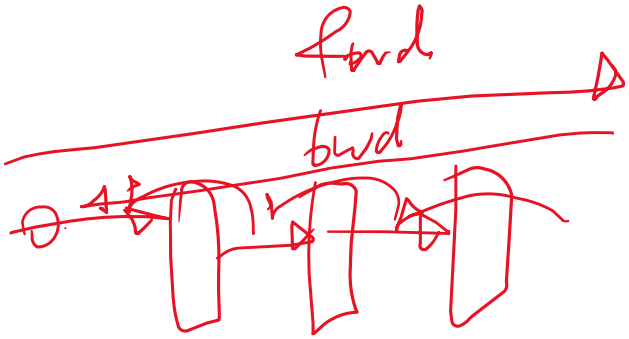
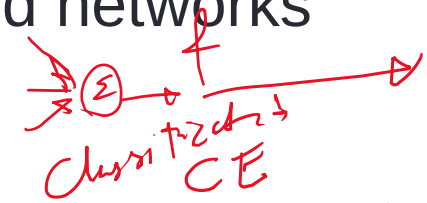
CONVOLUTIONAL AND RECURRENT NEURAL NETWORKS

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

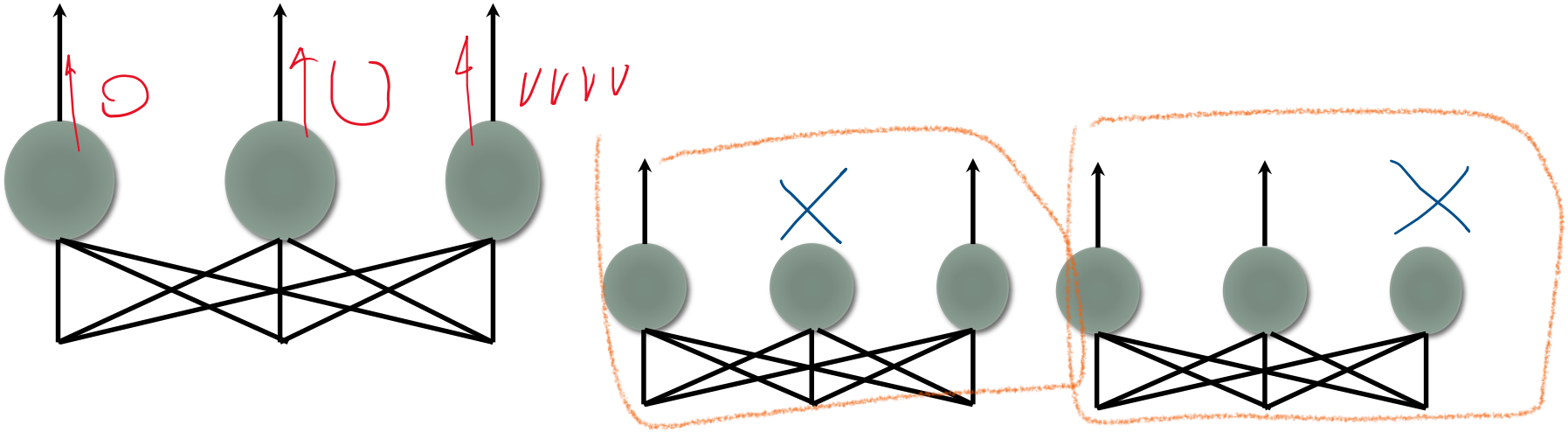
An implicit regularization technique for reducing overfitting

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)



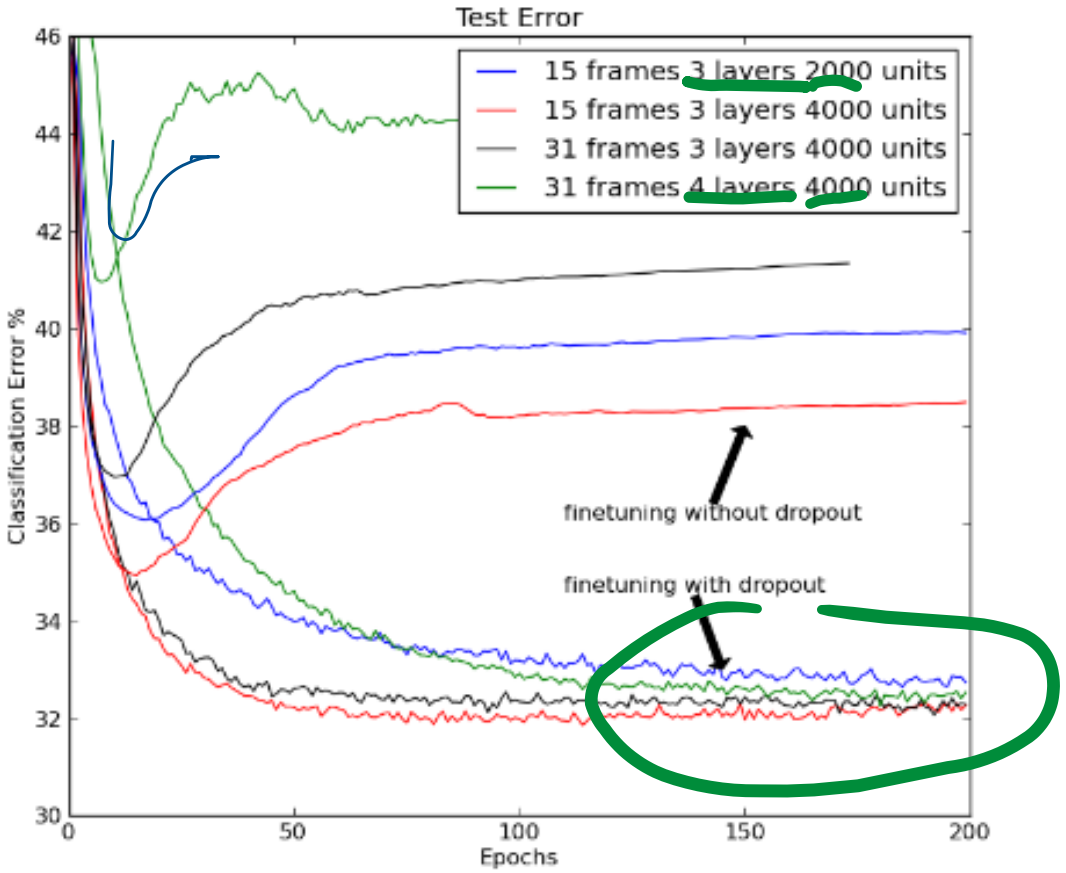
Model

Dropout rate of 0.33

Hinton, Geoffrey "Improving neural networks by preventing co-adaptation of feature detectors" 2012

Dropout on TIMIT

- A phoneme recognition task

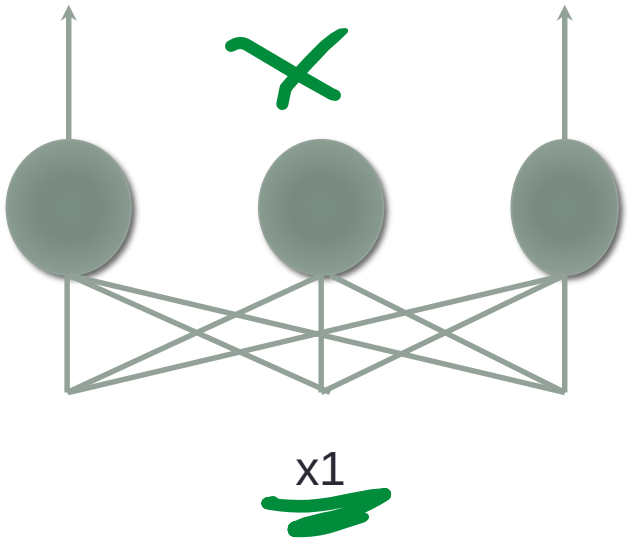


Hinton, Geoffrey "Improving neural networks by preventing co-adaptation of feature detectors" 2012

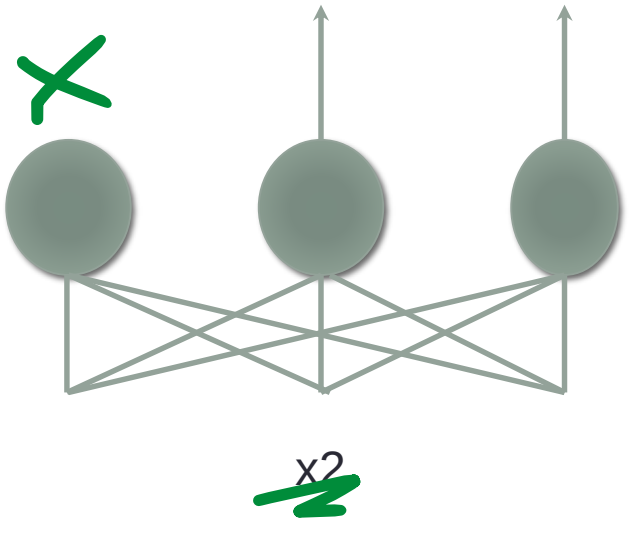
Dropout training time

Drop out rate 0.33

Randomly removed outputs
for each training sample



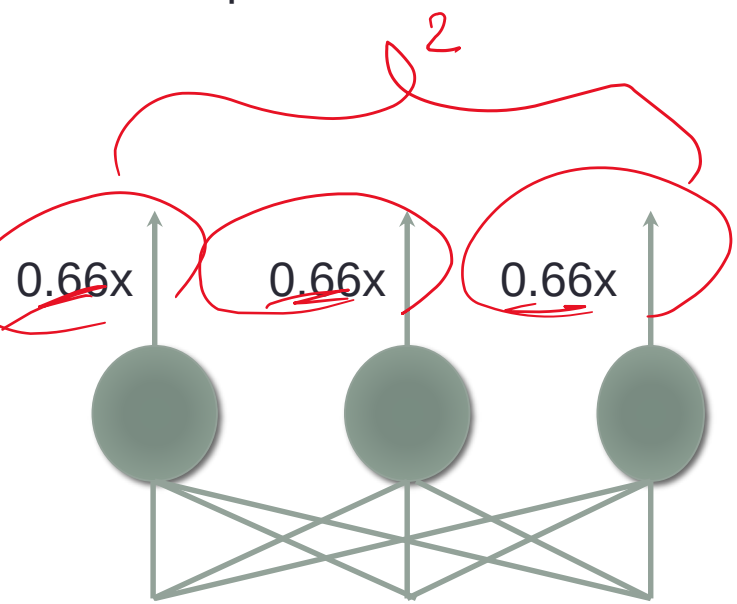
x1



x2

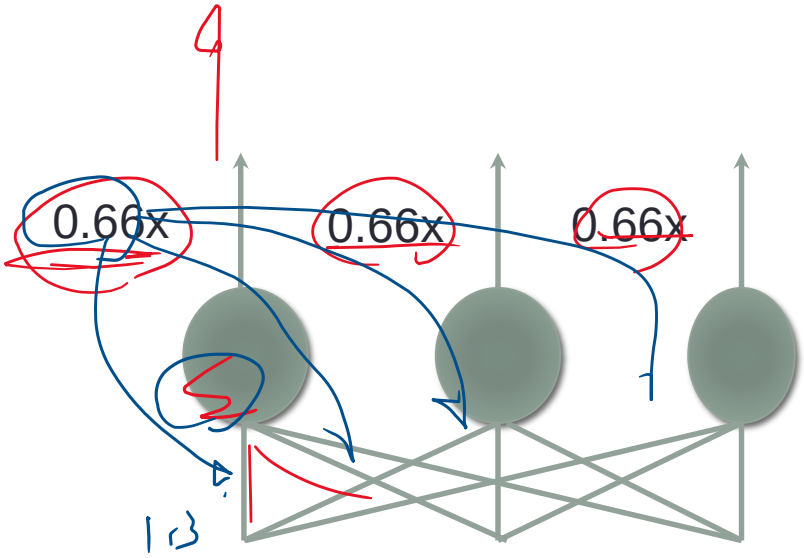
Dropout test time

Dropout rate 0.33



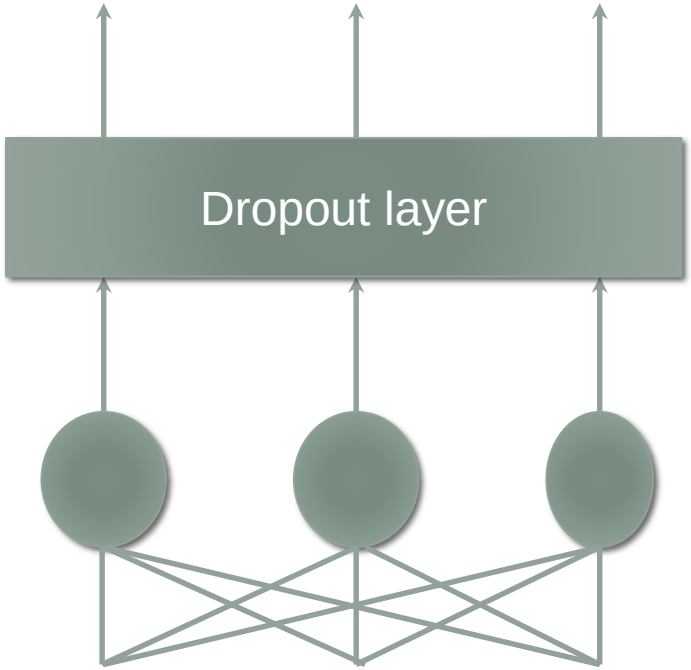
x1

Scale outputs so the output contribution is around the same



x2

Dropout implementation



x1

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

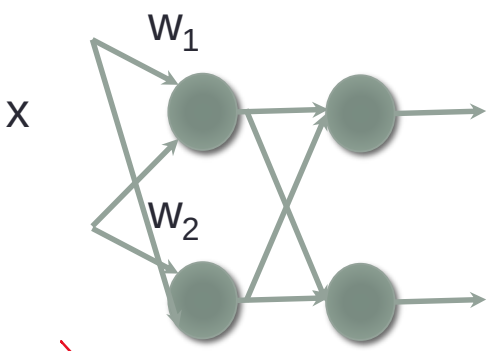
Convolution

Convolution Layers and Pooling Layers

CNN progress

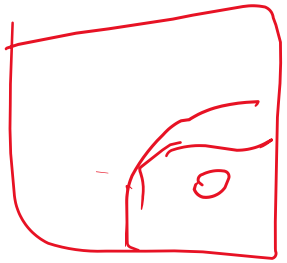
PCA as a filter

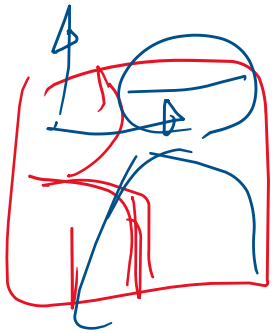
- $W^T x$



x = y

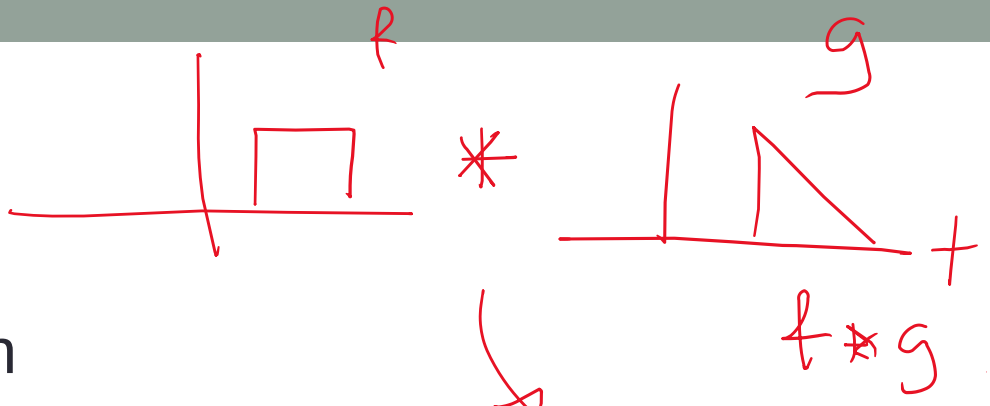
- What happens if I have a person that is off-frame?
- Need another filter that is shifted





translation equivalent

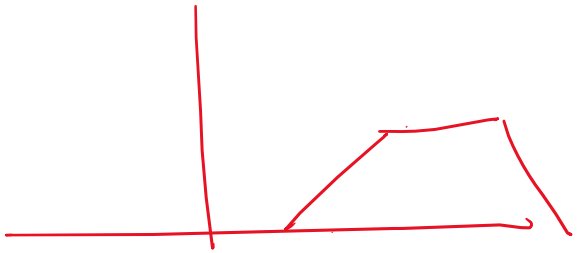
Convolution



- Continuous convolution

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

t=3

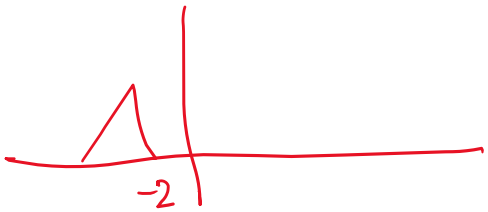


- Meaning of (t-T) : Flip then shift

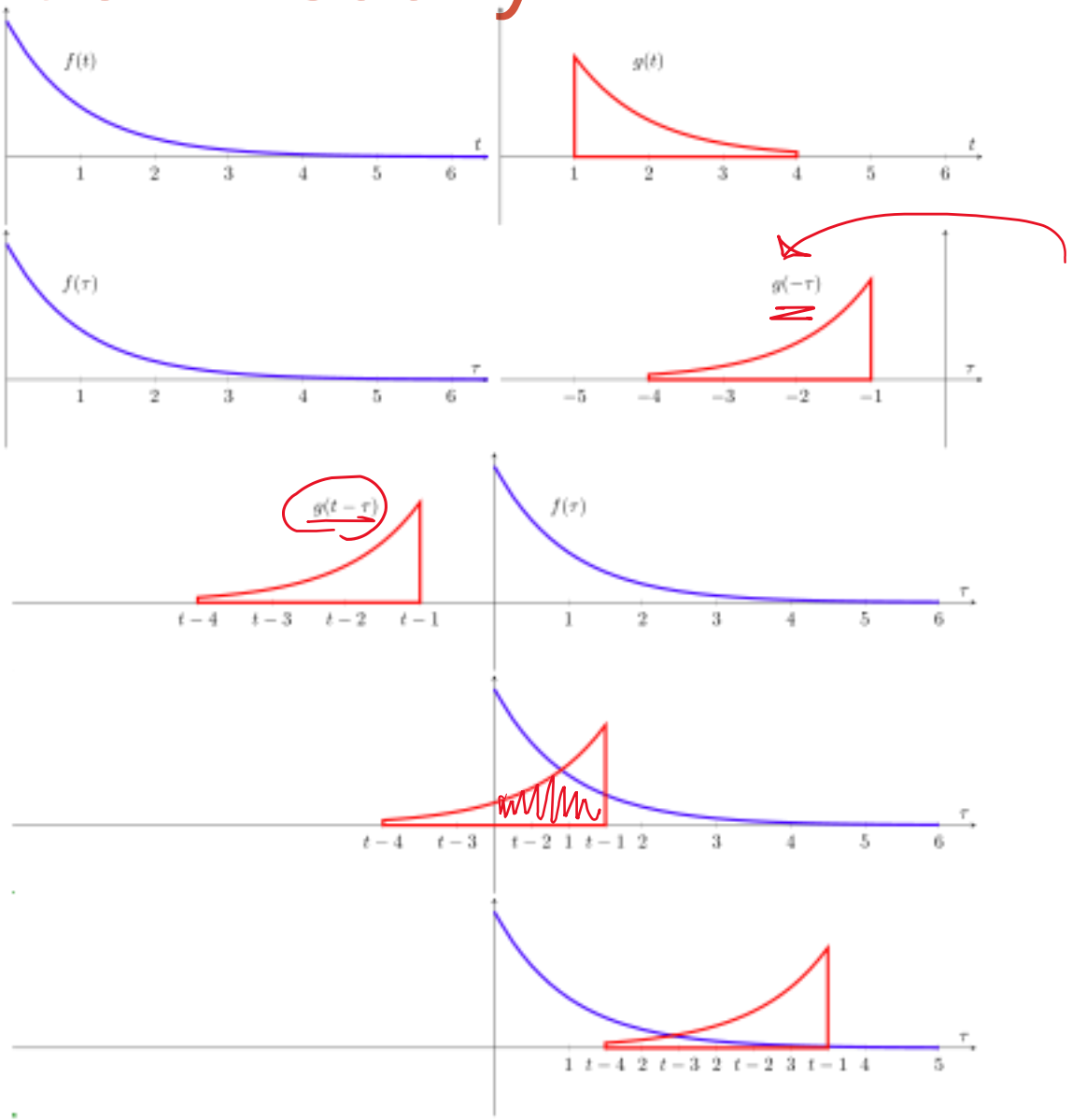
$\int f(\tau) g(3-\tau)\, d\tau$

$g(3-\tau)$

$g(-\tau)$



Convolution visually



https://en.wikipedia.org/wiki/Convolution

Convolution discrete

- Discrete convolution

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

- Continuous convolution

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

- Same concept as continuous version

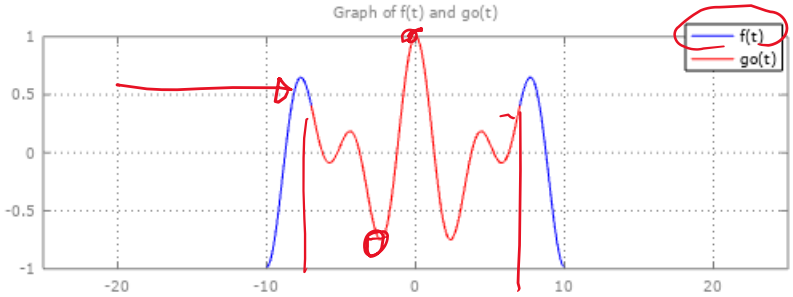
Matched filters

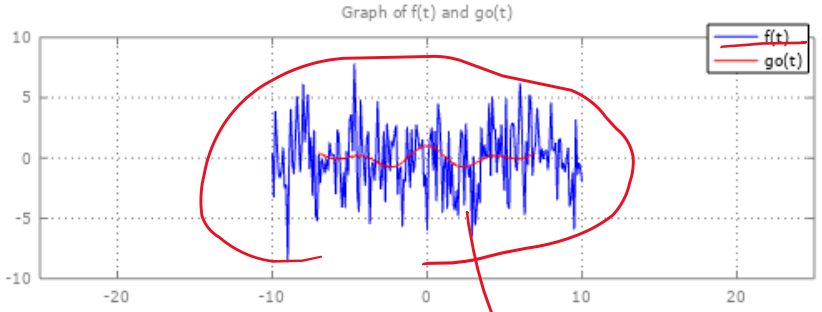
- We can use convolution to detect things that match our pattern
- Convolution can be considered as a filter
- If the filter detects our pattern, it will show up as a nice peak even if there are noise.

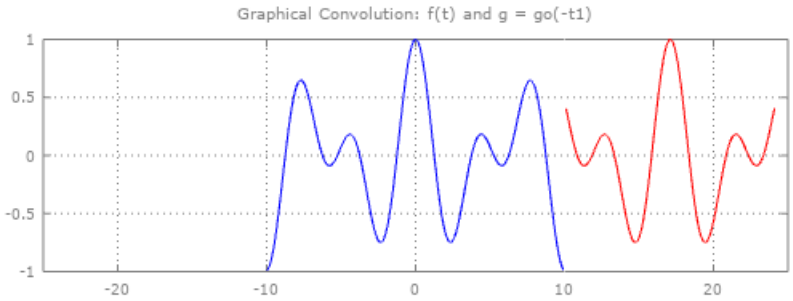
Matched filters

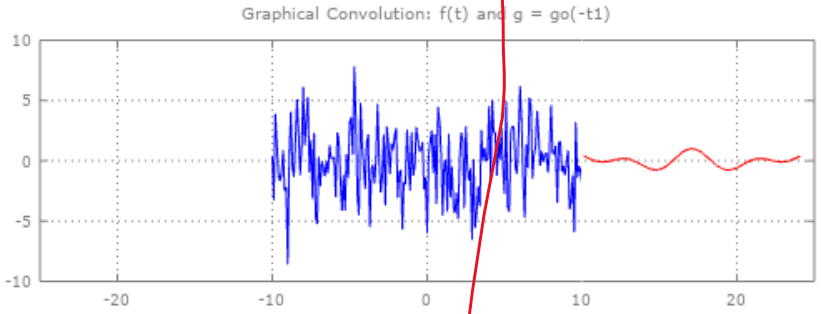
$f * g$

Red: matched filter
Blue: signal

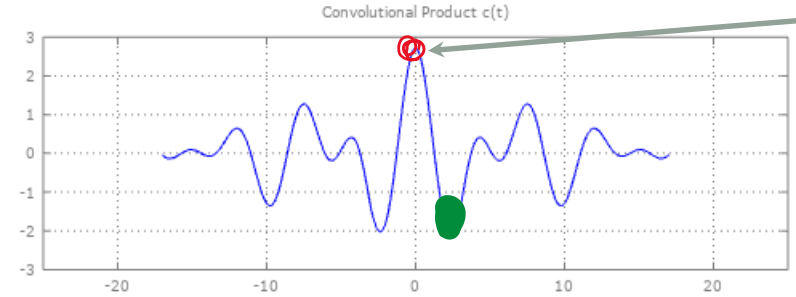


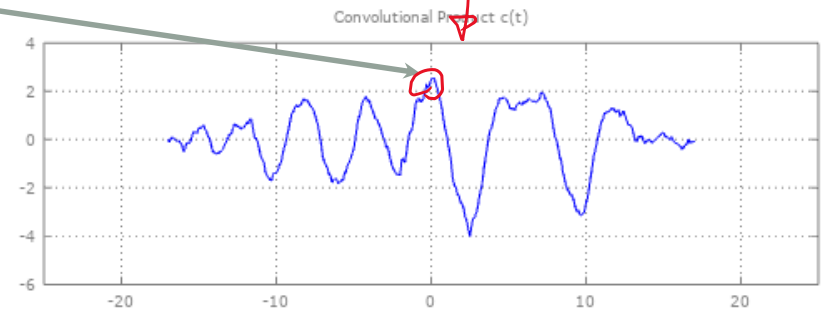






Matched peak





Convolution and Cross-Correlation

- Convolution

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

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

- (Cross)-Correlation

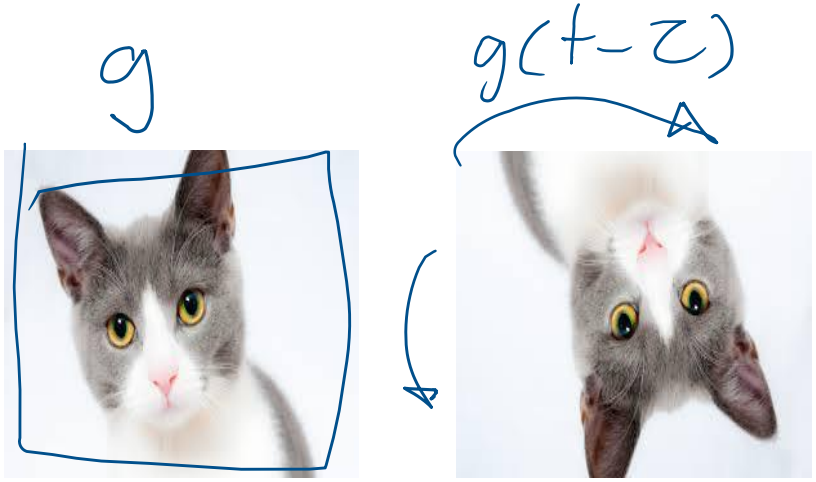
$$(f \star g)(\tau) \stackrel{\text{def}}{=} \int_{-\infty}^{\infty} f^*(t)\, g(t+\tau)\, dt,$$

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

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.

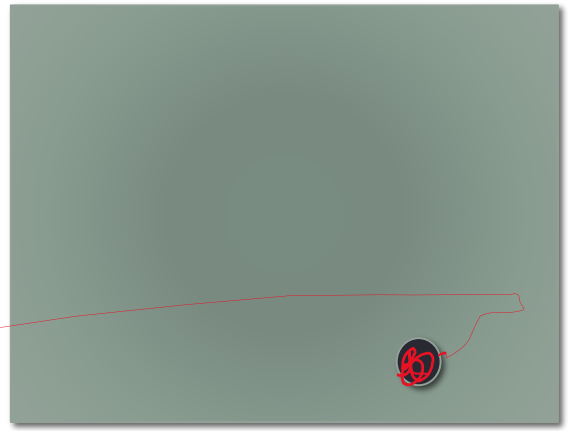
2D convolution

- Flip and shifts in 2D



- But, we no longer flips

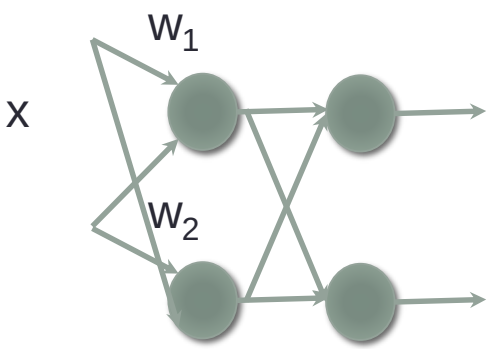
Our match filter



Will get some peak here

PCA as a filter

- $W^T x$

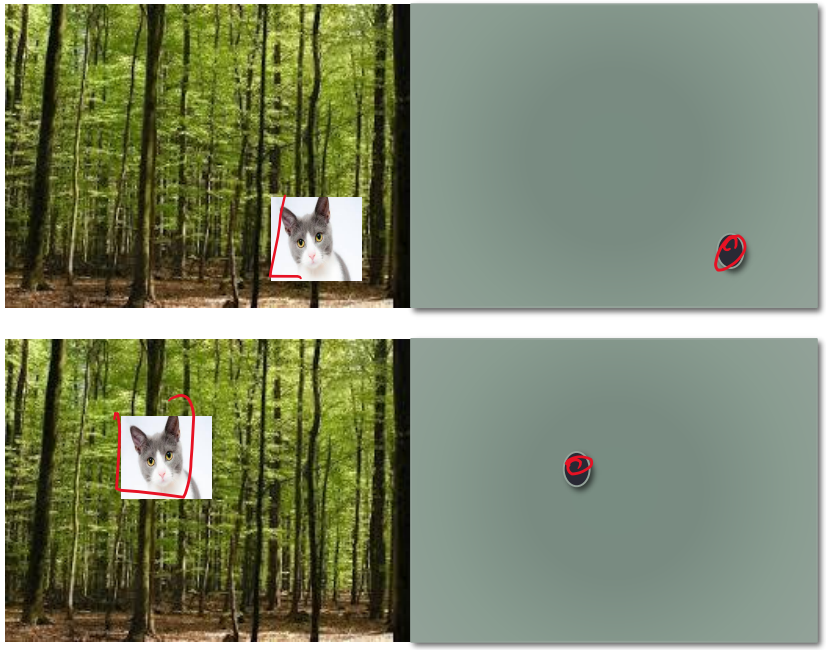


- What happens if I have a person that is off-frame?
- Need another filter that is shifted
- Ans: Convolution with W as filter

x = y

Convolutional Neural networks

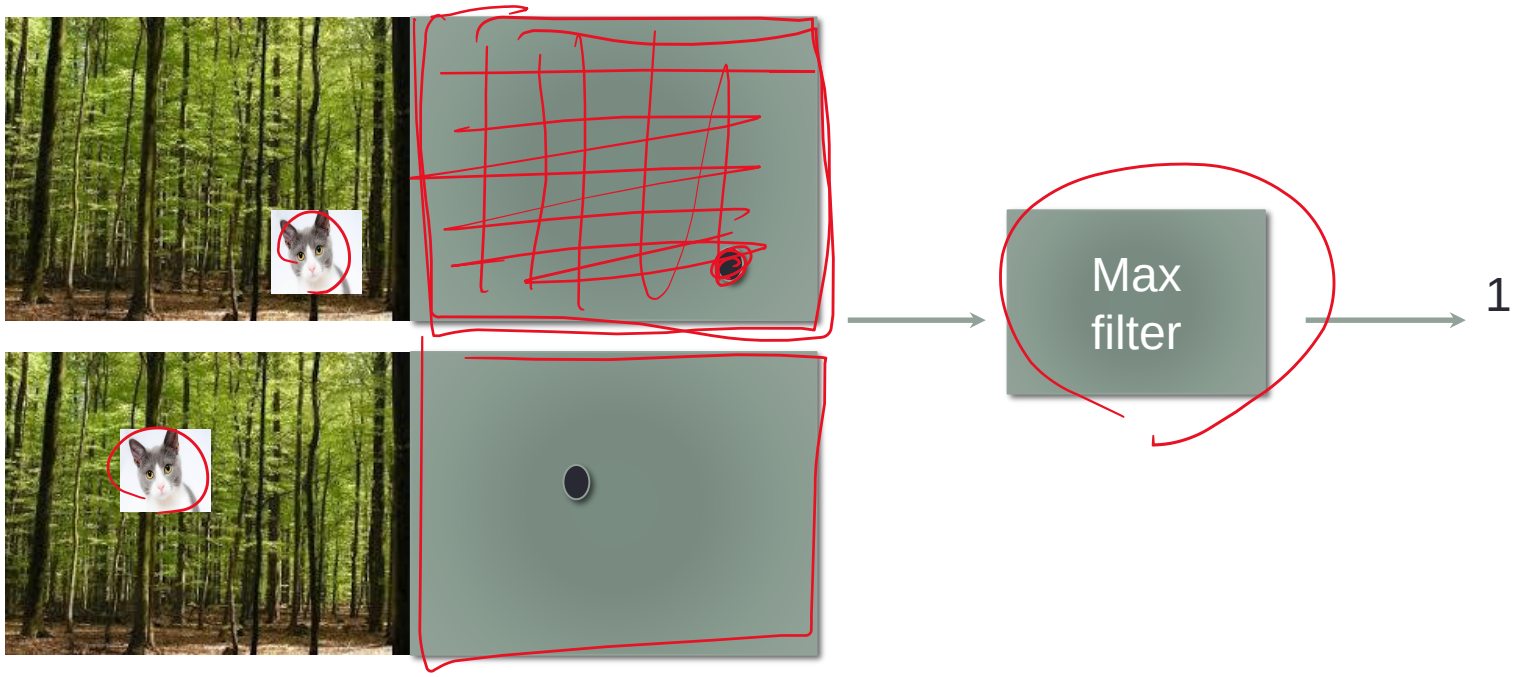
- A neural network with convolutions! (cross-correlation to be precise)
- But we have peaks at different location



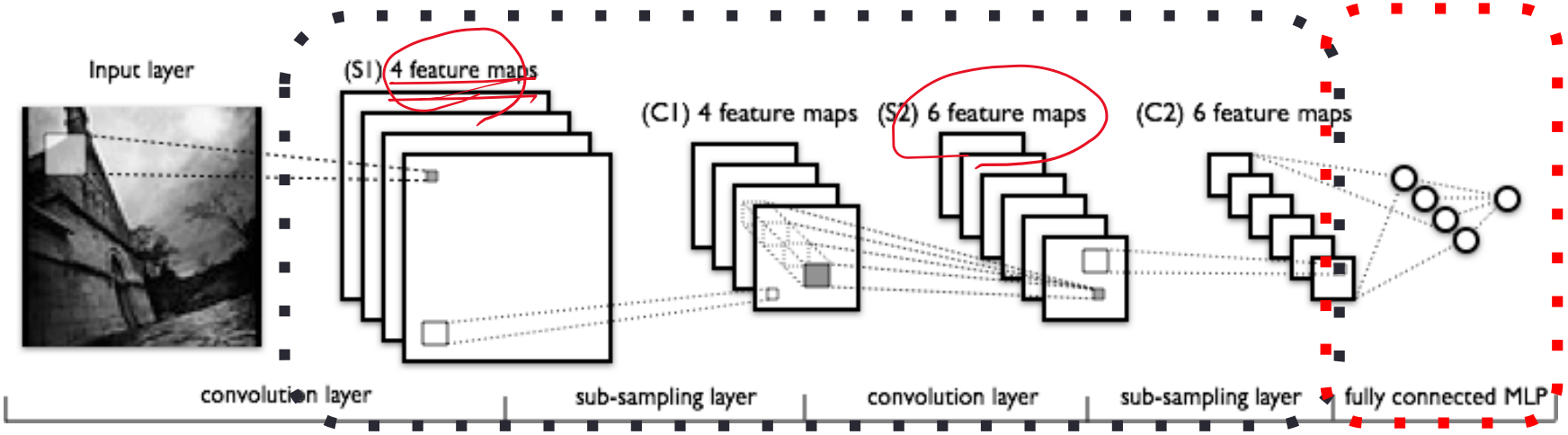
From the point of view of a network, these are two different things.

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)

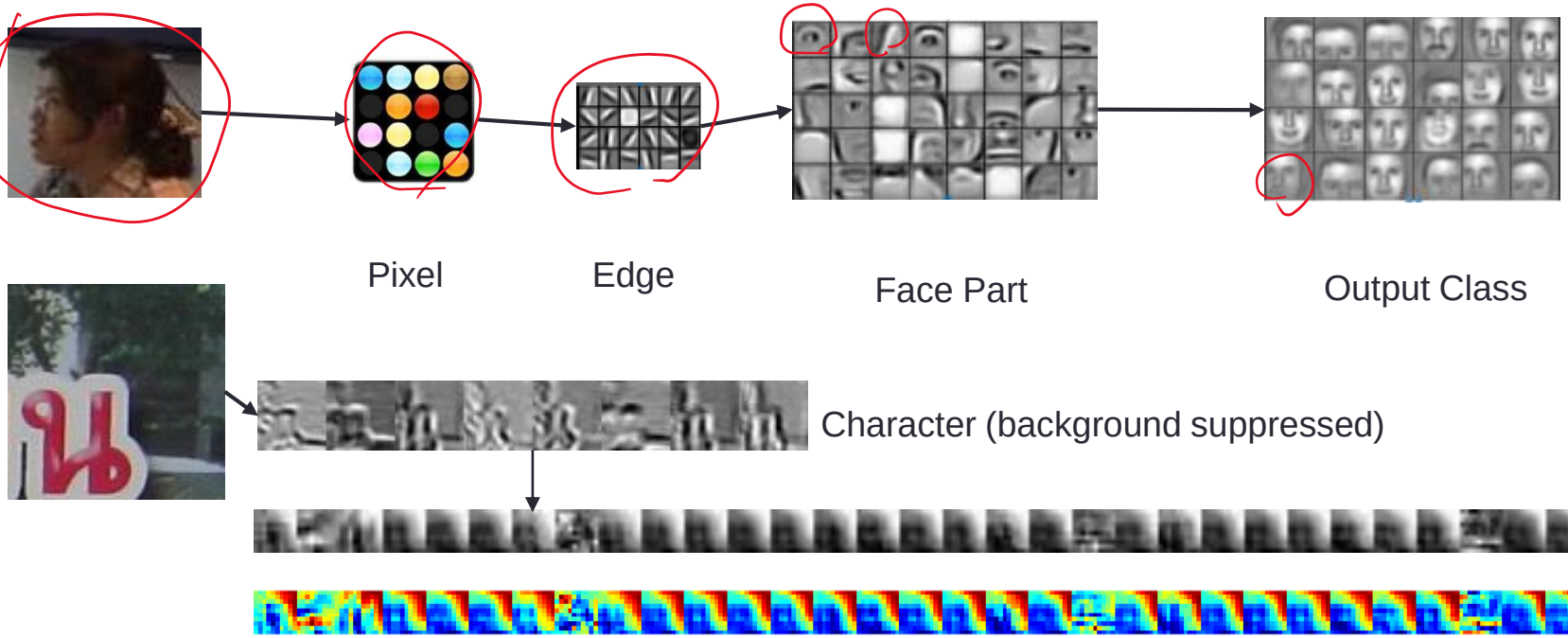


Trainable Feature Extractor Layer **Classification Layer**

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

Convolution Neural Network (CNN)

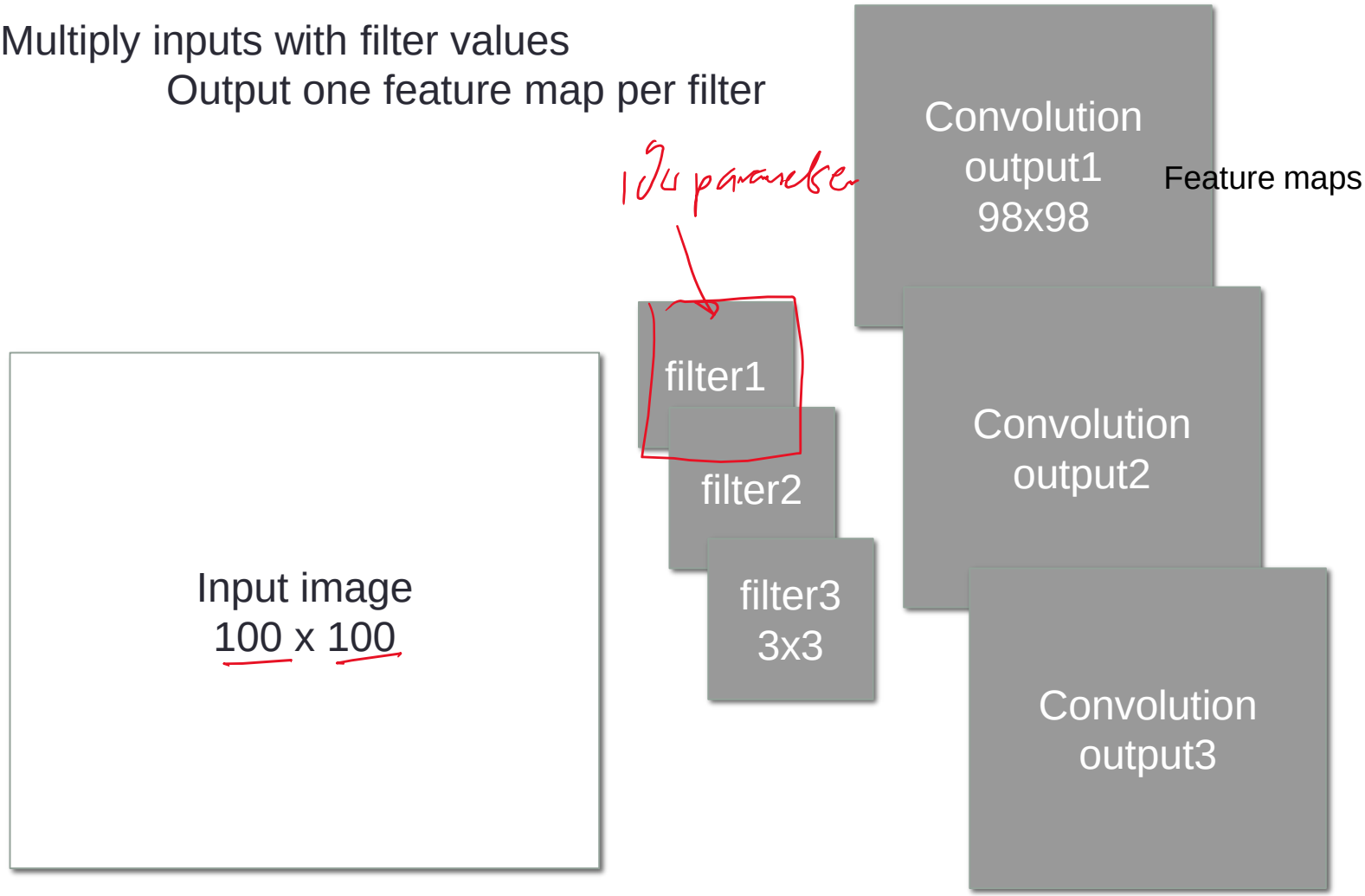
- Hierarchy of representations with increasing level of abstraction
- Each stage is a kind of trainable feature transform
- Image recognition: Pixel →edge → texture → part → object



