Inspire…Educate…Transform.

Convolution NNs

Dr. K. V. Dakshinamurthy
President, International School of Engineering

FC Networks

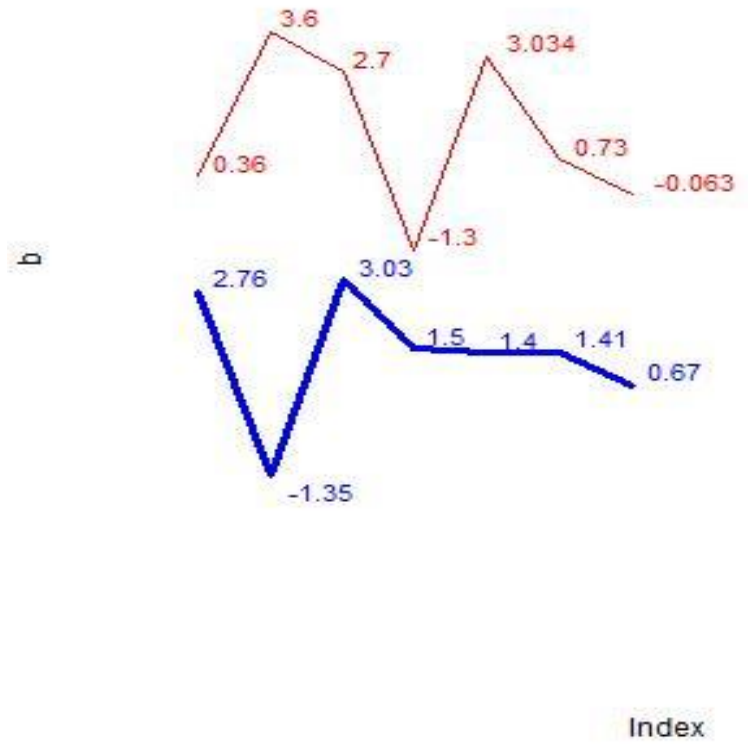
- Trillions of weights to learn in images
- Difficult to give context
- CNN and RNN are two ways out
- Let us do CNNs today

Convolution is a way out

- Let us understand the mathematics
- Then apply it to Neural nets

Convolution of two discrete functions

9.10



7.27

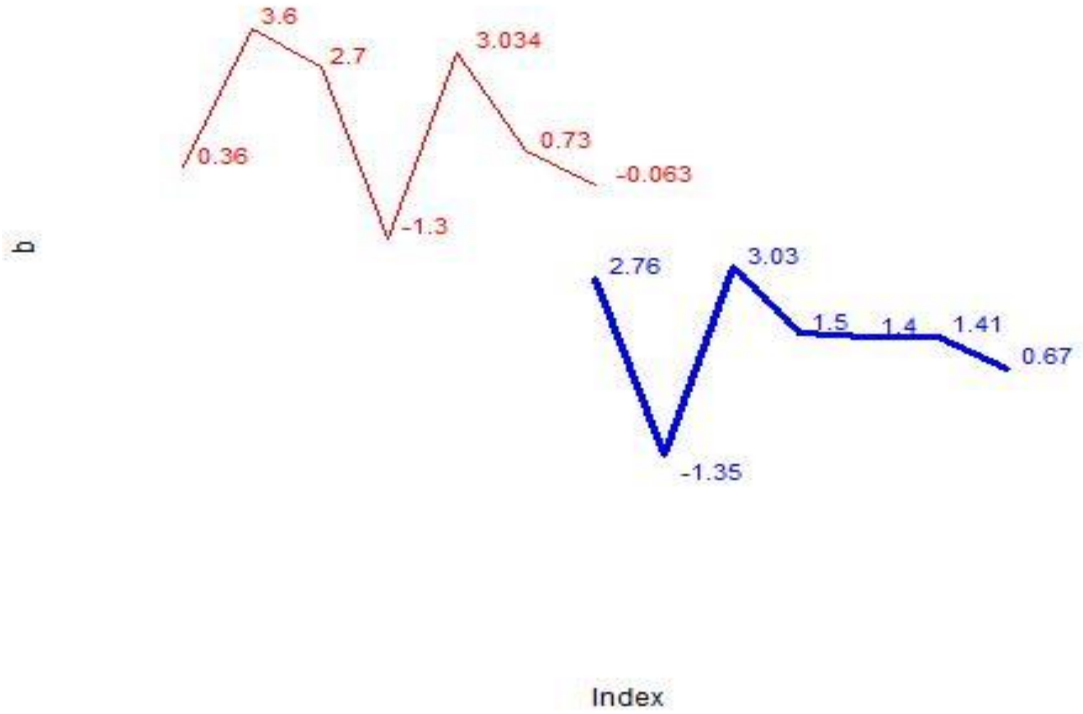
19.66

12.21

8.36

1.05

0.84



Formally convolution is defined as

$$s[t] = (x * w)(t) = \sum_{a=-\infty}^{\infty} x[a]w[t-a]$$

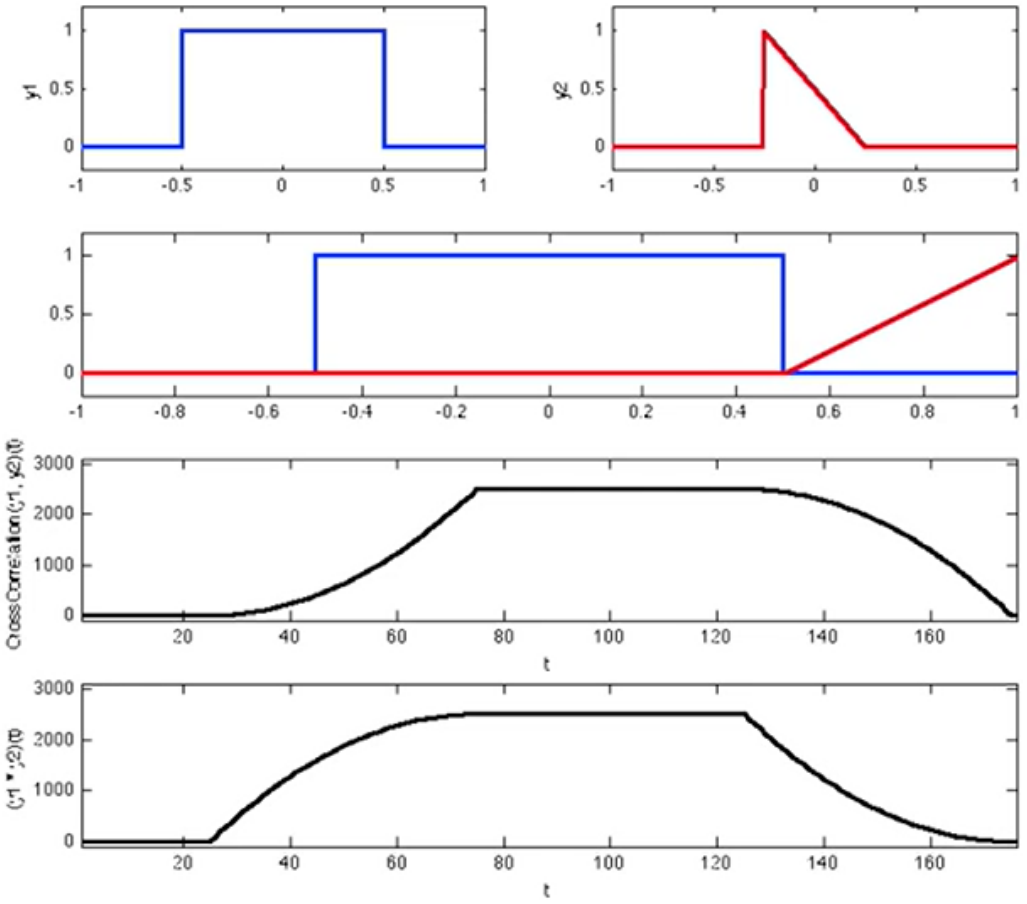
Convolution

$$s[i,j] = (I * K)[i,j] = \sum_m \sum_n I[i-m, j-n] K[m,n]$$

Cross-correlation,

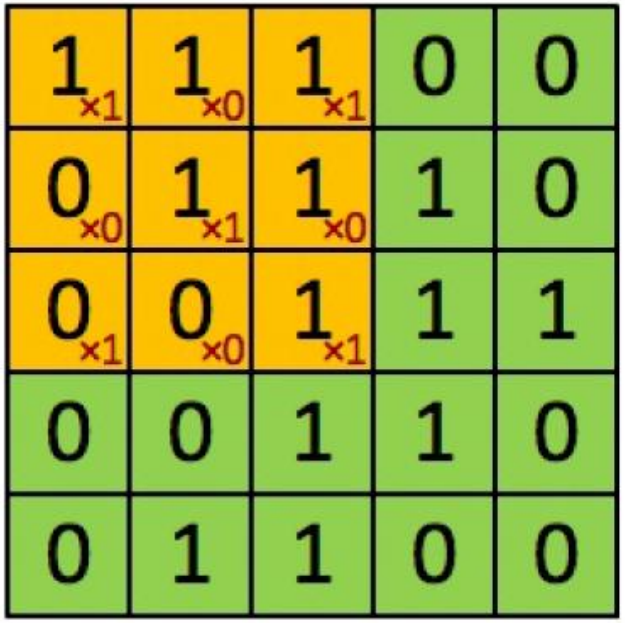
$$s[i,j] = (I * K)[i,j] = \sum_m \sum_n I[i+m, j+n] K[m,n]$$

Many machine learning libraries implement cross-correlation but call it convolution.



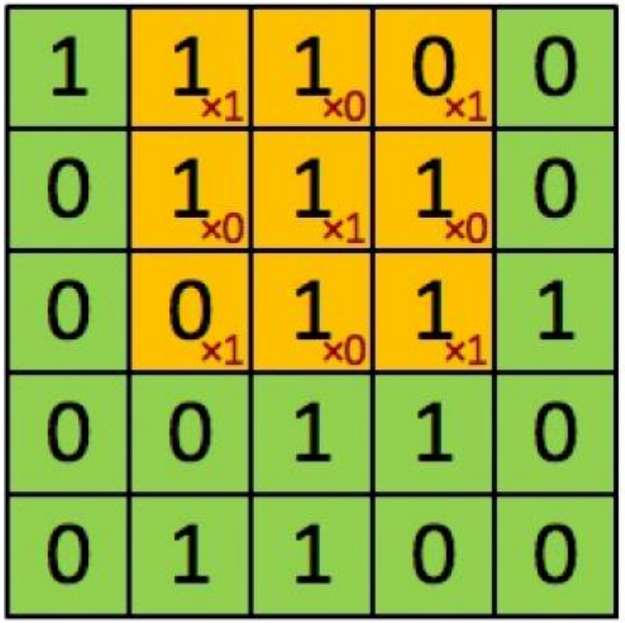
https://www.youtube.com/watch?v=MaOYONjMZLI

