched

[imgflip.com](#)

HW5



pytorch gave  
me an error

# Debugging guide

- [https://uvadlc-notebooks.readthedocs.io/en/latest/tutorial\\_notebooks/guide3/Debugging\\_PyTorch.html](https://uvadlc-notebooks.readthedocs.io/en/latest/tutorial_notebooks/guide3/Debugging_PyTorch.html)  
has list of common errors and best practices
- <http://karpathy.github.io/2019/04/25/recipe/>  
has guide for end-to-end model building (start simple and go more advance)