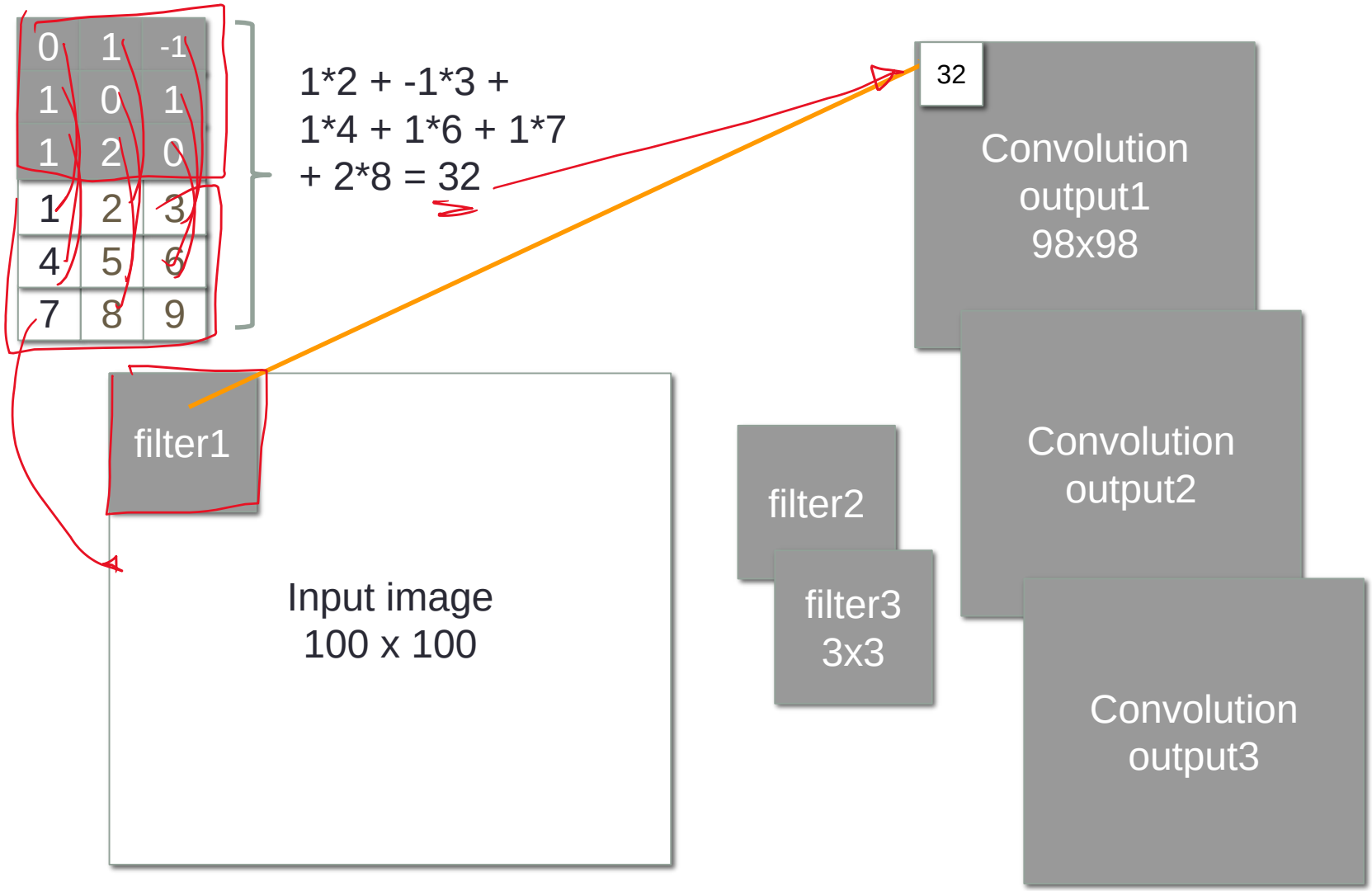
Convolution filters

Multiply inputs with filter values

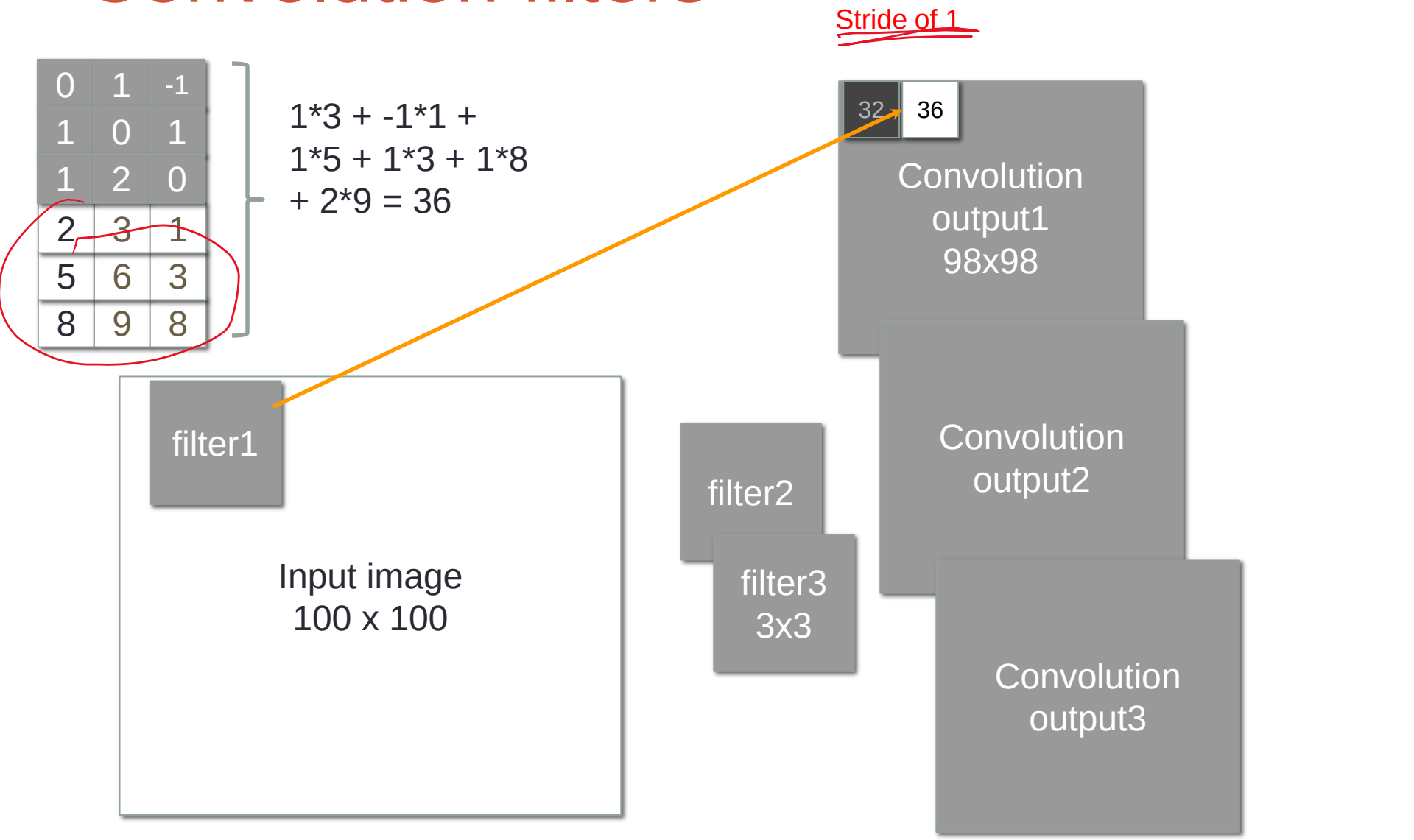
Output one feature map per filter



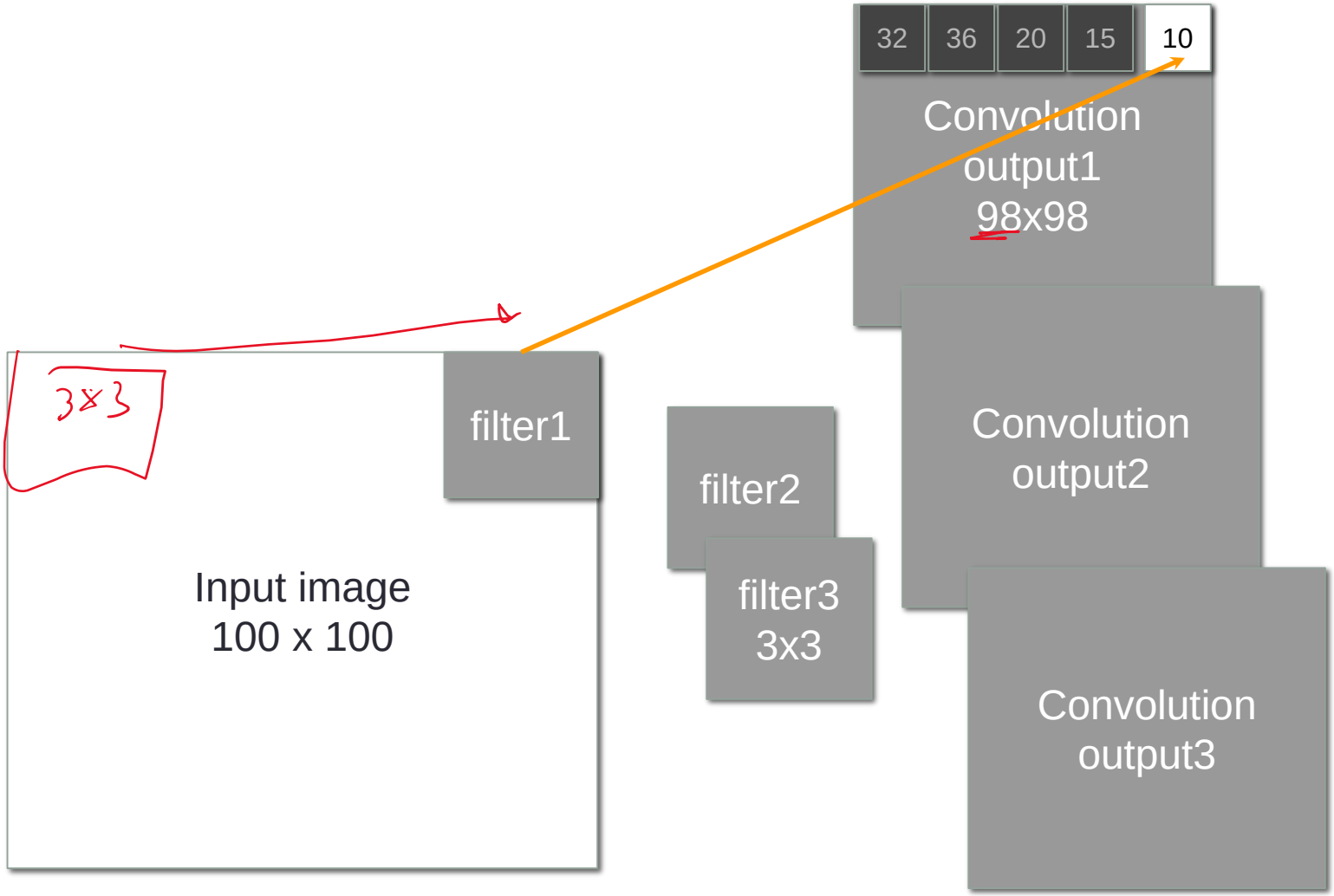
Convolution filters



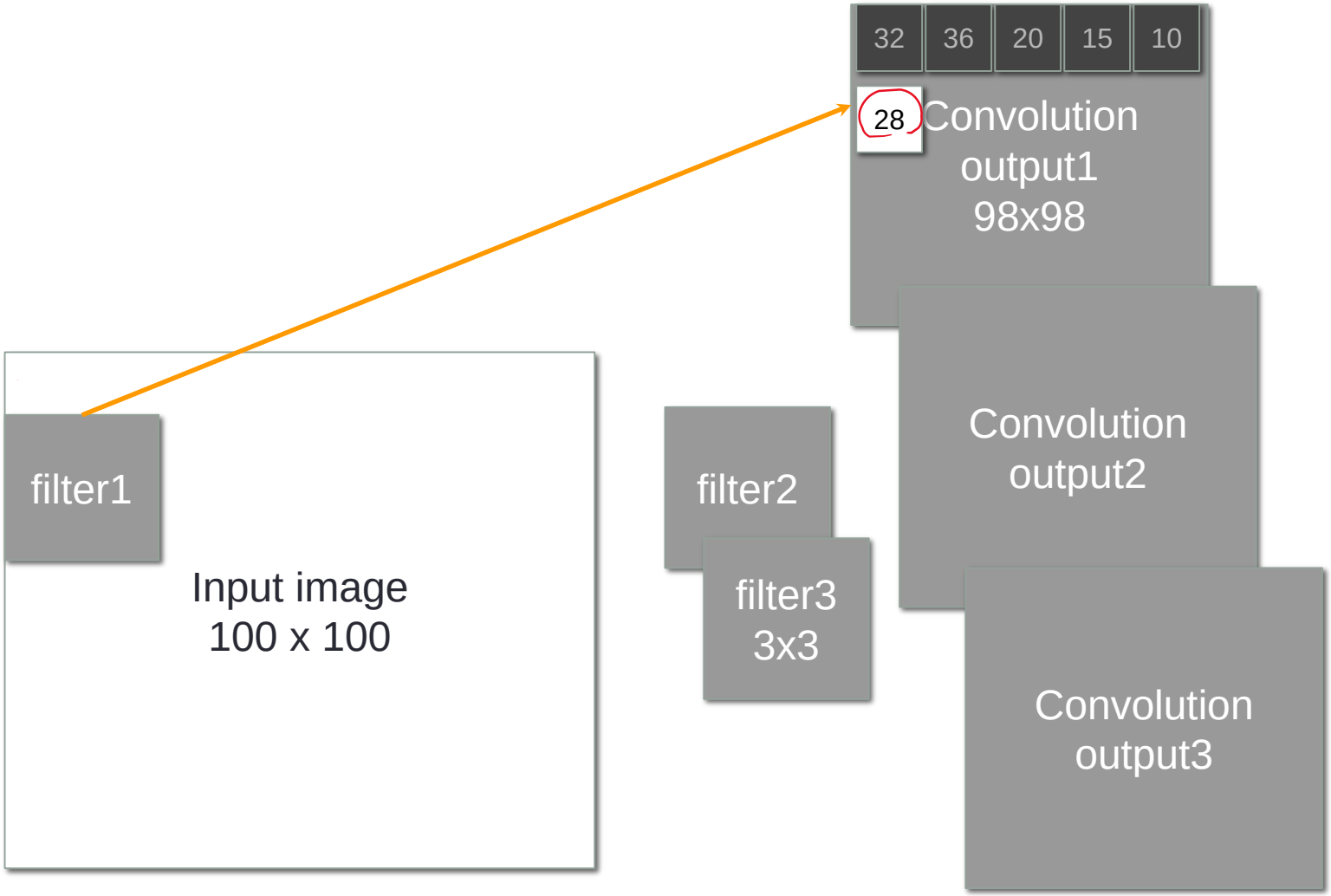
Convolution filters



Convolution filters



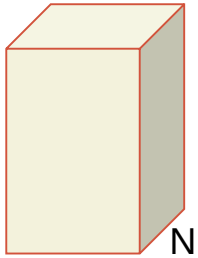
Convolution filters



Convolution filters

3 channel

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

16

filter2

filter1

filter3
3x3

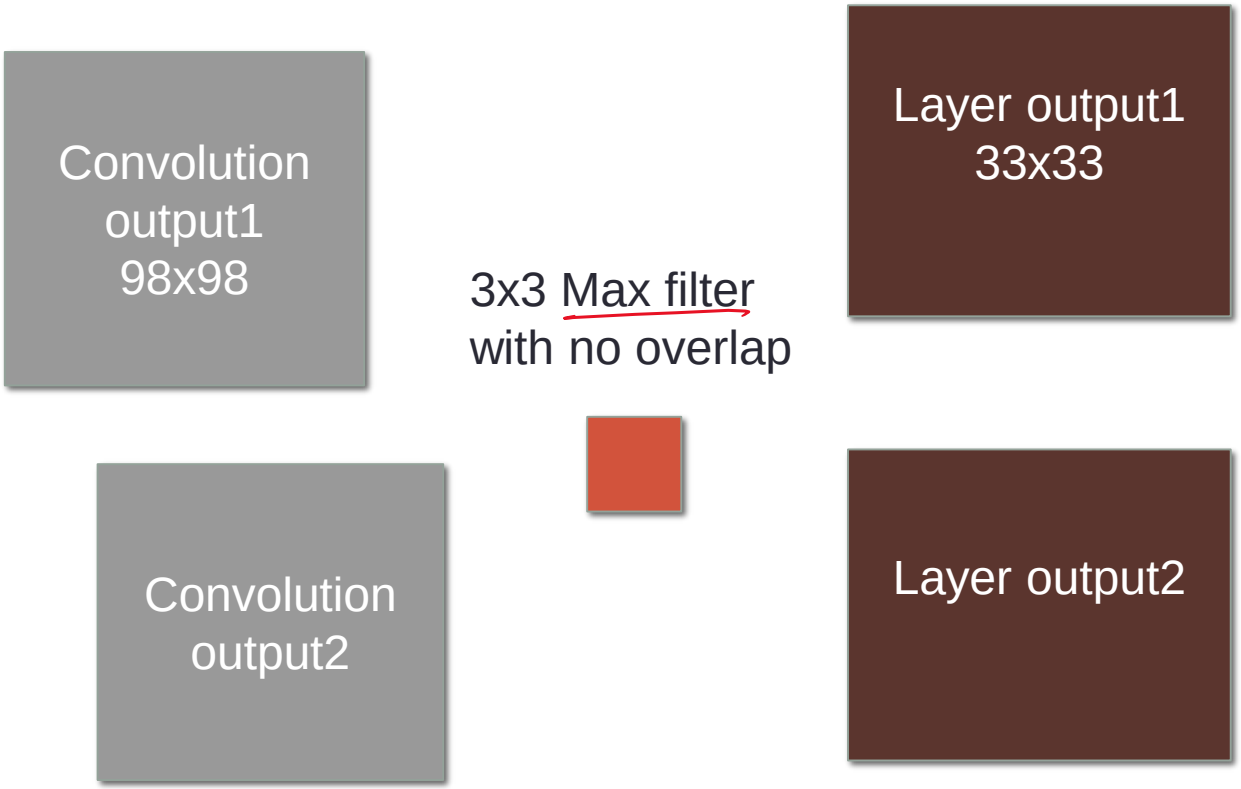
Input image
100 x 100

Convolution
output2

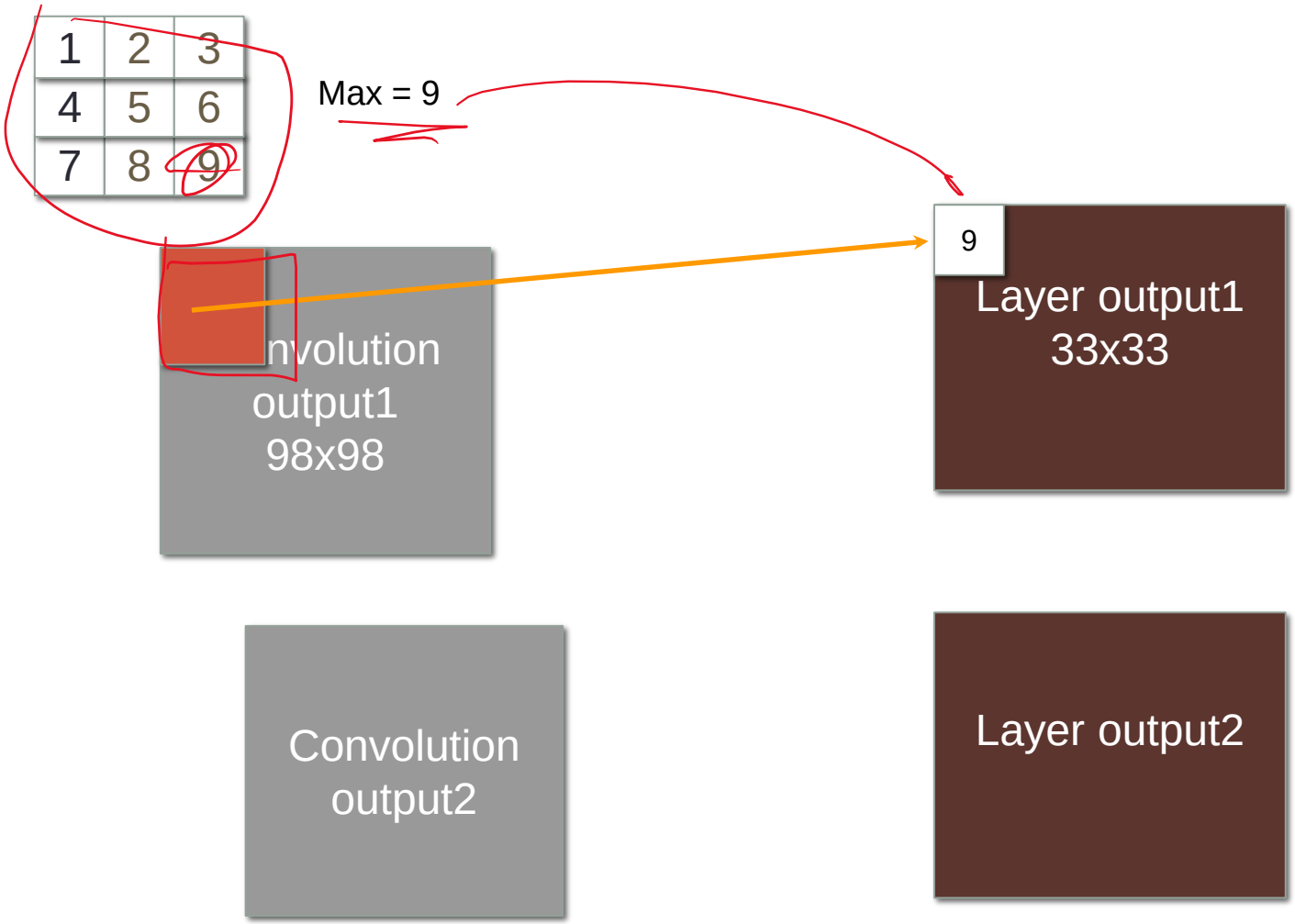
Convolution
output3

Pooling/subsampling

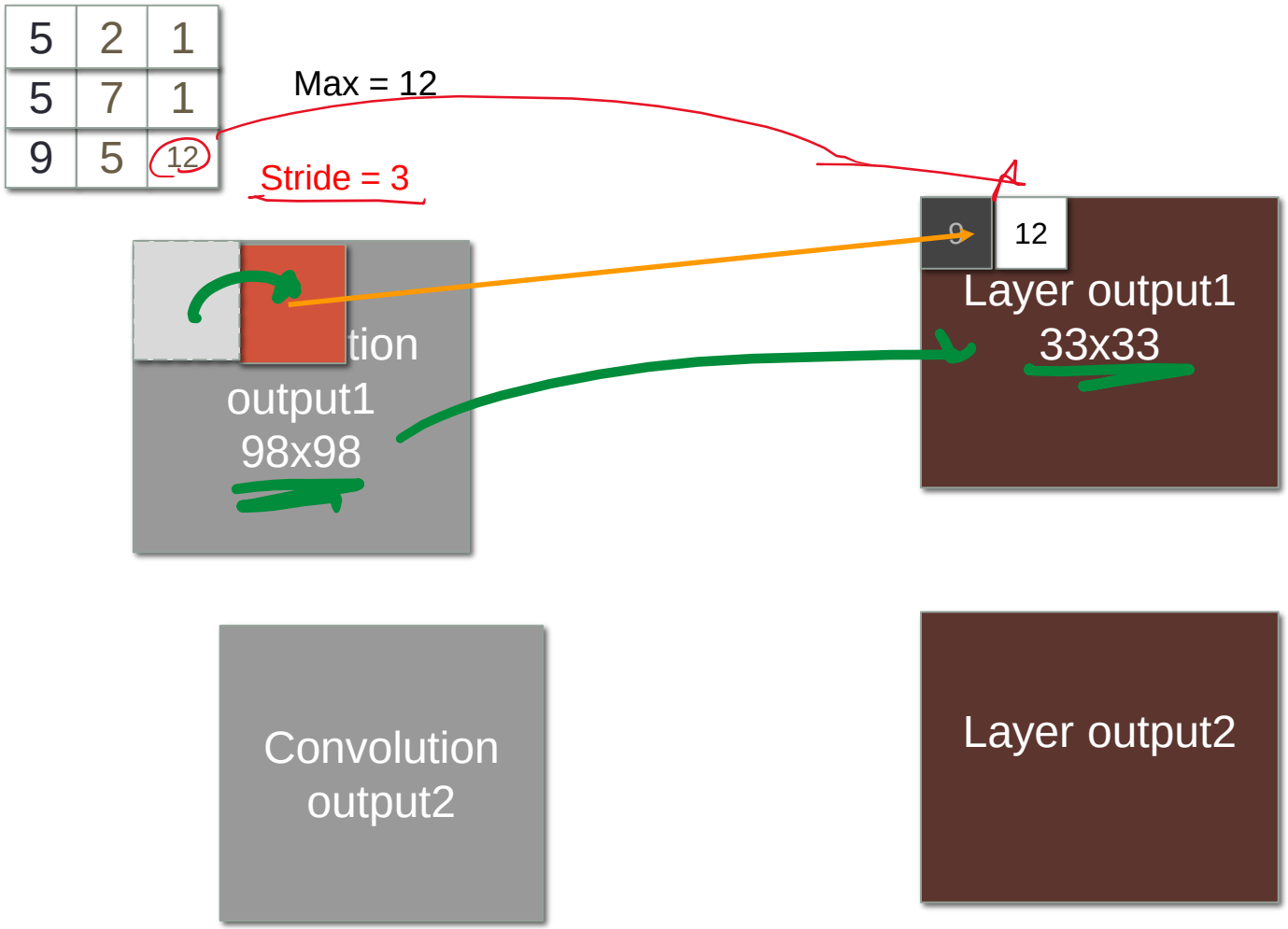
Reduce dimension of the feature maps



Pooling/subsampling

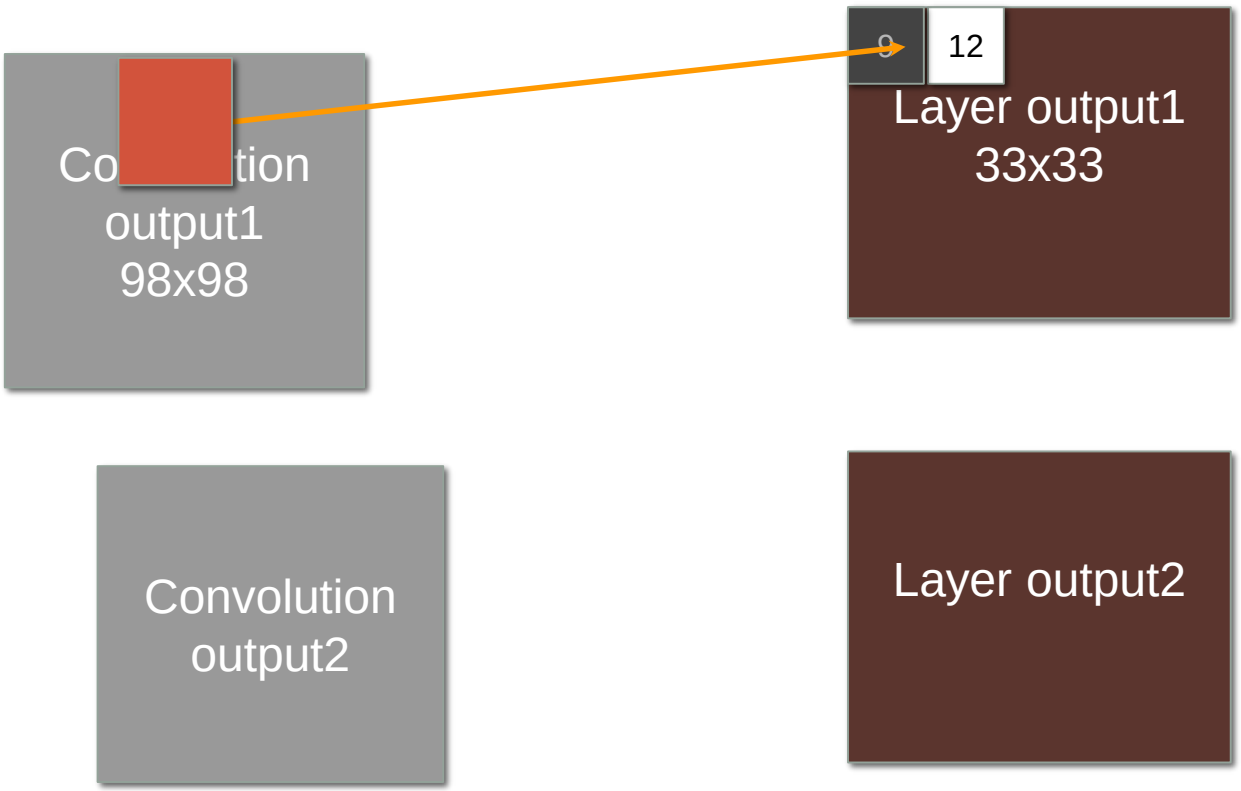


Pooling/subsampling



Pooling/subsampling

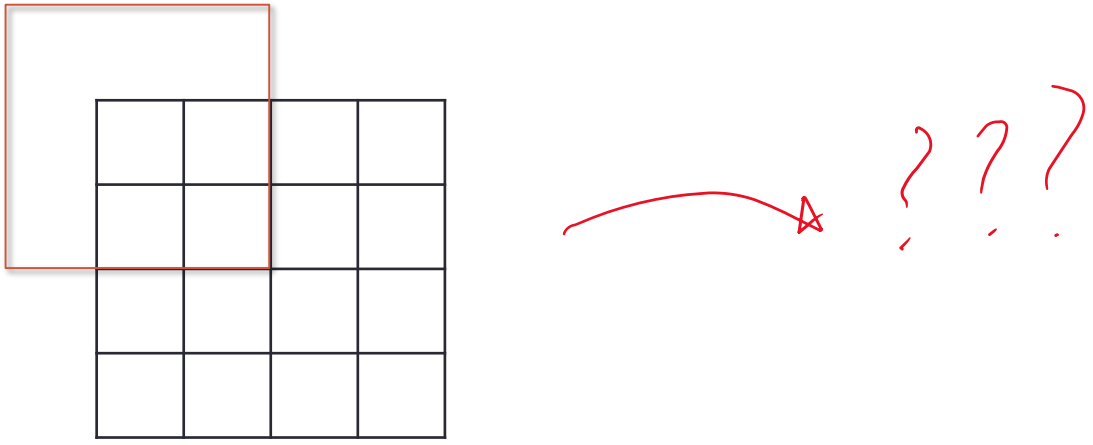
Can use other functions besides max
Example, average



Convolution puzzle

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

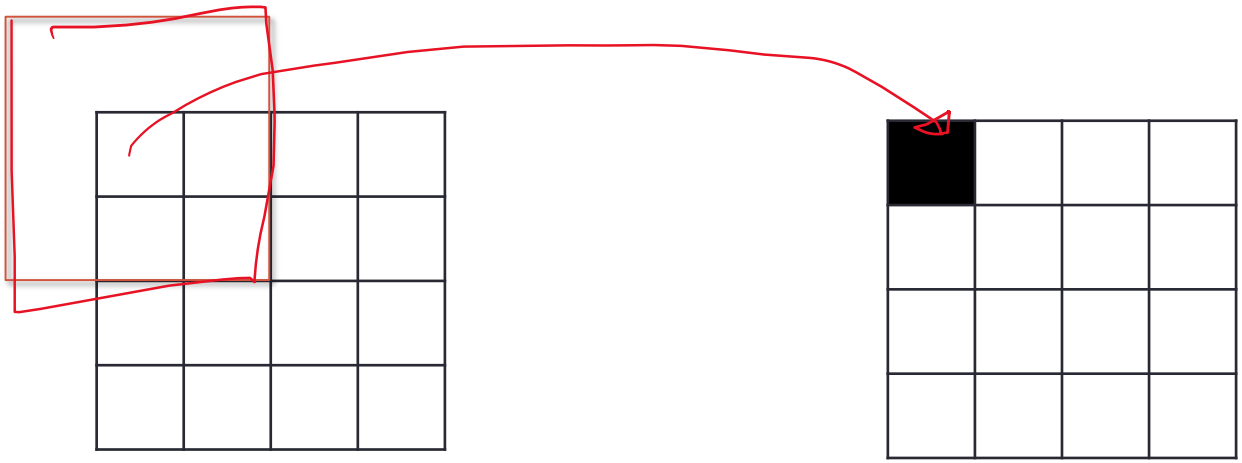
What is the output size?



Convolution puzzle

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

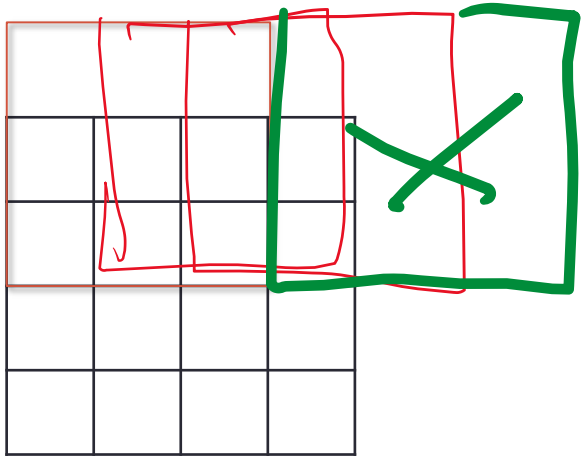
What is the output size?

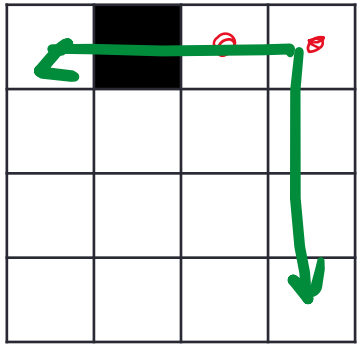


Convolution puzzle

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

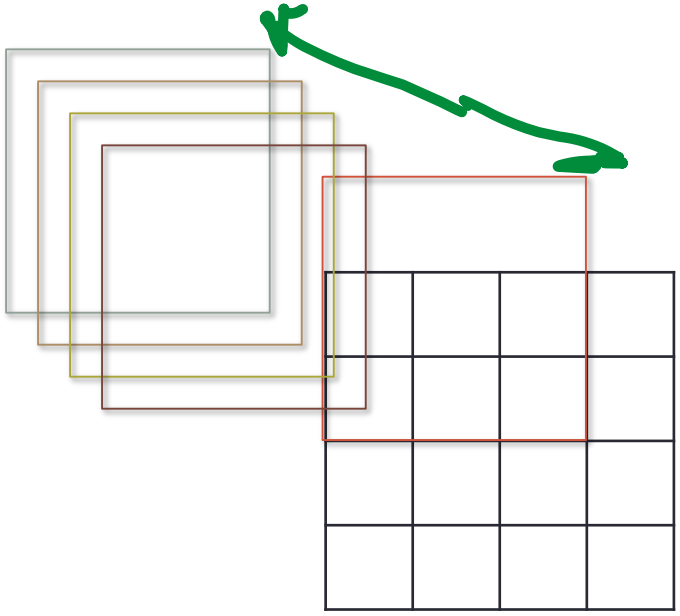
What is the output size?

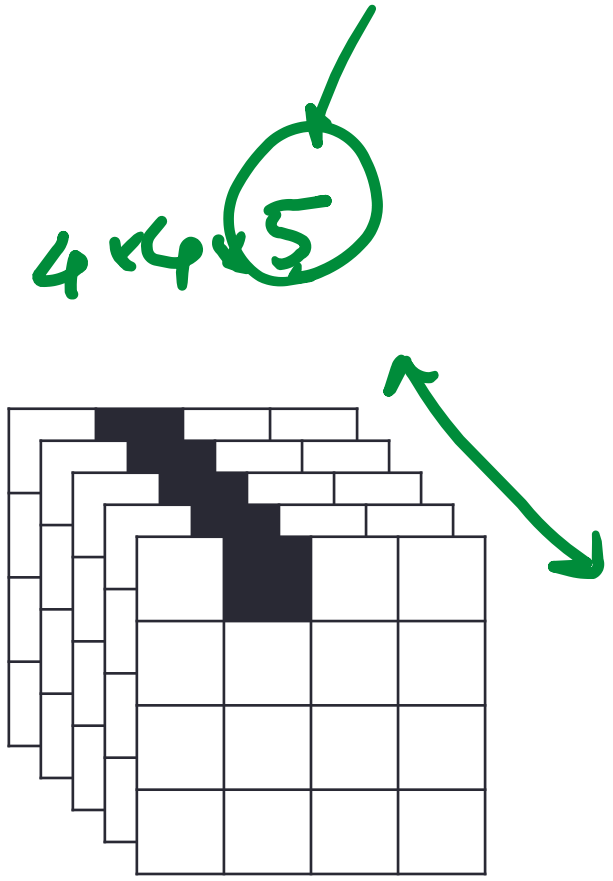




Convolution puzzle

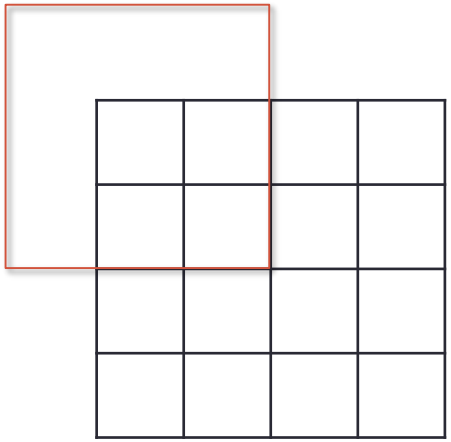
5 filters 3x3 filter pad, stride 1, pad 1





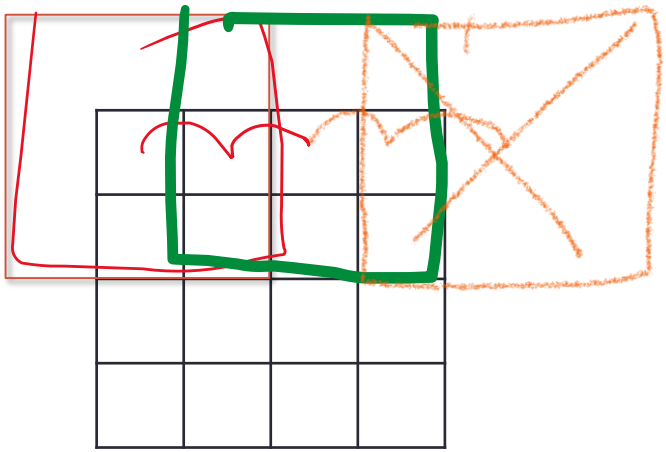
Convolution puzzle

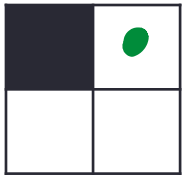
3x3 filter pad, stride 2, pad 1



Convolution puzzle

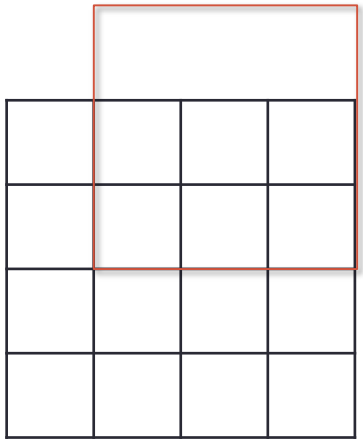
3x3 filter pad, stride 2, pad 1

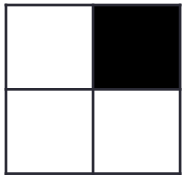




Convolution puzzle

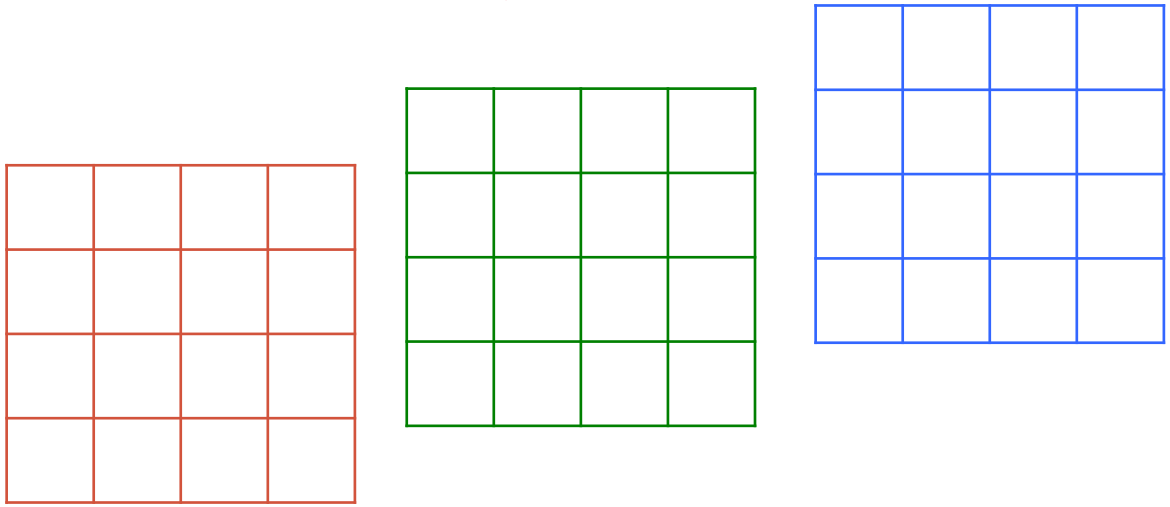
3x3 filter pad, stride 2, pad 1





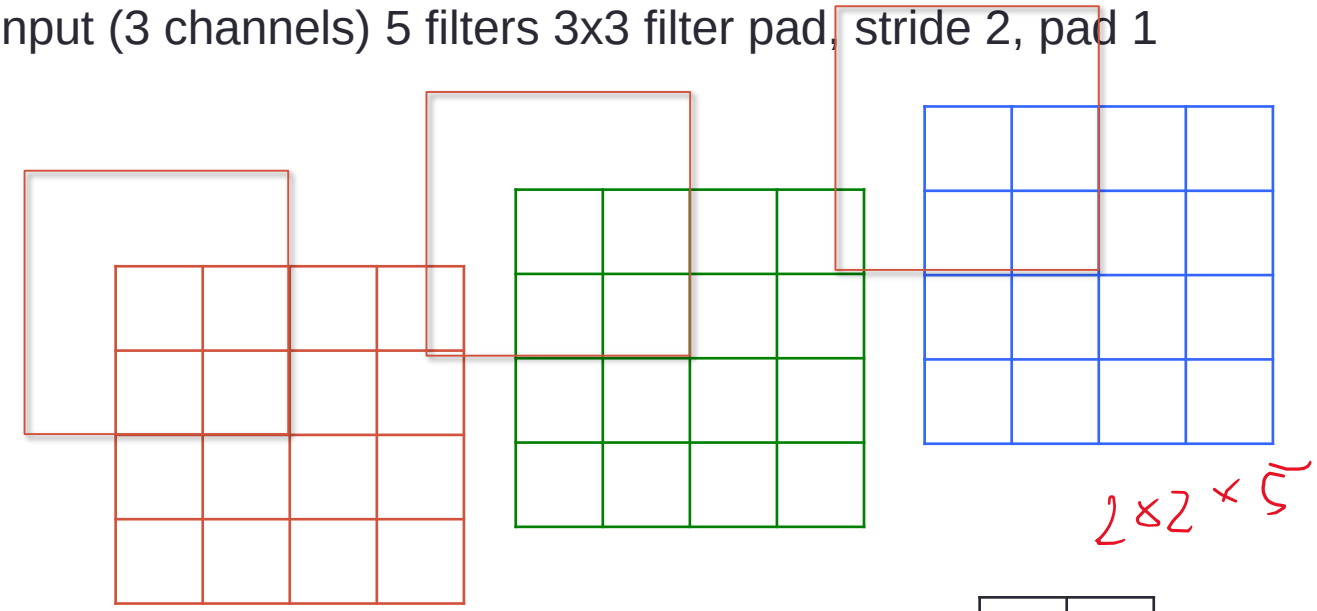
Convolution puzzle

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

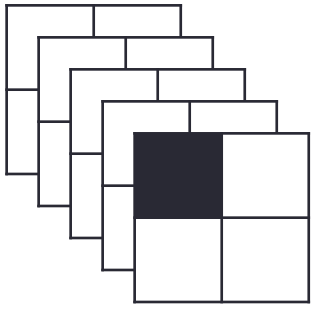


Convolution puzzle

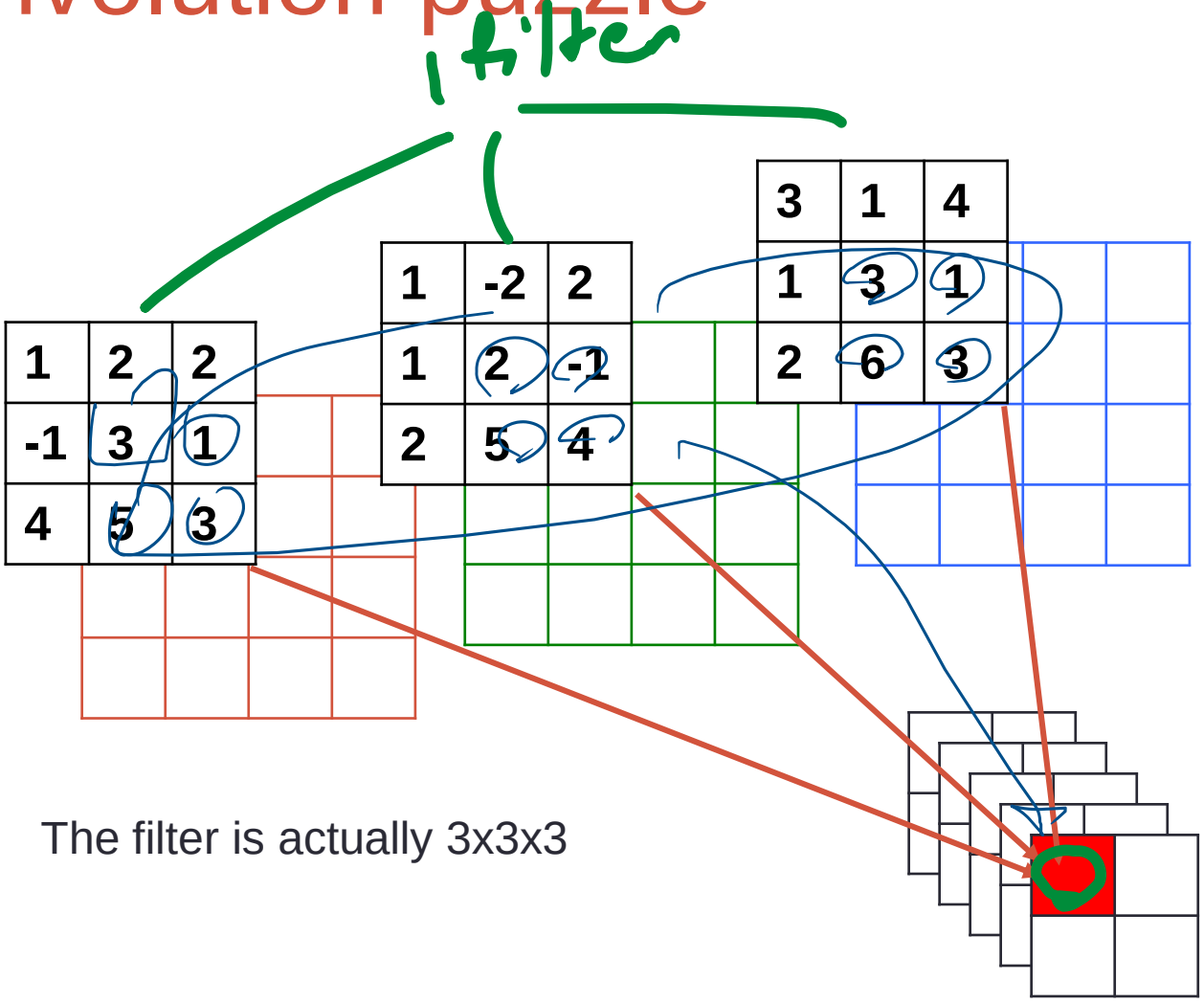
RGB input (3 channels) 5 filters 3x3 filter pad, stride 2, pad 1