CONVOLUTION IN 2D

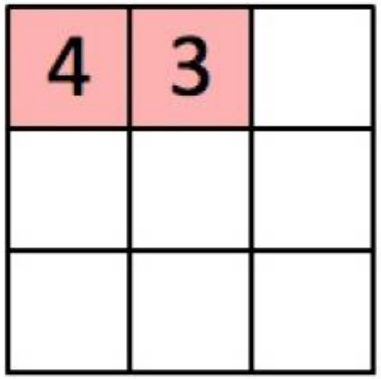


Image

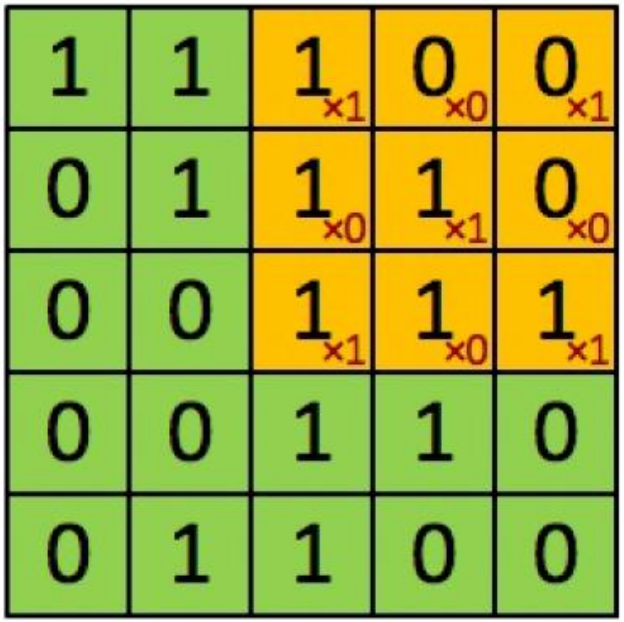
Convolved Feature



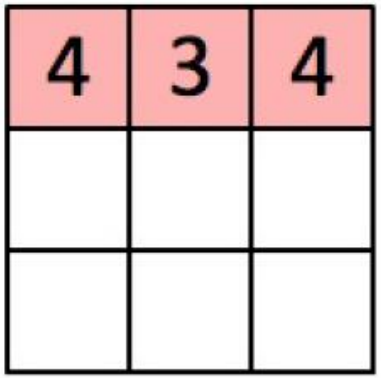
Image



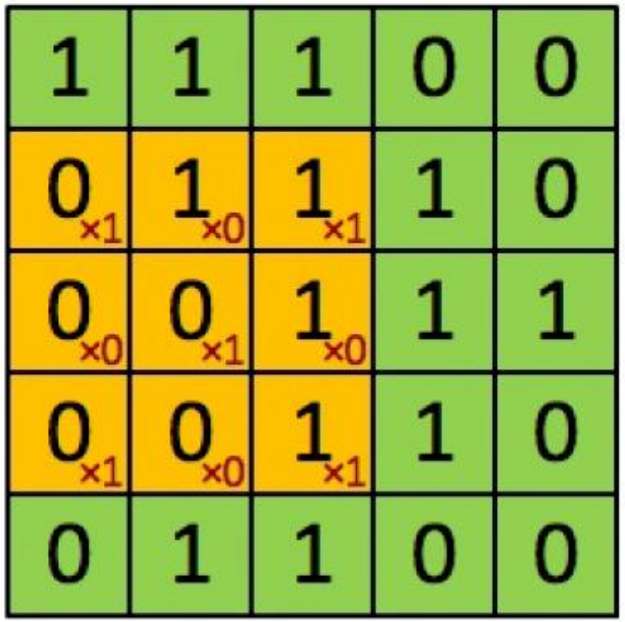
Convolved Feature

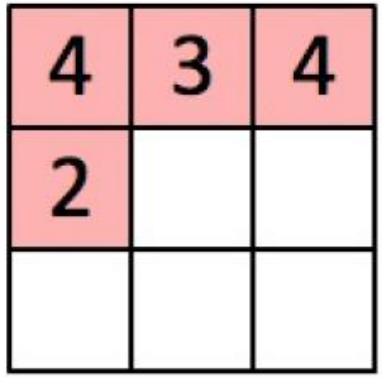


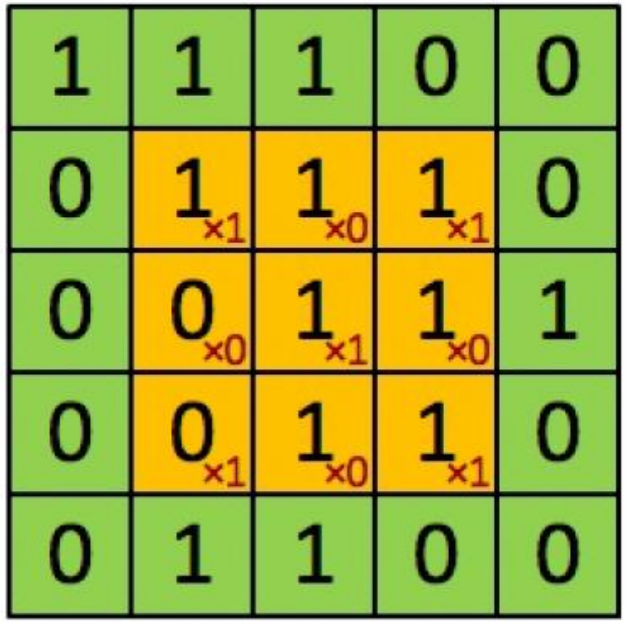
Image



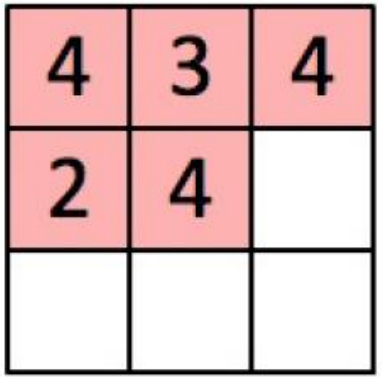
Convolved Feature


1 1 1 0 0
0 1 1 1 0
0 0 1 1 1
0 0 1 1 0
0 1 1 0 0
Image

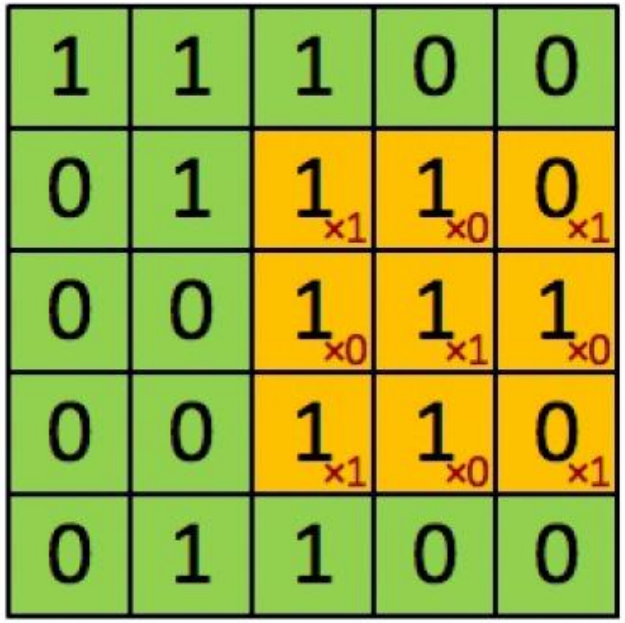

4 3 4
2
Convolved Feature



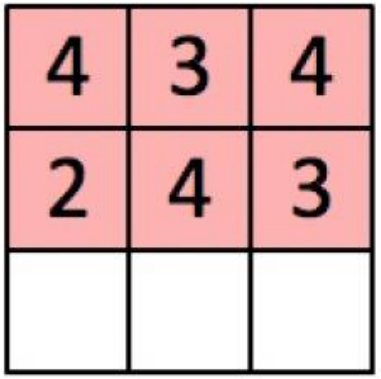
Image



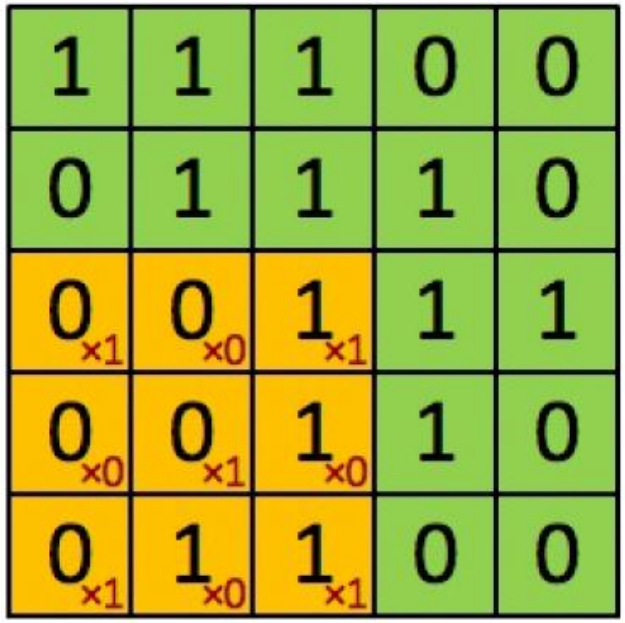
Convolved Feature



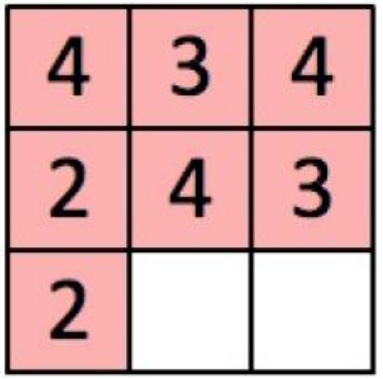
Image



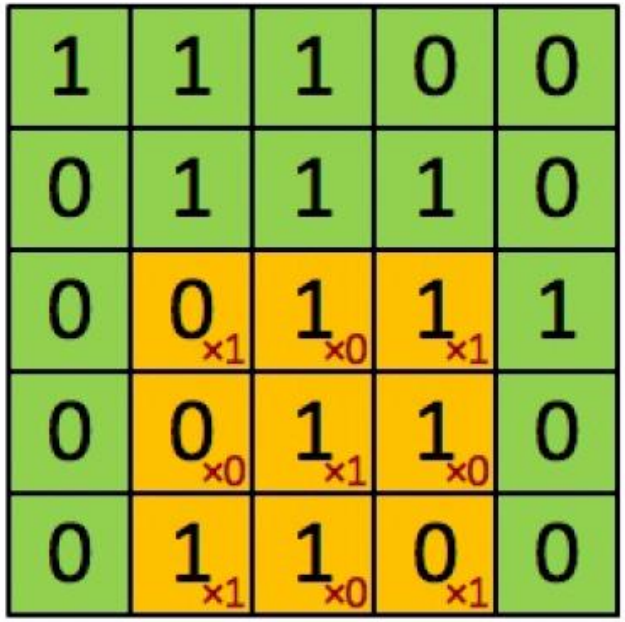
Convolved Feature



Image



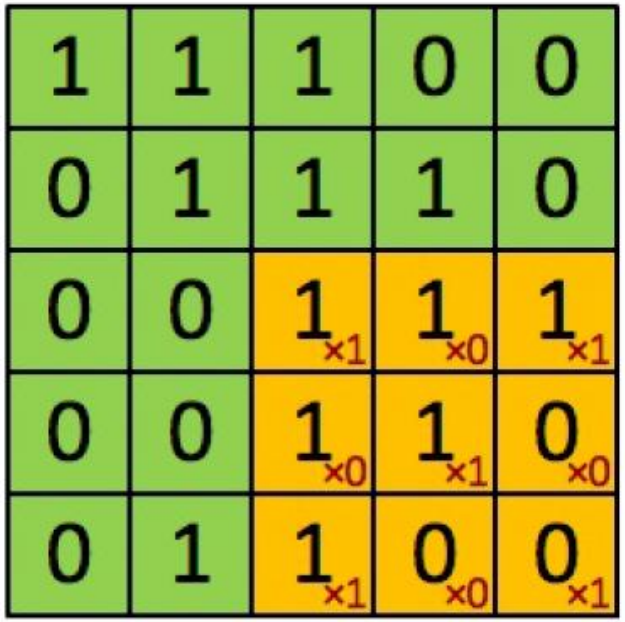
Convolved Feature



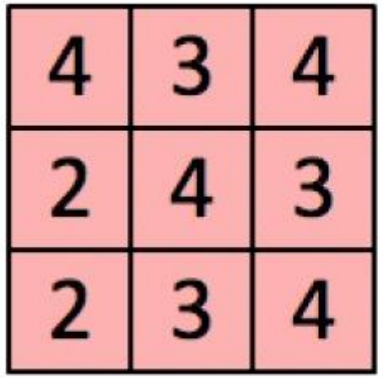
Image

4	3	4
2	4	3
2	3	

Convolved Feature



Image

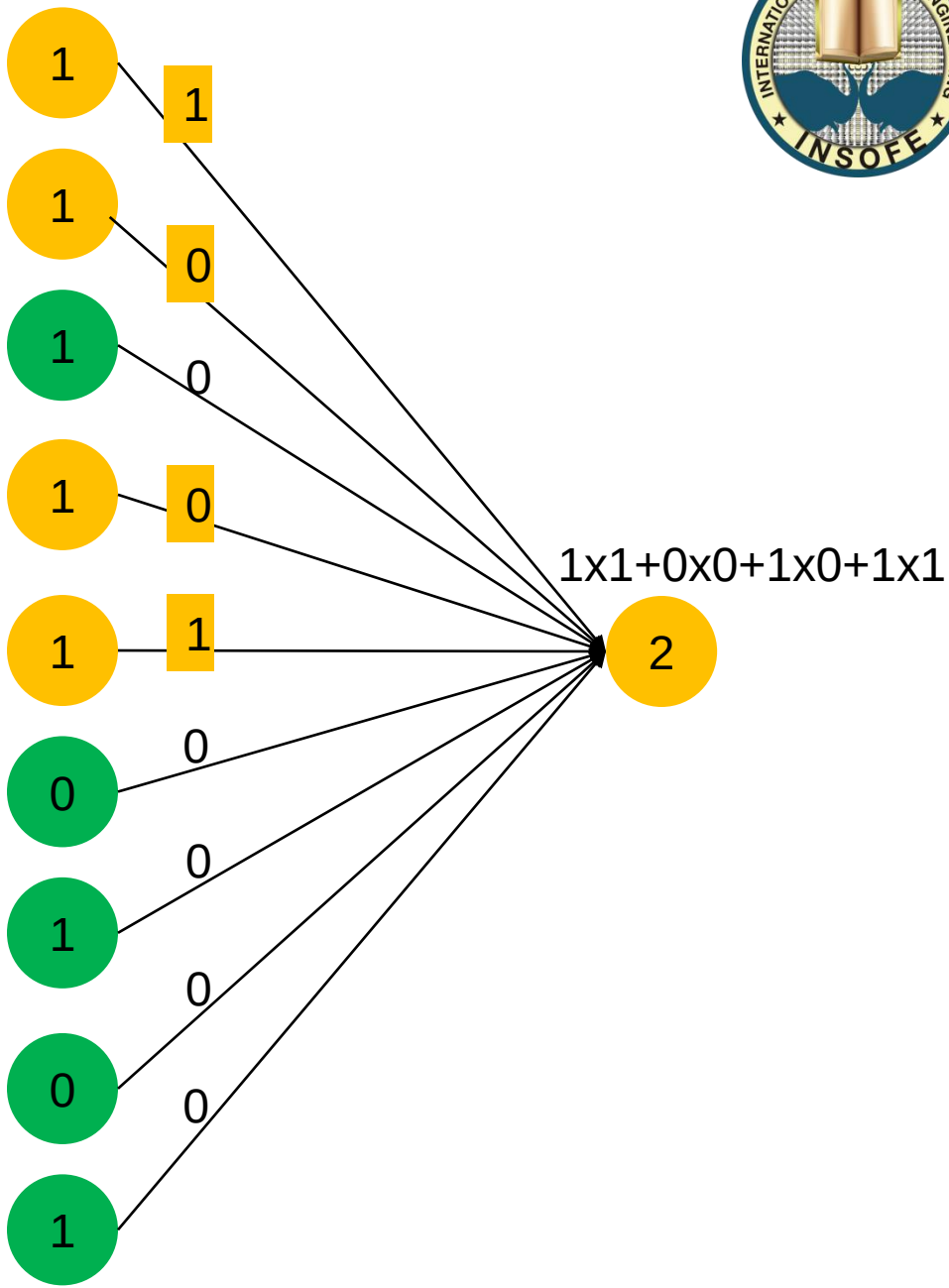


Convolved Feature

2D Convolution in 1D

1_{X1}	1_{X0}	1
1_{X0}	1_{X1}	0
1	0	1

2	



2D Convolution in 1D

1_{X1}	1_{X0}	1
1_{X0}	1_{X1}	0
1	0	1

2	

1	1_{X1}	1_{X0}
1	1_{X0}	0_{X1}
1	0	1

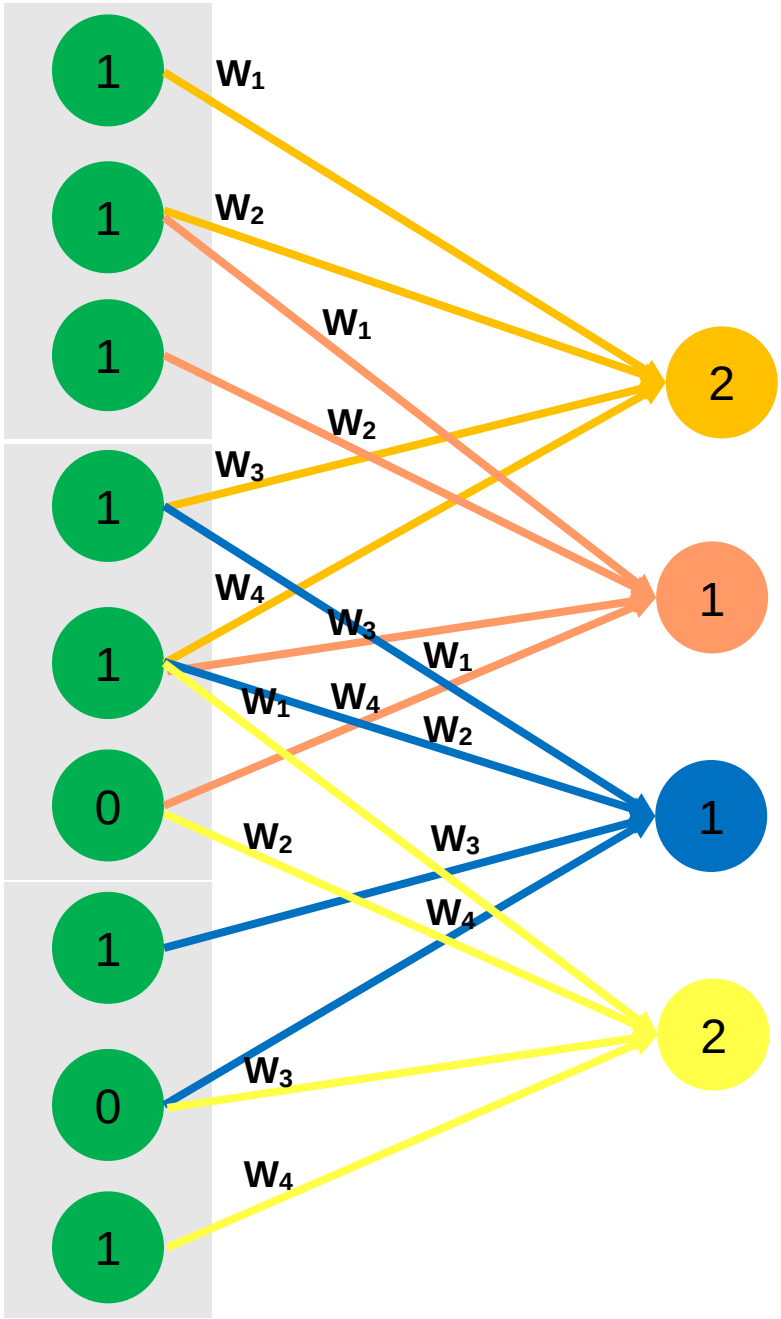
2	1

1	1	1
1_{X1}	1_{X0}	0
1_{X0}	0_{X1}	1

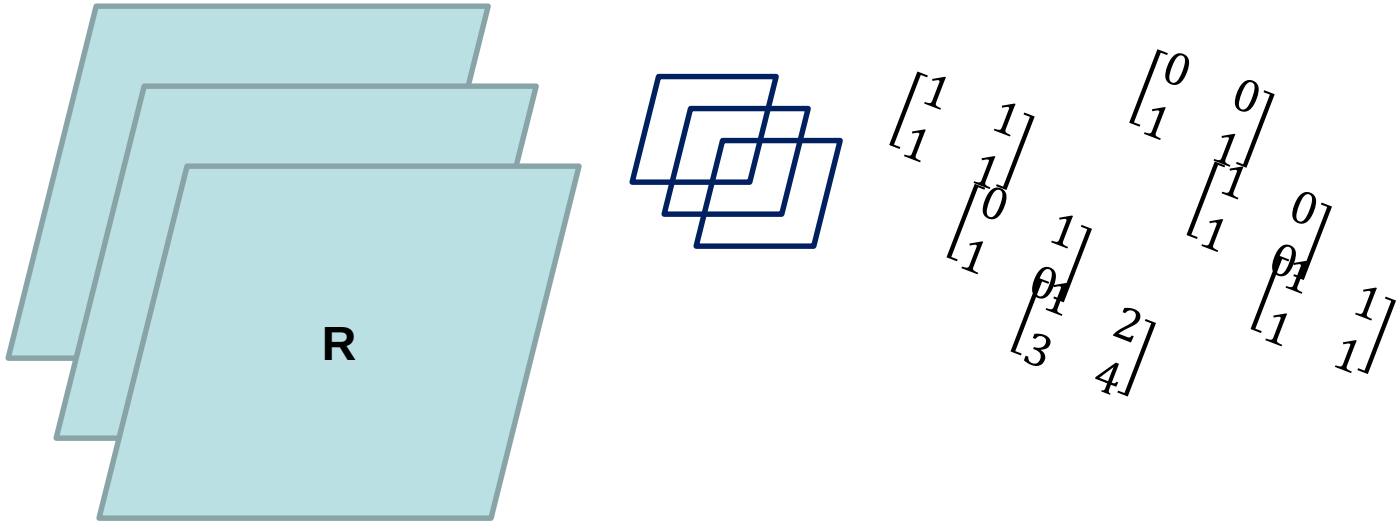
2	1
1	

1	1	1
1	1_{X1}	0_{X0}
1	0_{X0}	1_{X1}

2	1
1	2



Convolution in 3 layers



=(1*0)+(1*0)+(1*1)+(1*1)+
(0*1)+(1*0)+(1*1)+(0*0)+
(1*1)+(2*1)+(3*1)+(4*1)