The filter is actually 3x3x3

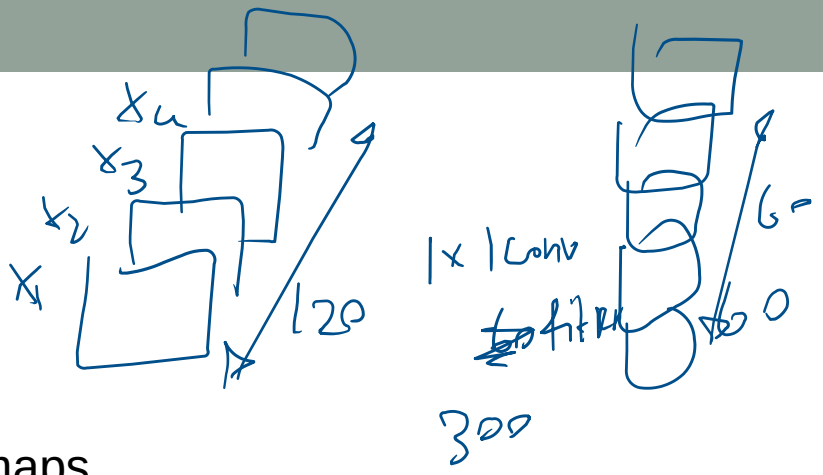


Convolution puzzle

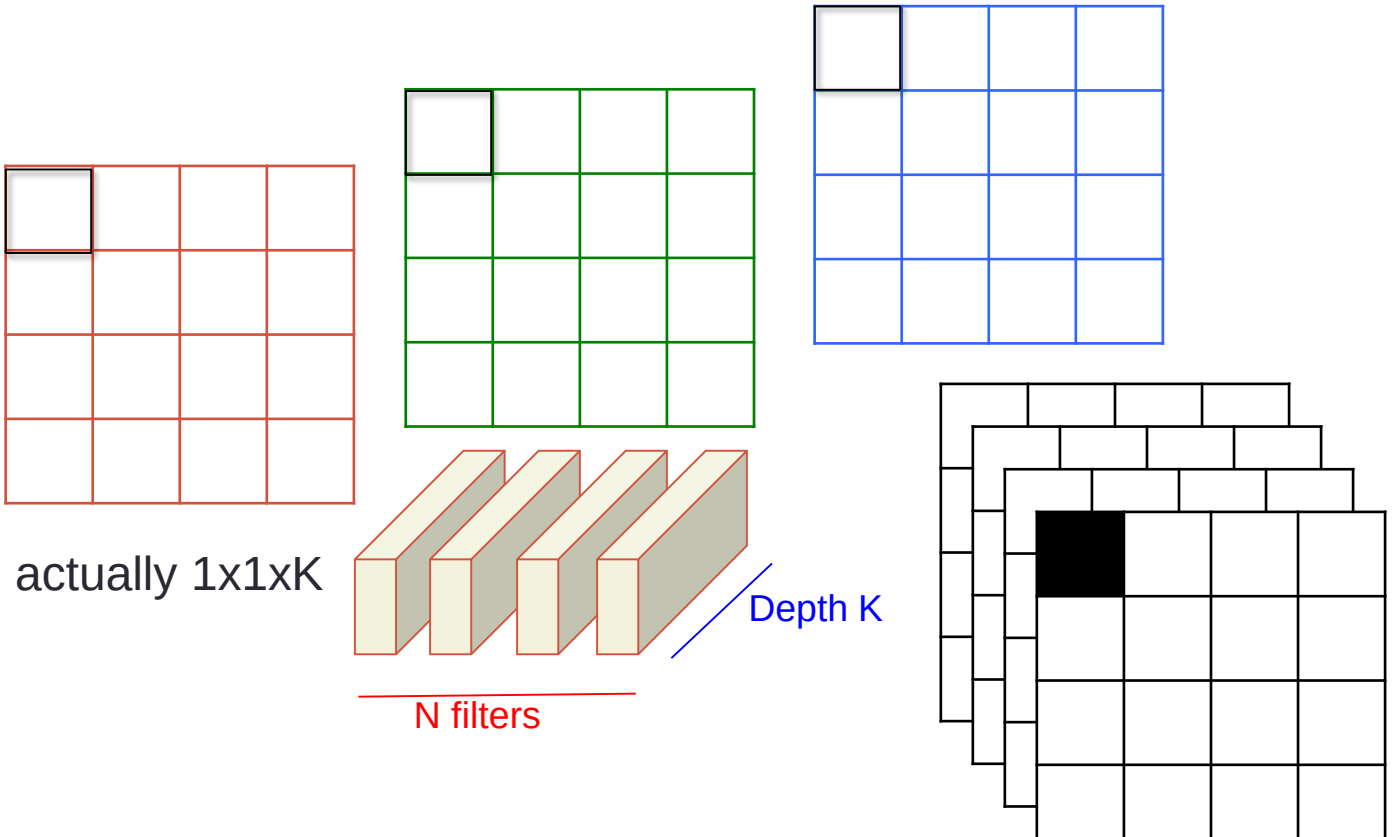


The filter is actually 3x3x3

1x1 convolution



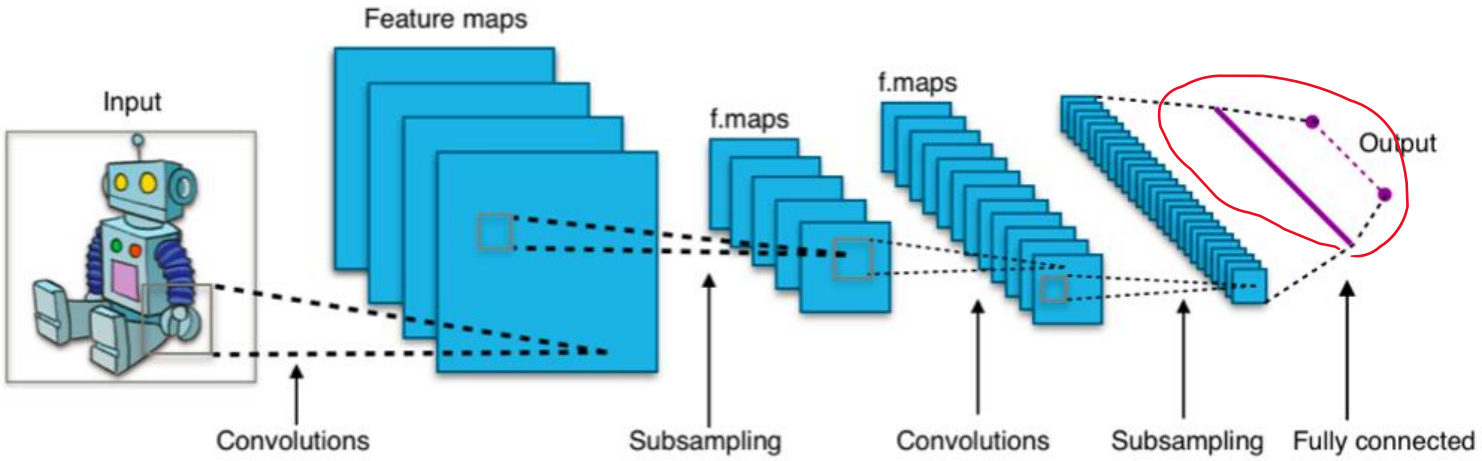
Reduces the dimension of feature maps



The filter is actually 1x1xK

CNN overview

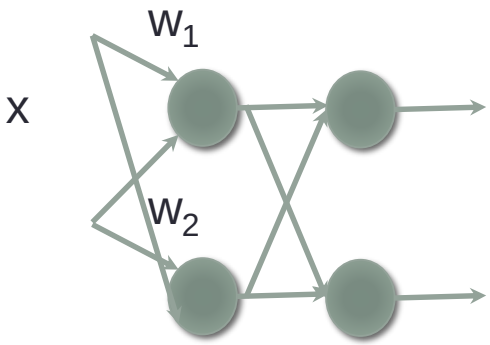
- Filter size, number of filters, filter shifts, and pooling rate are all parameters
- Usually followed by a fully connected network at the end
  - CNN is good at learning low level features
  - DNN combines the features into high level features and classify



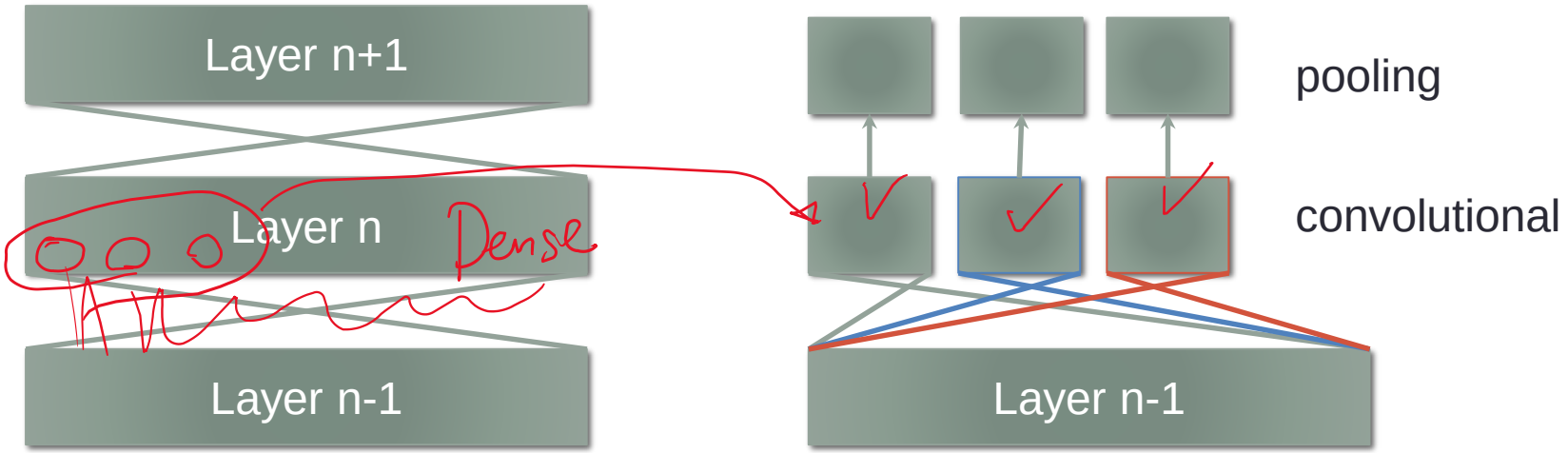
https://en.wikipedia.org/wiki/Convolutional_neural_network#/media/File:Typical_cnn.png

Inductive bias in convolution neural networks

- $W^T x$

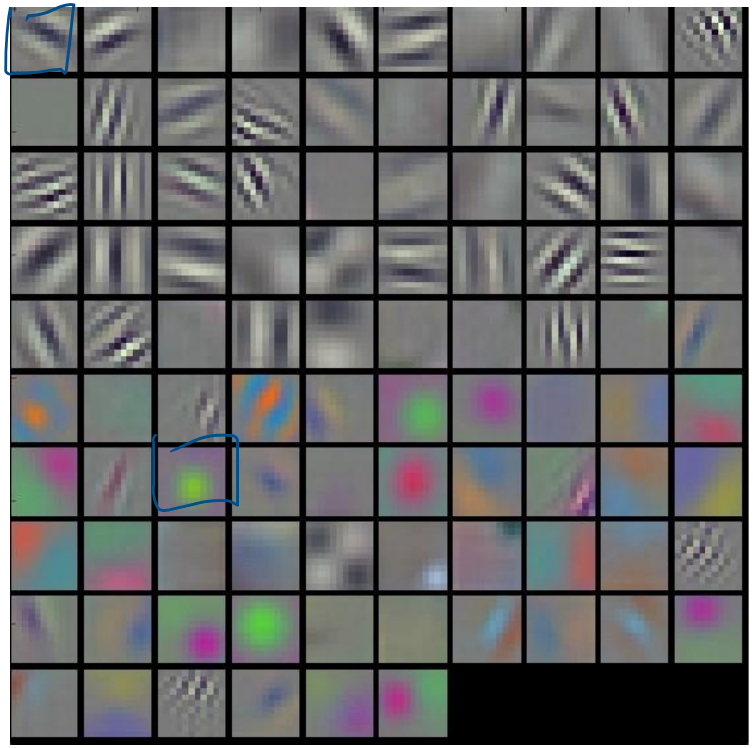


- Cats at different location might need two neurons for different locations in fully connect NNs.
- CNN shares the parameters in 1 filter
- The network is no longer fully connected



Visualizing convolutional layers

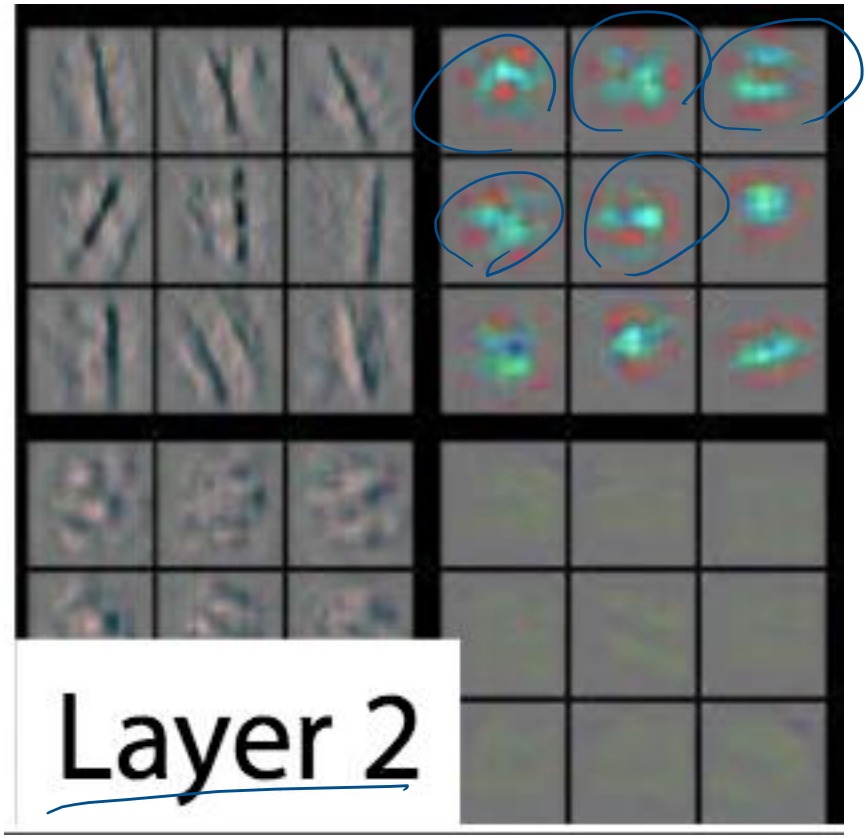
- Just like PCA, we can visualize the weights of a transform
- “Matched filters”



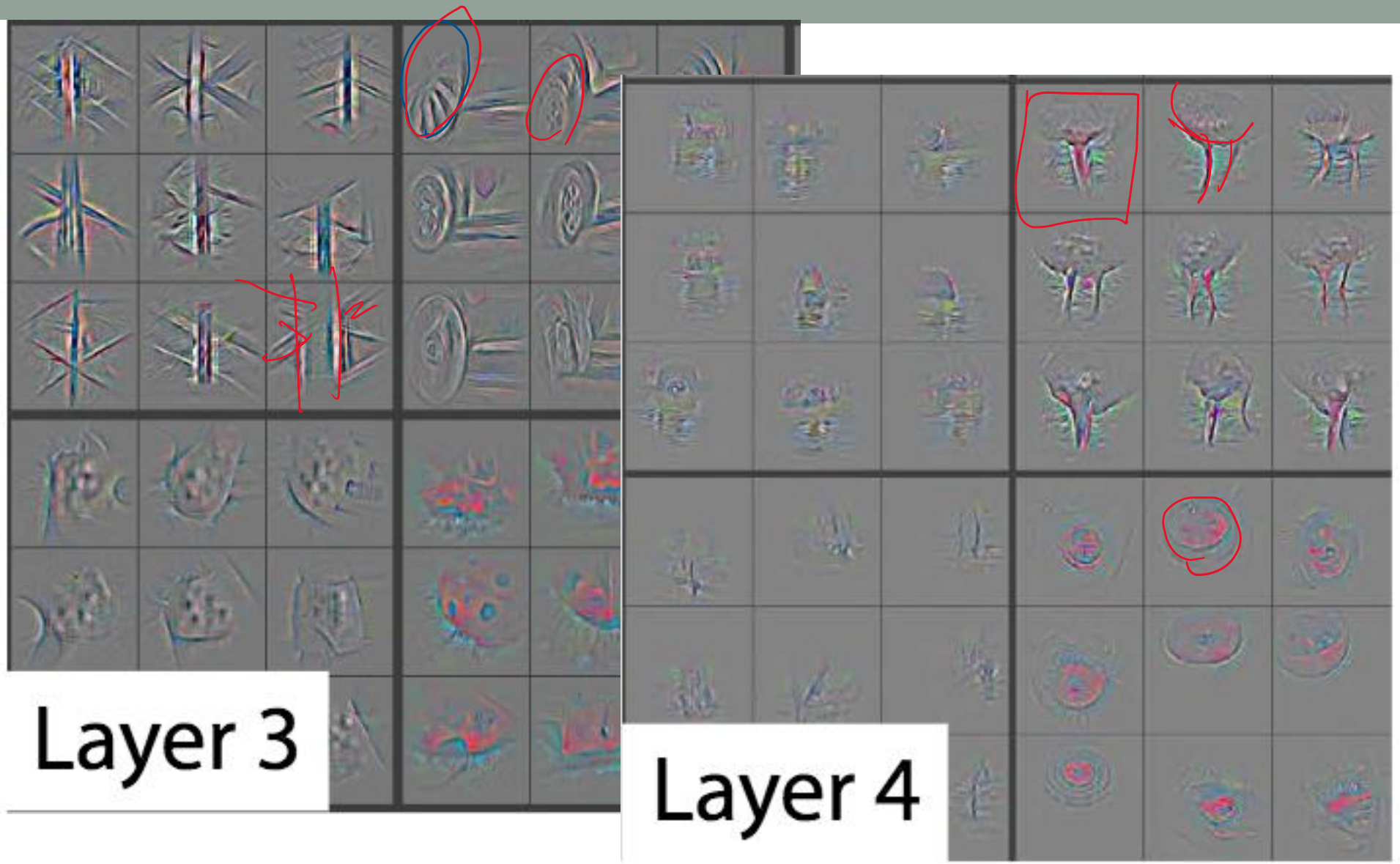
http://cs231n.github.io/understanding-cnn/

Higher layer captures higher-level concepts

Layer 1



https://arxiv.org/pdf/1311.2901.pdf

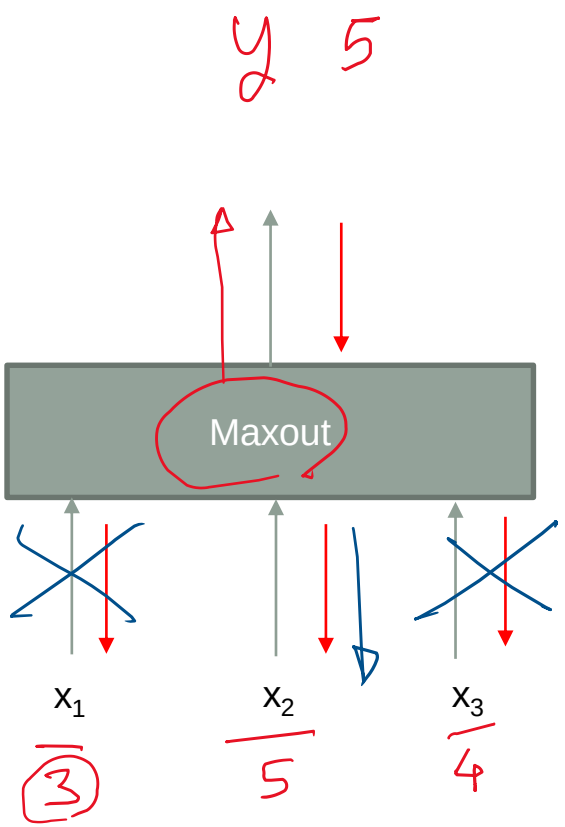

Layer 3
Layer 4

Pooling/subsampling

- Max filter -> Maxout
  - Backward pass?

max → 1

other → 0



$\frac{dy}{dx_1} = ?\ 0$

$\frac{dy}{dx_2} = ?\ 1$

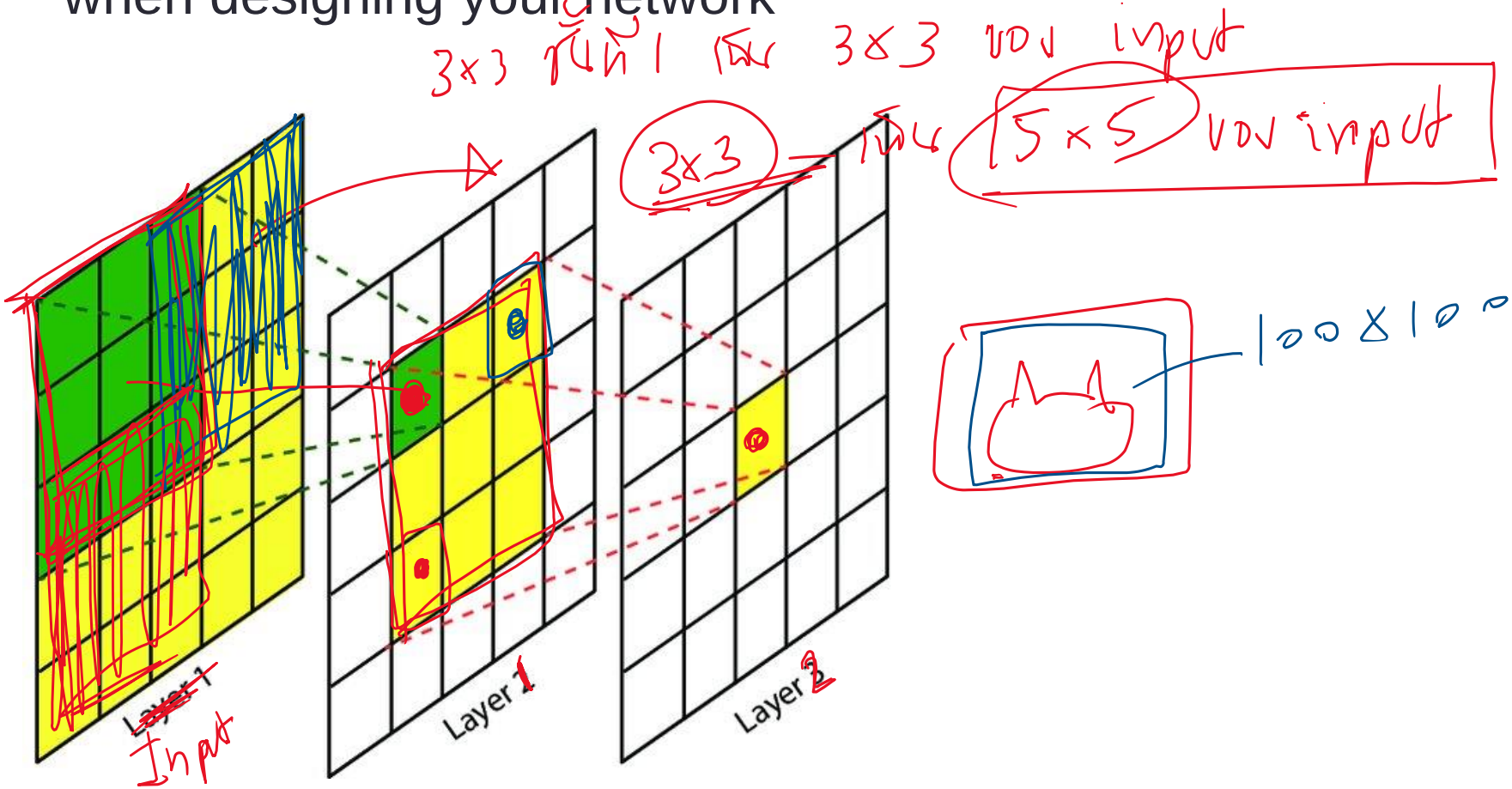
$\frac{dy}{dx_3} = ?\ 0$

Common schemes

- INPUT -> [CONV -> RELU -> POOL]*N -> [FC -> RELU]*M -> FC
- INPUT -> [CONV -> RELU -> CONV -> RELU -> POOL]*N -> [FC -> RELU]*M -> FC

Receptive field

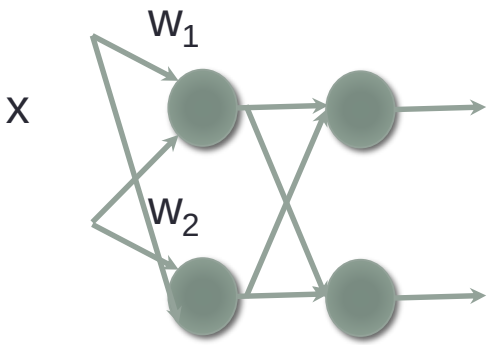
You might want to consider about how large is your pattern when designing your network

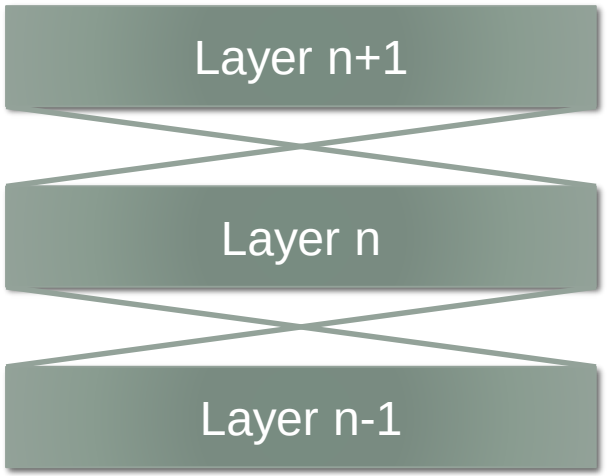


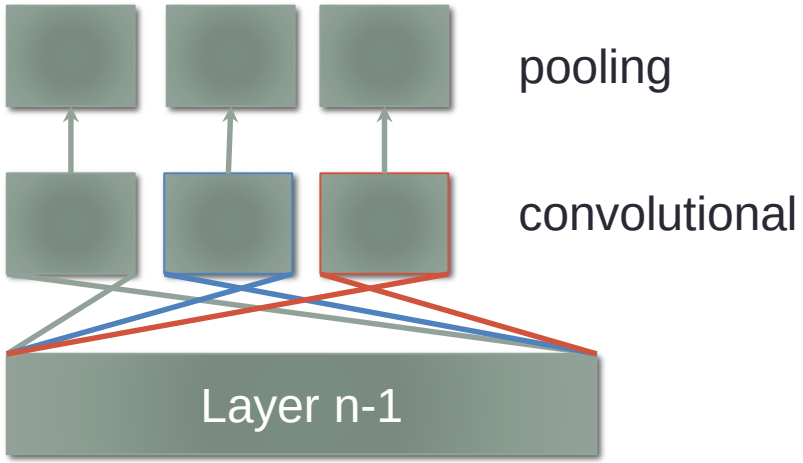
source image

Inductive bias for time?

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

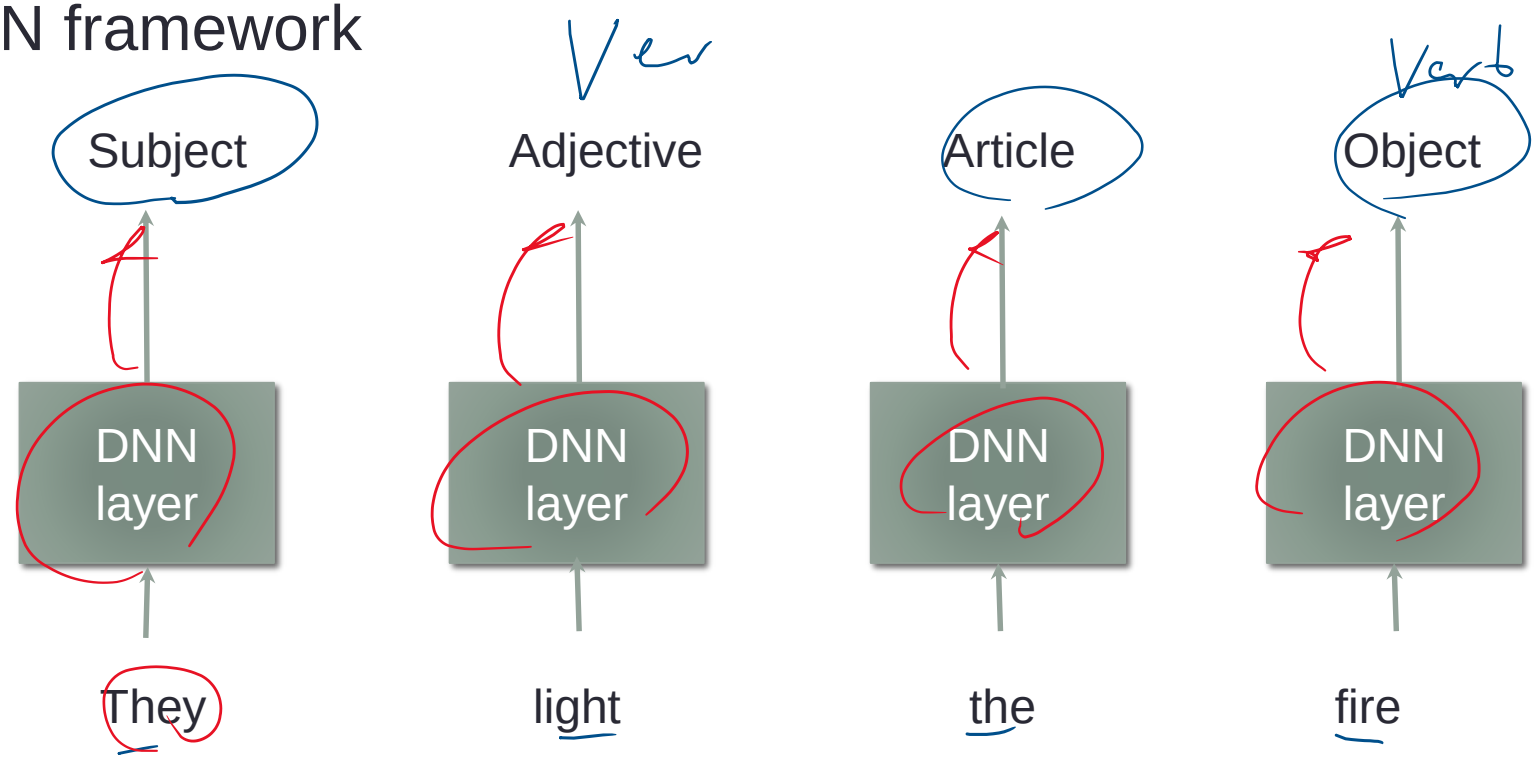






Recurrent neural network (RNN)

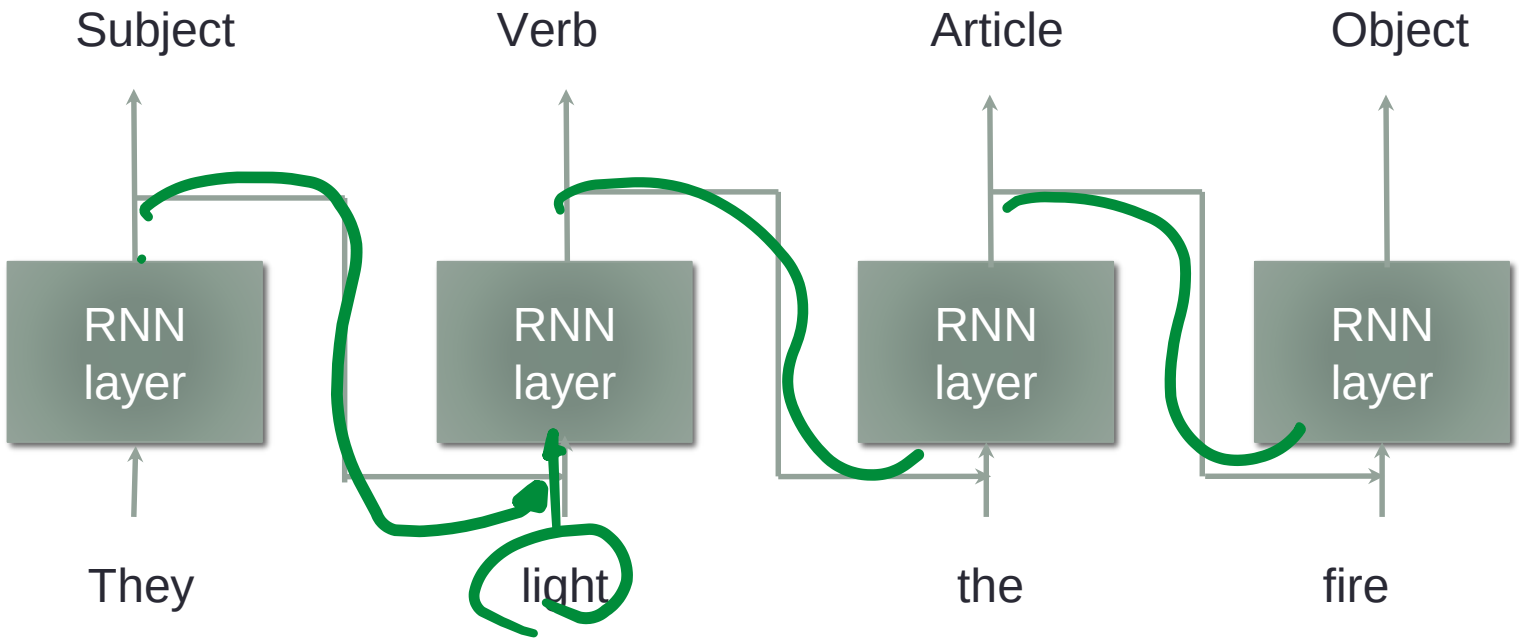
- DNN framework



Problem 1: need a way to remember the past

Recurrent neural network (RNN)

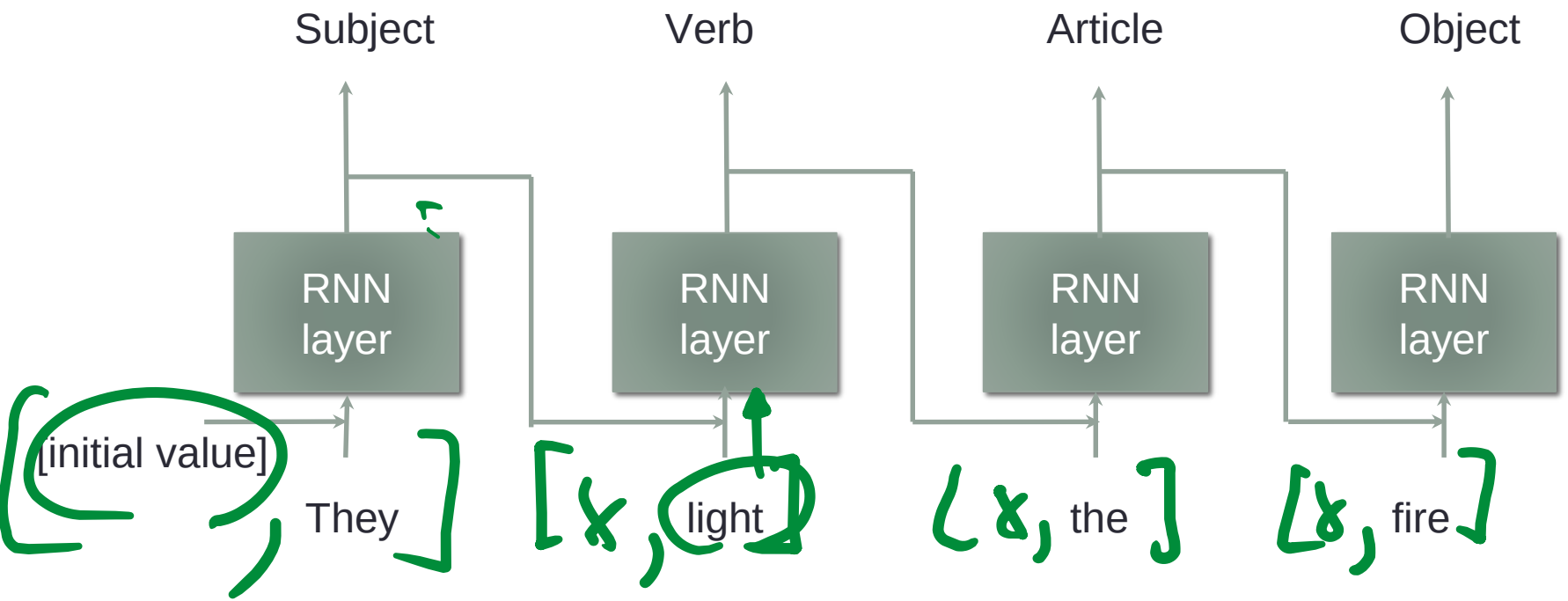
- RNN framework



Output of the layer encodes something meaningful about the past

Recurrent neural network (RNN)

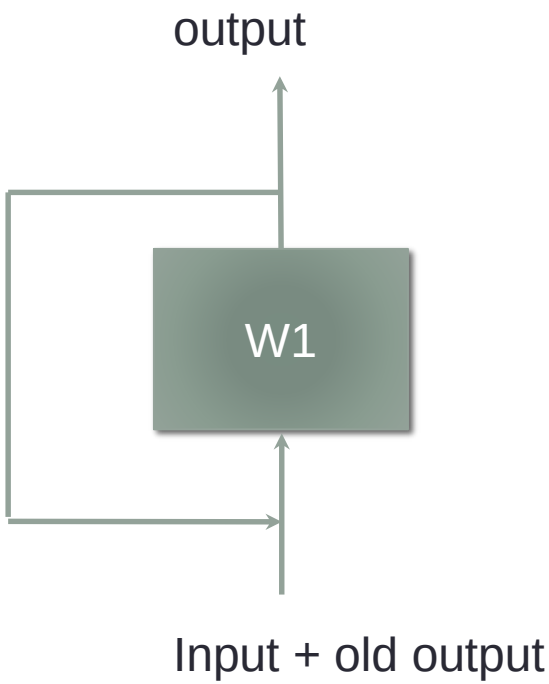
- RNN framework



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

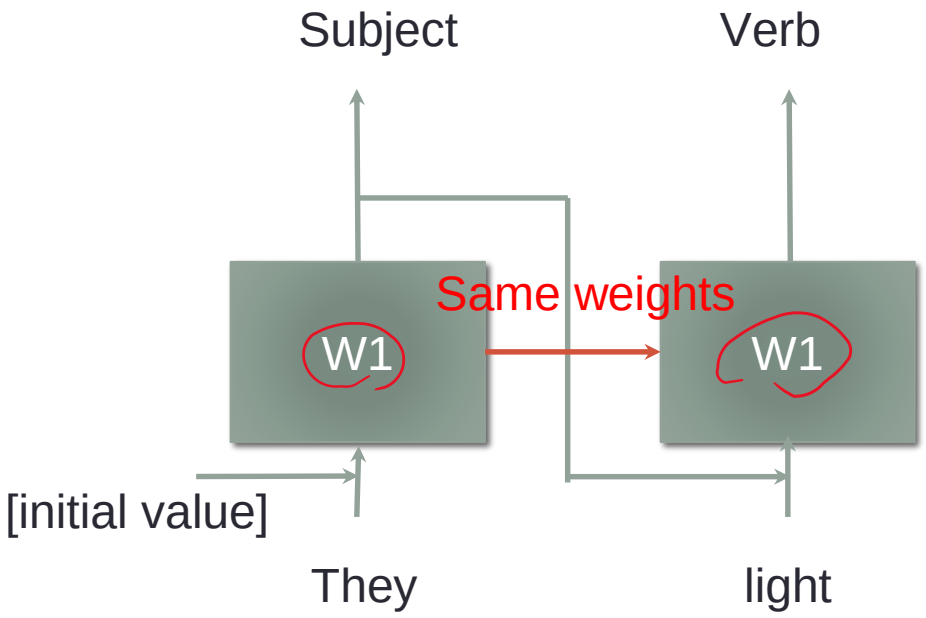
Recurrent neural network (RNN)

- Unrolling of a recurrent layer.