= 13

Even a 3D convolution gives a 2D output

Convolution also engineers features

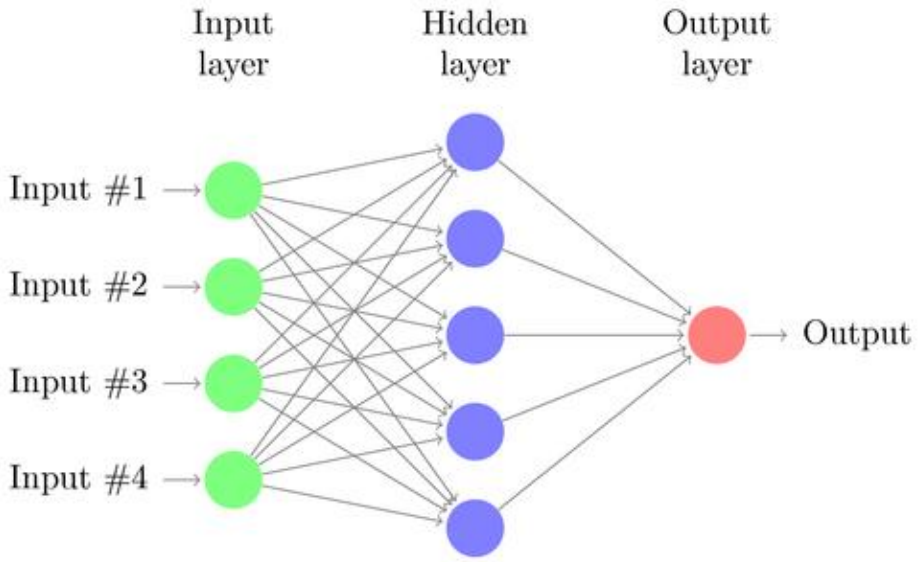
$$\begin{bmatrix} 1 & 1 & 1 \\ 0 & 0 & 0 \\ -1 & -1 & -1 \end{bmatrix}$$

$$\begin{bmatrix} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{bmatrix}$$

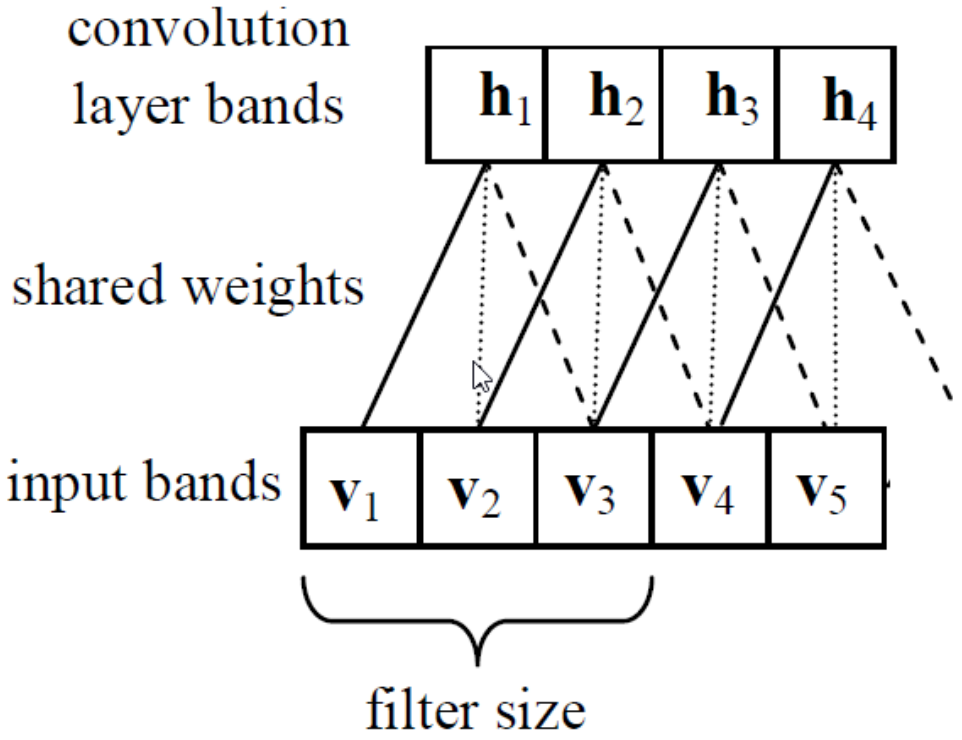
ADVANTAGES OF CONVOLUTION

Do autoencoders work for images?

- Number of weights to learn is drastically low in CNNs



Autoencoders

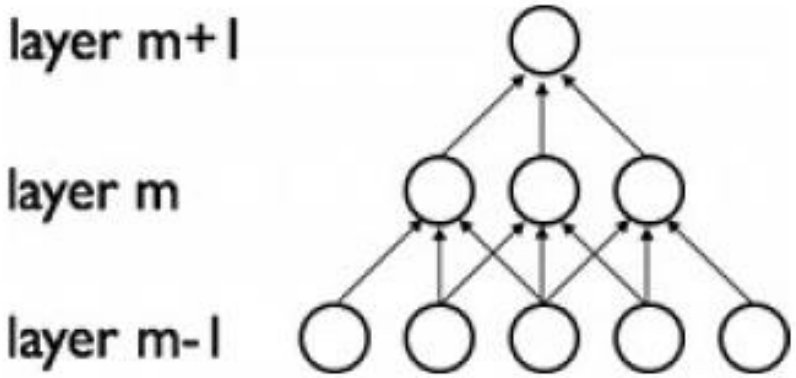


Why?

Regular Neural Nets don't scale well to full images. In CIFAR-10, images are only of size 32x32x3 (32 wide, 32 high, 3 color channels), so a single fully-connected neuron in a first hidden layer of a regular Neural Network would have 32*32*3 = 3072 weights. This amount still seems manageable, but clearly this fully-connected structure does not scale to larger images.

For example, an image of more respectable size, e.g. 200x200x3, would lead to neurons that have 200*200*3 = 120,000 weights. Moreover, we would almost certainly want to have several such neurons, so the parameters would add up quickly! Clearly, this full connectivity is wasteful and the huge number of parameters would quickly lead to overfitting.

Local connectivity

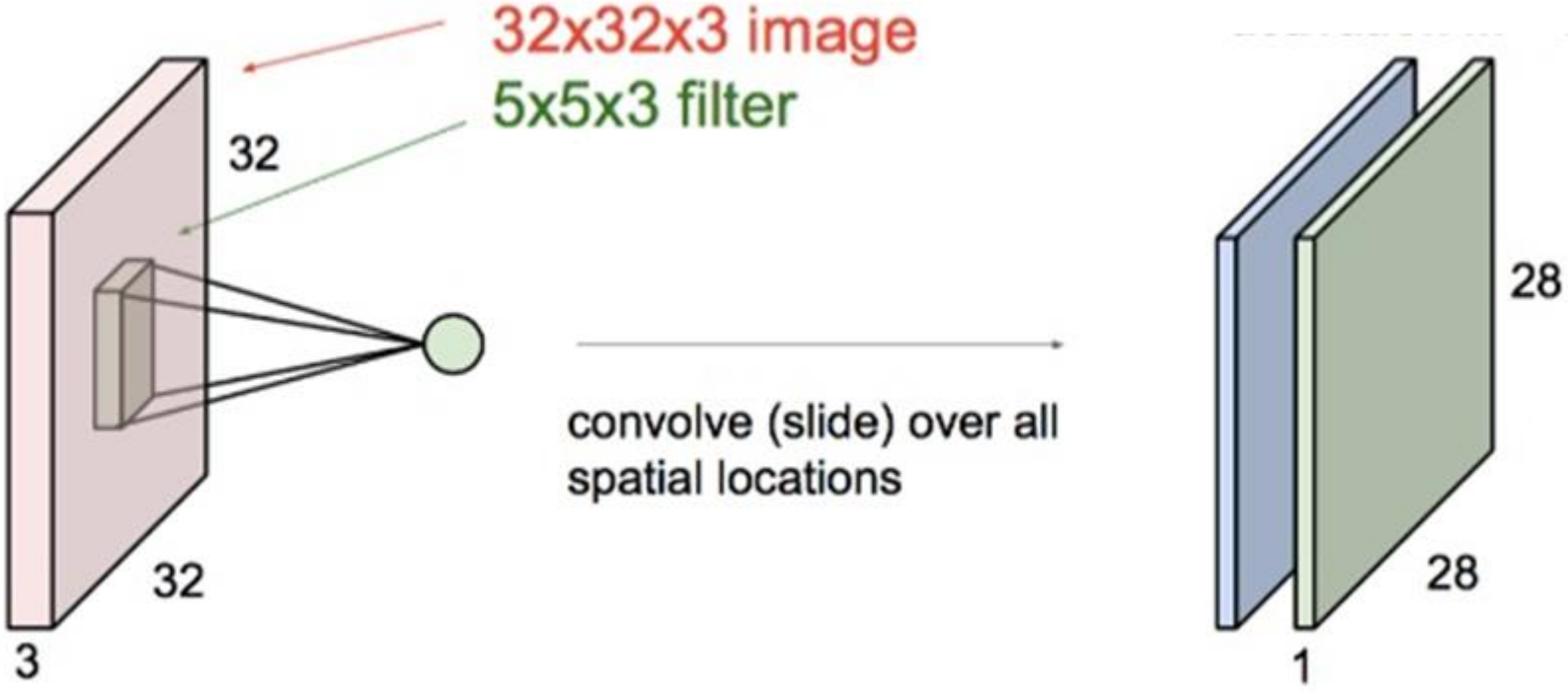


Imagine that layer m-1 is the input retina. In the above figure, units in layer m have receptive fields of width 3 in the input retina and are thus only connected to 3 adjacent neurons in the retina layer.

Units in layer m+1 have a similar connectivity with the layer below. We say that their receptive field with respect to the layer below is also 3, but their receptive field with respect to the input is larger (5).

Replicating units in this way allows for features to be detected regardless of their position in the visual field. Additionally, weight sharing increases learning efficiency by greatly reducing the number of free parameters being learnt. The constraints on the model enable CNNs to achieve better generalization on vision problems.

CNNs use convolution at least in one layer



While we decide on length and width, the depth of the connectivity is always equal to the depth of the input volume. The connections are local in space (along width and height), but always full along the entire depth of the input volume.

During the forward pass, we slide (more precisely, convolve) each filter across the width and height of the input volume and compute dot products between the entries of the filter and the input at any position.

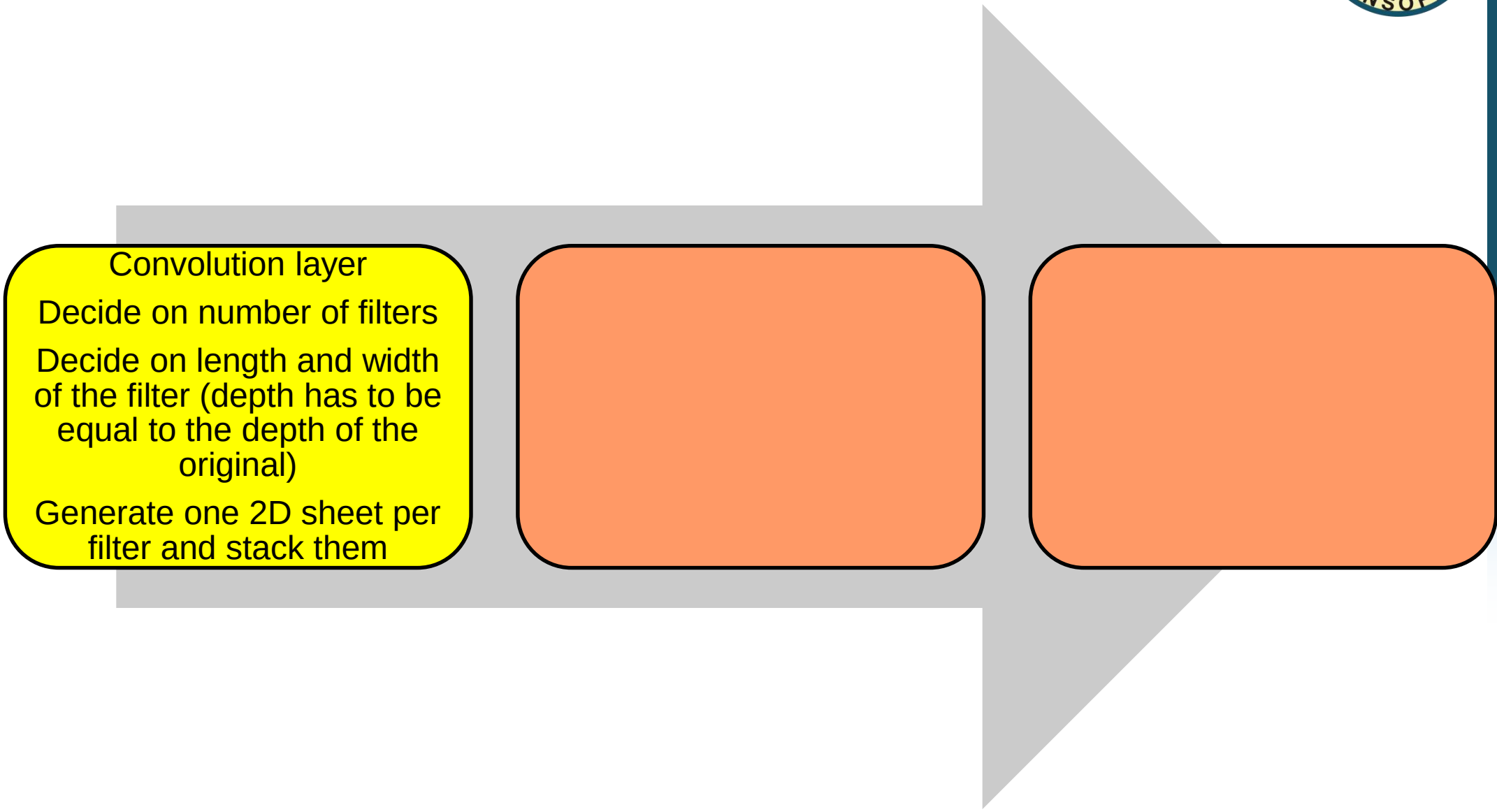
Suppose the input volume has size [32x32x3]. If the receptive field (or the filter size) is 5x5, then each neuron in the Conv Layer will have weights to a [5x5x3] region in the input volume, for a total of 5*5*3 = 75 weights (and +1 bias parameter). Notice that the extent of the connectivity along the depth axis must be 3, since this is the depth of the input volume.

Intuitively, the network will learn filters that activate when they see some type of visual feature such as an edge of some orientation or a blotch of some color on the first layer, or eventually entire honeycomb or wheel-like patterns on higher layers of the network.

As we slide the filter over the width and height of the input volume we will produce a 2-dimensional activation map that gives the responses of that filter at every spatial position.

Now, we will have an entire set of filters in each CONV layer (e.g. 12 filters), and each of them will produce a separate 2-dimensional activation map. We will stack these activation maps along the depth dimension and produce the output volume.

A simple CNN


Convolution layer
Decide on number of filters
Decide on length and width of the filter (depth has to be equal to the depth of the original)
Generate one 2D sheet per filter and stack them

Four hyperparameters control the size of the output volume

- We already discussed the depth (number of filters) and size (height and width)
- Stride: We must specify the stride with which we slide the filter. When the stride is 1 then we move the filters one pixel at a time. When the stride is 2 (or uncommonly 3 or more, though this is rare in practice) then the filters jump 2 pixels at a time as we slide them around. This will produce smaller output volumes spatially.
- Zero padding: Some times it will be convenient to pad the input volume with zeros around the border. The nice feature of zero padding is that it will allow us to control the spatial size of the output volumes. We will use it to exactly preserve the spatial size of the input volume so the input and output width and height are the same.

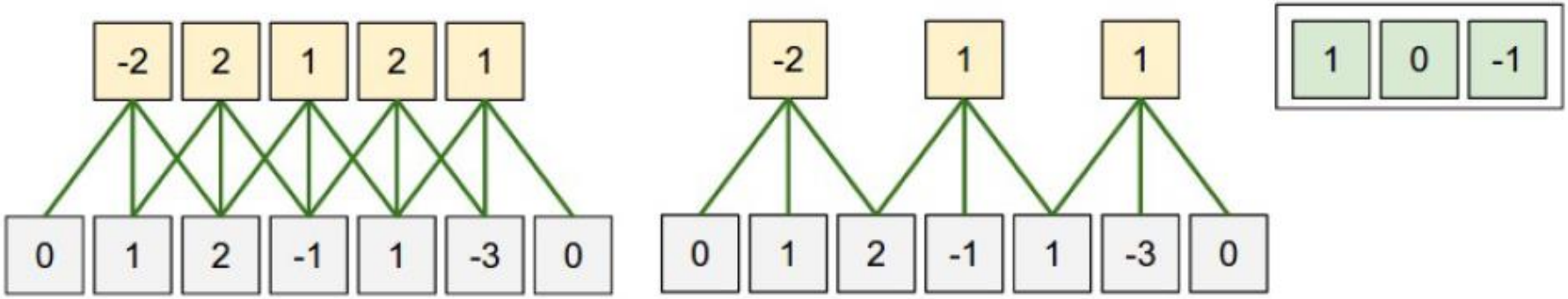


Illustration of spatial arrangement. In this example there is only one spatial dimension (x-axis), one neuron with a receptive field size of F = 3, the input size is W = 5, and there is zero padding of P = 1. Left: The neuron strided across the input in stride of S = 1, giving output of size (5 - 3 + 2)/1+1 = 5. Right: The neuron uses stride of S = 2, giving output of size (5 - 3 + 2)/2+1 = 3. Notice that stride S = 3 could not be used since it wouldn't fit neatly across the volume. In terms of the equation, this can be determined since (5 - 3 + 2) = 4 is not divisible by 3.

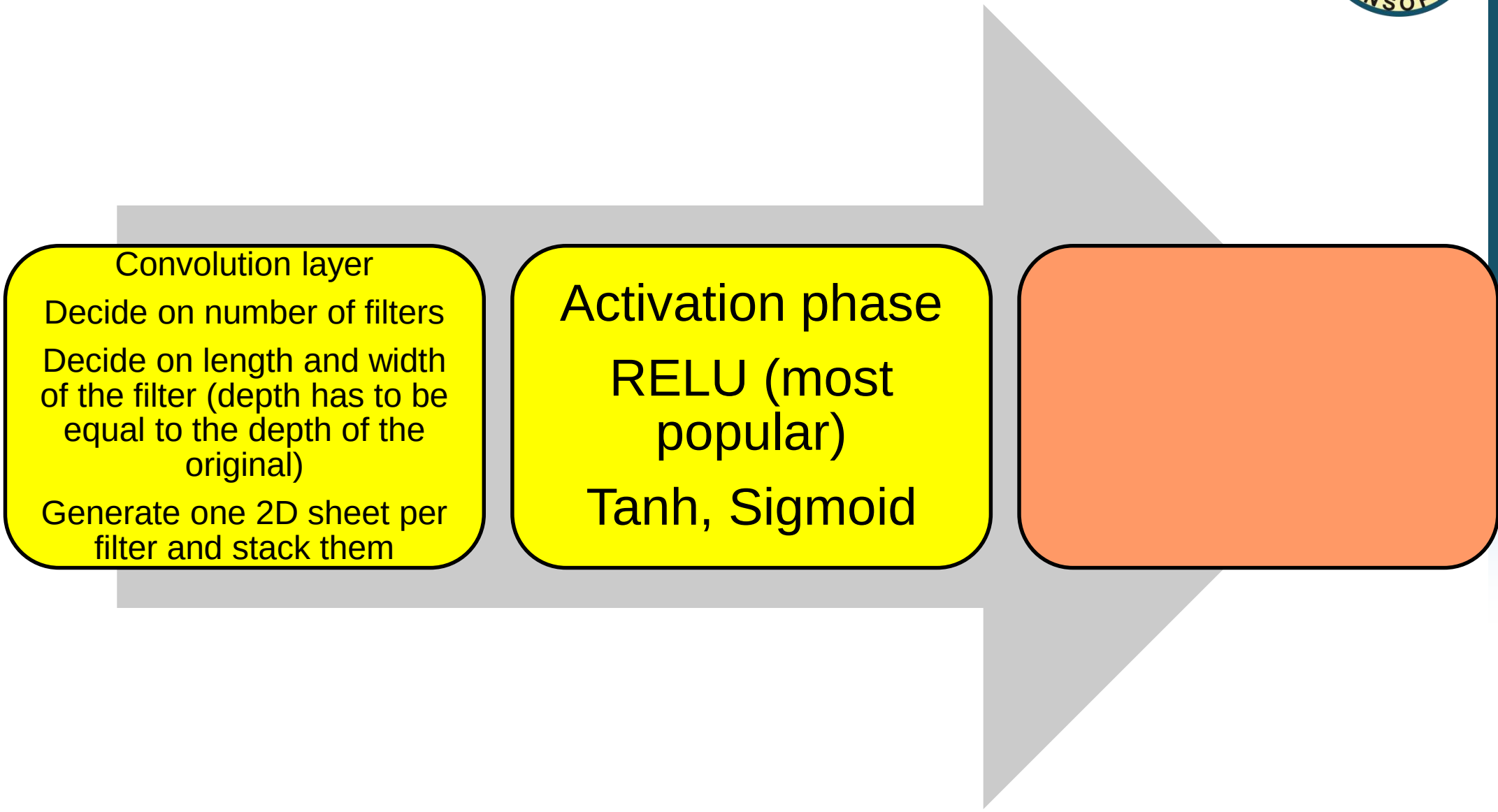
The neuron weights are in this example [1,0,-1] (shown on very right), and its bias is zero. These weights are shared across all yellow neurons (see parameter sharing below).