Recurrent neural network (RNN)

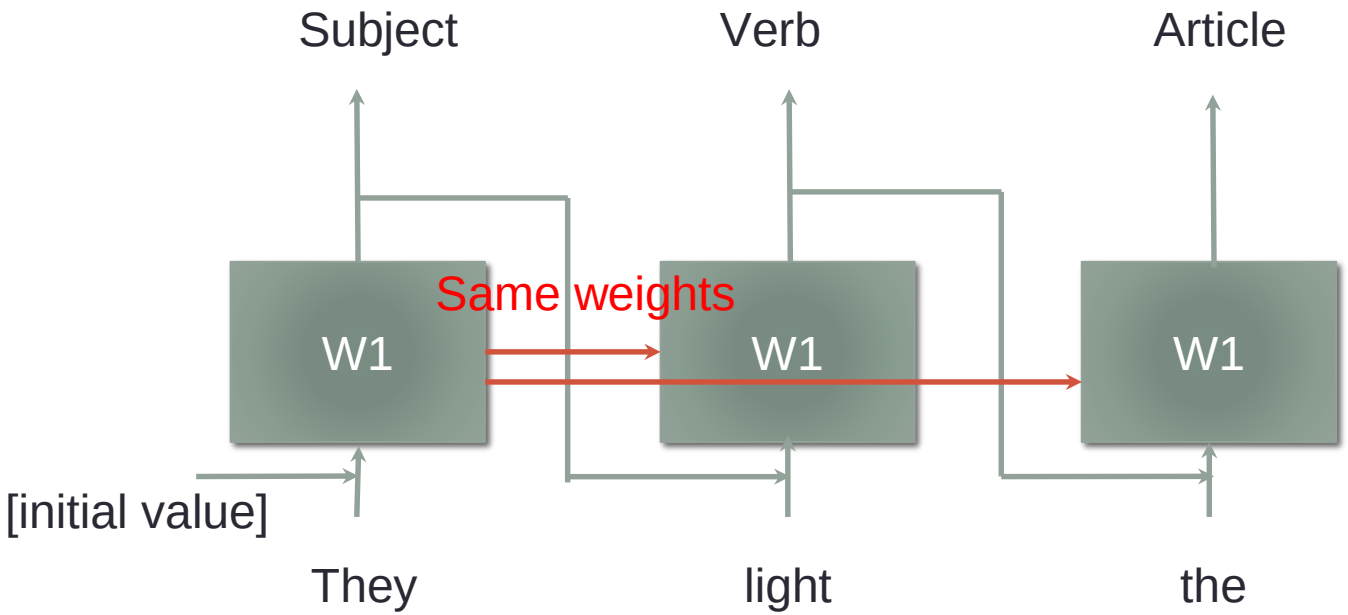
- Unrolling of a recurrent layer.



Parameter sharing across time

Recurrent neural network (RNN)

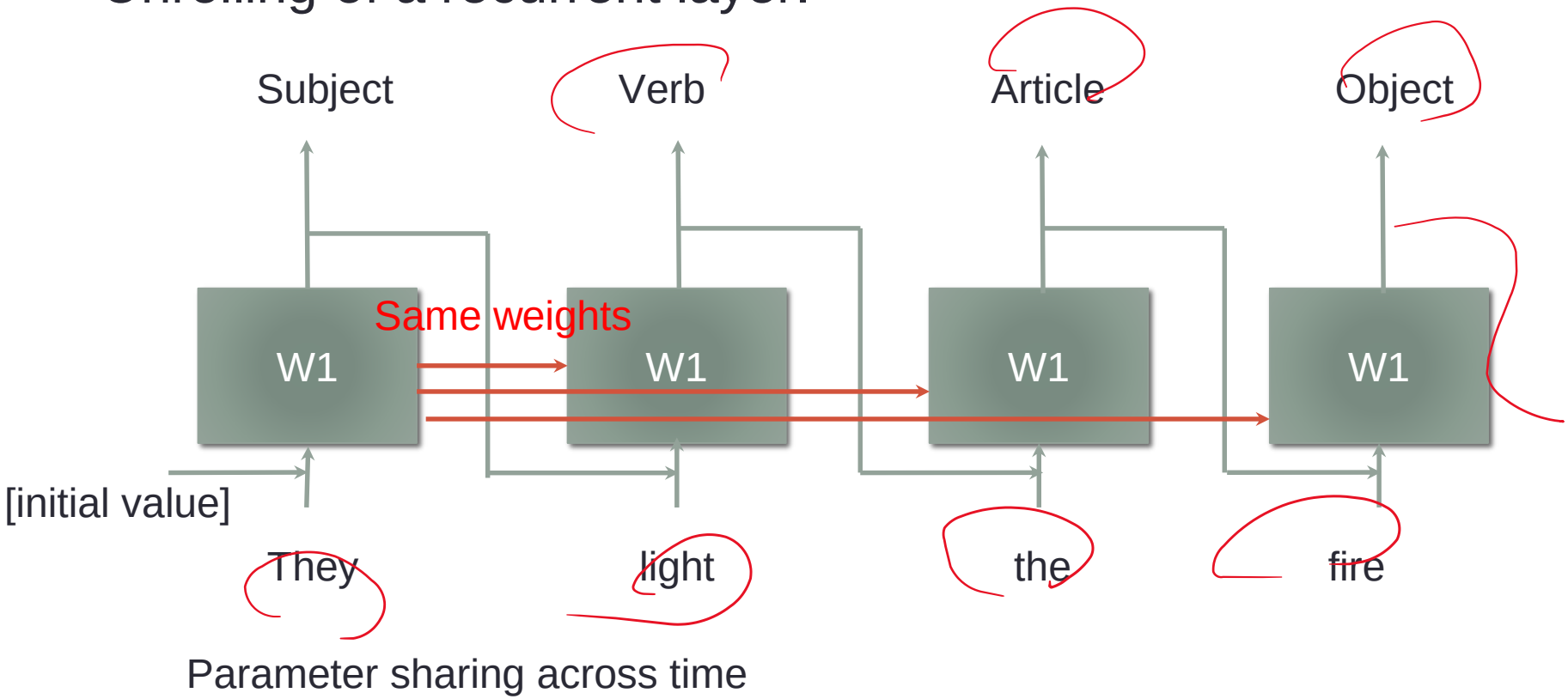
- Unrolling of a recurrent layer.



Parameter sharing across time

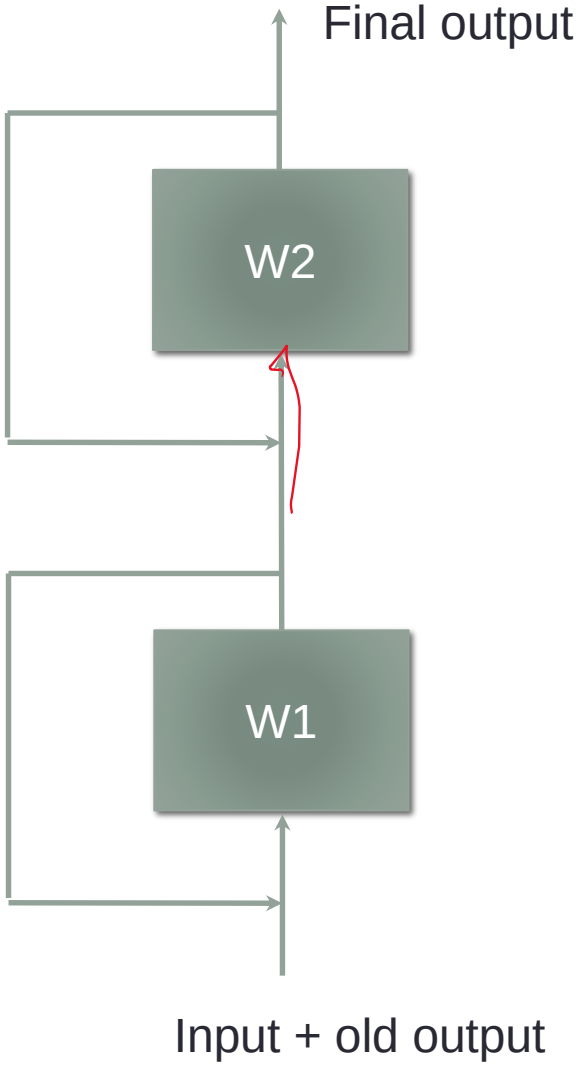
Recurrent neural network (RNN)

- Unrolling of a recurrent layer.



Recurrent neural network (RNN)

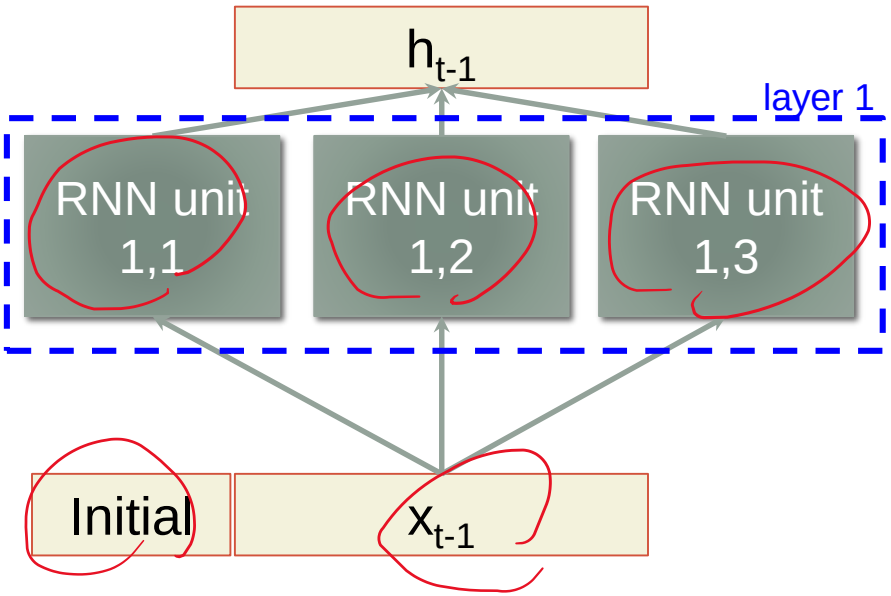
- Stacks of recurrent layer



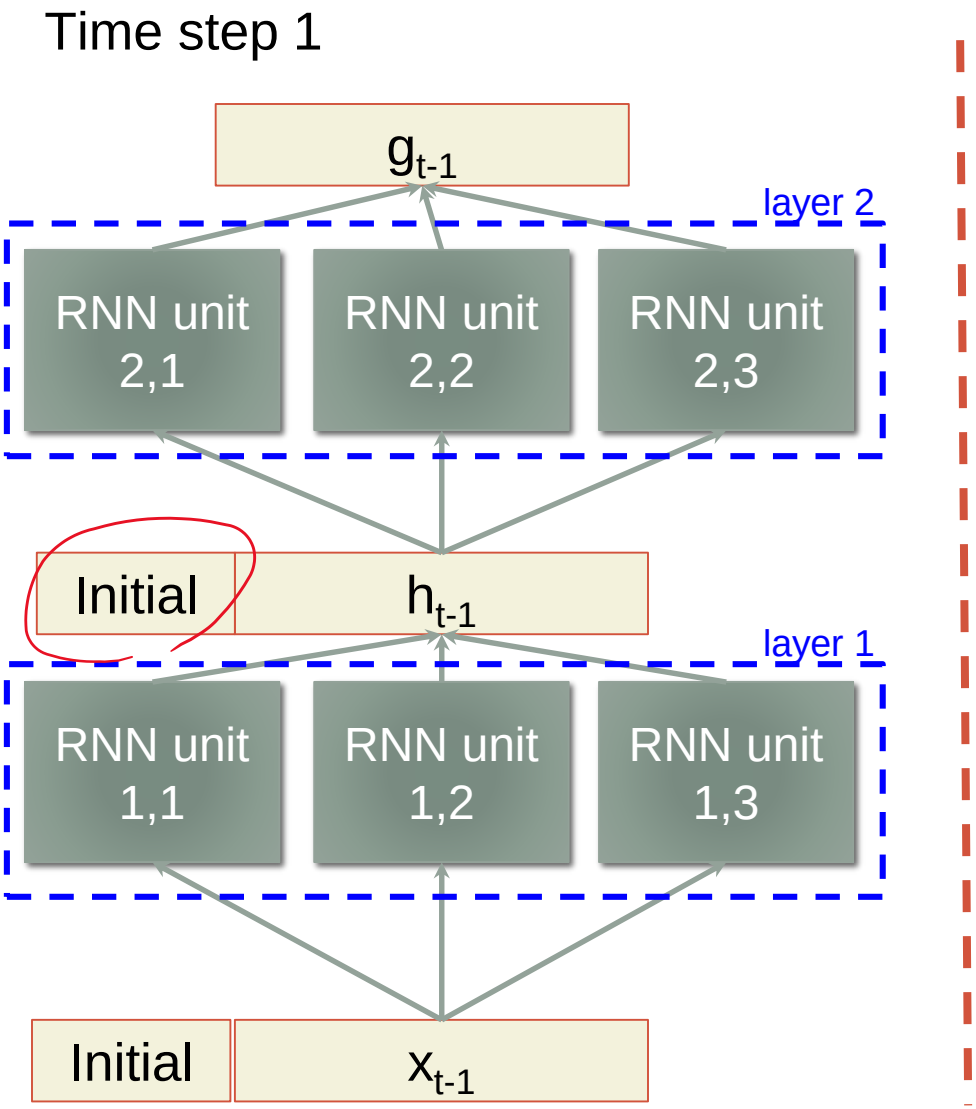
RNN layers (expanded in time)

Time step 1

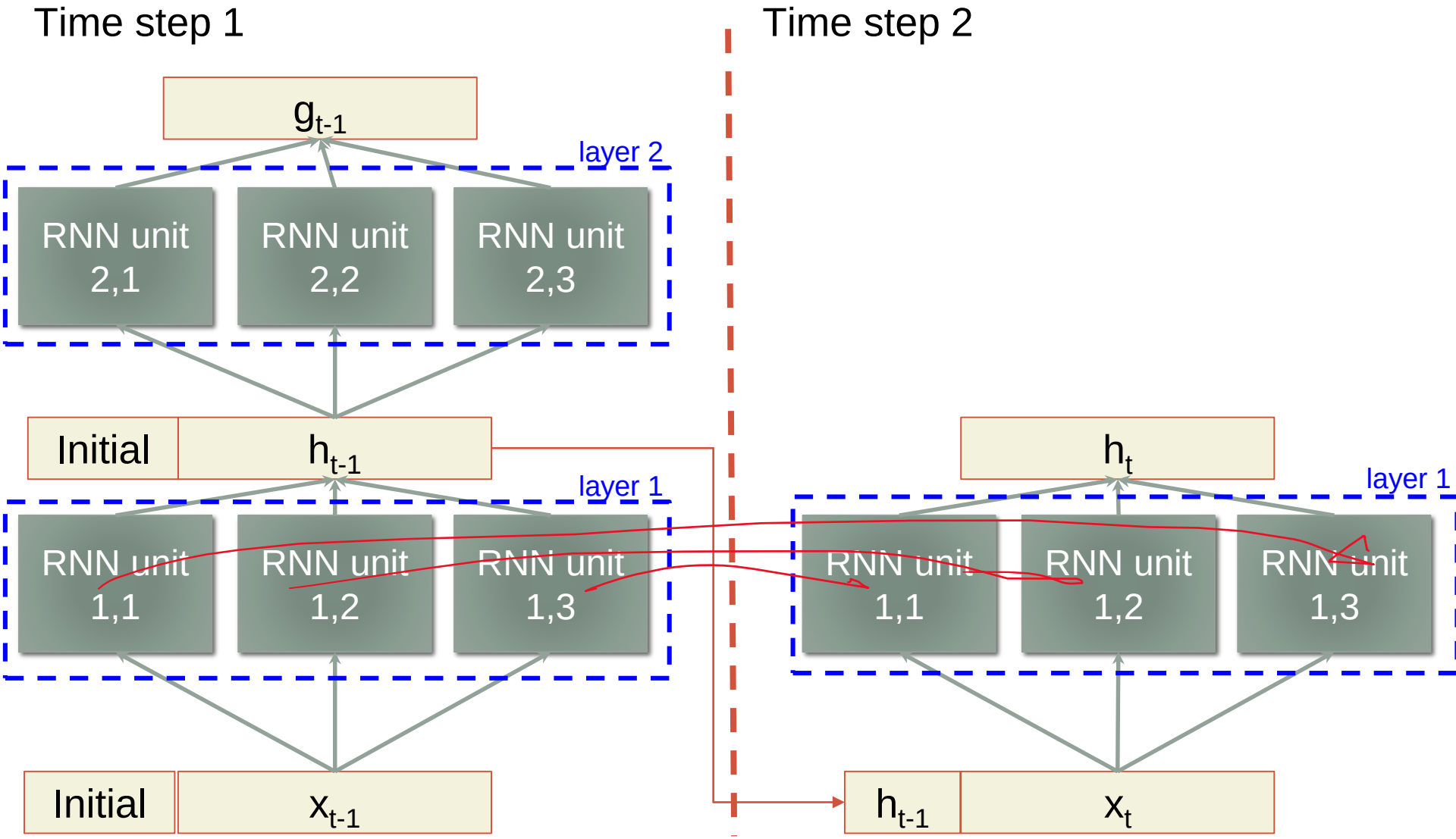
Time step 2



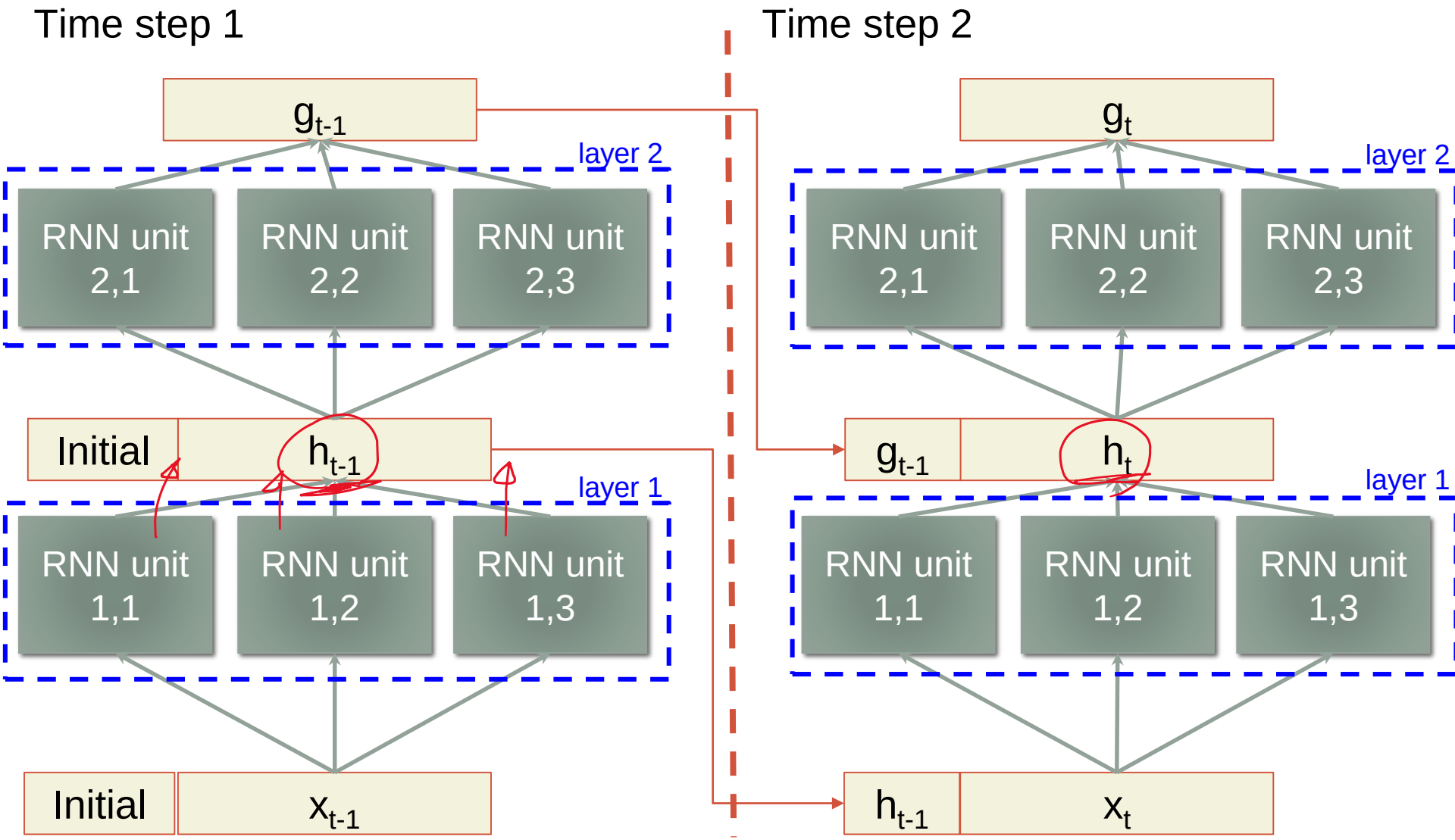
RNN layers (expanded in time)



RNN layers (expanded in time)

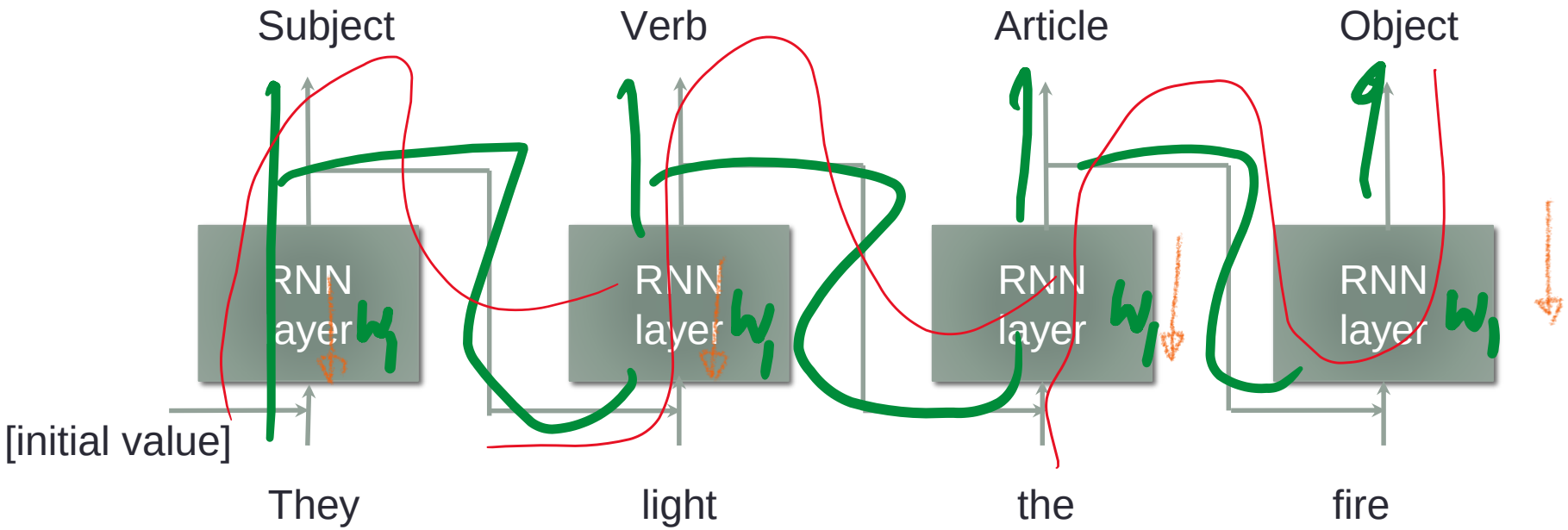


RNN layers (expanded in time)



Training a recurrent neural network

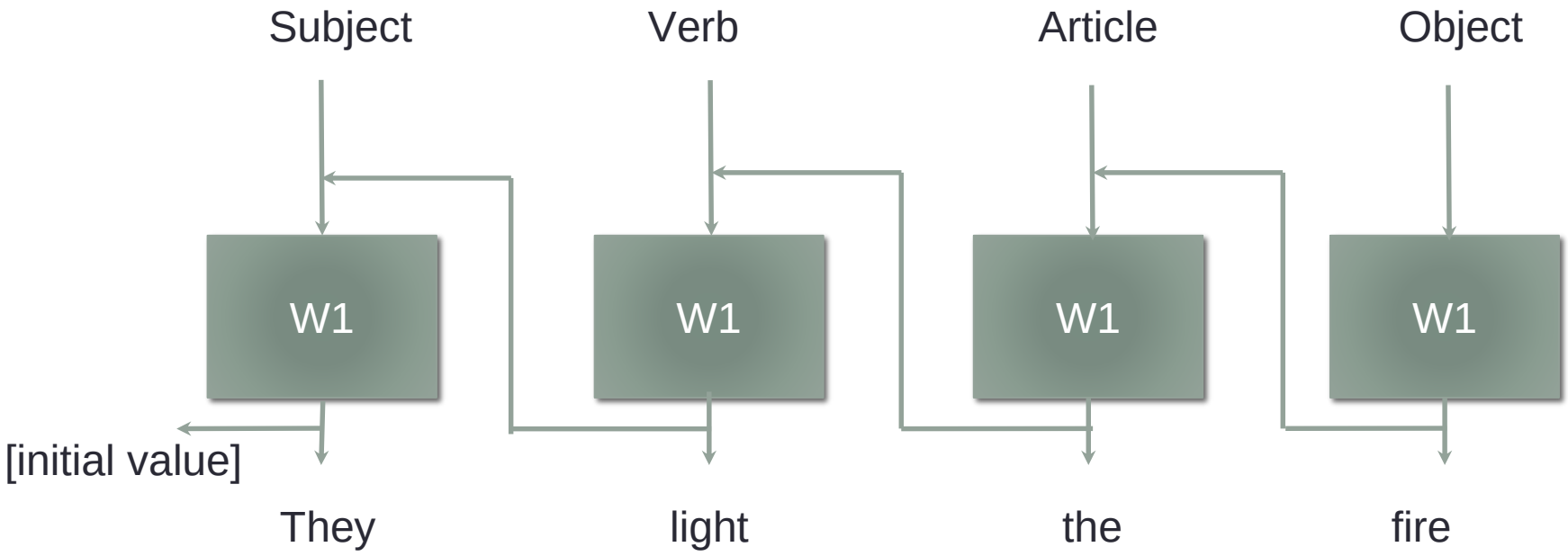
- RNN framework



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

Training a recurrent neural network

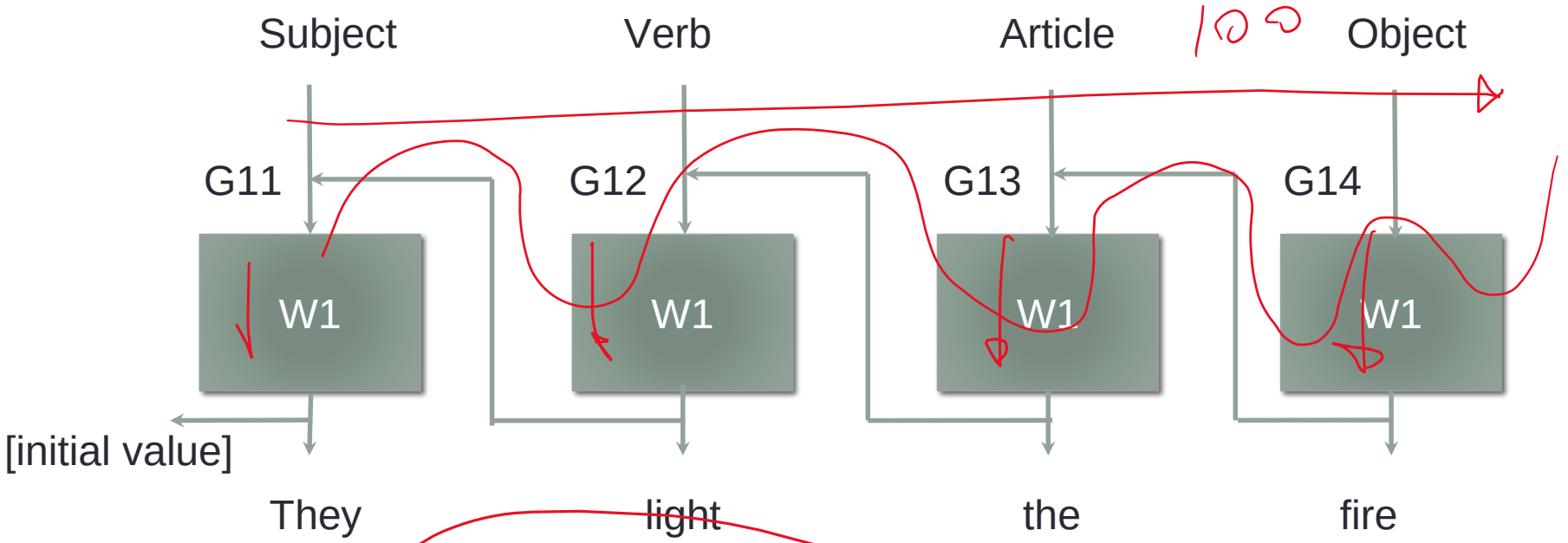
- Backward Computation graph



Backpropagation through time (BPTT)

BPTT

- Backward Computation graph

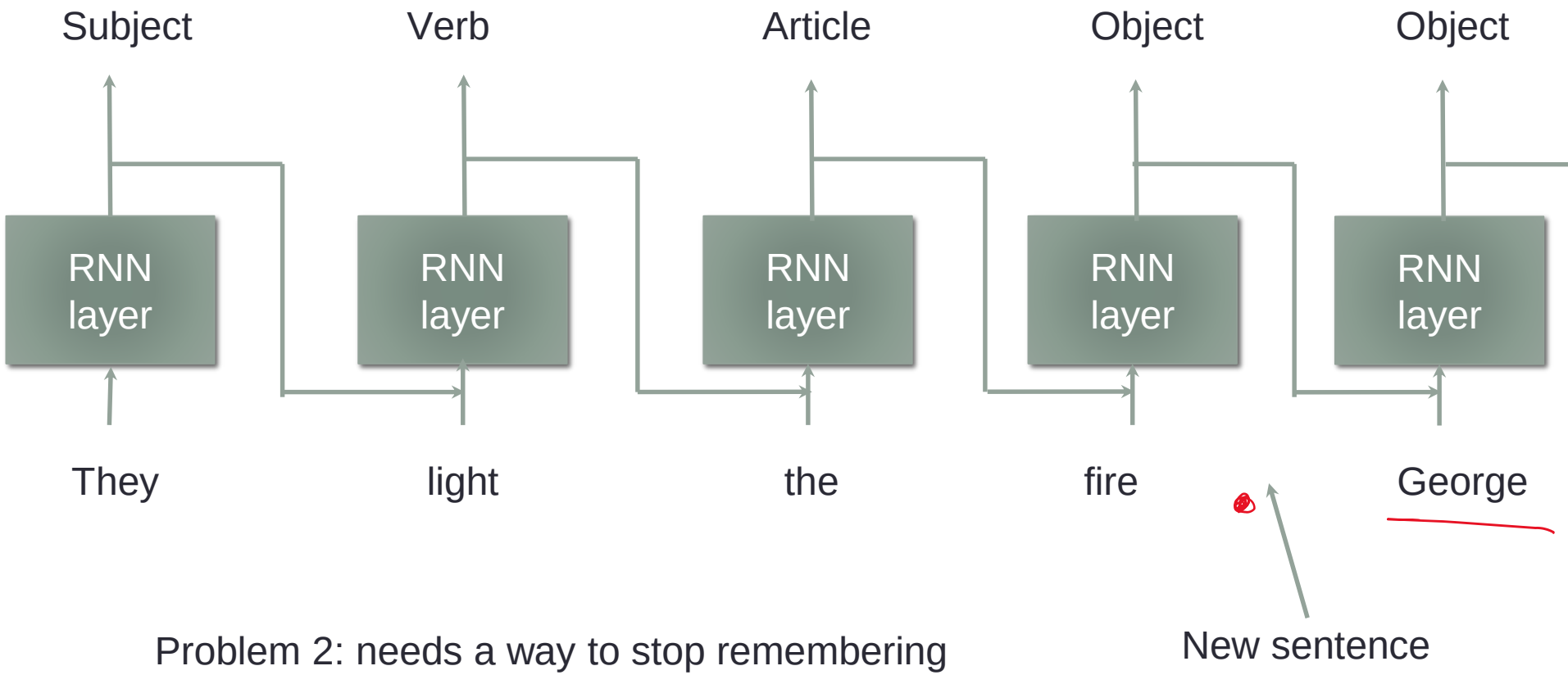


W1 <- W1 + G11 + G12 + G13 + G14

Problem 1: cannot deal with infinitely long recurrent

Gradient explosion, vanishing gradient

Recurrent neural network (RNN)

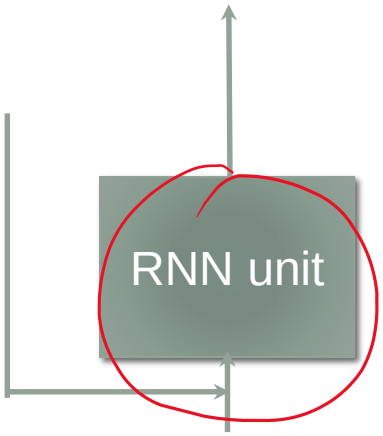


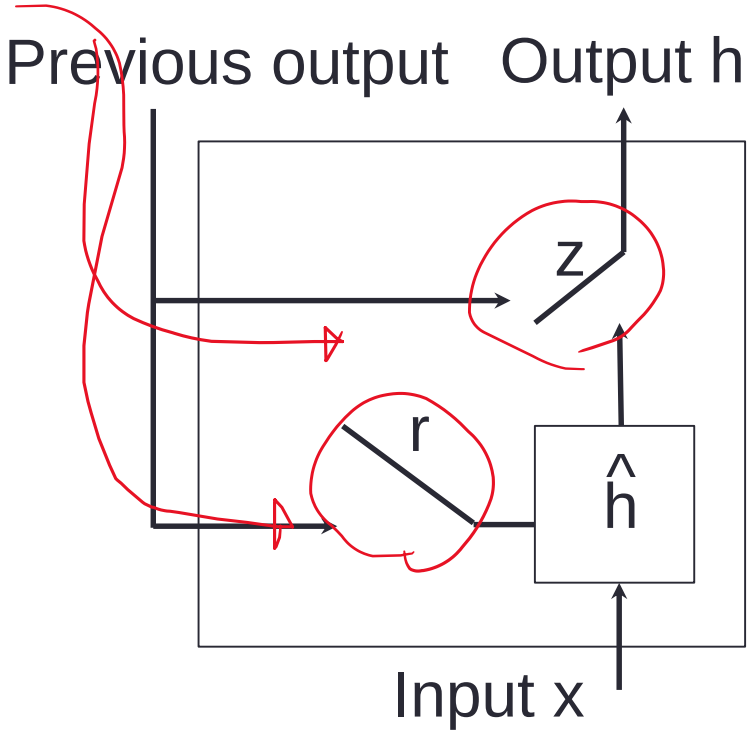
Problem 2: needs a way to stop remembering

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)





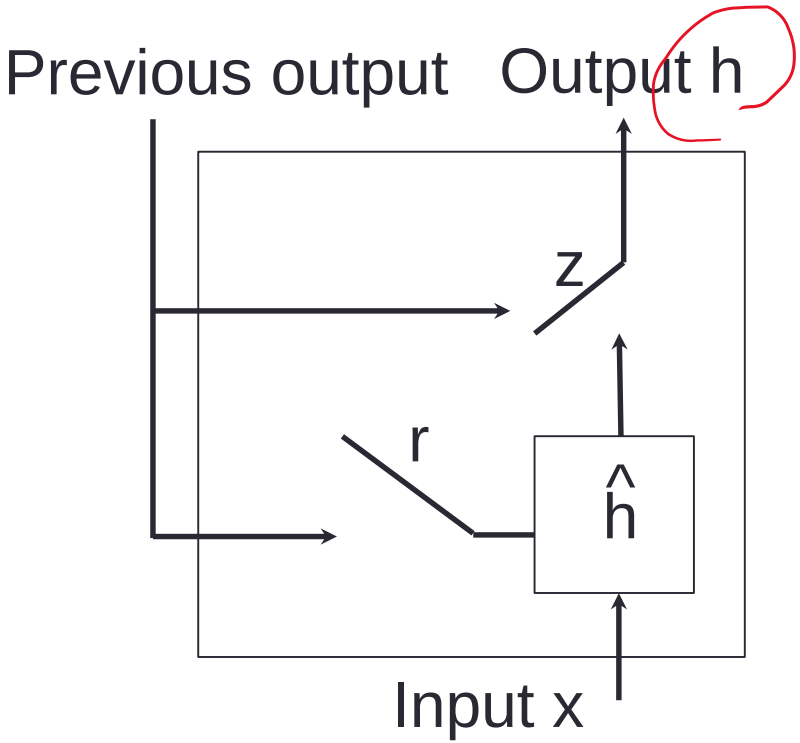
Gated Recurrent Unit (GRU)

Neuron index

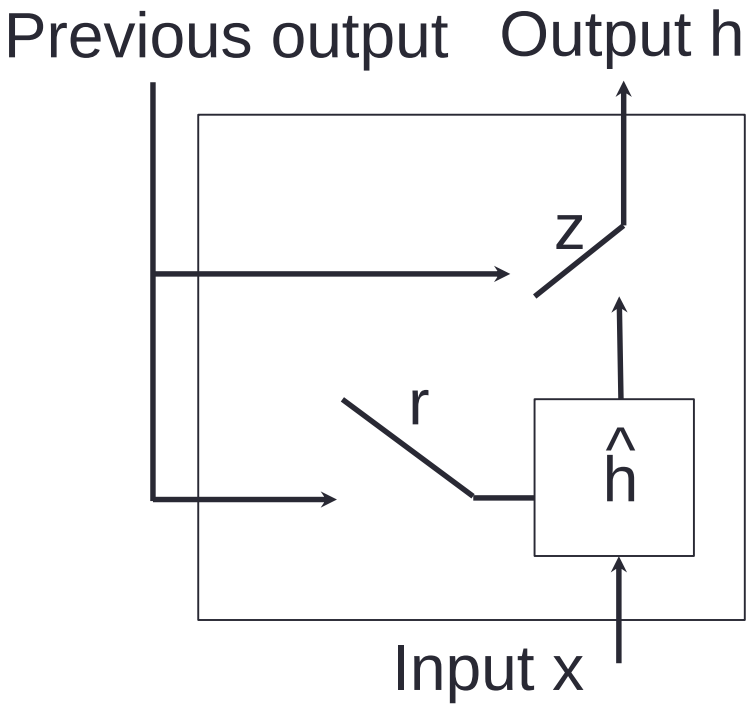
$$h_t^j = (1 - z_t^j) h_{t-1}^j + z_t^j \hat{h}_t^j$$

time index

update gate



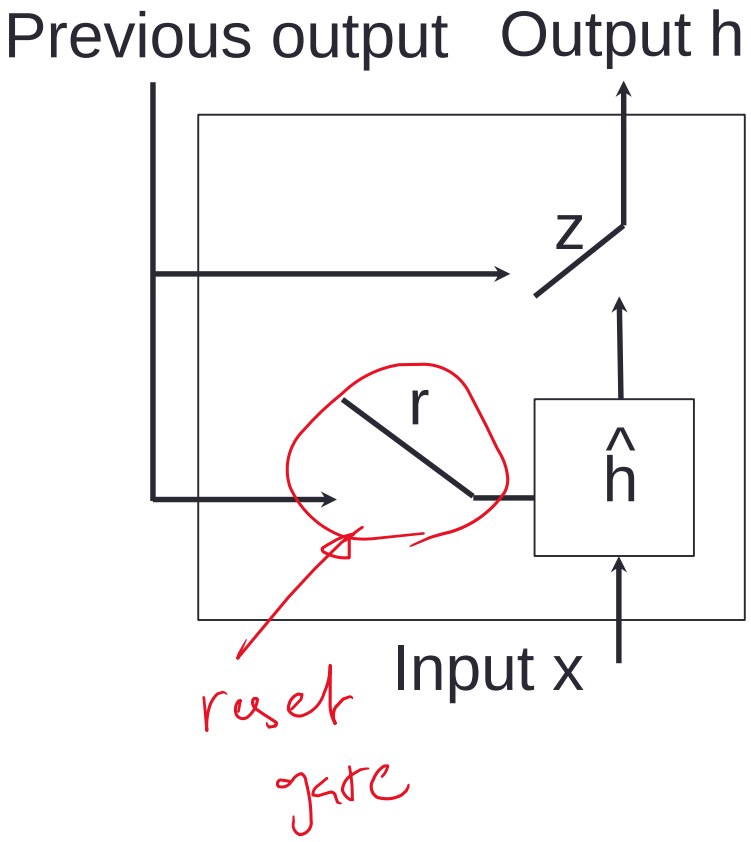
Gated Recurrent Unit (GRU)



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

One GRU neuron output (scalar)

Gated Recurrent Unit (GRU)



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

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

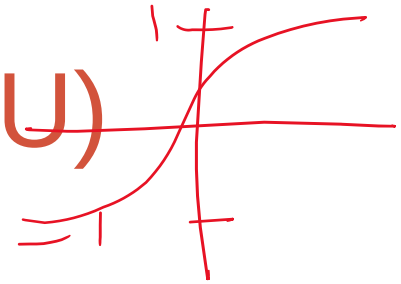
Element-wise product

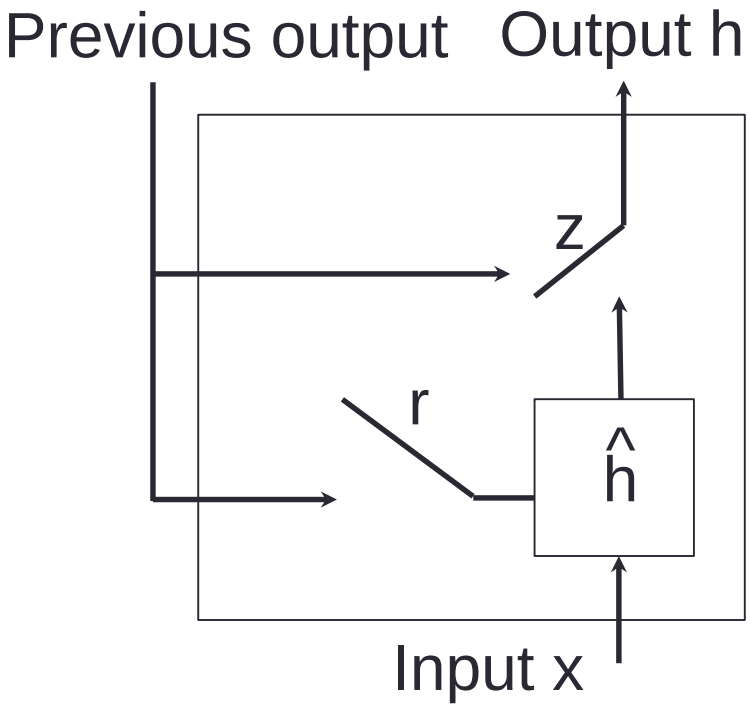
Linear transform with matrix multiply

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

$x_t^j = h_t^j$

Gated Recurrent Unit (GRU)