Convolution Layer

- Accepts a volume of size $W_1 \times H_1 \times D_1$
- Requires four hyperparameters:
  - Number of filters K,
  - their spatial extent F,
  - the stride S,
  - the amount of zero padding P.
- Produces a volume of size $W_2 \times H_2 \times D_2$ where:
  - $W_2 = (W_1 - F + 2P)/S + 1$
  - $H_2 = (H_1 - F + 2P)/S + 1$ (i.e. width and height are computed equally by symmetry)
  - $D_2 = K$
- With parameter sharing, it introduces $F \cdot F \cdot D_1$ weights per filter, for a total of $(F \cdot F \cdot D_1) \cdot K$ weights and K biases.
- In the output volume, the d-th depth slice (of size $W_2 \times H_2$) is the result of performing a valid convolution of the d-th filter over the input volume with a stride of S, and then offset by d-th bias.

http://cs231n.github.io/convolutional-networks/

A simple CNN


Convolution layer
Decide on number of filters
Decide on length and width of the filter (depth has to be equal to the depth of the original)
Generate one 2D sheet per filter and stack them
Activation phase
RELU (most popular)
Tanh, Sigmoid

Rectified Linear Units (RELU)

In the context of artificial neural networks, the **rectifier** is an activation function defined as

$$f(x) = \max(0, x),$$

where *x* is the input to a neuron. This is also known as a ramp function and is analogous to half-wave rectification in electrical engineering. This activation function has been argued to be more biologically plausible[1] than the widely used logistic sigmoid (which is inspired by probability theory; see logistic regression) and its more practical[2] counterpart, the hyperbolic tangent. The rectifier is, as of 2015, the most popular activation function for deep neural networks.[3]

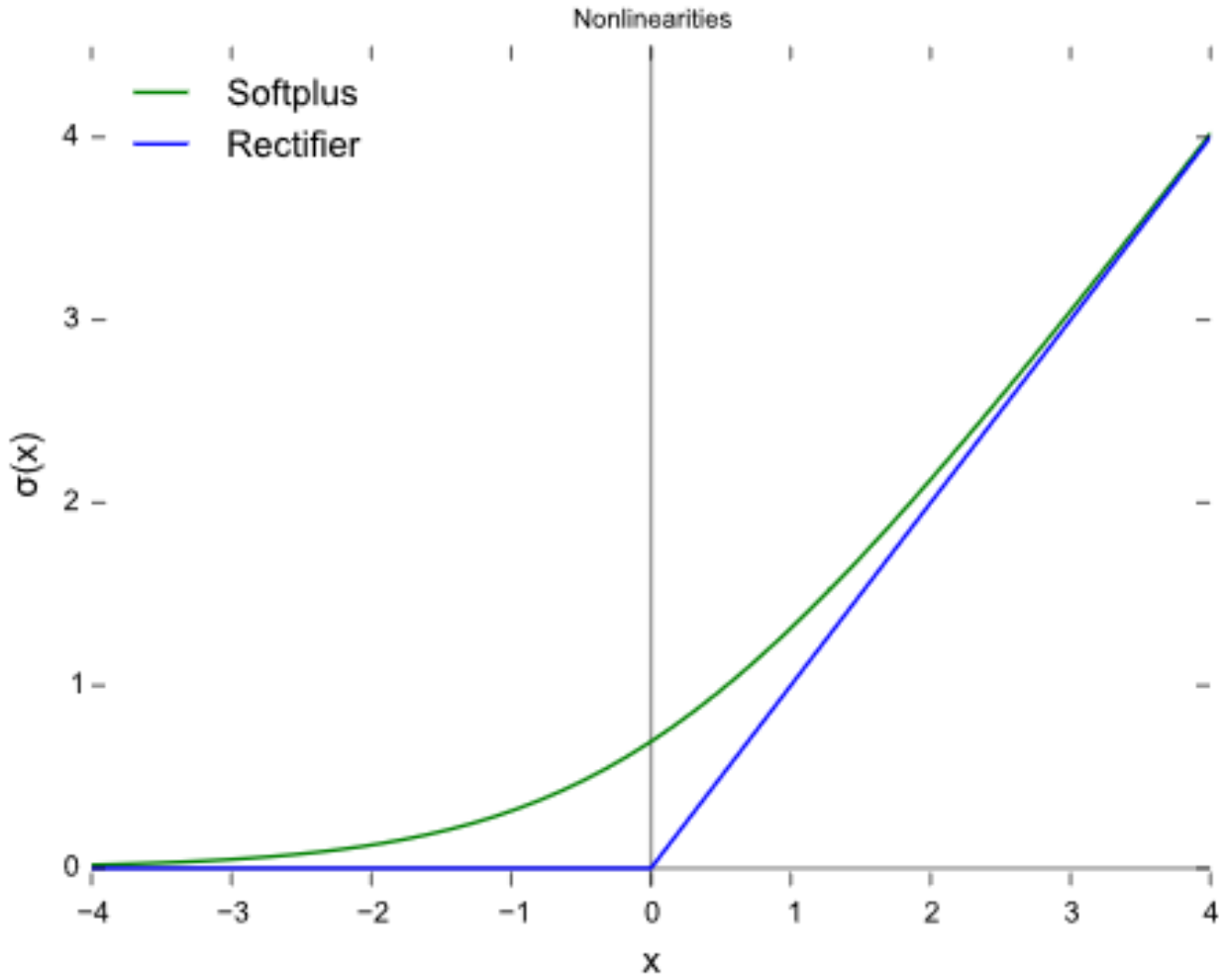
A unit employing the rectifier is also called a **rectified linear unit** (**ReLU**).[4]

A smooth approximation to the rectifier is the analytic function

$$f(x) = \ln(1 + e^x),$$

which is called the **softplus** function.[5] The derivative of softplus is $f'(x) = e^x/(e^x + 1) = 1/(1 + e^{-x})$, i.e. the logistic function.

Rectified linear units find applications in computer vision[1] and speech recognition[6][7] using deep neural nets.


Nonlinearities
Softplus
Rectifier
σ(x)
4
3
2
1
0
−4
−3
−2
−1
0
1
2
3
4
x

Noisy ReLUs [edit]

Rectified linear units can be extended to include Gaussian noise, making them noisy ReLUs, giving[4]

$$f(x) = \max(0, x + Y), \text{ with } Y \sim \mathcal{N}(0, \sigma(x))$$

Noisy ReLUs have been used with some success in restricted Boltzmann machines for computer vision tasks.[4]

Leaky ReLUs [edit]

Leaky ReLUs allow a small, non-zero gradient when the unit is not active.[7]

$$f(x) = \begin{cases} x & \text{if } x > 0 \\ 0.01x & \text{otherwise} \end{cases}$$

Parametric ReLUs take this idea further by making the coefficient of leakage into a parameter that is learned along with the other neural network parameters.[8]

$$f(x) = \begin{cases} x & \text{if } x > 0 \\ ax & \text{otherwise} \end{cases}$$

Note that for $a < 0$, this is equivalent to

$$f(x) = \max(x, ax)$$

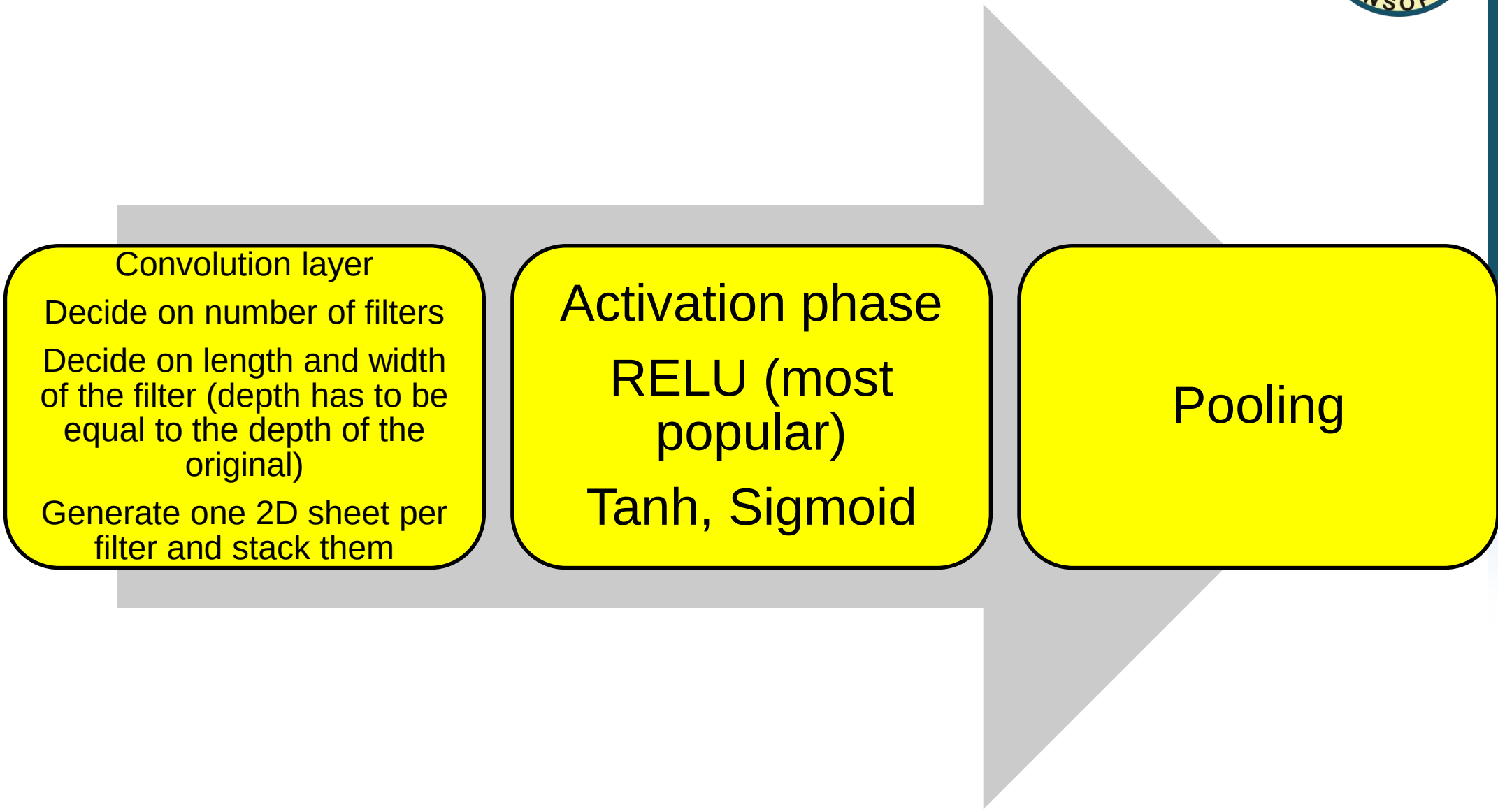
and thus has a relation to "maxout" networks.[8]