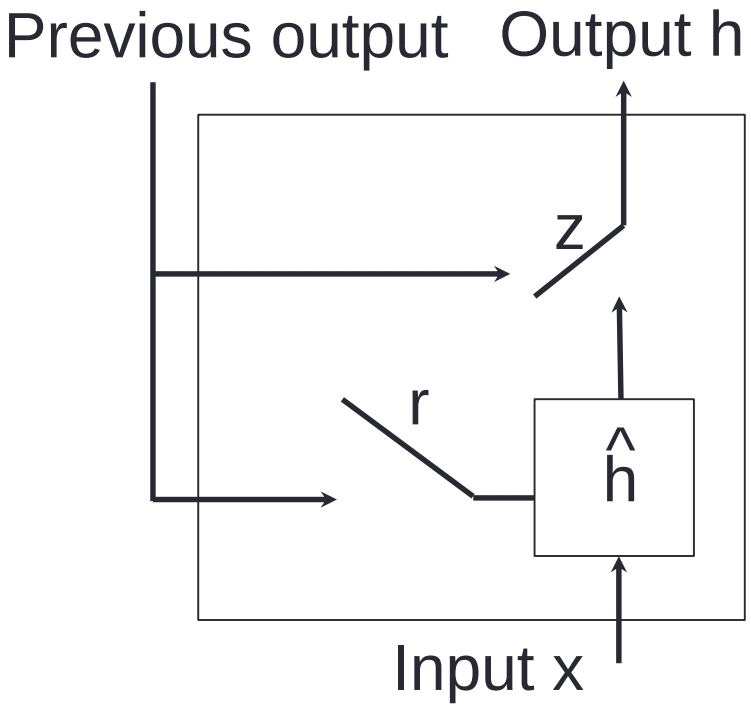
$$h_t^j = (1 - z_t^j) h_{t-1}^j + z_t^j \hat{h}_t^j$$

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

Takes the j-th element

Bounds the output

Gated Recurrent Unit (GRU)



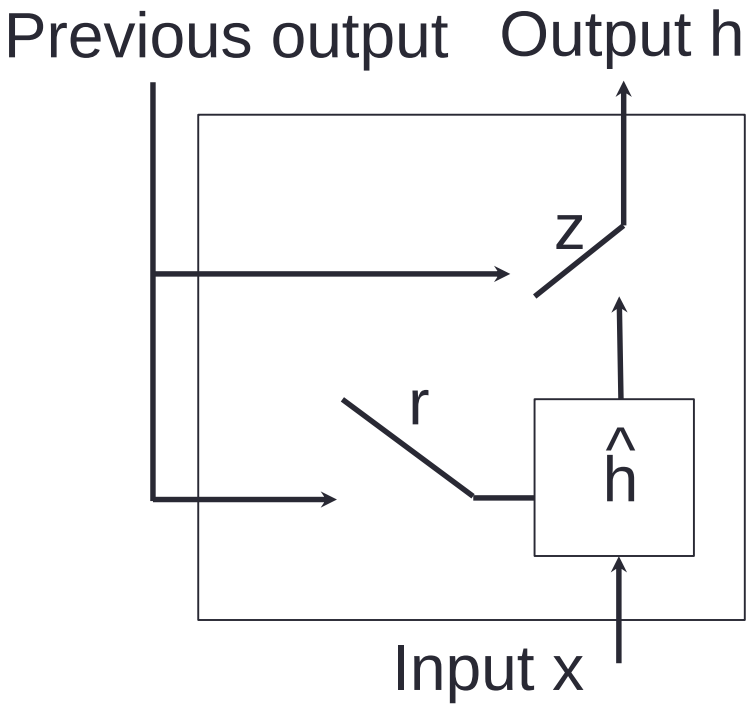
$$h_t^j = (1 - z_t^j) h_{t-1}^j + z_t^j \hat{h}_t^j$$

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

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

Indicates a different set of weights

Gated Recurrent Unit (GRU)



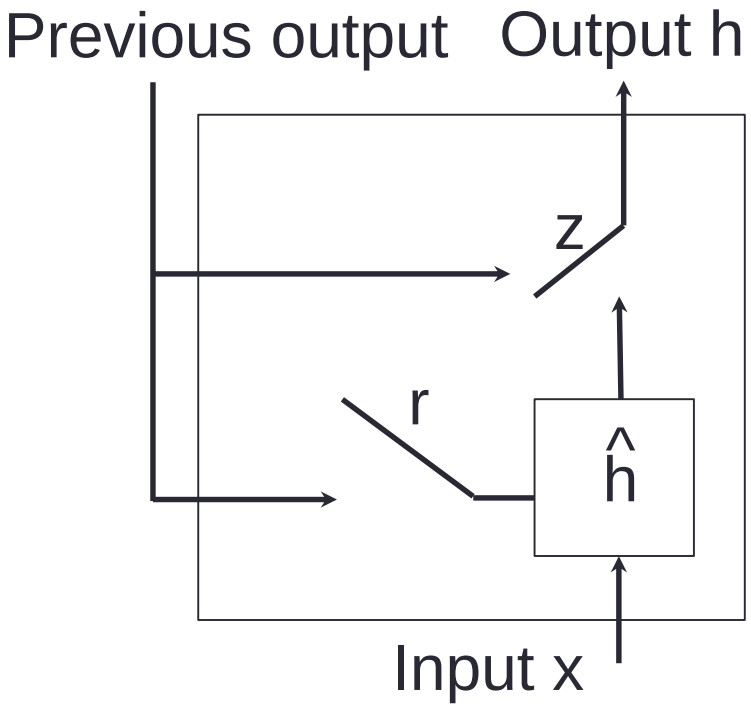
$$h_t^j = (1 - z_t^j) h_{t-1}^j + z_t^j \hat{h}_t^j$$

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

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

Bounds the output to 0 to 1 for interpolation

Gated Recurrent Unit (GRU)



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

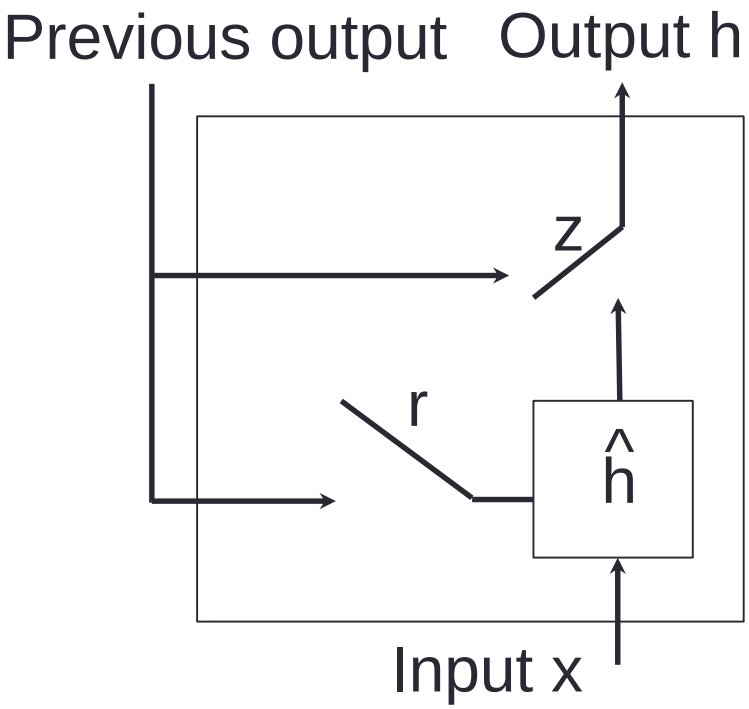
$$\hat{h}_t^j = tanh^j(W\mathbf{x}_t + U(\mathbf{r}_t \odot \mathbf{h}_{t-1}))$$

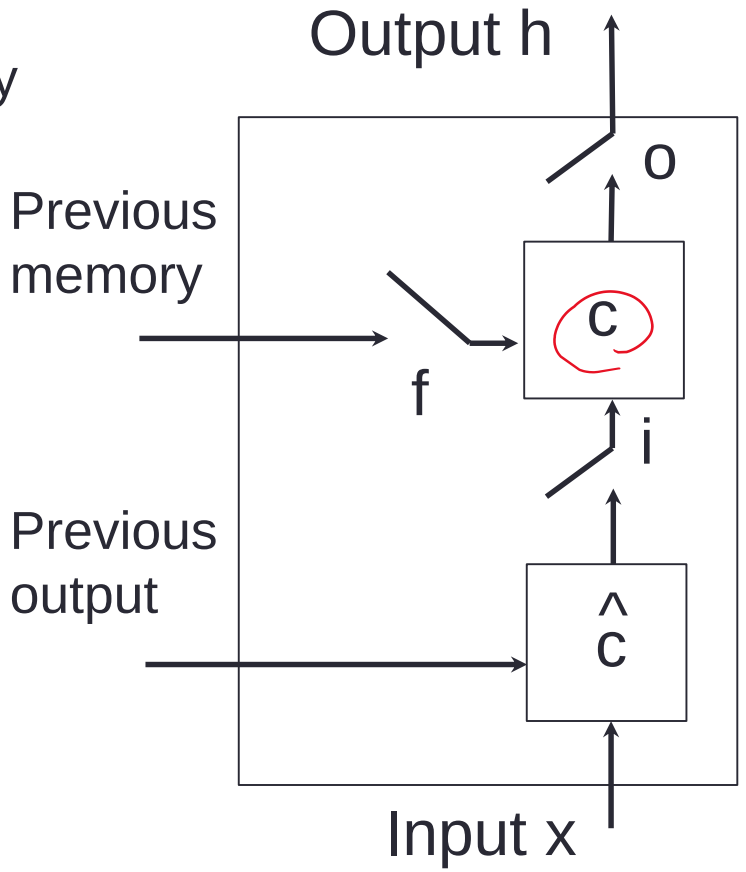
$$z_t^j = sigmoid^j(W_z\mathbf{x}_t + U_z\mathbf{h}_{t-1})$$

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

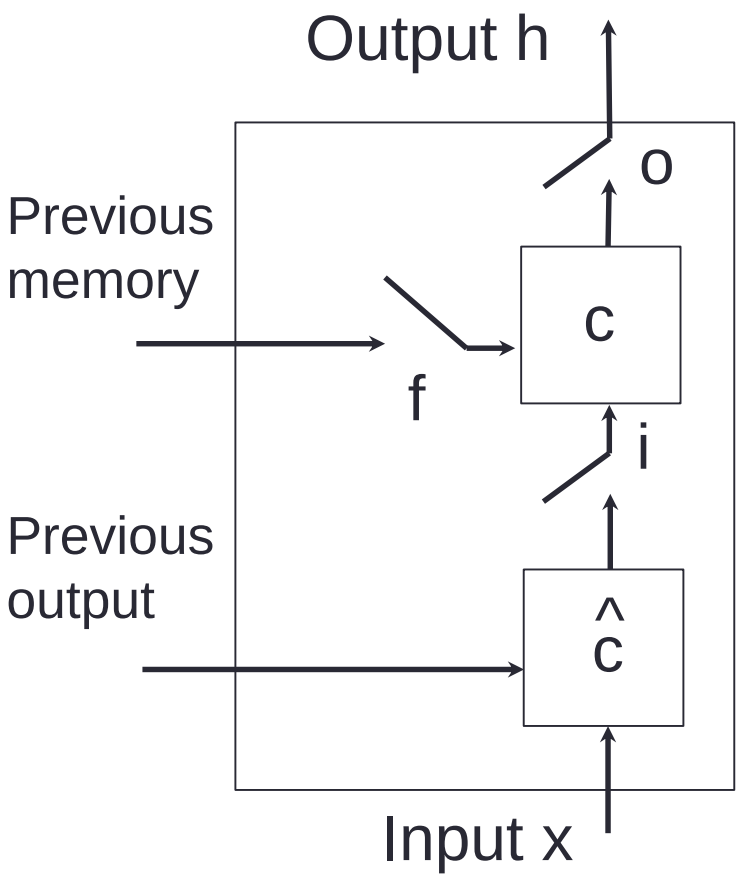
Long Short-Term Memory (LSTM)

- 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)



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

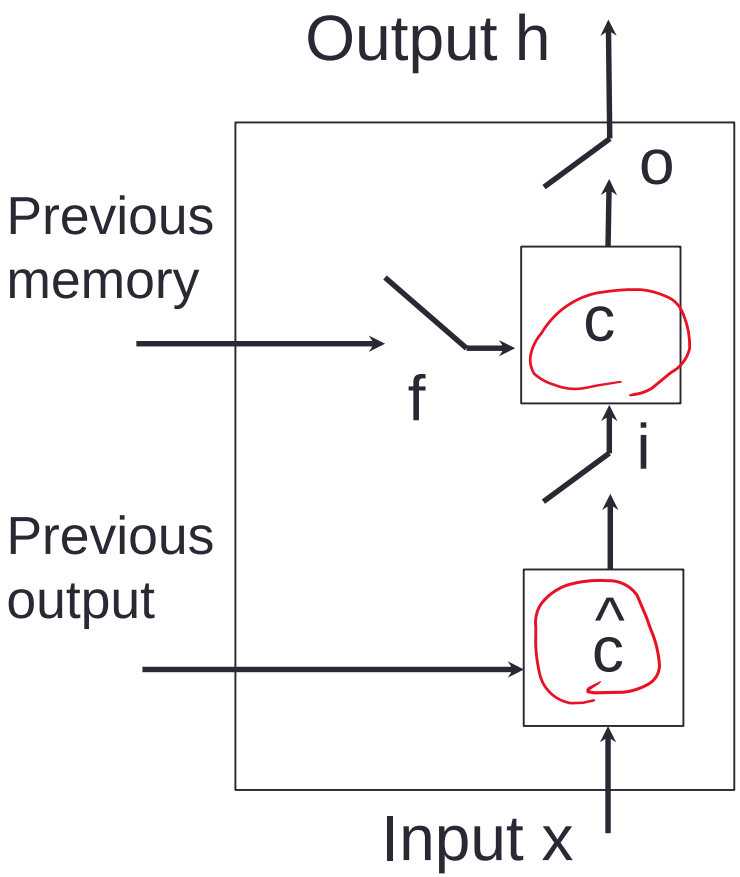
$$o_t^j = F^j(W_o\mathbf{x}_t + U_o\mathbf{h}_{t-1} + V_o\mathbf{c_t})$$

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

Contribution from memory "Peephole connection"

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

Long Short-Term Memory (LSTM)



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

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

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

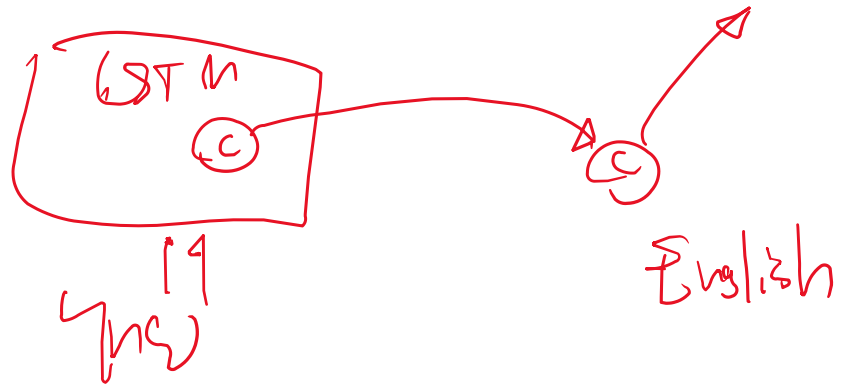
$$h_t^j = o_t^j tanh(c_t^j)$$

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

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

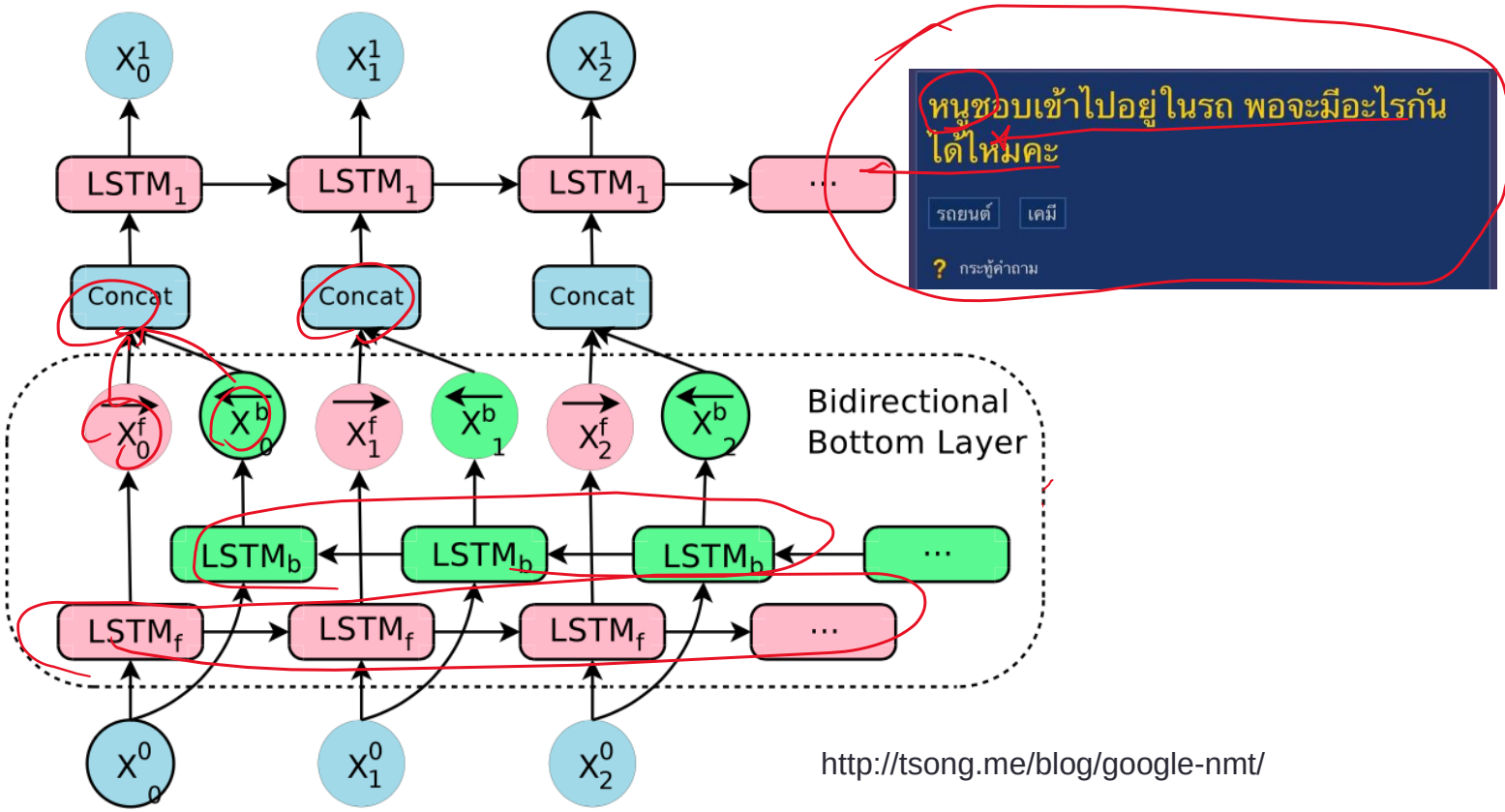
GRU vs LSTM

- 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!



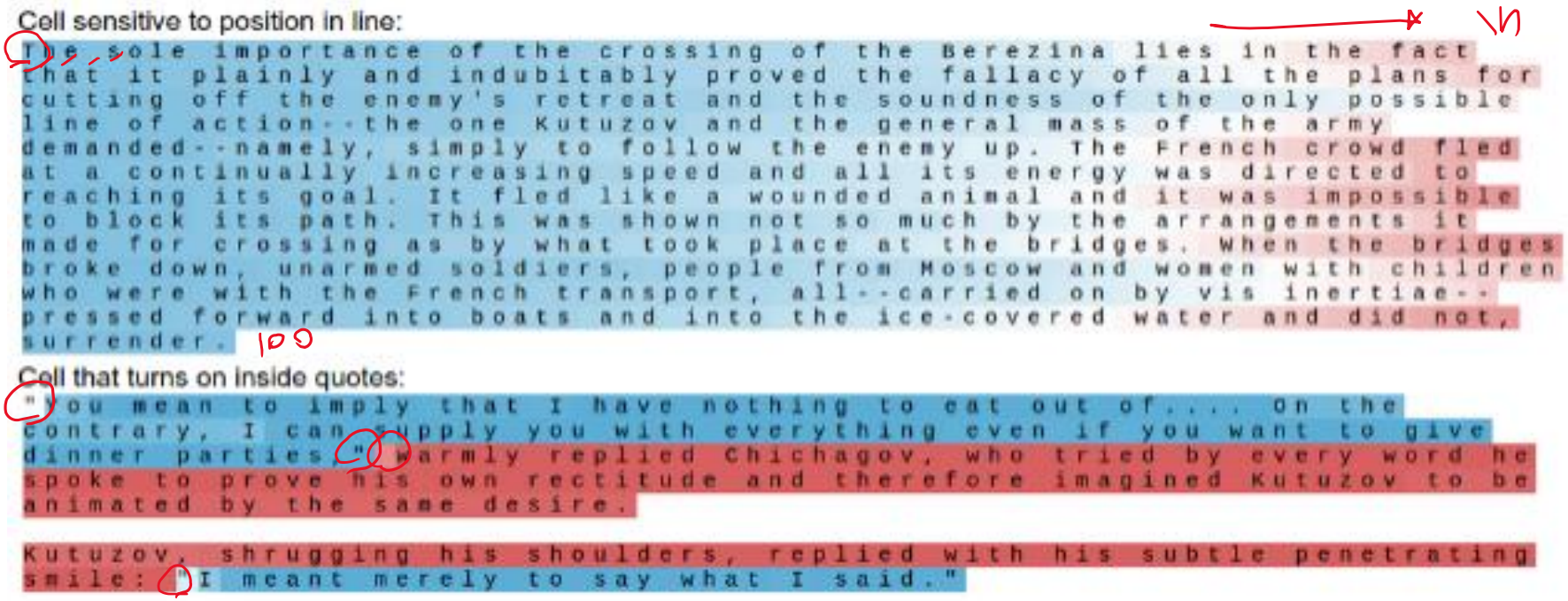
Bi-directional LSTM / GRU

- 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



http://tsong.me/blog/google-nmt/

LSTM remembers meaningful things



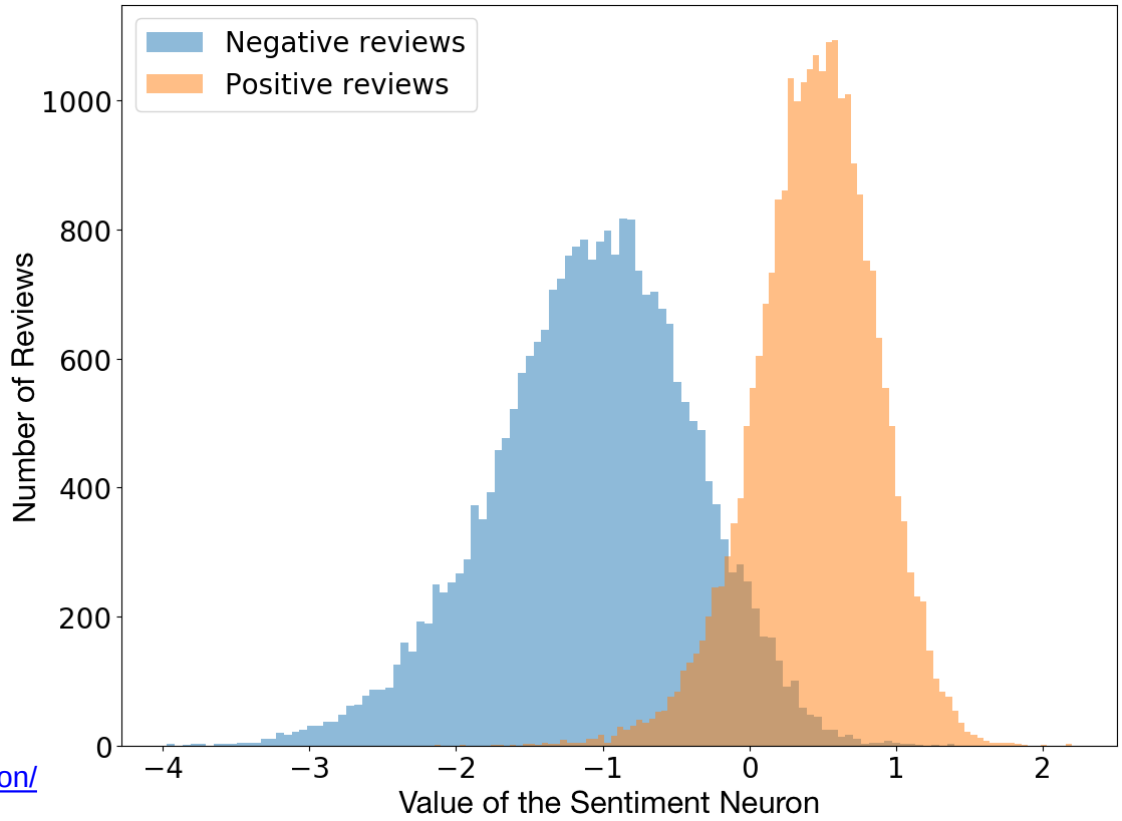
Andrej Karpathy, Visualizing and Understanding Recurrent Networks, 2015, https://arxiv.org/abs/1506.02078

Sentiment Neuron

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

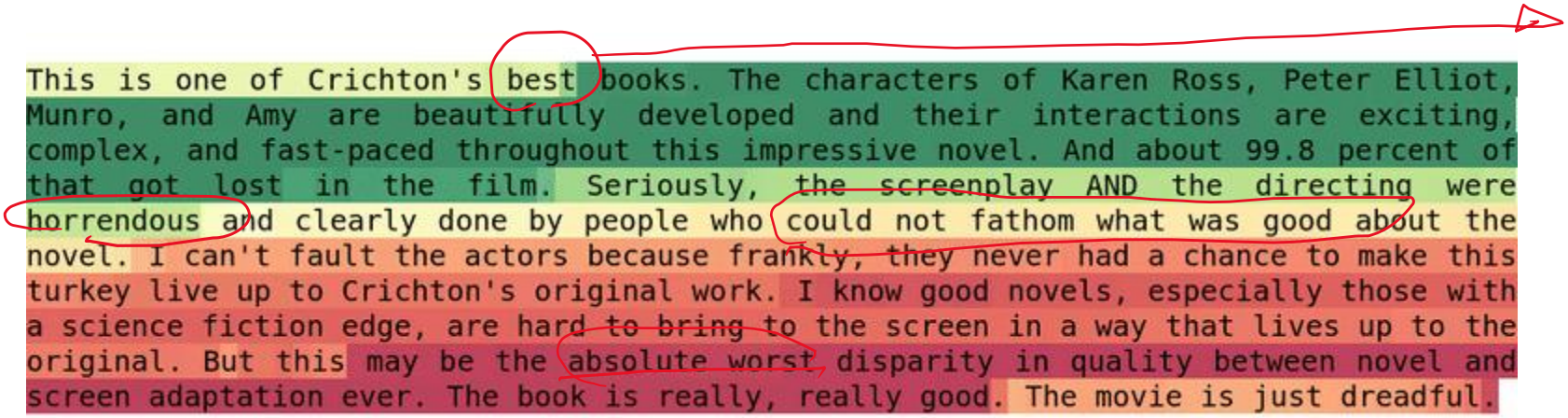
LSTM with 4096 hidden units

One unit seems to be learning sentiment



https://openai.com/blog/unsupervised-sentiment-neuron/

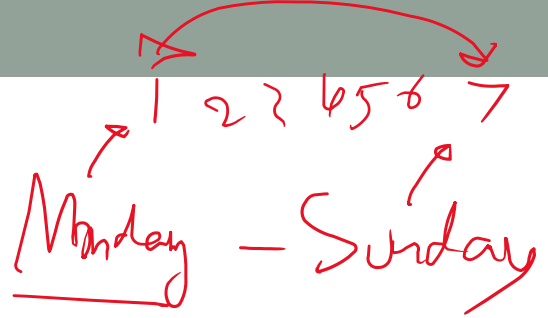
Sentiment neuron



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

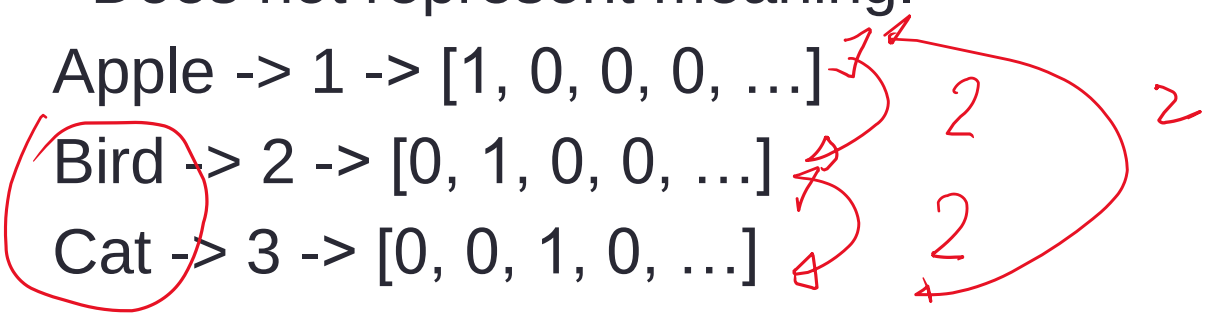
One hot encoding

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

Apple -> 1 -> [1, 0, 0, 0, …]

Bird -> 2 -> [0, 1, 0, 0, …]

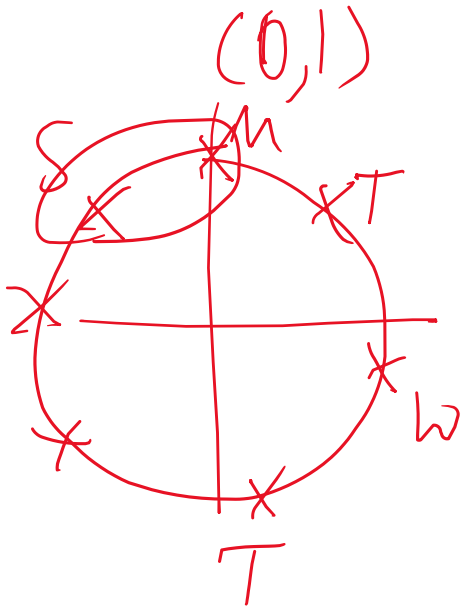
Cat -> 3 -> [0, 0, 1, 0, …]



|Apple – Bird| = |Bird – Cat|

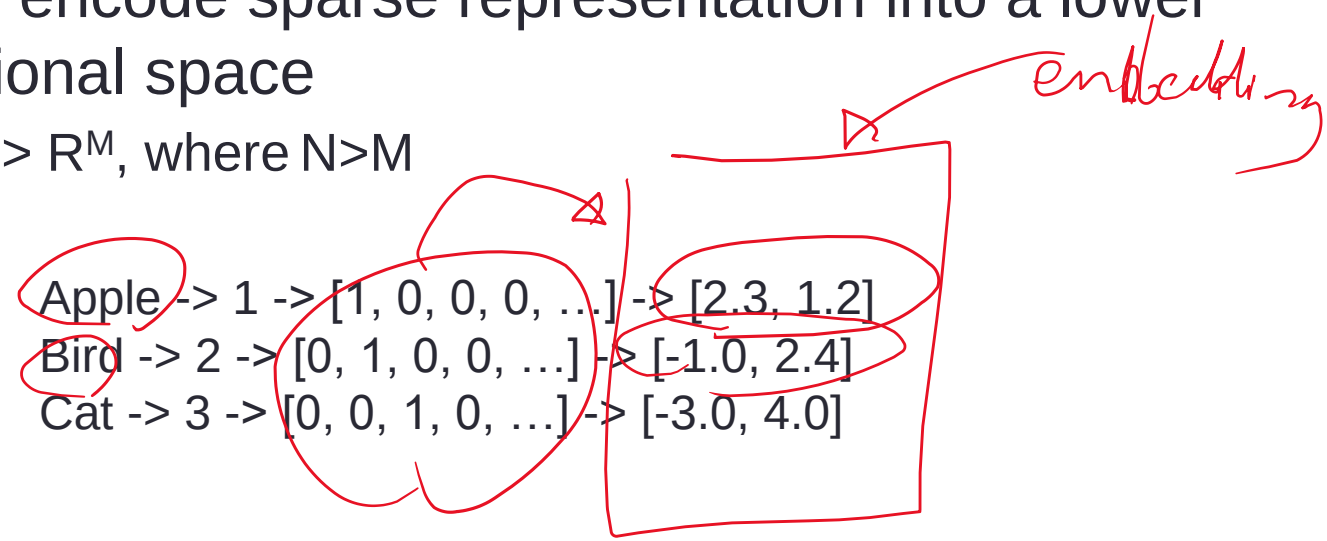
Getting meaning into the feature vectors

- 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]
- ก = [0 0 0 1 0 0 0 0, 0, 1]
- า = [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



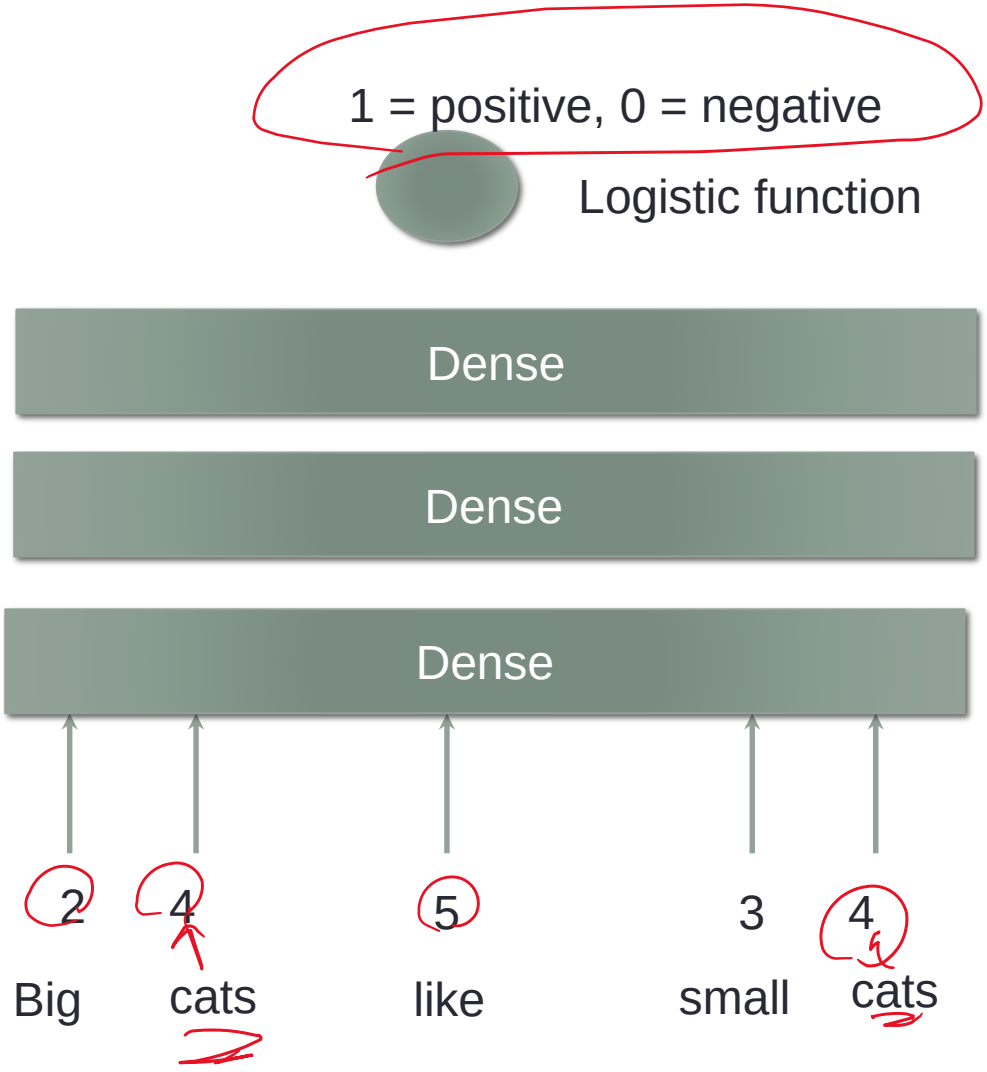
Dense representation

- We can encode sparse representation into a lower dimensional space
  - F: R^N -> R^M, where N>M



- We can do this by using an embedding layer

Sentiment analysis with fully connected networks

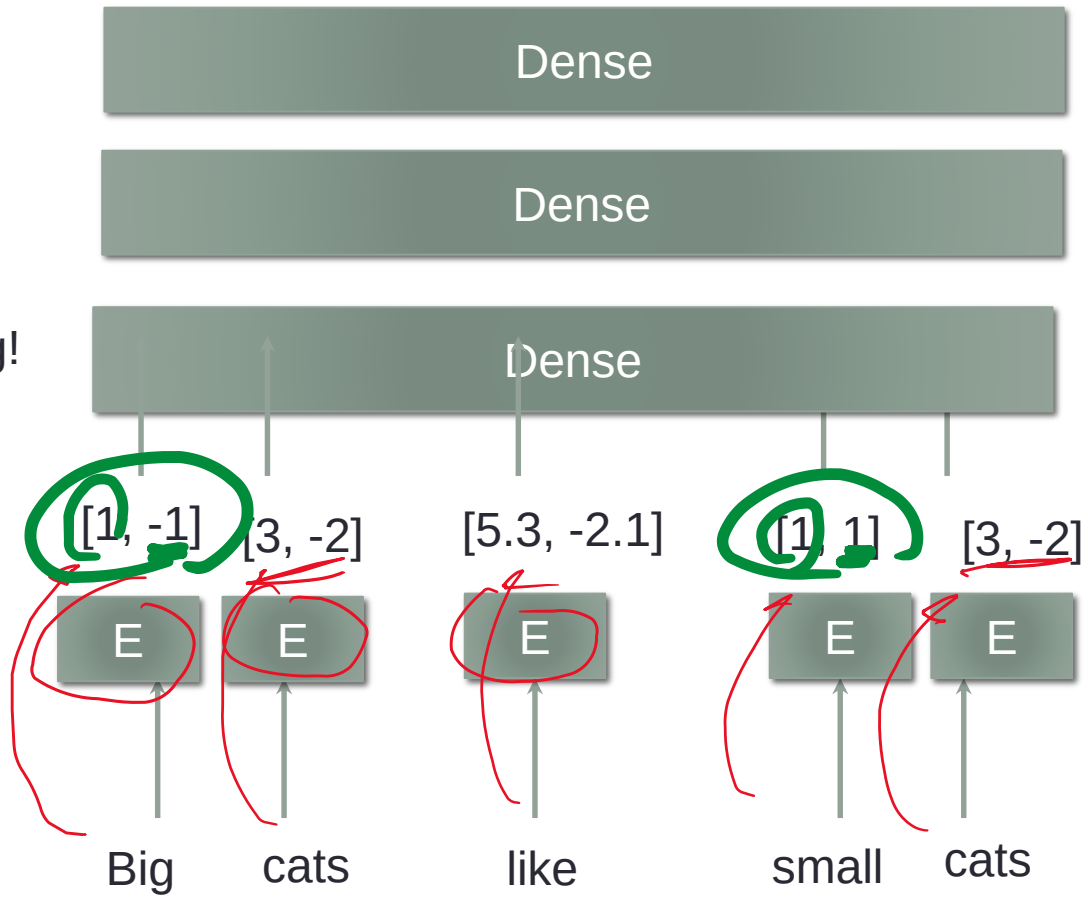


Adding embedding layer

Embedding layer shares the same weights

Parameter sharing!

More on embeddings in the next two lectures!



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

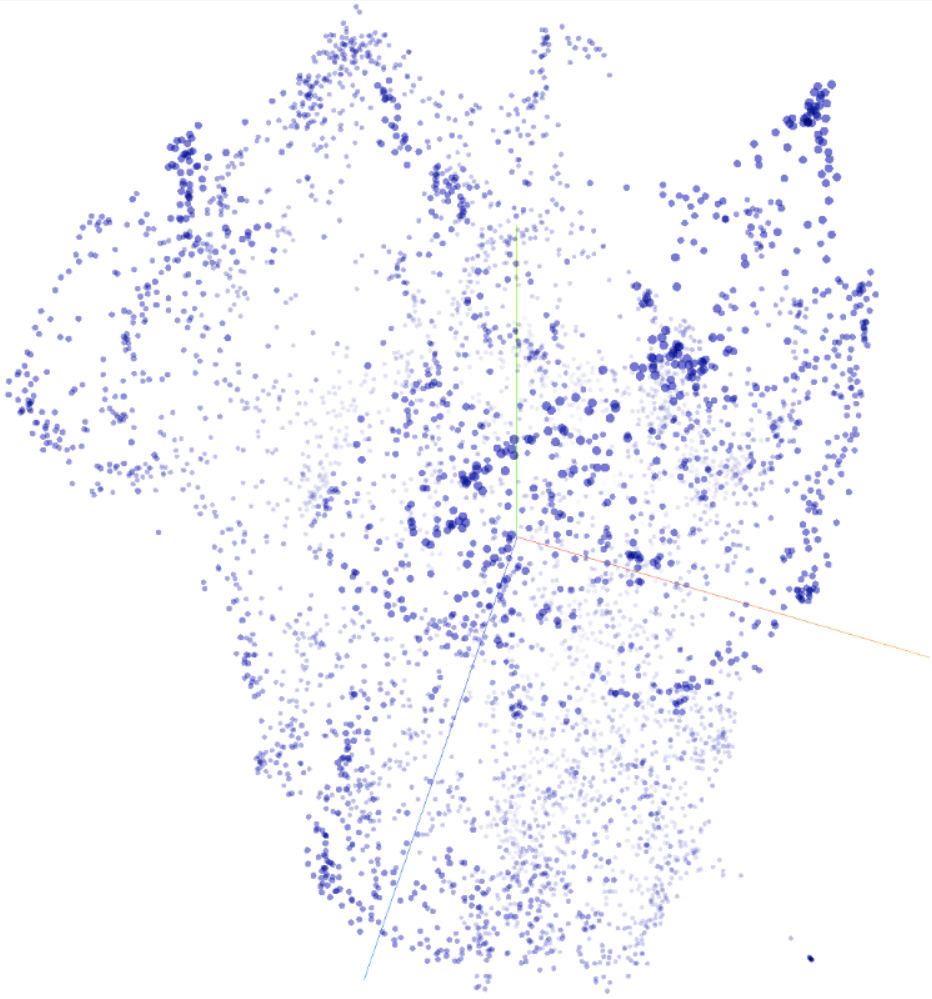
Neighbors 15

Run

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

Learn more about UMAP.

Points: 10000 | Dimension: 200



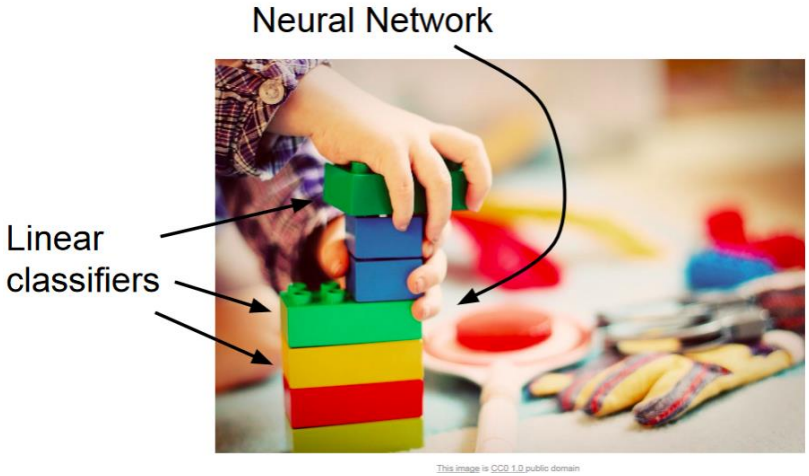
Embedding vs PCA

- 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

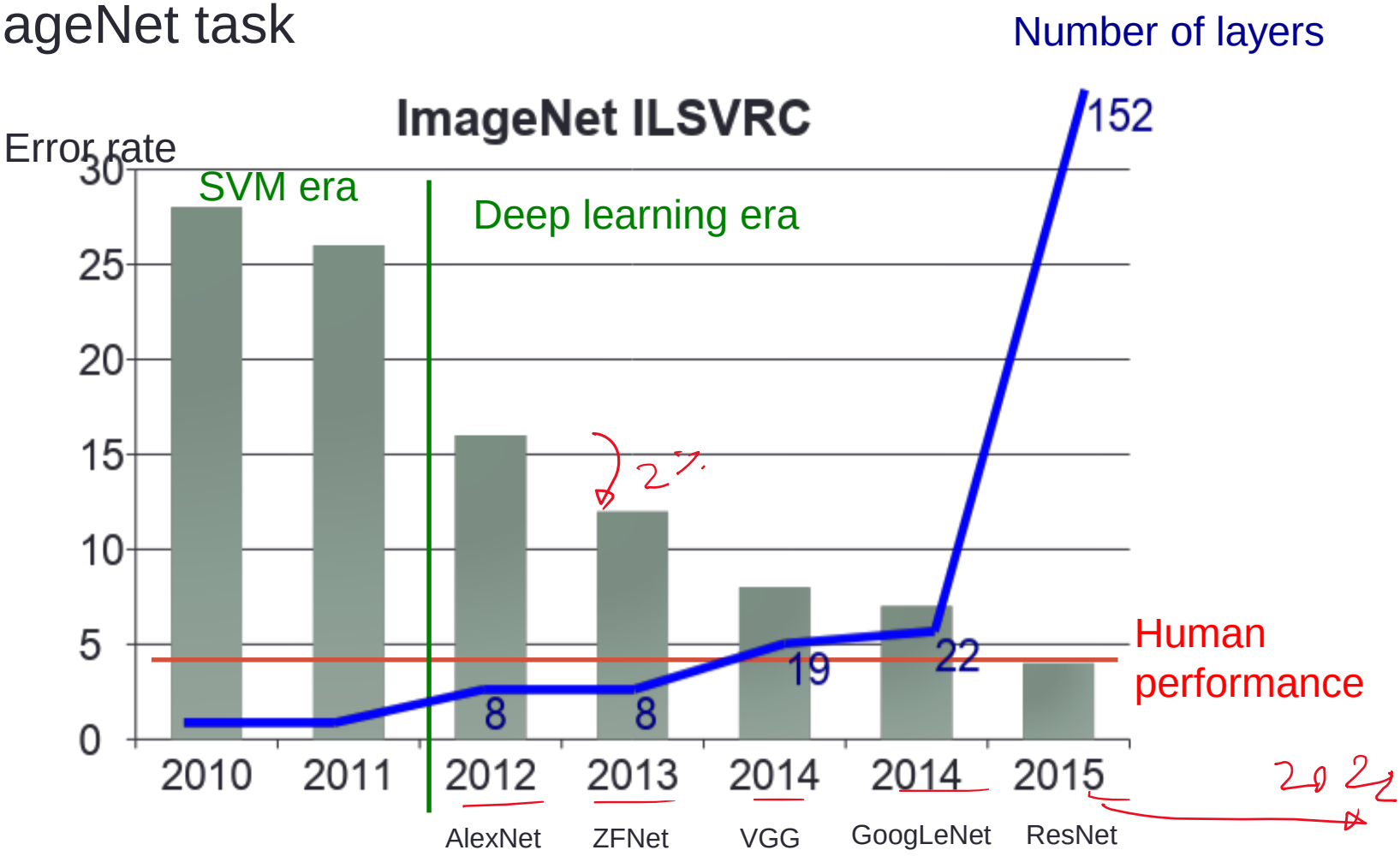
- Typical models now consists of all 3 types
  - CNN: local structure in the feature. Used for feature learning.
  - LSTM: remembering longer term structure or across time
  - DNN: Good for mapping features for classification. Usually used in final layers
- DNN structures encode inductive bias



https://www.analyticsvidhya.com/blog/2017/08/10-advanced-deep-learning-architectures-data-scientists/

A brief history of imagenet architectures

- ImageNet task



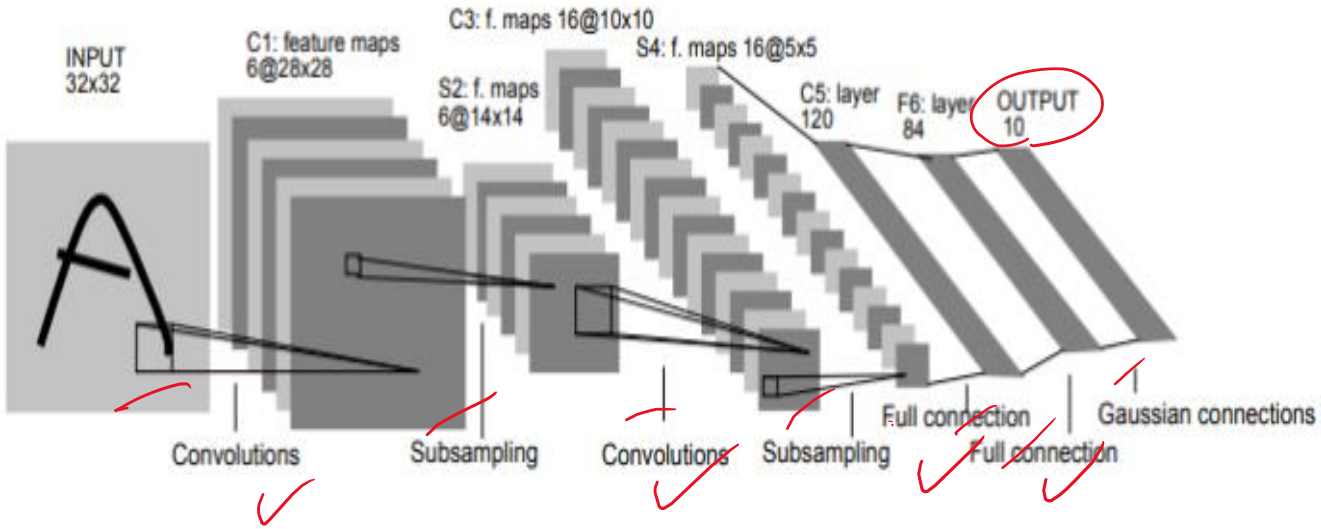
Olga Russakovsky, ImageNet Large Scale Visual Recognition Challenge, 2014 https://arxiv.org/abs/1409.0575

LeNet

Convolutions and poolings followed by fully connected layers

Tanh activations

Ability to handle larger images limited by compute

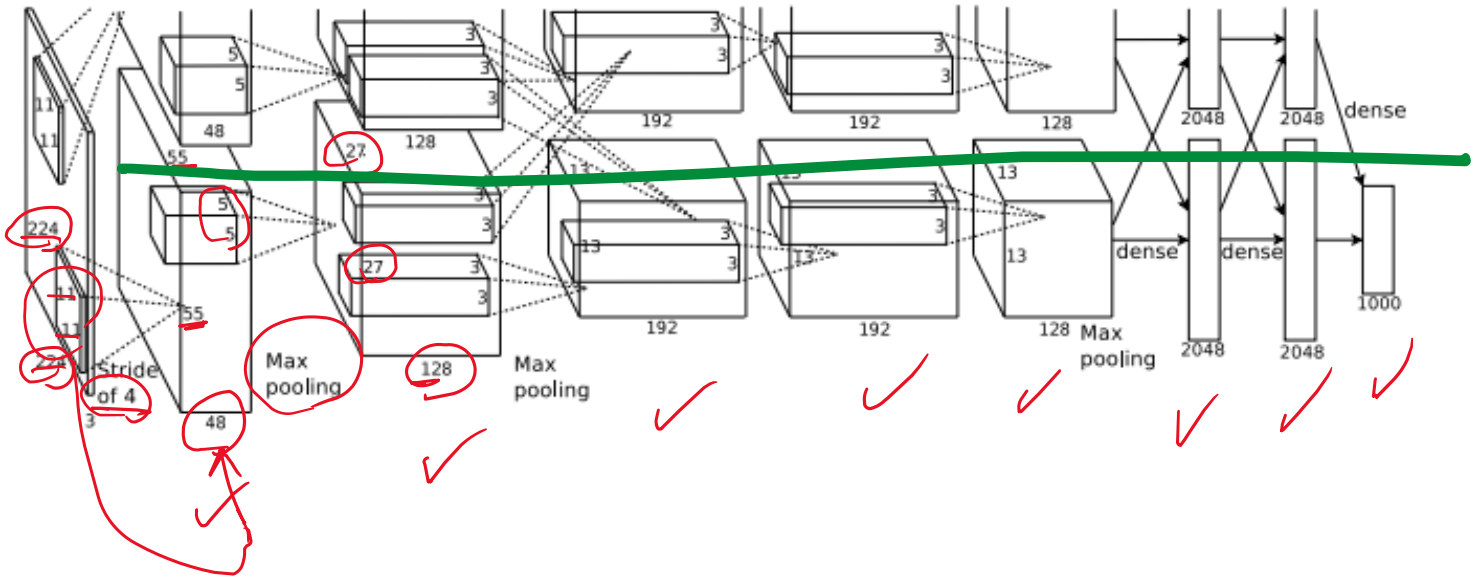


Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. “Gradient-based learning applied to document recognition,” 1998.

AlexNet

Convolutions, max pooling, dropout, data augmentation, ReLU activations, SGD with momentum

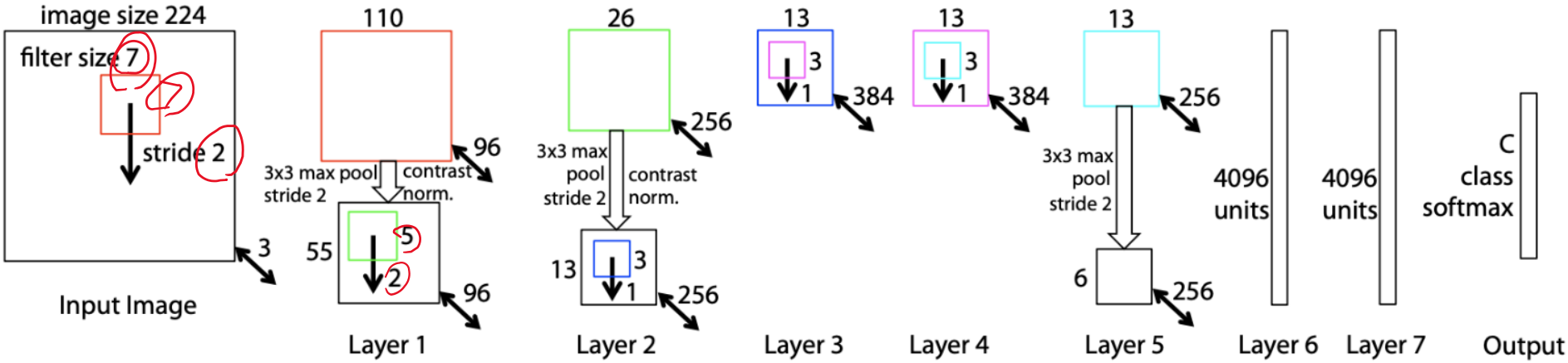
Two pipelines to fit into two GPUs



Alex Krizhevsky, et al. “ImageNet Classification with Deep Convolutional Neural Networks”, 2012.

ZFNet

Tweaking hyperparameters



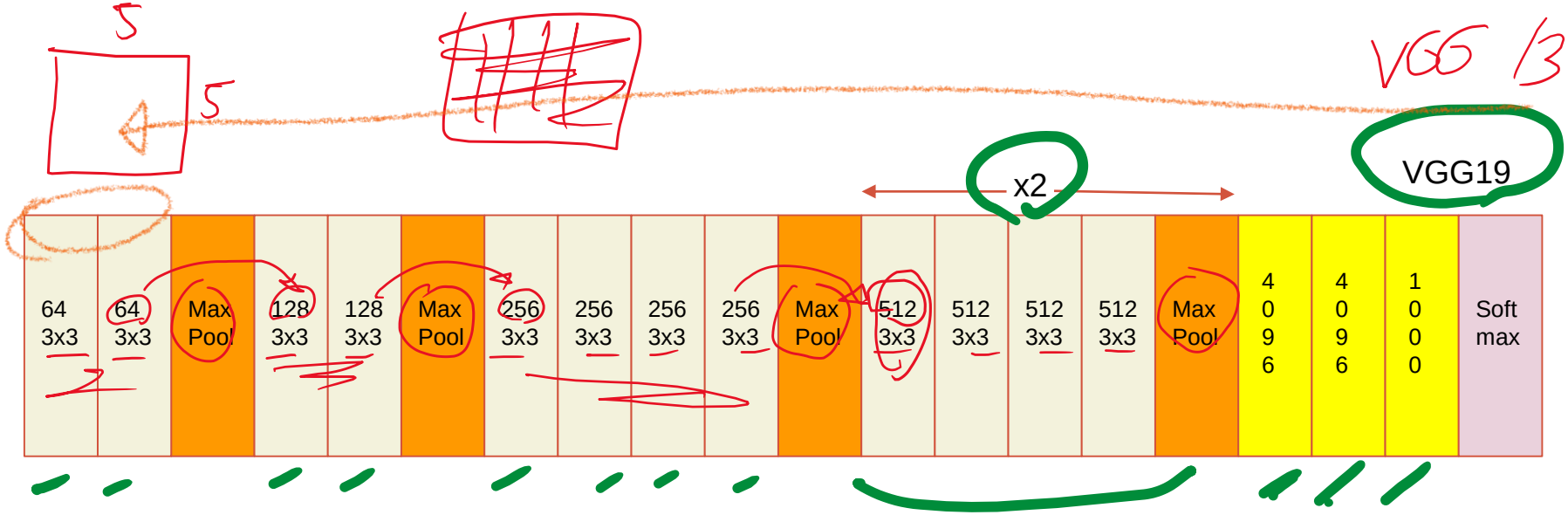
Matthew D. Zeiler, Rob Fergus, “Visualizing and Understanding Convolutional Networks,” 2013

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



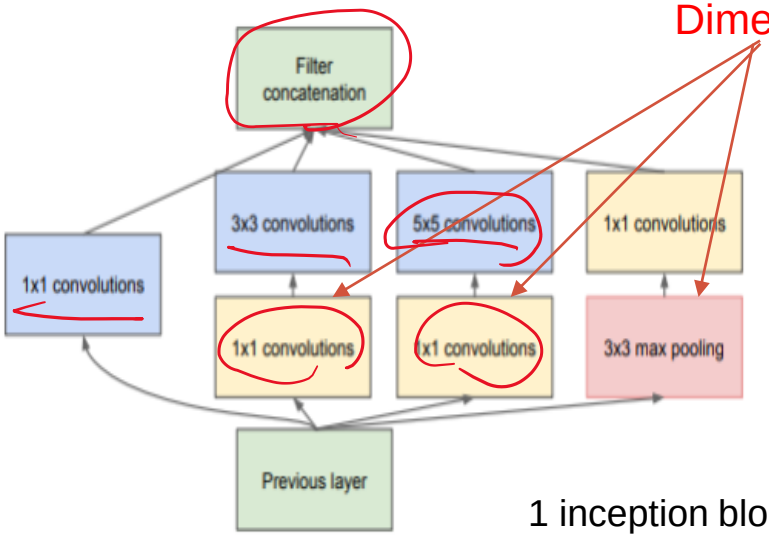
Karen Simonyan, Andrew Zisserman“Very Deep Convolutional Networks for Large-Scale Image Recognition” 2014

GoogLeNet (Inception v1)

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

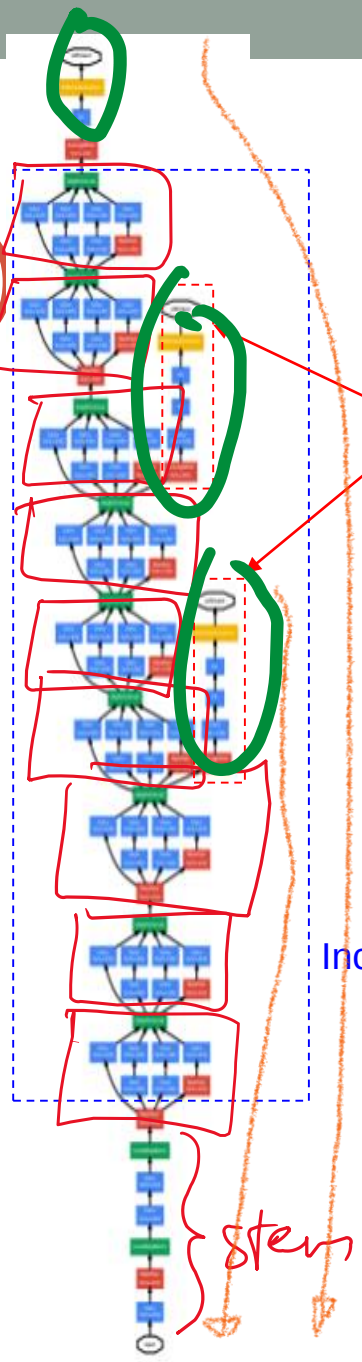
Dimensionality reduction via 1x1 convolution

Multiple softmax losses to help the gradient problem



Dimensionality reduction

1 inception block

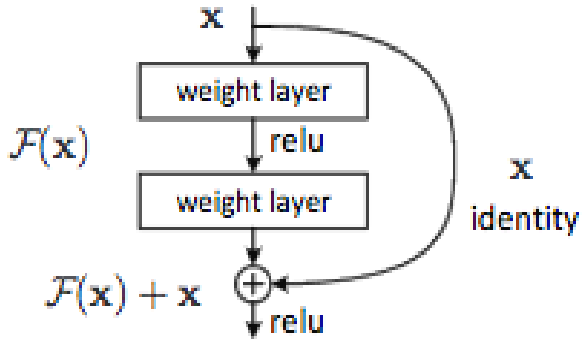


Christian Szegedy, et al. “Going Deeper with Convolutions,” 2014

Vanishing/Exploding gradient

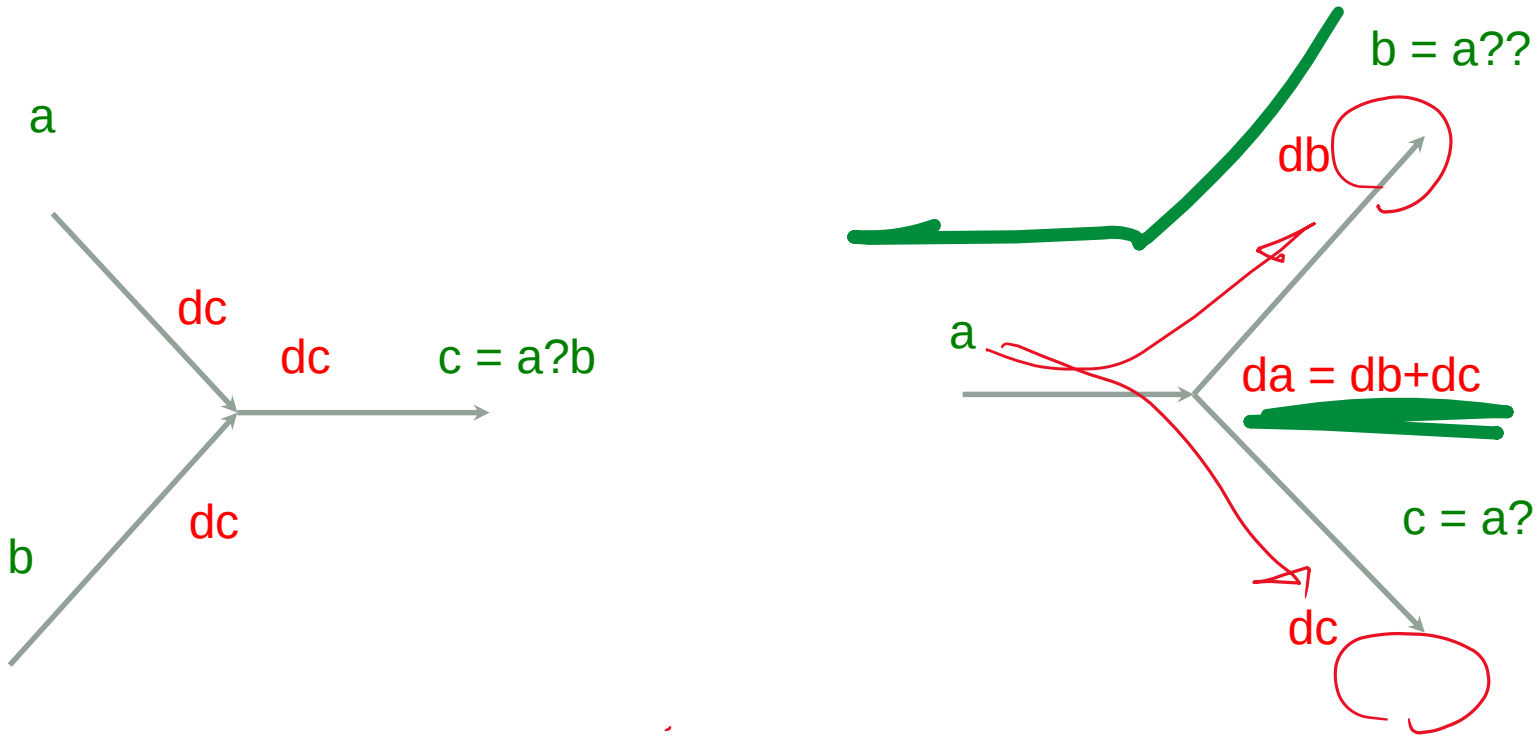
- Backprop introduces many multiplications down chain
- The gradient value gets smaller and smaller
  - The deeper the network the smaller the gradient in the lower layers
  - Lower layers changes too slowly (or not at all)
  - Hard to train very deep networks (>6 layers)
- The opposite can also be true.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

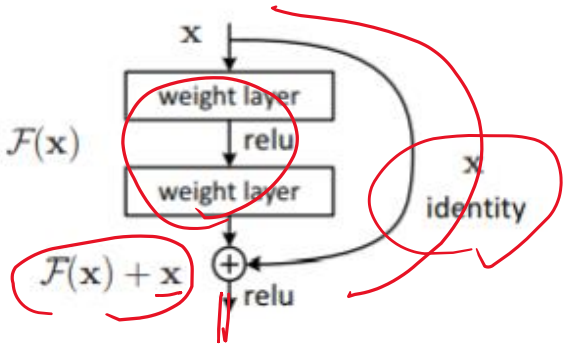


Forward and backward pass acts differently at forks

ResNet (Residual Network)

Batch norm

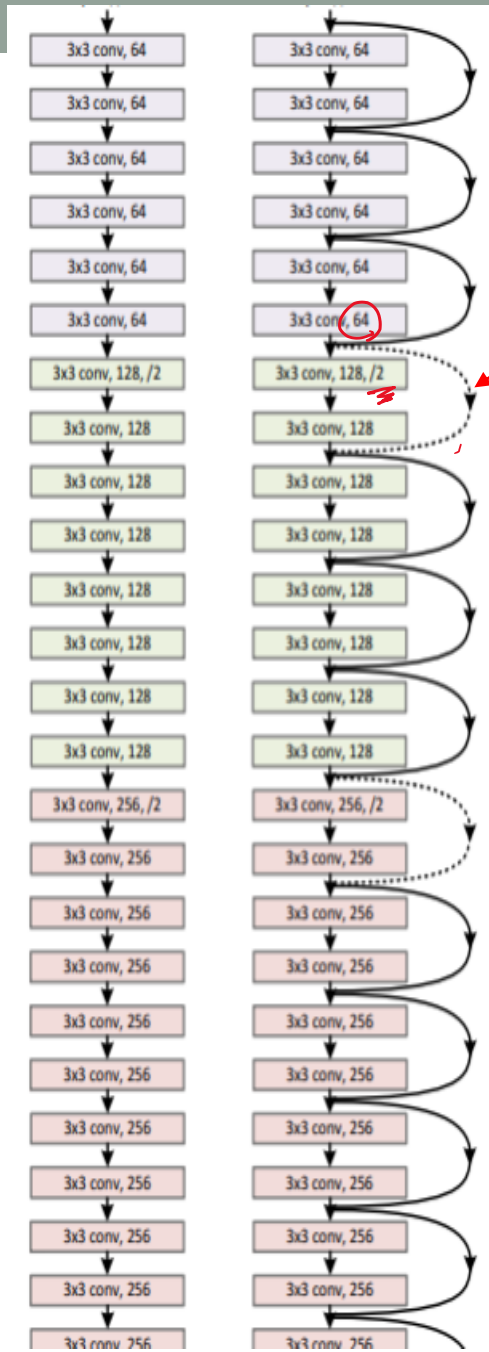
Extra “skip connections” to reduce the vanishing gradient problem



$y = F(x) + X$

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

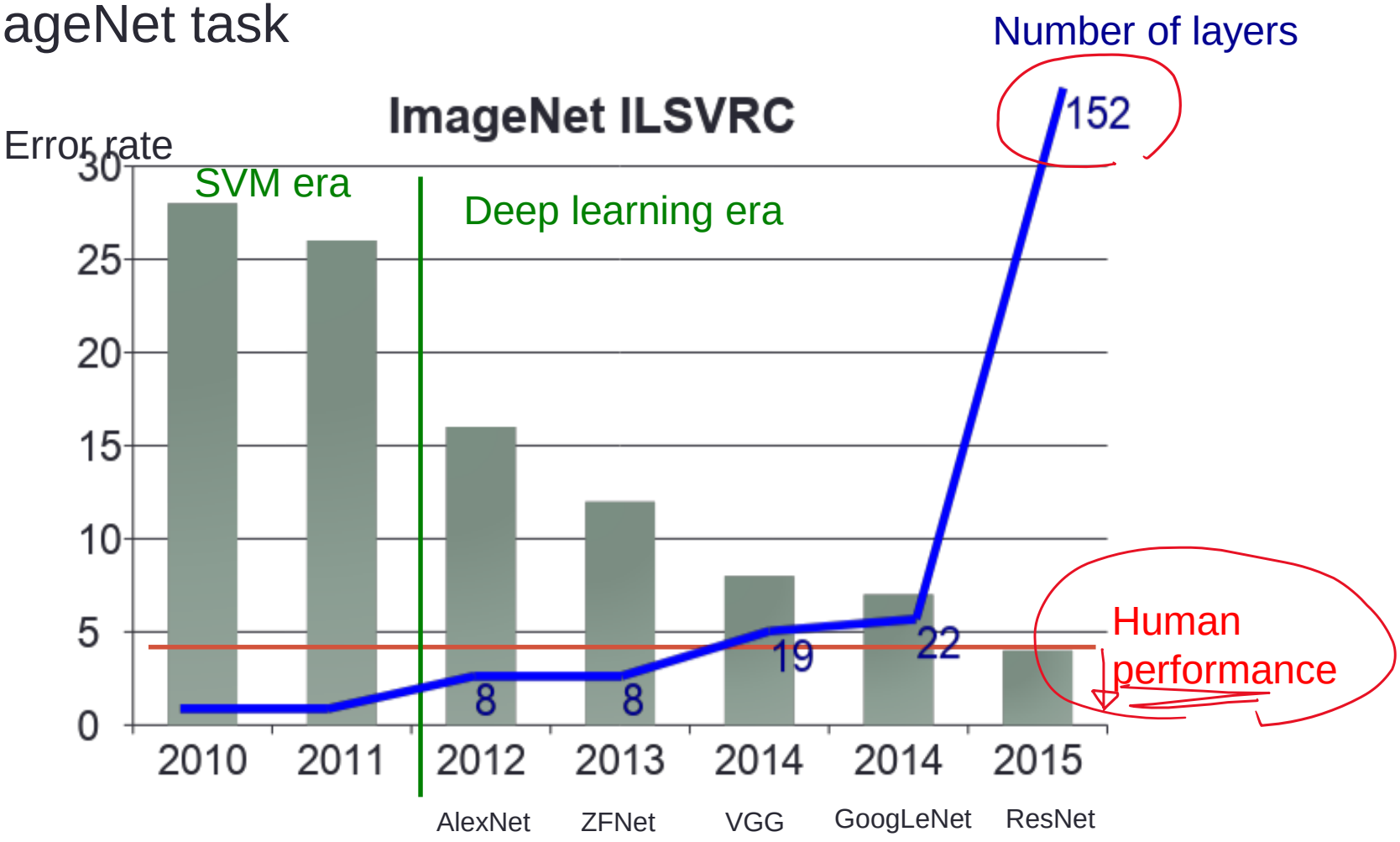
residual $y - X = F(x)$



Need extra operations to make the size the same

A brief history of imagenet architectures

- ImageNet task



Olga Russakovsky, ImageNet Large Scale Visual Recognition Challenge, 2014 https://arxiv.org/abs/1409.0575

Inception v2+v3

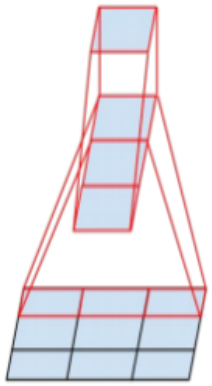
Implement 5x5 with two 3x3s

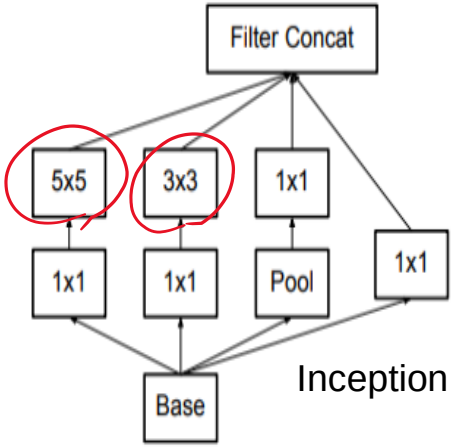
Factorized convolution

3x3 -> 3x1 and 1x3

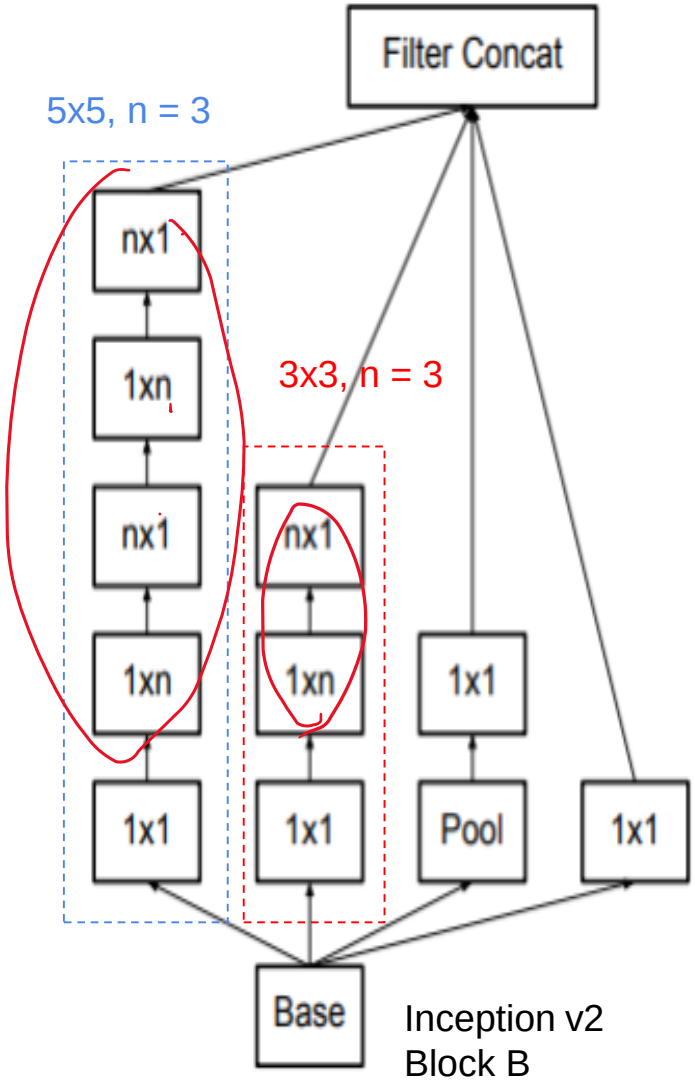
3 types of inception blocks

RMSprop, Batch norm, label smoothing





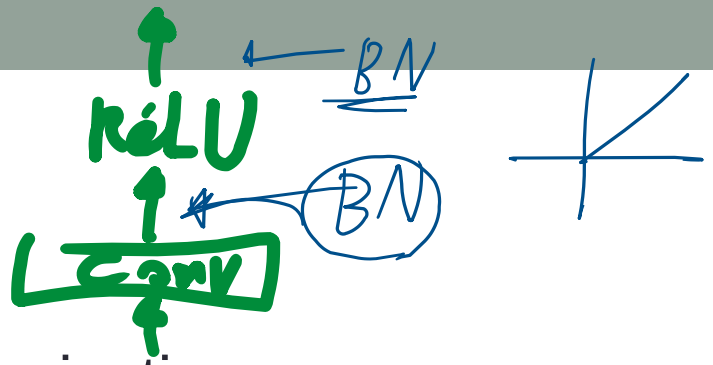
Inception v1



Inception v2
Block B

Christian Szegedy, “Rethinking the Inception Architecture for Computer Vision,” 2015

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

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

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

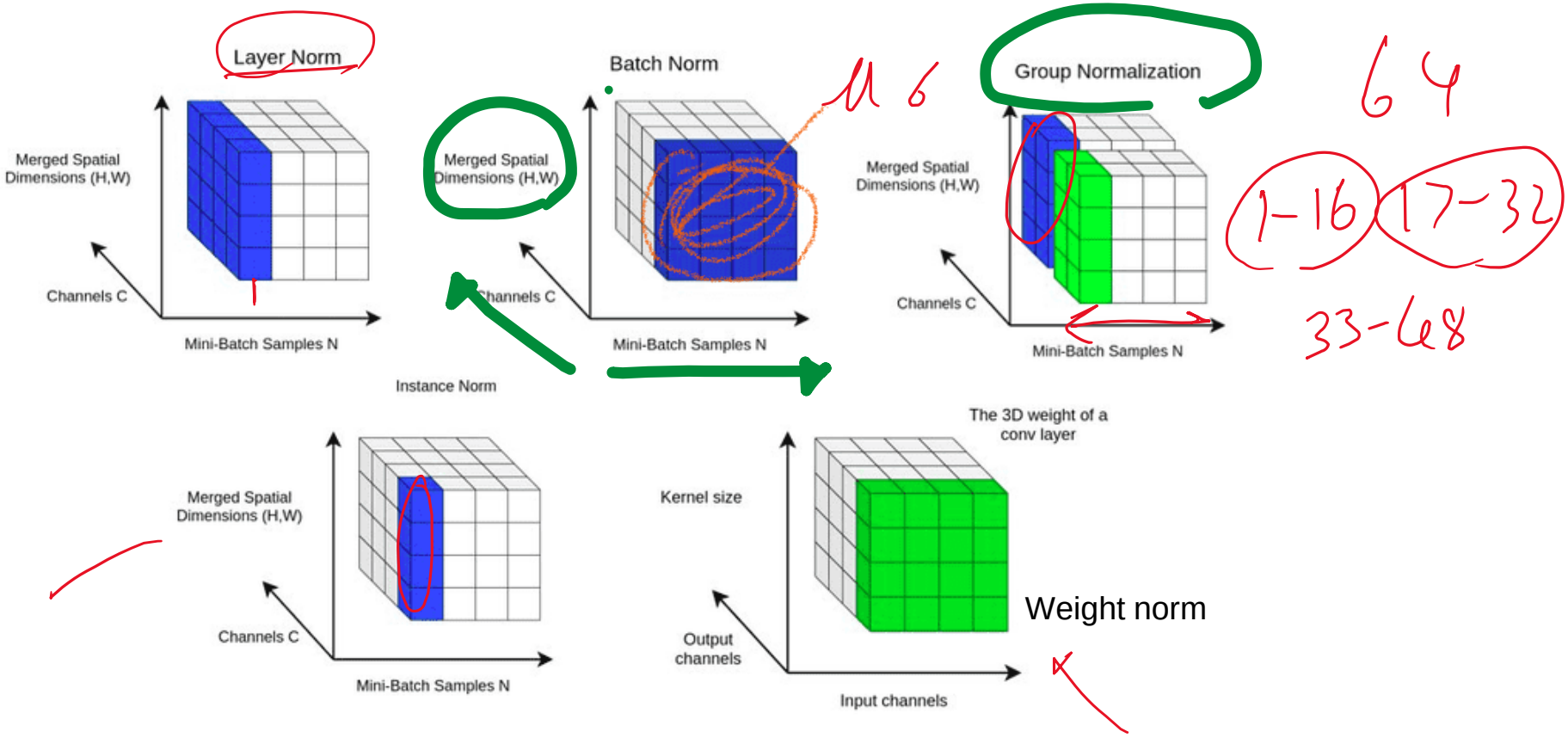
Replaces dropout in some networks

https://arxiv.org/abs/1502.03167

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

Other normalizations

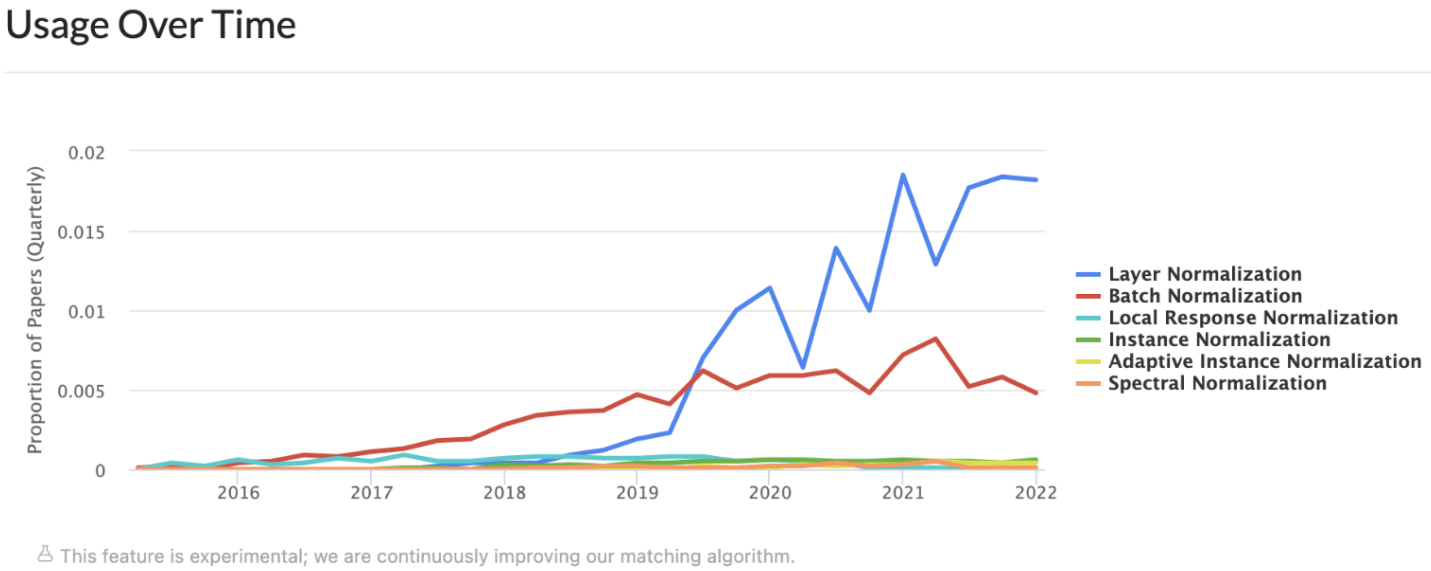
- Other normalizations are out there



https://theaisummer.com/normalization/

Other normalizations

Trend of normalization used in papers



NLP – layer norm
Image – Batch norm/layer norm
Small batches - Group norm (batch norm is not reliable in small batches)

https://paperswithcode.com/method/layer-normalization

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 .