Advantages

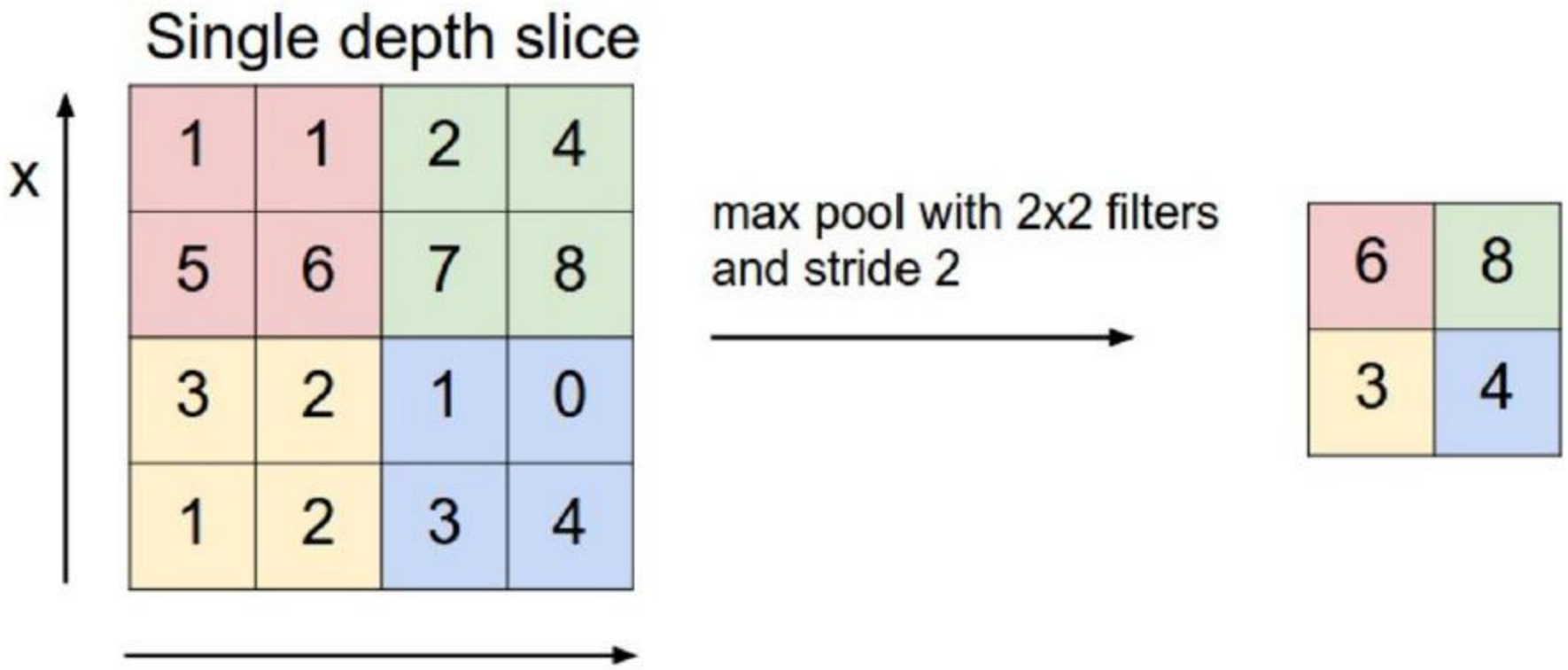
Biological plausibility: One-sided, compared to the antisymmetry of tanh.
Sparse activation: For example, in a randomly initialized network, only about 50% of hidden units are activated (having a non-zero output).
Efficient gradient propagation: No vanishing gradient problem or exploding effect.
Efficient computation: Only comparison, addition and multiplication.

For the first time in 2011,[1] the use of the rectifier as a non-linearity has been shown to enable training deep supervised neural networks without requiring unsupervised pre-training. Rectified linear units, compared to sigmoid function or similar activation functions, allow for faster and effective training of deep neural architectures on large and complex datasets.

A simple CNN


Convolution layer
Decide on number of filters
Decide on length and width of the filter (depth has to be equal to the depth of the original)
Generate one 2D sheet per filter and stack them
Activation phase
RELU (most popular)
Tanh, Sigmoid
Pooling

Max pooling

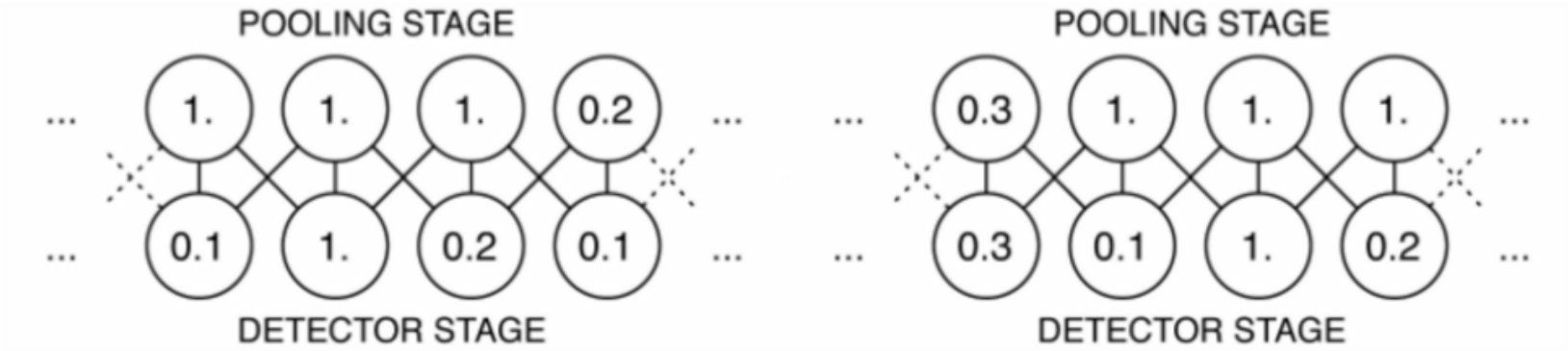


The average of a rectangular neighborhood.

The L2 norm of a rectangular neighborhood.

A weighted average based on the distance from the central pixel.

pooling helps to make the representation become invariant to small translations of the input.



In Right panel, the input has been shifted to the right by 1 pixel. Every value in the bottom row has changed, but only half of the values in the top row have changed.

Invariance to local translation can be a very useful property if we care more about whether some feature is present than exactly where it is.

For example: In a face, we need not know the exact location of the eyes.

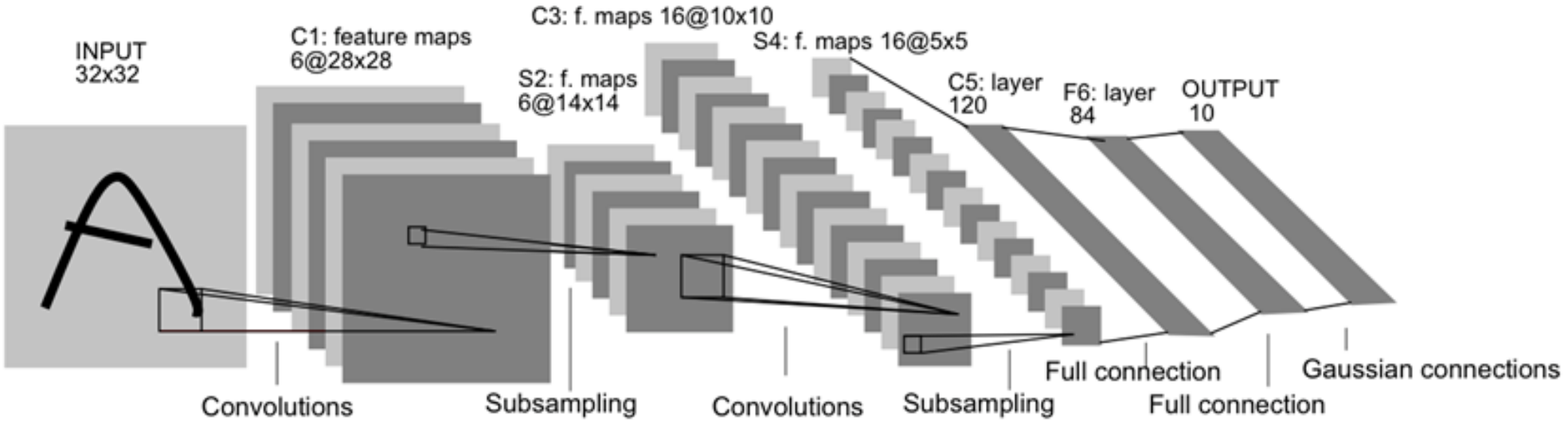
Why do CNNs work?

- The networks can be large and hence bias is minimized

- The weights are mostly zero because of convolution. RELU and dropout makes even fewer weights. Hence, variance is minimized

CNN ARCHITECTURES

- INPUT [32x32x3] will hold the raw pixel values of the image, in this case an image of width 32, height 32, and with three color channels R,G,B.
- CONV layer will compute the output of neurons that are connected to local regions in the input, each computing a dot product between their weights and a small region they are connected to in the input volume. This may result in volume such as [32x32x12] if we decided to use 12 filters.
- RELU layer will apply an elementwise activation function, such as the $max(0, x)$ thresholding at zero. This leaves the size of the volume unchanged ([32x32x12]).
- POOL layer will perform a downsampling operation along the spatial dimensions (width, height), resulting in volume such as [16x16x12].
- FC (i.e. fully-connected) layer will compute the class scores, resulting in volume of size [1x1x10], where each of the 10 numbers correspond to a class score, such as among the 10 categories of CIFAR-10. As with ordinary Neural Networks and as the name implies, each neuron in this layer will be connected to all the numbers in the previous volume.



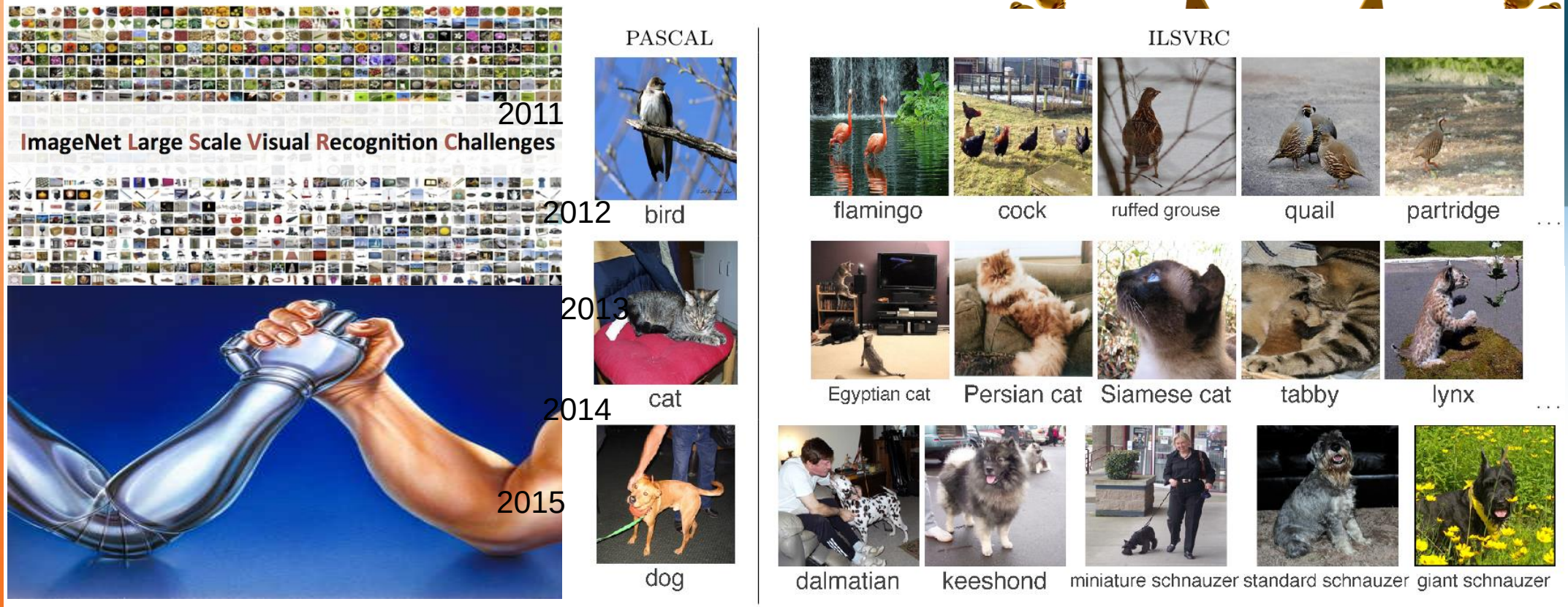
An early (Le-Net5) Convolutional Neural Network design, LeNet-5, used for recognition of digits

MODERN AGE ARCHITECTING

Deep learning and Bayesian graphical models

- The most accurate models for solving the last riddle

  ILSVRC (ImageNet Large-Scale Visual Recognition Challenge)