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

The mean and standard-deviation are calculated per-dimension over the mini-batches and γ and β are learnable parameter vectors of size C (where C is the input size). By default, the elements of γ are set to 1 and the elements of β 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.

https://pytorch.org/docs/stable/generated/torch.nn.BatchNorm2d.html

Xception

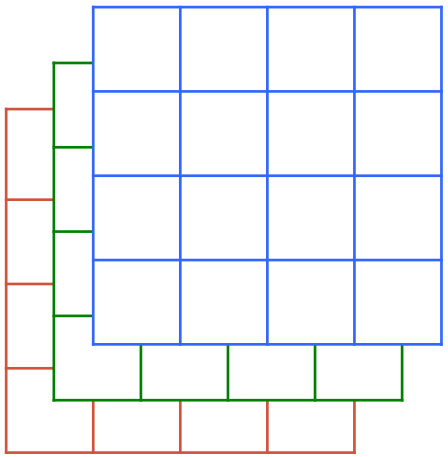
Depthwise separable convolution: two-step convolution

1. Depthwise convolution
2. 1x1 convolution

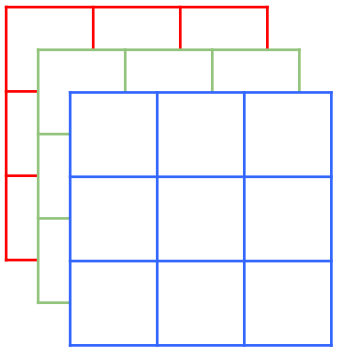
Typical convolution 3x3 filter is 3x3xinput channel

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

input



One regular convolution filter



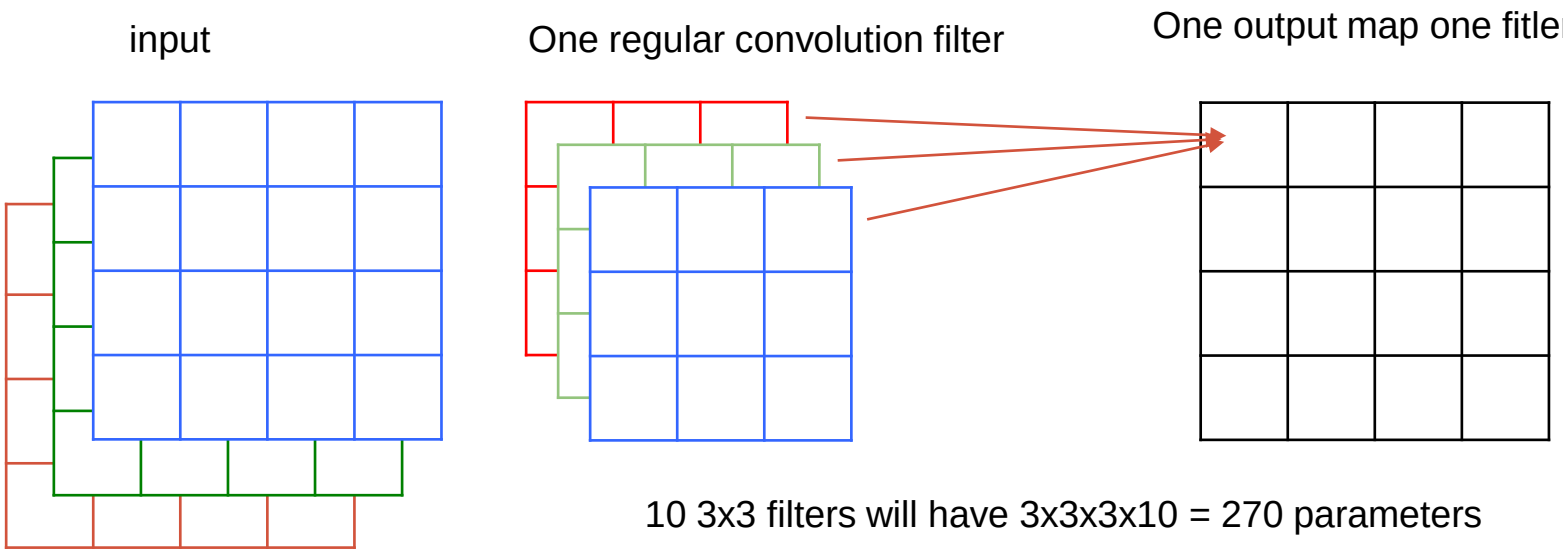
Xception

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

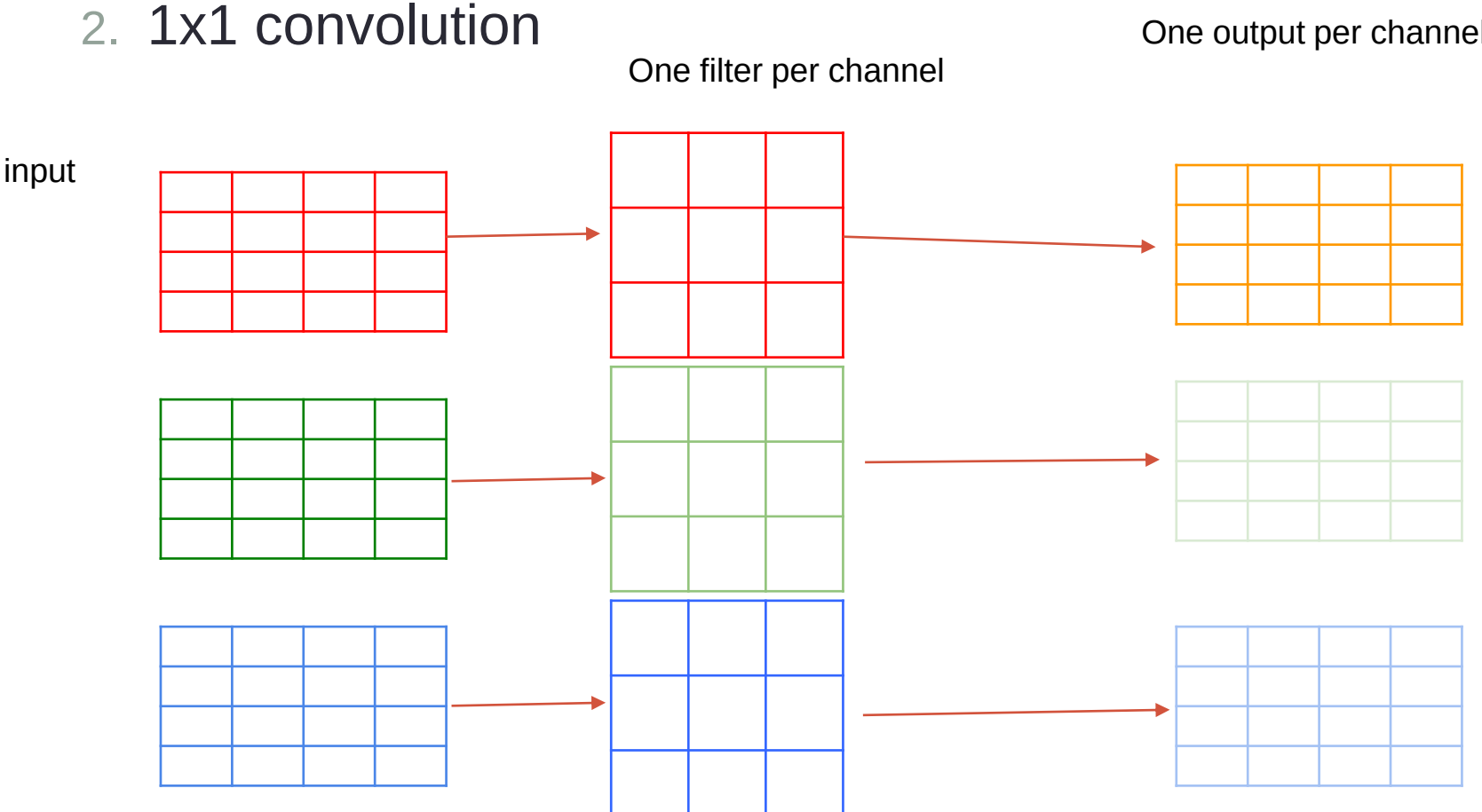


10 3x3 filters will have 3x3x3x10 = 270 parameters

Xception

Depthwise separable convolution: two-step convolution

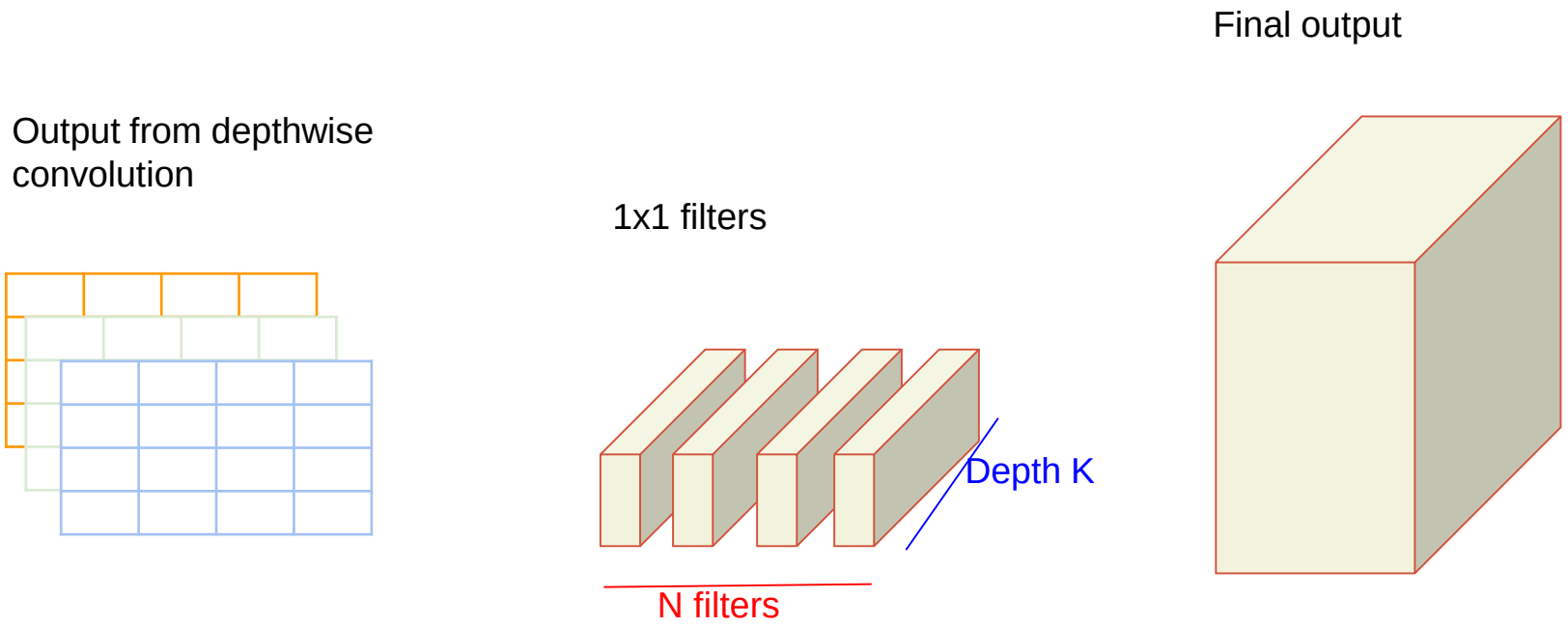
1. Depthwise convolution (learn from each channel, spatial info)
2. 1x1 convolution



Xception

Depthwise separable convolution: two-step convolution

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



10 3x3 depthwise separable convoltuion will have 3x3x3 + 3x10 = 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

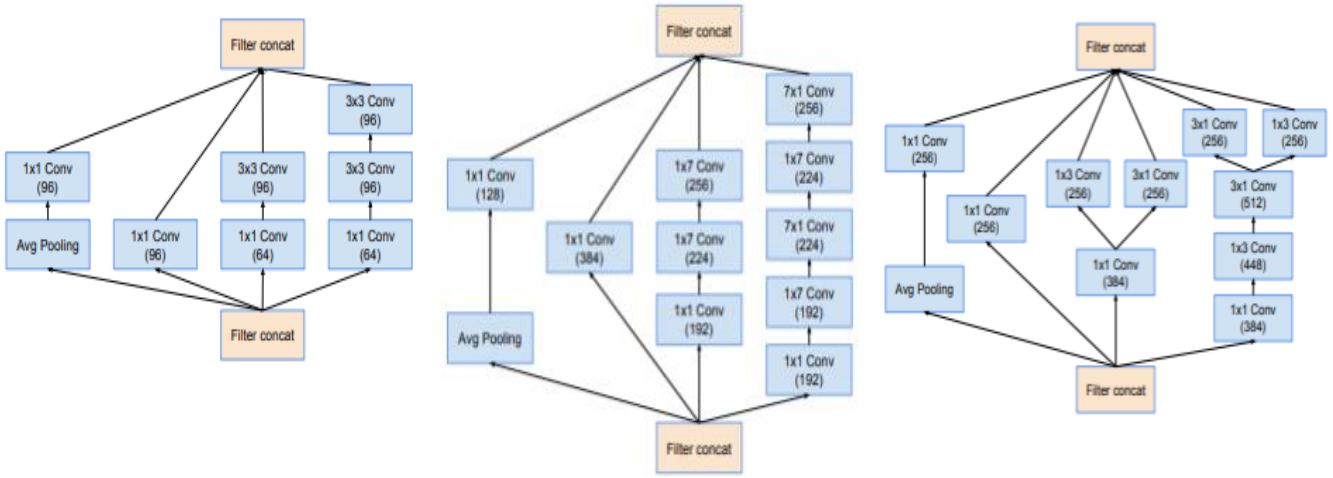
François Chollet, “Xception: Deep Learning with Depthwise Separable Convolutions” 2016.
Andrew G. Howard, et al., “MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications” 2017.

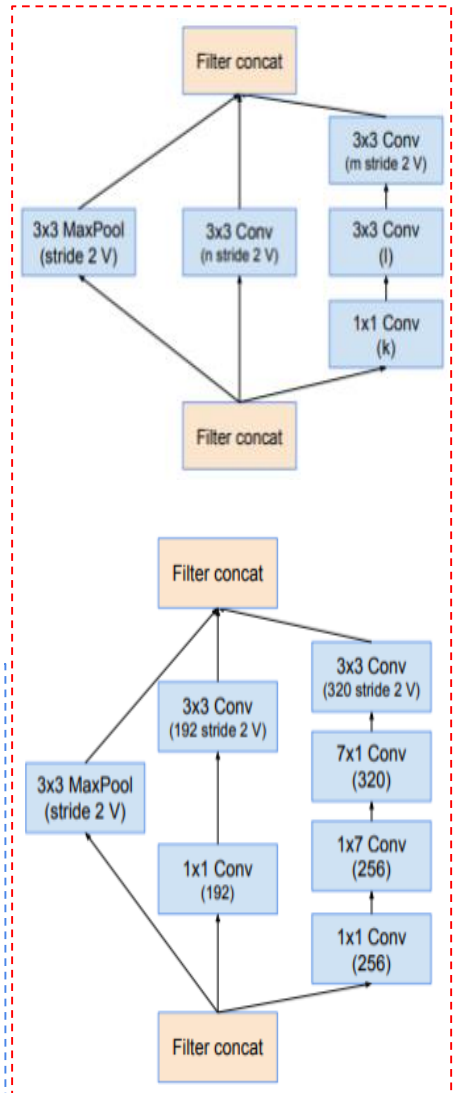
Inception v4

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

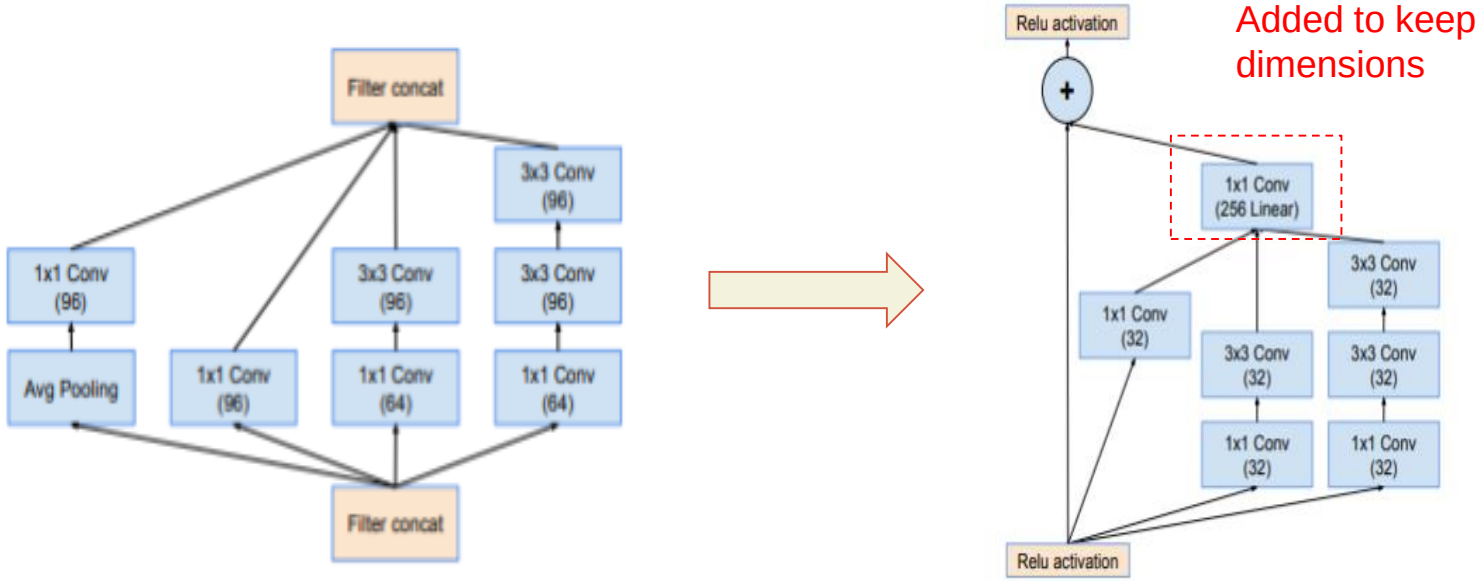
Christian Szegedy, “Inception-v4, Inception-ResNet and the Impact of Residual Connections on Learning” 2016.

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



Christian Szegedy, “Inception-v4, Inception-ResNet and the Impact of Residual Connections on Learning” 2016.

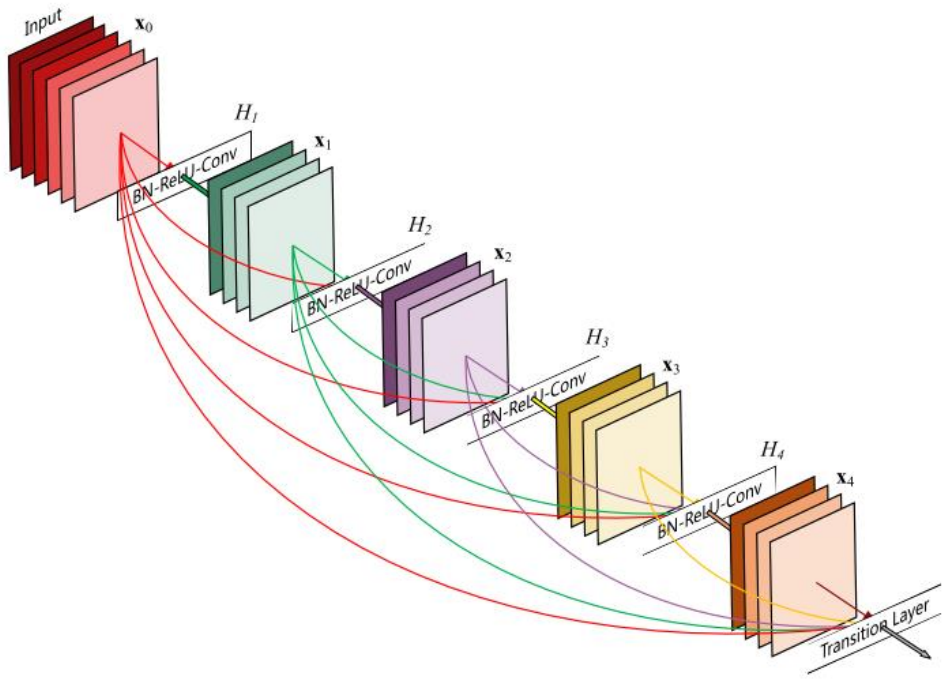
DenseNet

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

Densely connect multiple layers

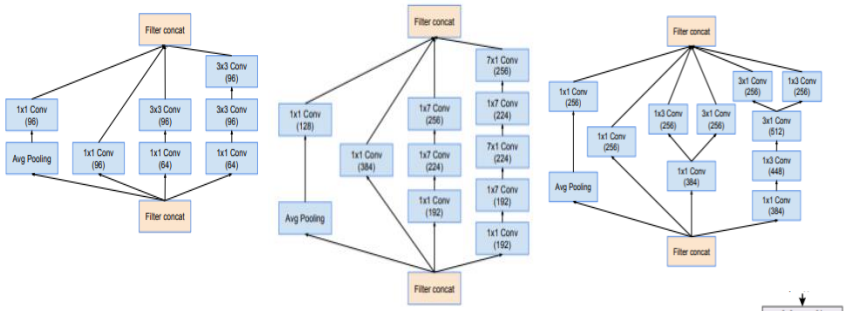
Multi-scale feature maps

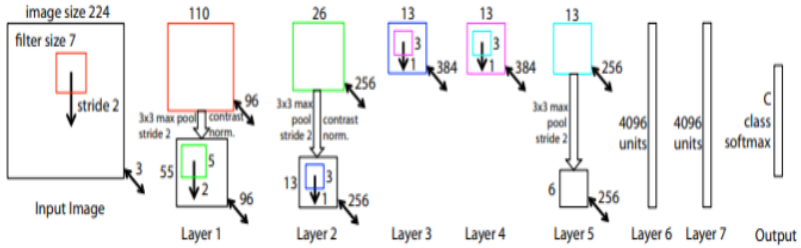
Smaller model due to smaller number of channels

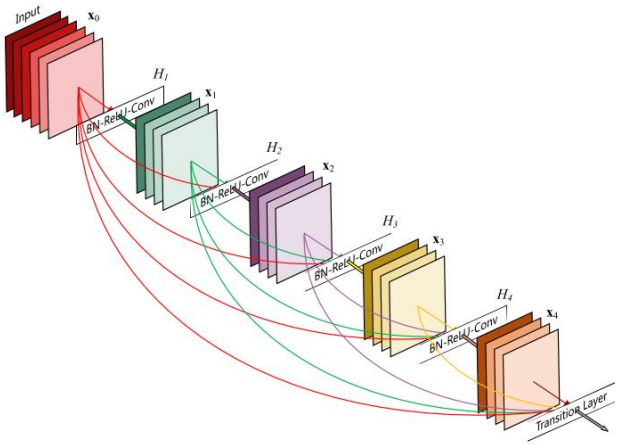


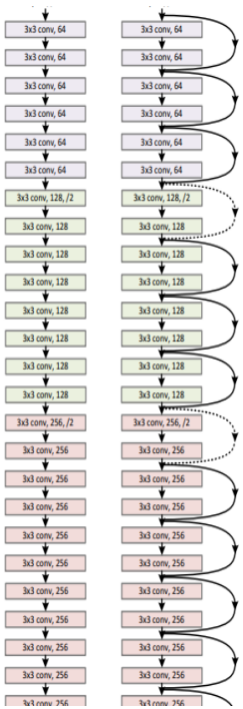
Densely Connected Convolutional Networks 2016

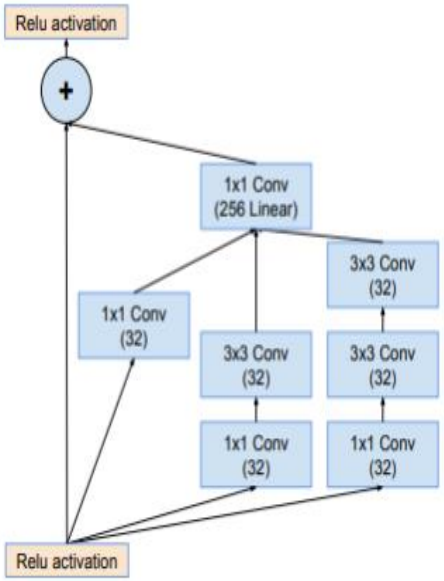
Do people keep tweaking network architectures until the end of time?









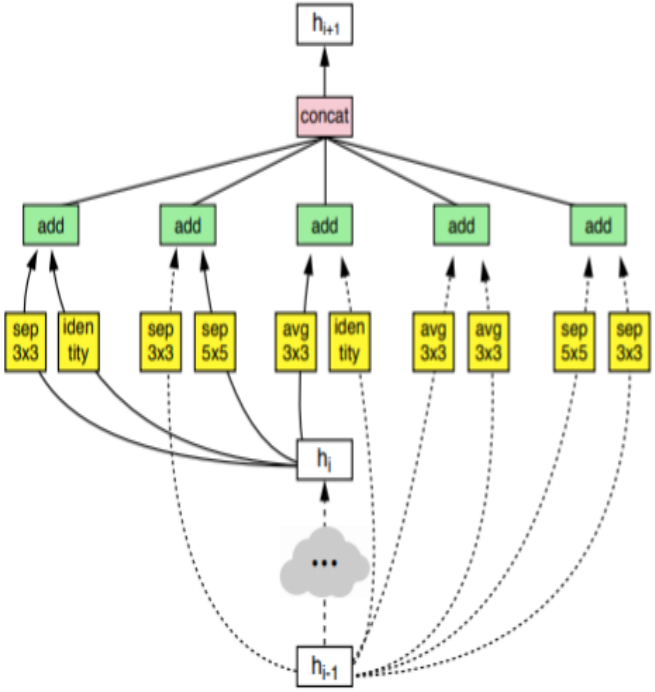


NasNet

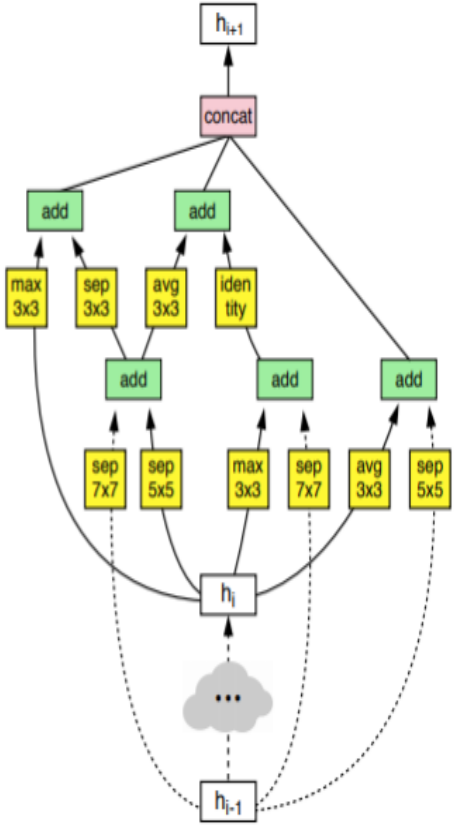
Neural Architecture Search Network

Use reinforcement learning to search for the best network configuration

Lowers compute (FLOPs) while maintaining high accuracy



Inception block



Reduction block

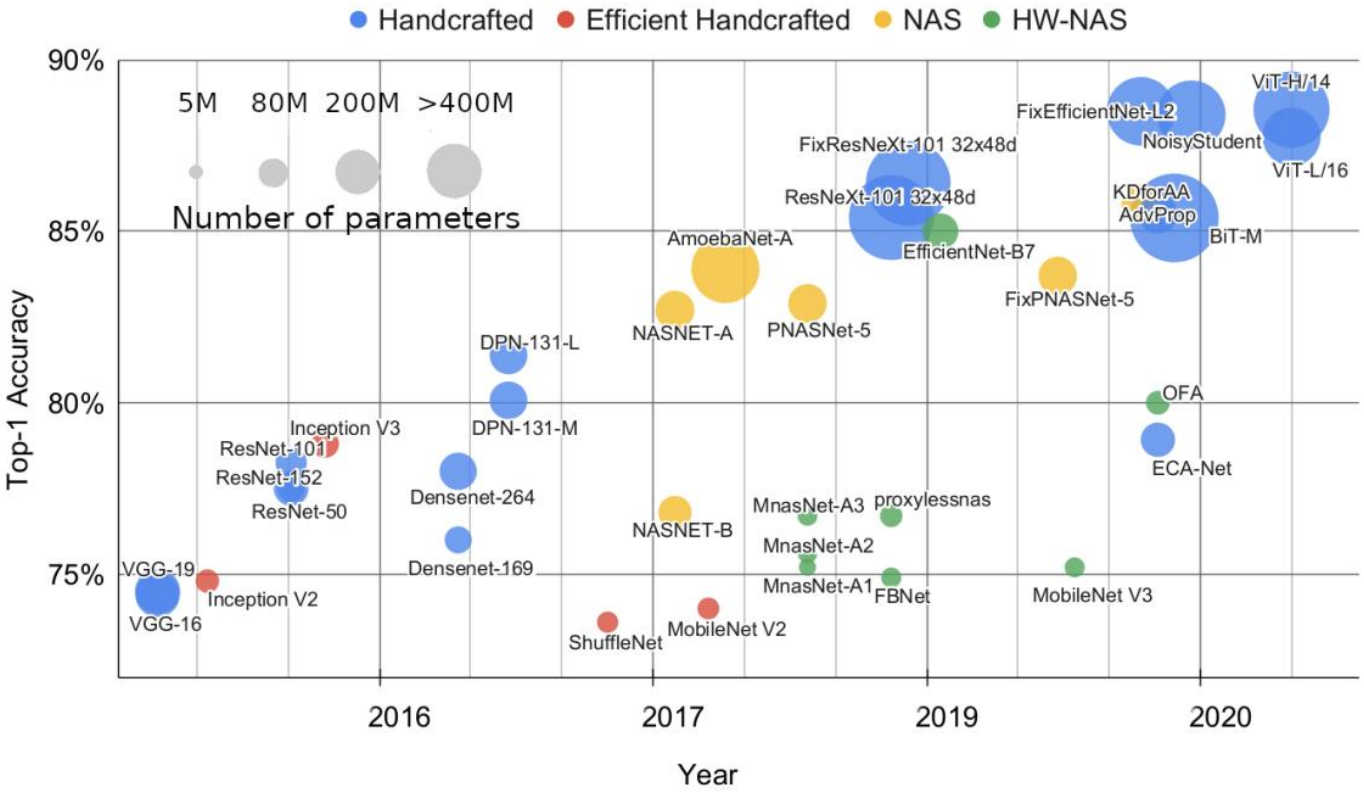
Barret Zoph, et al. “Learning Transferable Architectures for Scalable Image Recognition,” 2017.

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

$$Elapse\ Time = CPU\ Time + I/O\ Time + Memory\ Time$$

- FLOPs is not a good measure of performance



https://arxiv.org/abs/2101.09336

EfficientNetv2

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

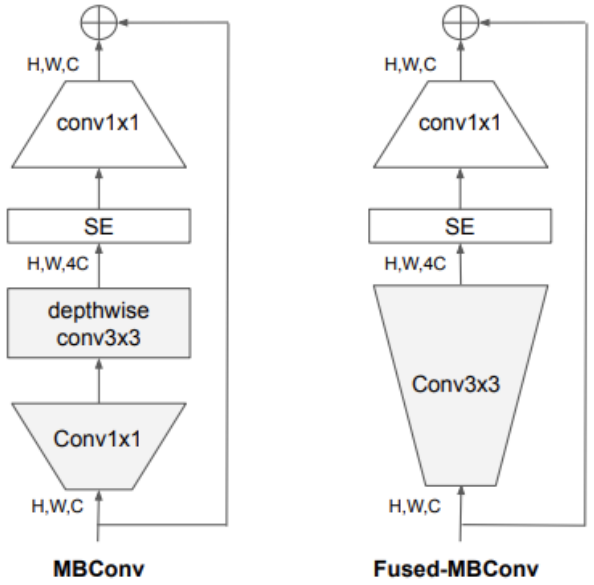


Figure 2. Structure of MBConv and Fused-MBConv.

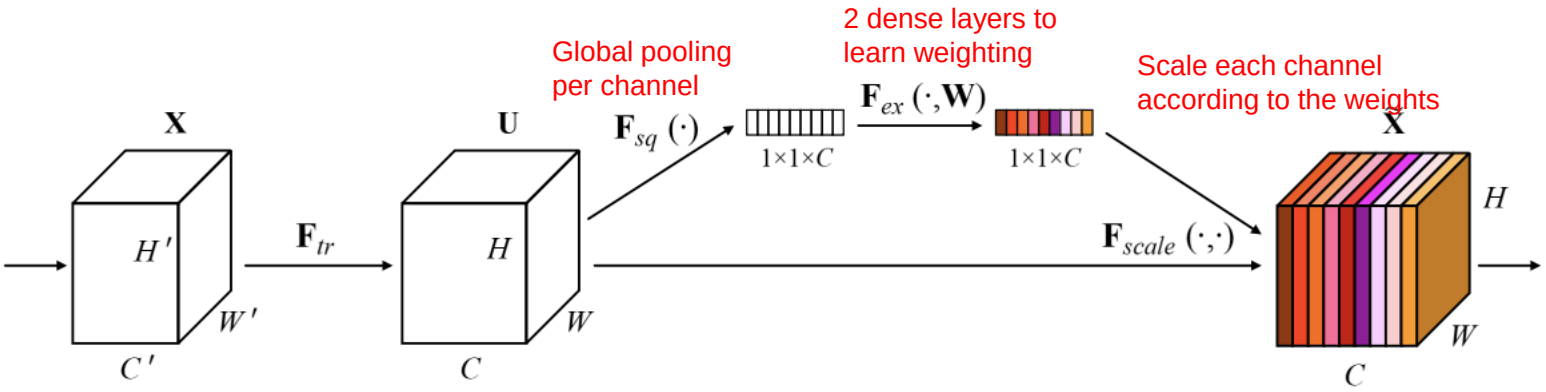
https://arxiv.org/abs/1905.11946
https://arxiv.org/abs/2104.00298

Modern convolutional blocks

- Squeeze and excitation
- Mobile inverted bottleneck

Squeeze and excitation

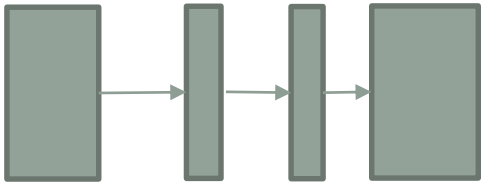
- Maybe each channel should be weighted differently?
  - A picture with wave patterns should look for fish features
  - A picture with grass patterns should look for cow features
- “Attention” on channel info



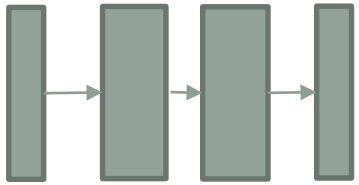
https://arxiv.org/abs/1709.01507

Inverted bottleneck

- A bottleneck block
  - reduce channels using 1x1
  - Regular conv on fewer channels
  - Increase channel using 1x1



- Inverted bottleneck
  - Increase channels using 1x1
  - Regular conv on more channels
  - Reduce channels using 1x1



- Bottleneck combines information between channels
- With residual connections, inverted bottleneck uses less runtime memory during inference (DAG computation graph)

https://arxiv.org/abs/1801.04381

Mobile inverted Bottleneck

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

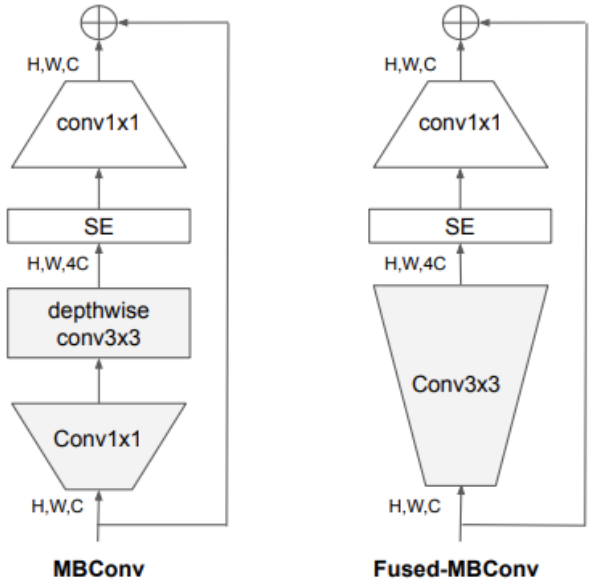
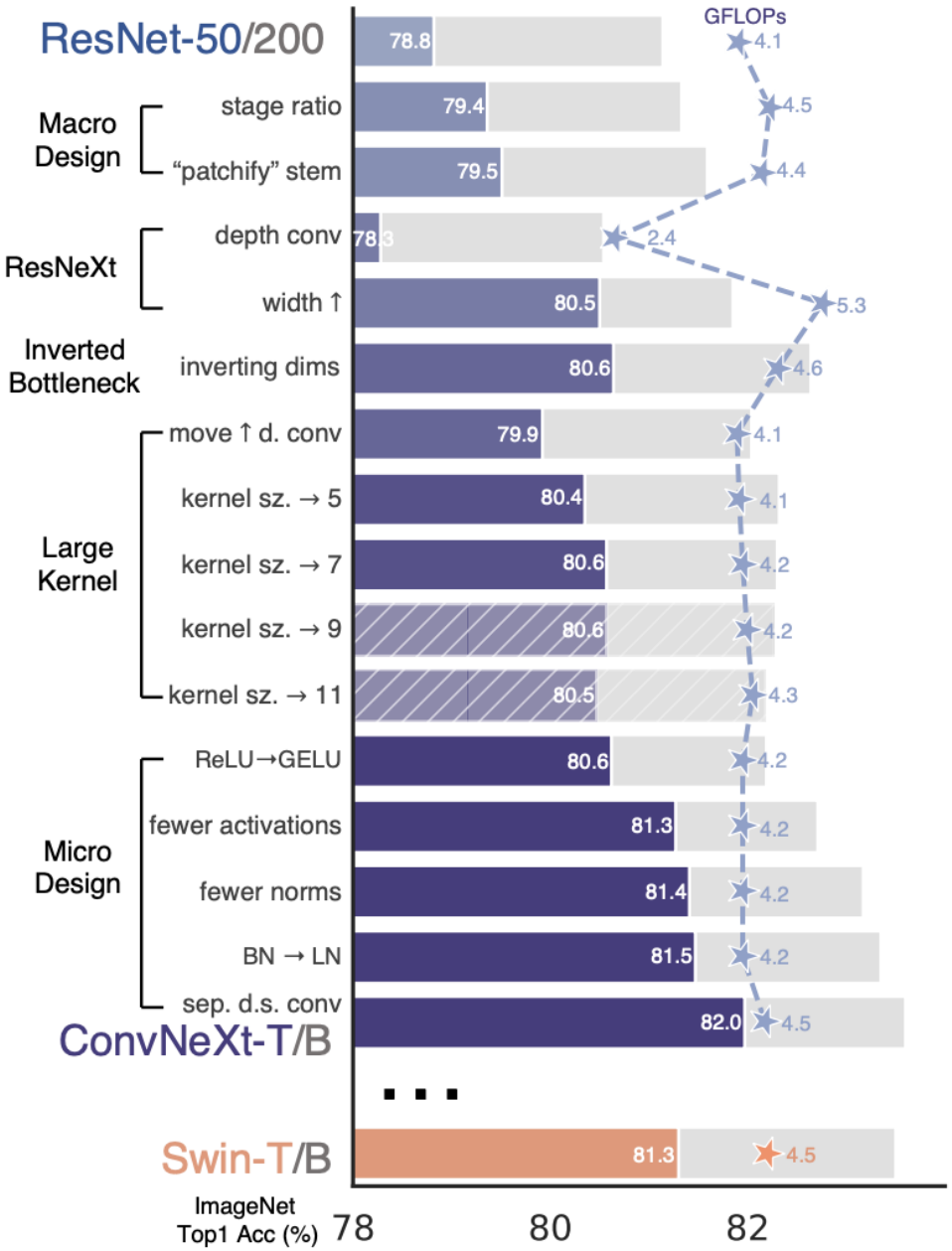


Figure 2. Structure of MBConv and Fused-MBConv.

ConvNext

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



https://arxiv.org/abs/2201.03545

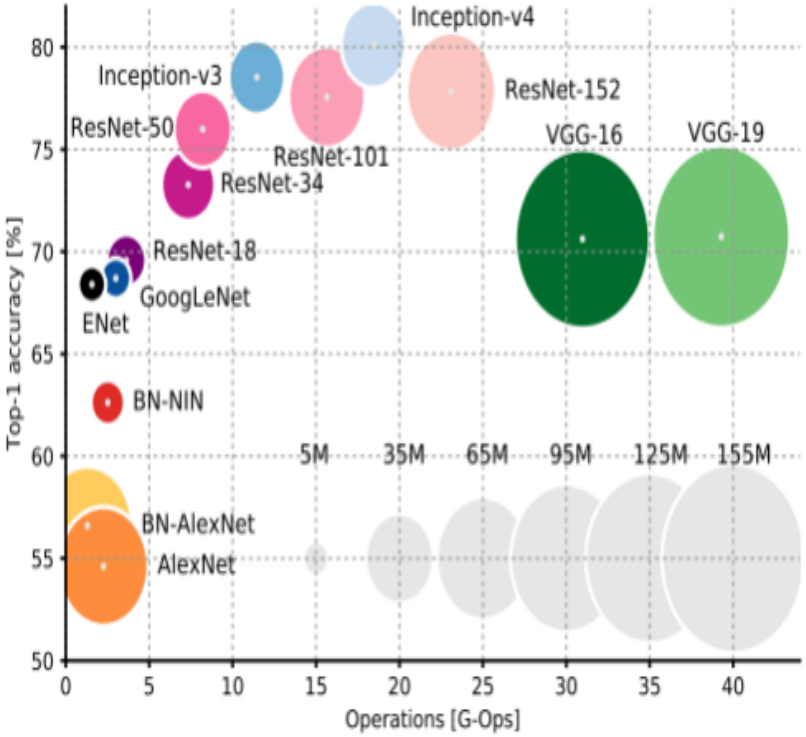
Object classifiers summary

Most successful tricks:

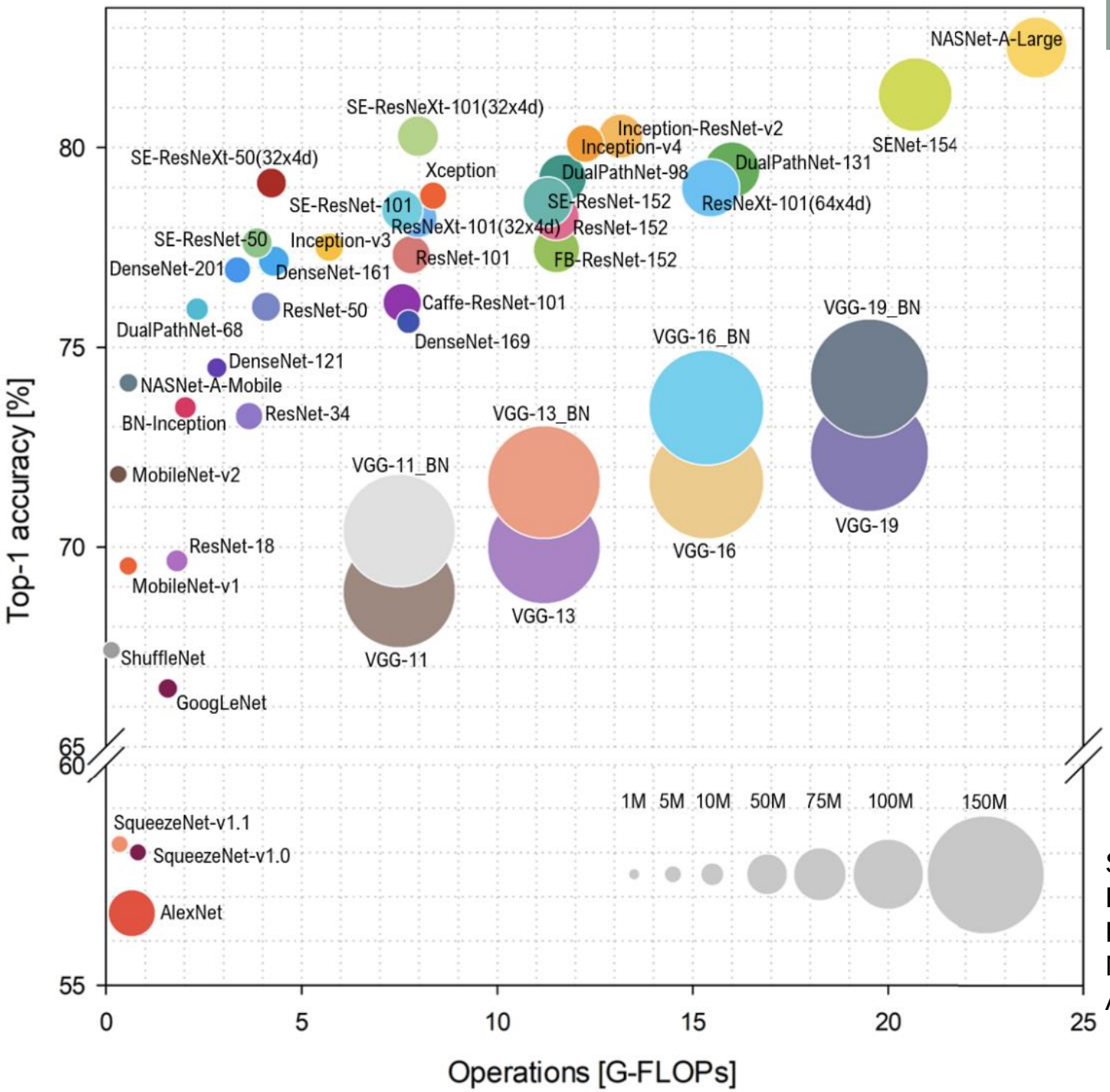
Dropout, residual, batch norms, multi-path, data augmentation

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

Comparing accuracy, model size, transferability and compute.



Alfredo Canziani, et al. “An analysis of deep neural network models for practical applications,” 2017



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

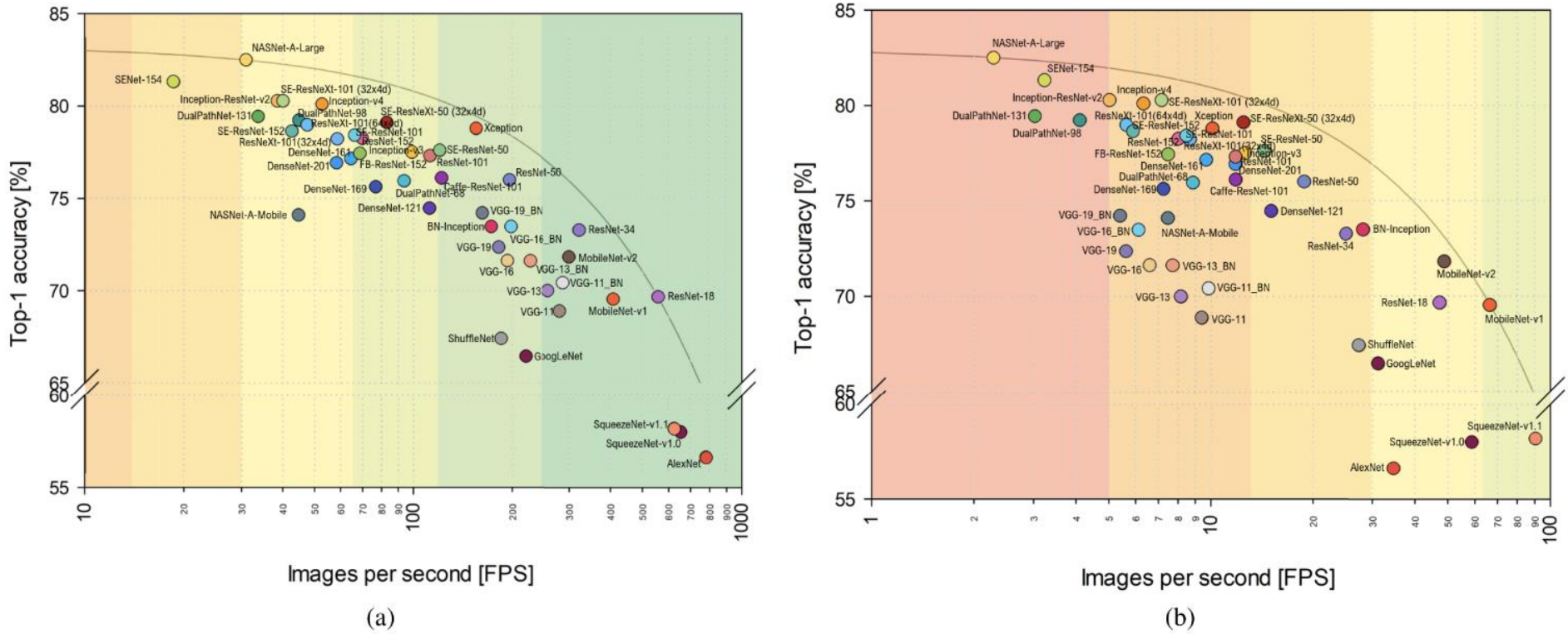


FIGURE 3. **Top-1 accuracy *vs.* number of images processed per second (with batch size 1) using the Titan Xp (a) and Jetson TX1 (b).**

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

Other stuff to look for

Layers

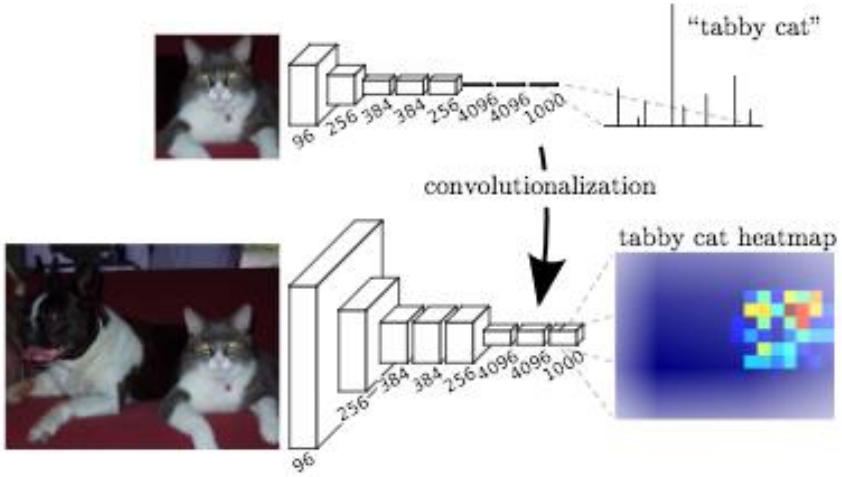
Dilated convolution https://towardsdatascience.com/understanding-2d-dilated-convolution-operation-with-examples-in-numpy-and-tensorflow-with-d376b3972b25

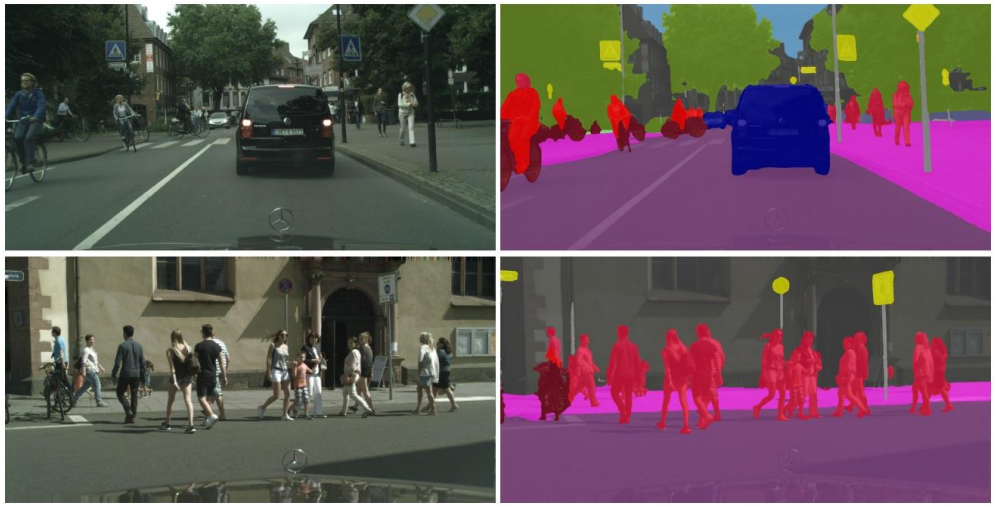
Necks

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



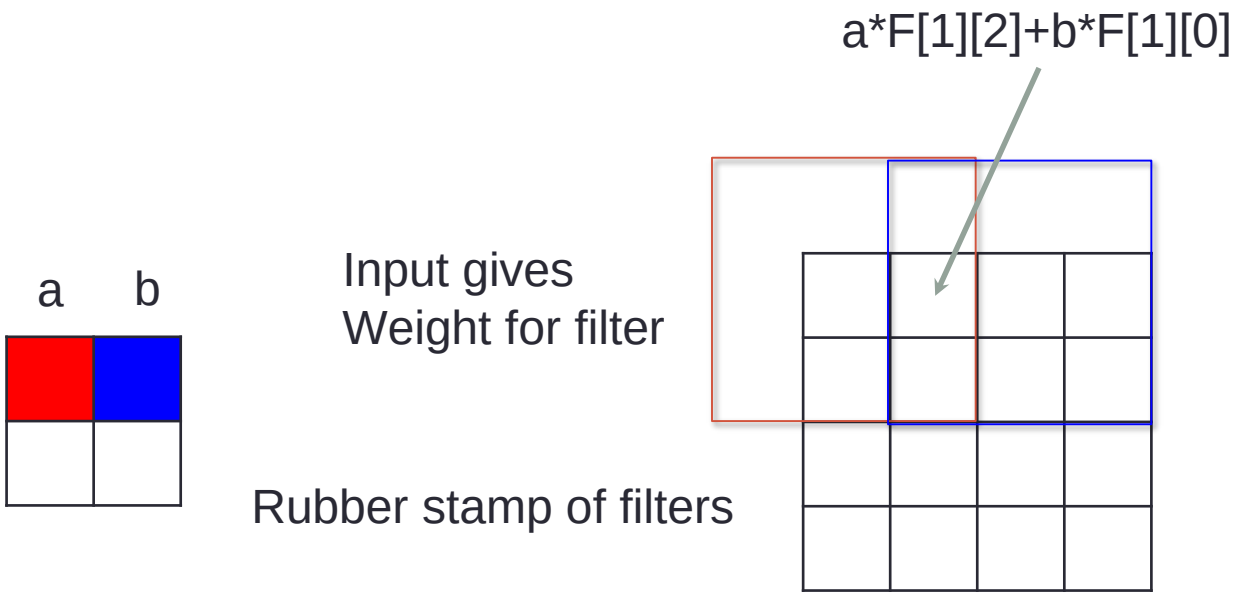


2 main approaches to upsample
De-convolution
resize (unpooling) +convolution

https://people.eecs.berkeley.edu/~jonlong/long_shelhamer_fcn.pdf
http://vladlen.info/publications/feature-space-optimization-for-semantic-video-segmentation/

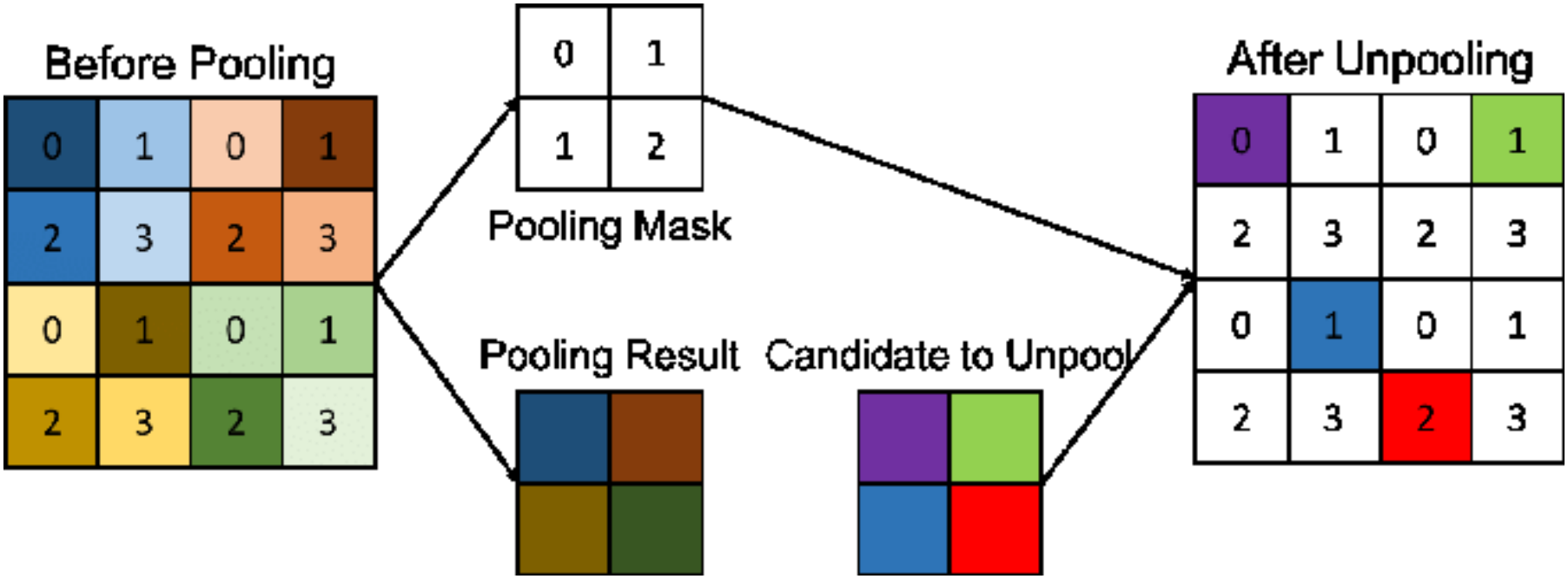
De-convolution (Upsampling)

- 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

Fully Automatic Acute Ischemic Lesion Segmentation in DWI Using Convolutional Neural Networks, 2017.

Upsampling notes

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

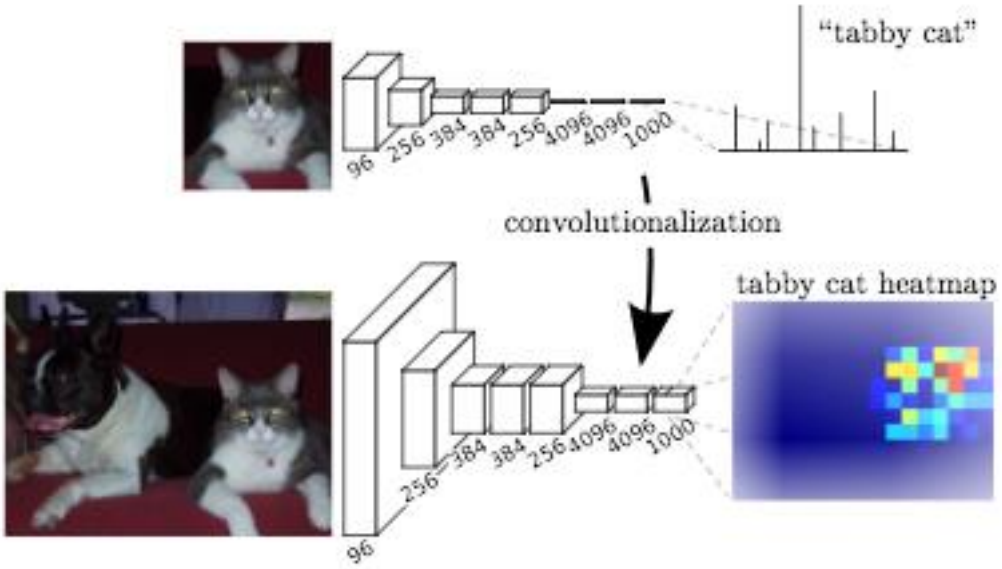
Unpooling can be replaced with regular image resizing techniques (interpolation)

Image using deconv

Image using resize upsampling

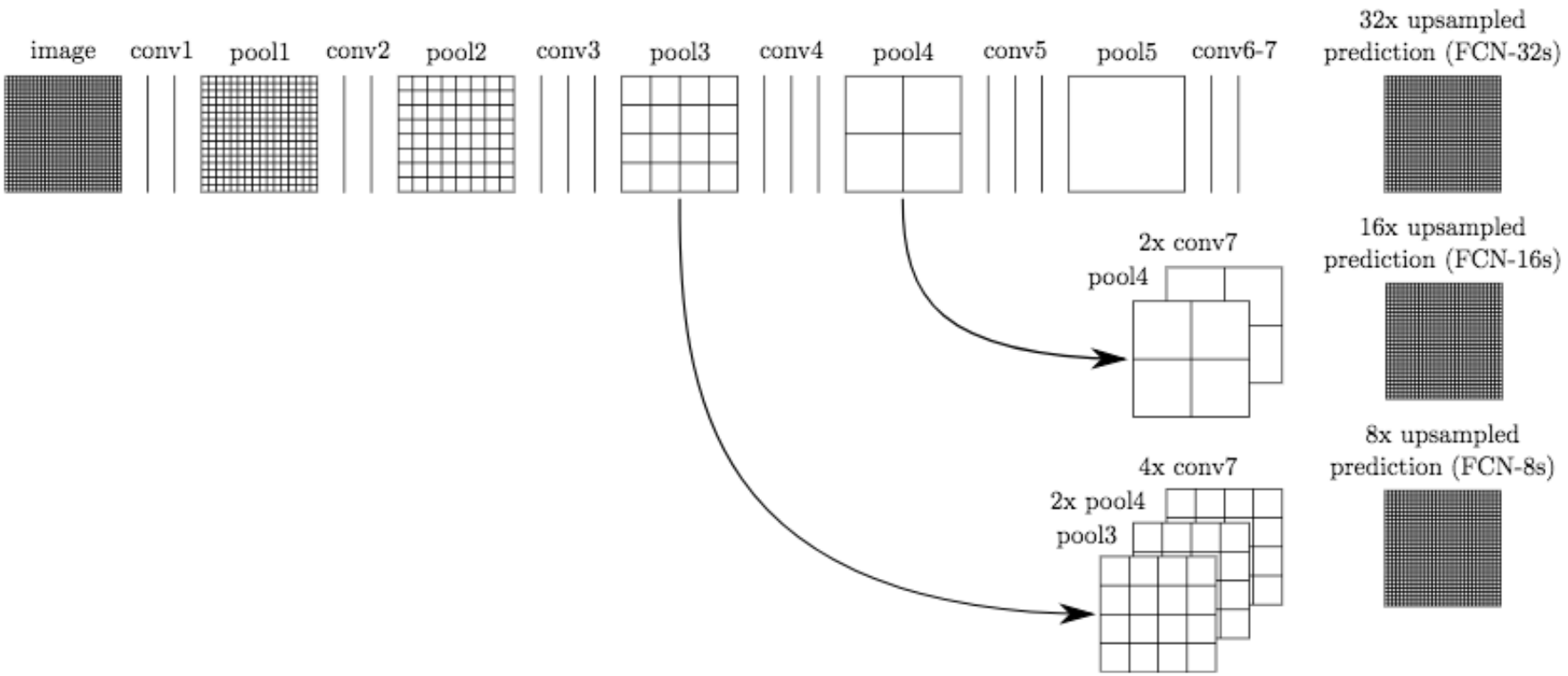
https://distill.pub/2016/deconv-checkerboard/

De-convolution for segmentation



https://people.eecs.berkeley.edu/~jonlong/long_shelhamer_fcn.pdf

De-convolution for segmentation



https://people.eecs.berkeley.edu/~jonlong/long_shelhamer_fcn.pdf

De-convolution for segmentation

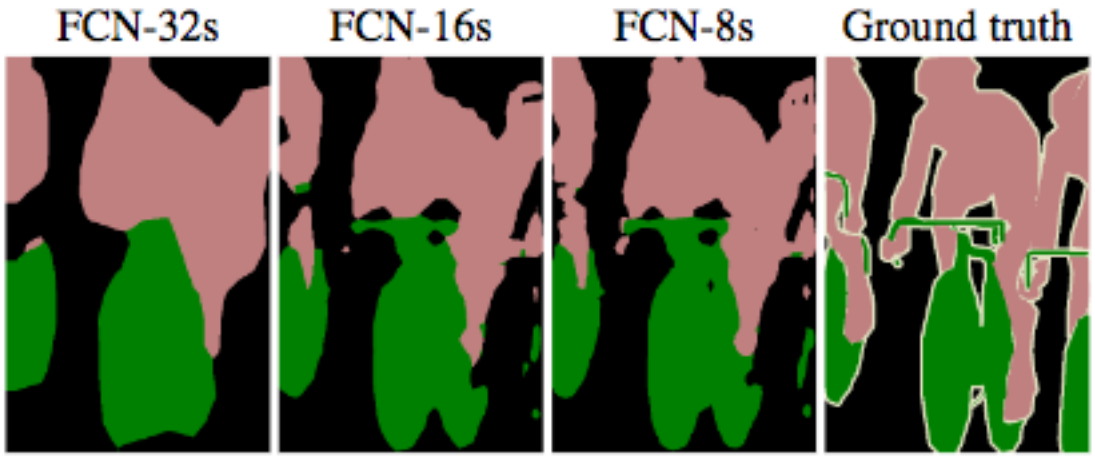


Figure 4. Refining fully convolutional nets by fusing information from layers with different strides improves segmentation detail. The first three images show the output from our 32, 16, and 8 pixel stride nets (see Figure 3).

https://people.eecs.berkeley.edu/~jonlong/long_shelhamer_fcn.pdf

CNN Summary

Building blocks

Matched filters and pooling

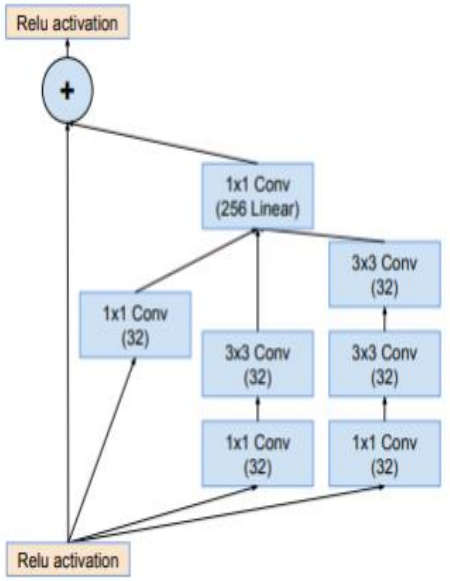
Filter factorization

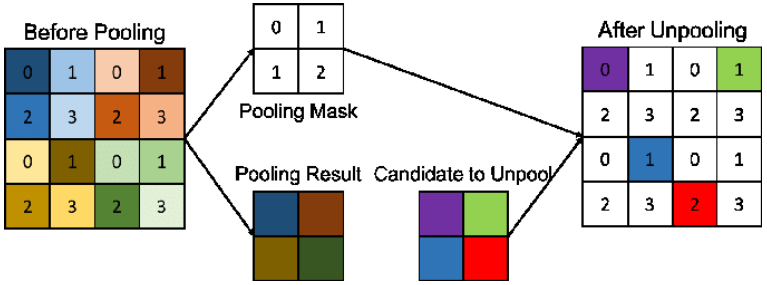
5x5 -> two 3x3

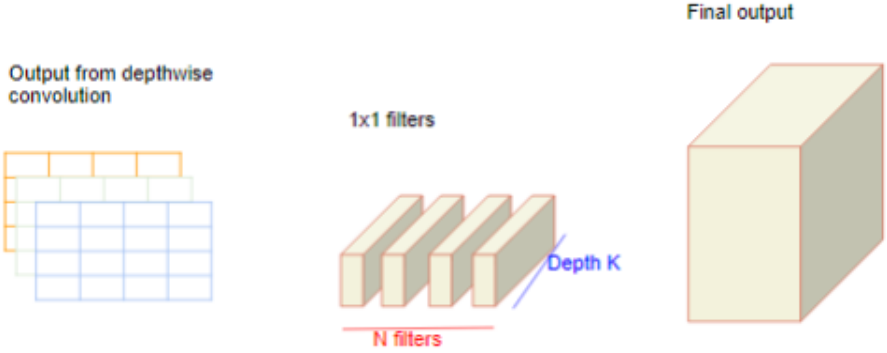
depthwise vs matrix factorization

1x1 convolution

Deconvolution and upsampling

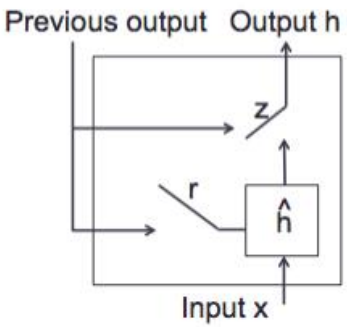


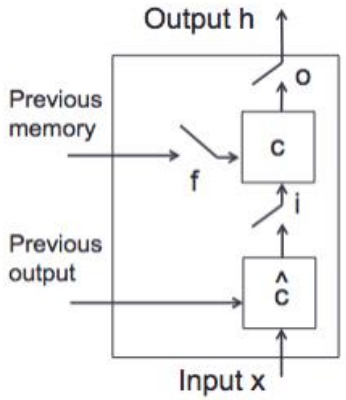


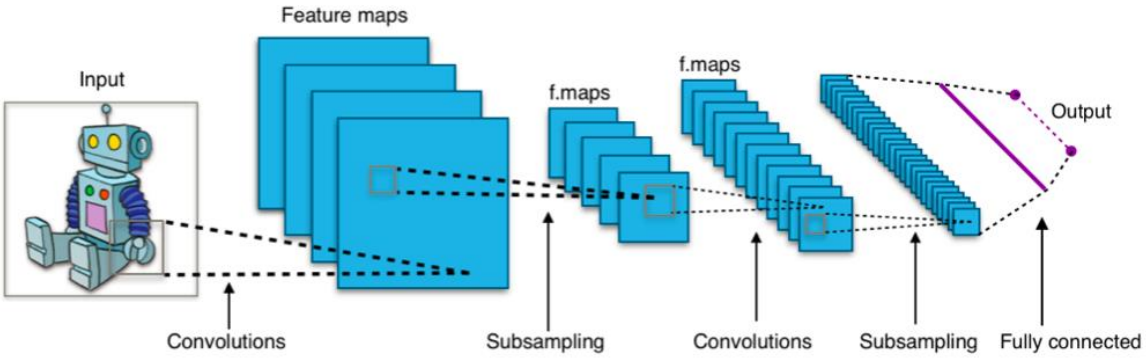


Neural networks

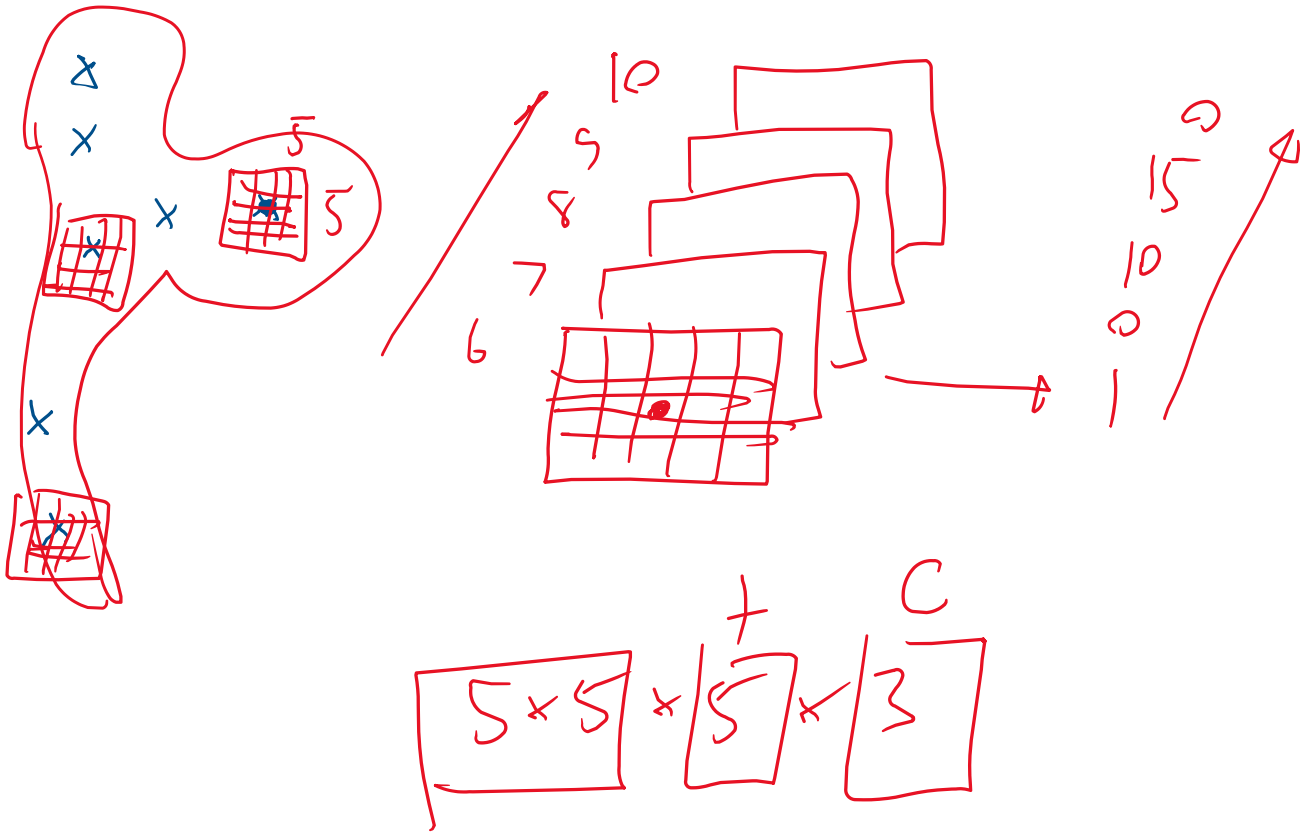
- Fully connected networks
  - SGD, backprop
- CNN
- RNN, LSTM, GRU





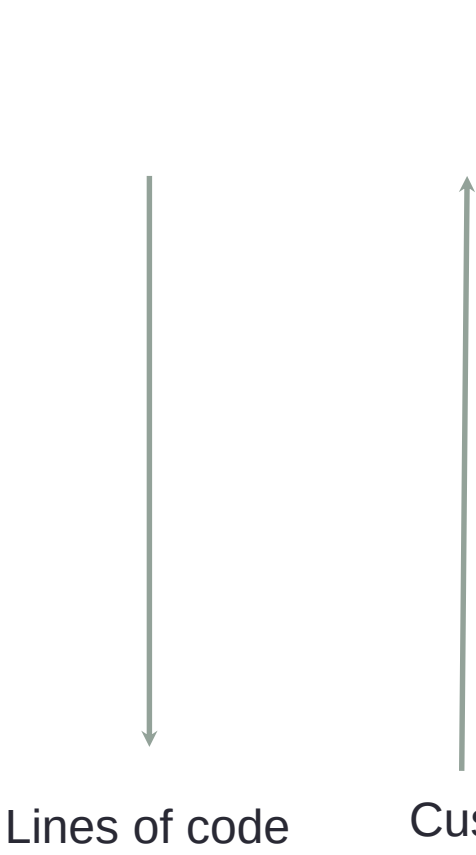


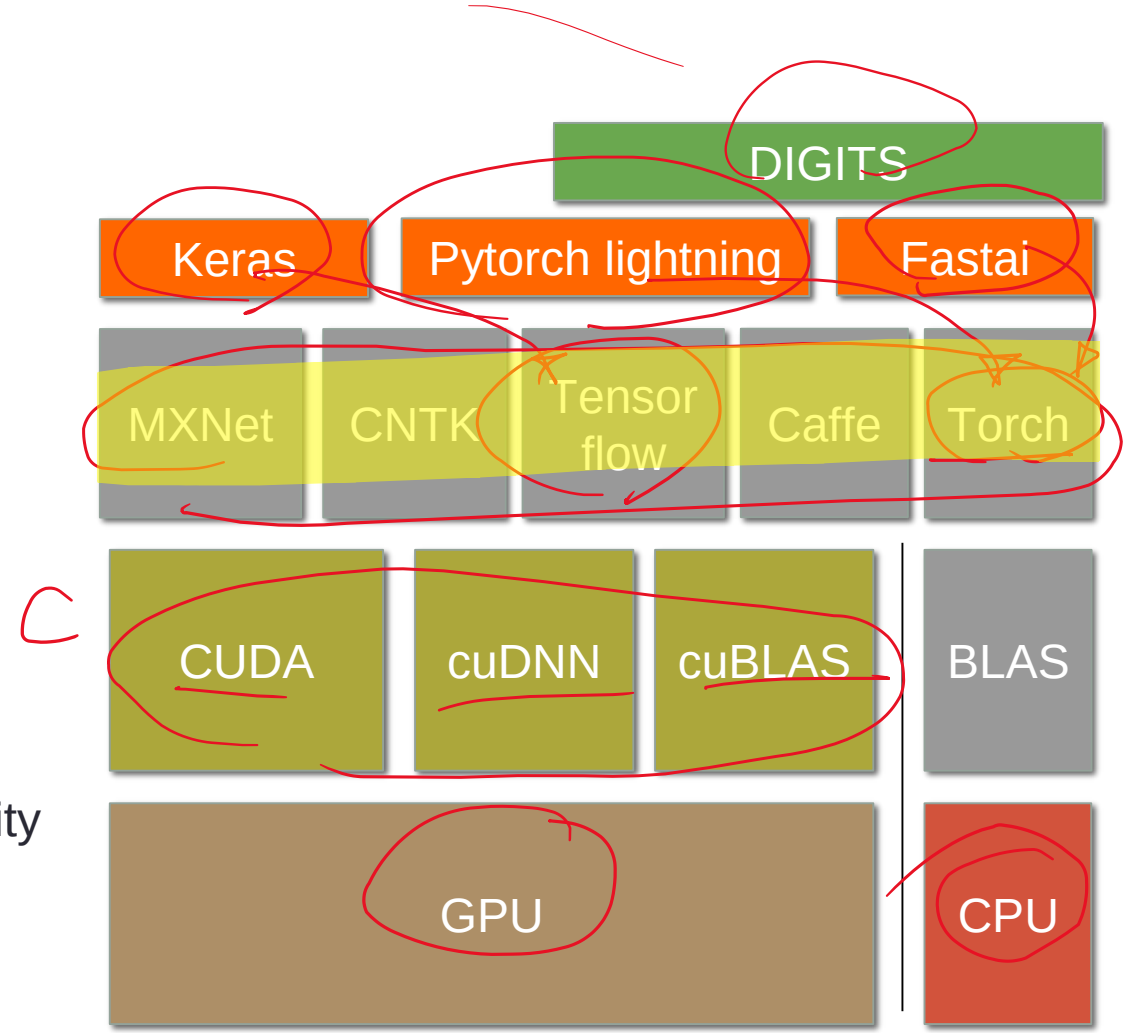
Generation ⟵ Next lecture


5
5
10
9
8
7
6
15
10
0
1
5 × 5 + 5 + 3
C

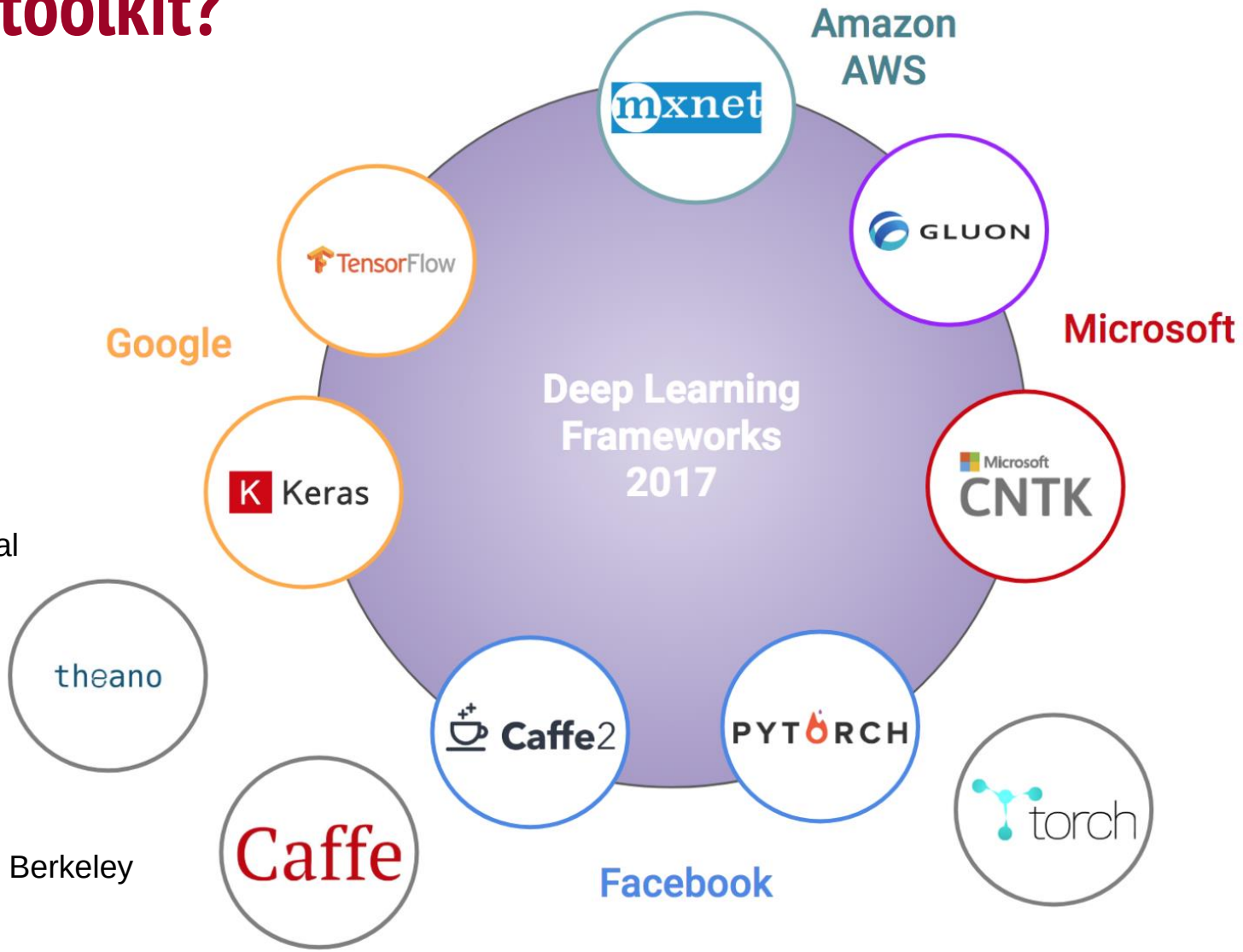
What toolkit

Tradeoff between customizability and ease of use



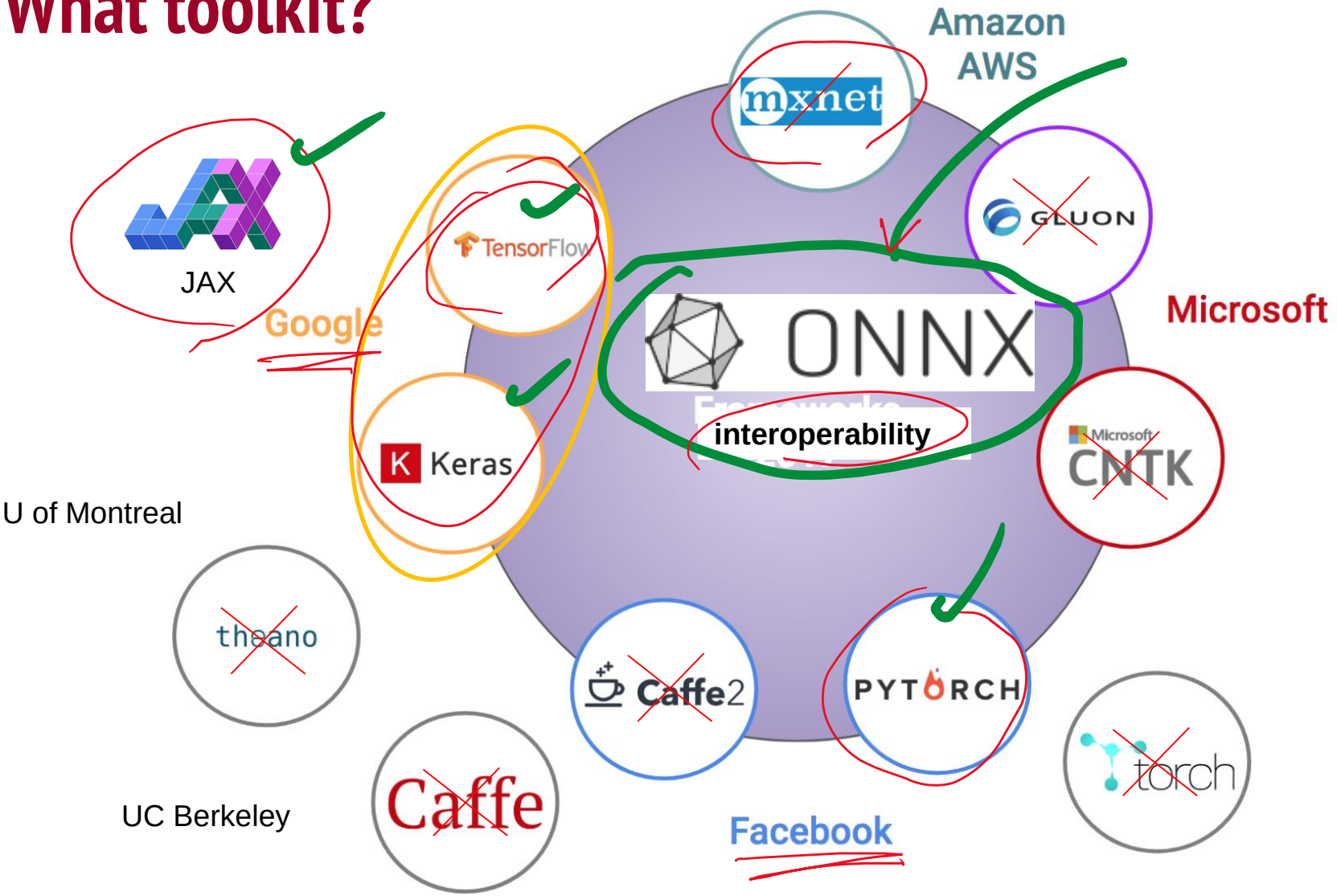


What toolkit?



https://towardsdatascience.com/battle-of-the-deep-learning-frameworks-part-i-cff0e3841750

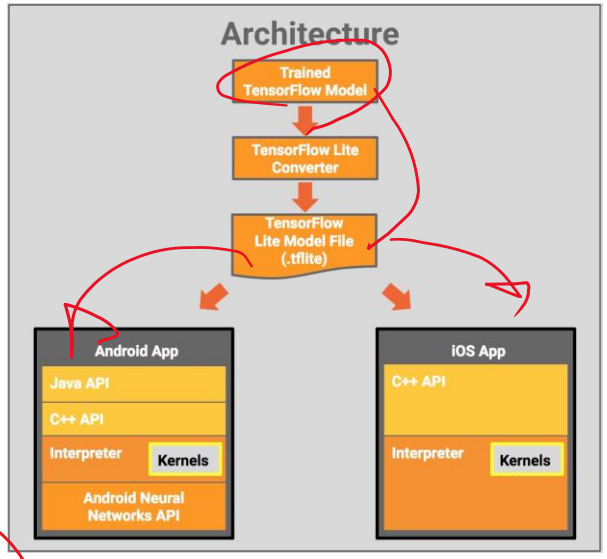
What toolkit?

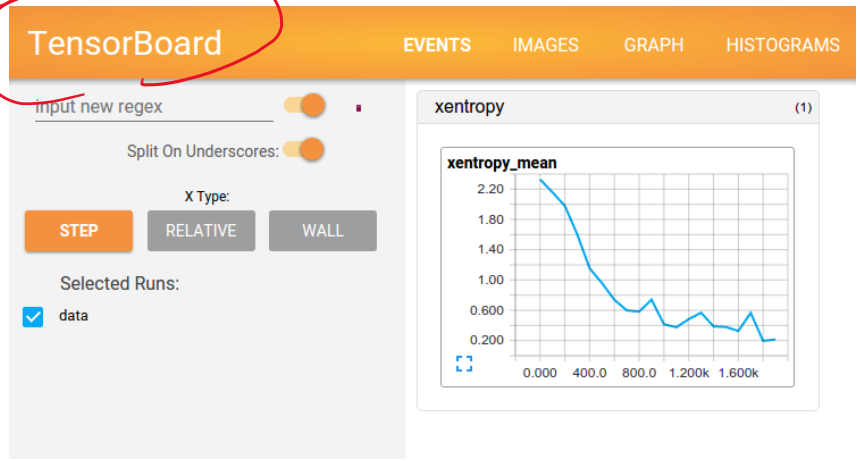


https://towardsdatascience.com/battle-of-the-deep-learning-frameworks-part-i-cff0e3841750

Which?

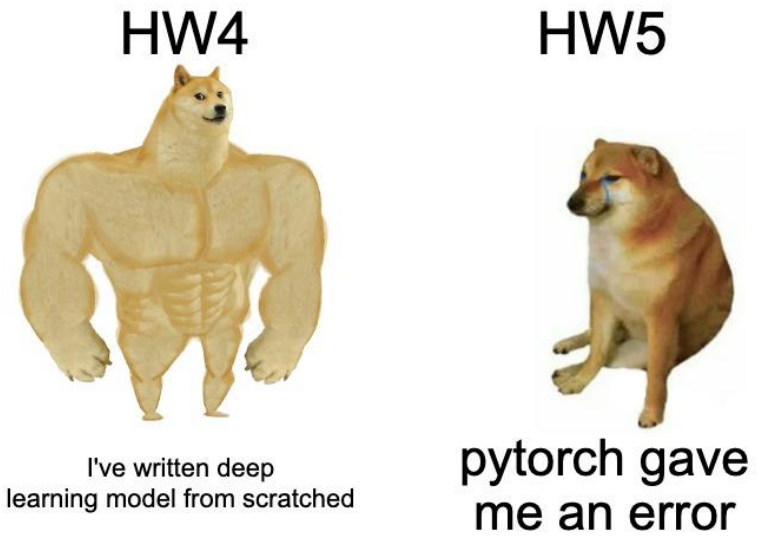
- Easiest to use and play with deep learning: Keras
- Easiest to use and tweak: pytorch
- Easiest to deploy: tensorflow
  - Tensorflow lite for mobile
  - TensorRT support
  - Javaruntime support
- Best tools: increasingly pytorch
- Community: increasingly pytorch
- In the end you should know some tf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



Debugging guide

- 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)