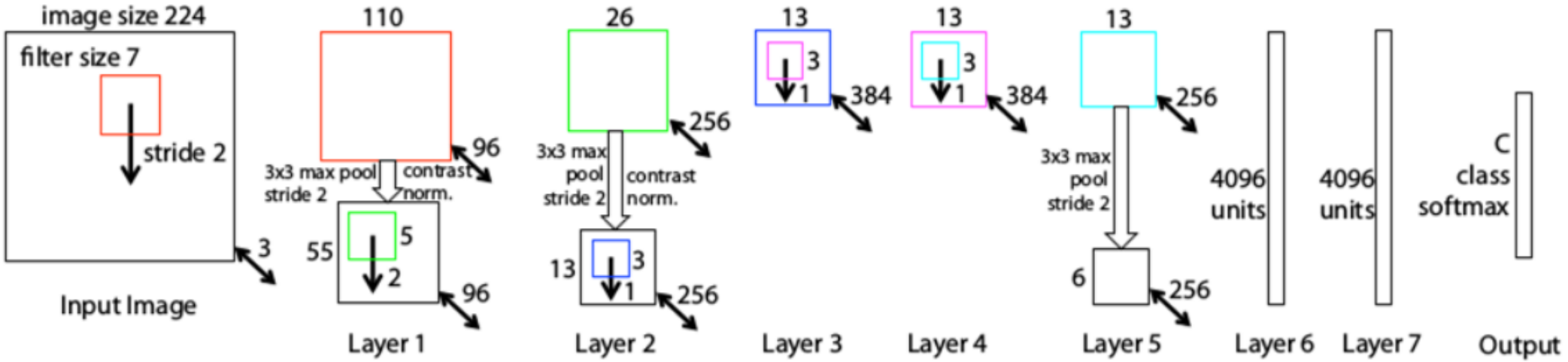


Alex Net

- They used a relatively simple layout, compared to modern architectures. The network was made up of 5 conv layers, max-pooling layers, dropout layers, and 3 fully connected layers. The network they designed was used for classification with 1000 possible categories.

- 2012 marked the first year where a CNN was used to achieve a top 5 test error rate of 15.4% (Top 5 error is the rate at which, given an image, the model does not output the correct label with its top 5 predictions). The next best entry achieved an error of 26.2%, which was an astounding improvement that pretty much shocked the computer vision community.

ZF net



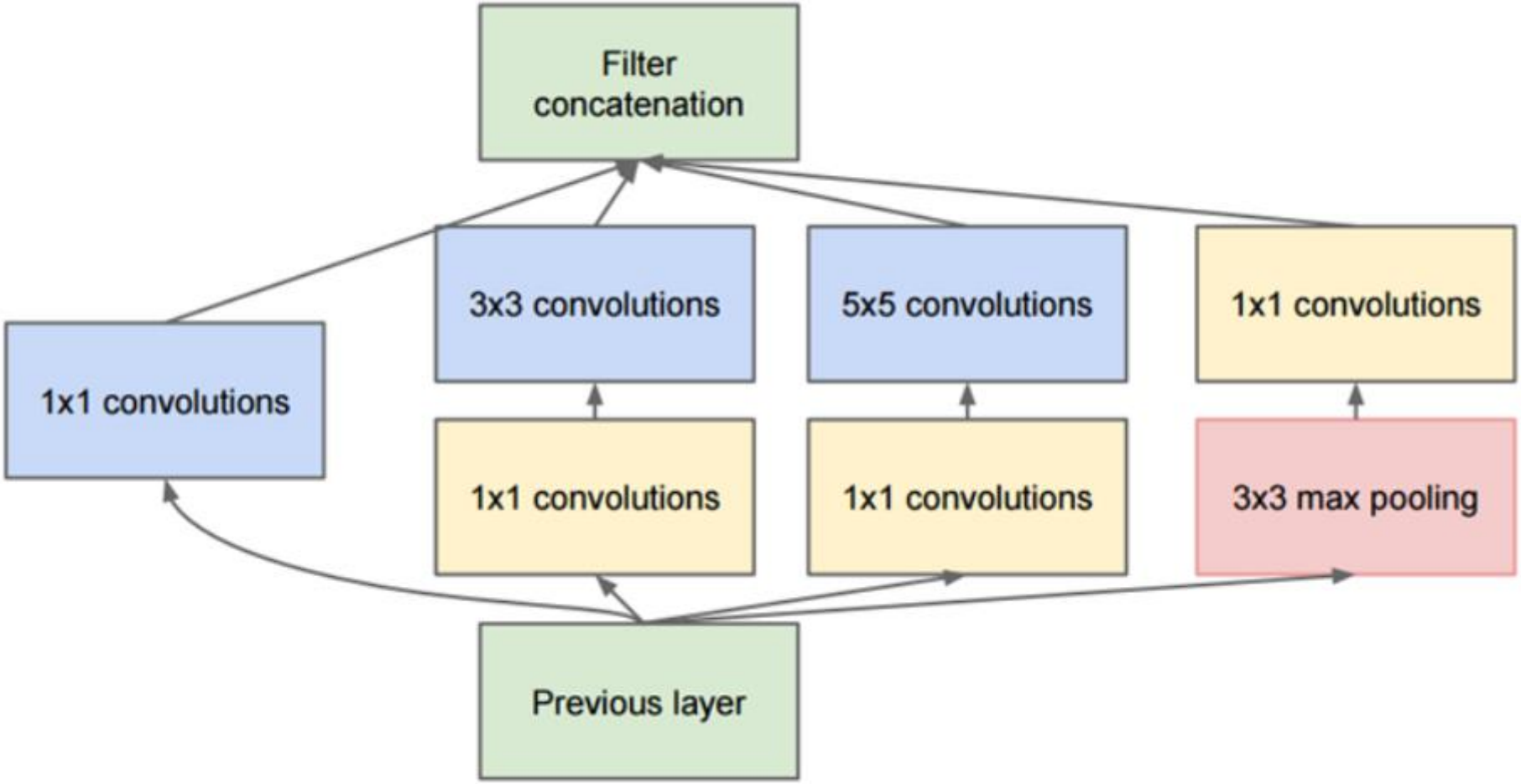
11.2% error

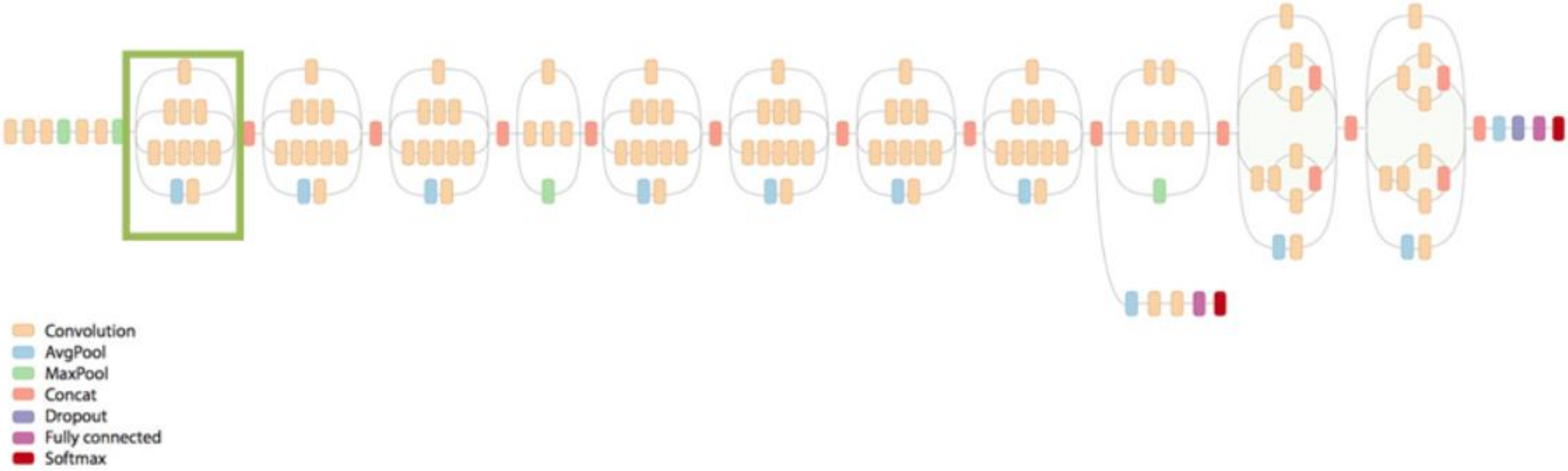
VGG net

- Karen Simonyan and Andrew Zisserman of the University of Oxford created a 19 layer CNN that strictly used 3x3 filters with stride and pad of 1, along with 2x2 maxpooling layers with stride 2.

ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224 × 224 RGB imag)					
conv3-64	conv3-64 **LRN**	conv3-64 **conv3-64**	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 **conv3-128**	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool					
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 **conv1-256**	conv3-256 conv3-256 **conv3-256**	conv3-256 conv3-256 conv3-256 **conv3-256**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

7.3% error rate
Runner up


Filter concatenation
3x3 convolutions
5x5 convolutions
1x1 convolutions
1x1 convolutions
1x1 convolutions
1x1 convolutions
3x3 max pooling
Previous layer


Convolution
AvgPool
MaxPool
Concat
Dropout
Fully connected
Softmax

Microsoft's ResNet

- ResNet is a new 152 layer network architecture that set new records in classification, detection, and localization through one incredible architecture. Aside from the new record in terms of number of layers, ResNet won ILSVRC 2015 with an incredible error rate of 3.6% (Depending on their skill and expertise, humans generally hover around a 5-10% error rate.

Revolution of Depth

AlexNet, 8 layers
(ILSVRC 2012)

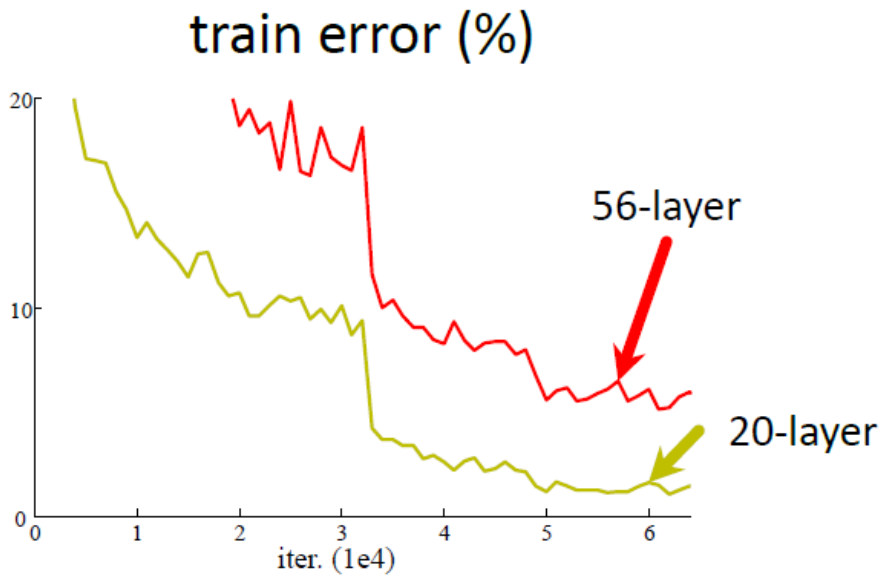
VGG, 19 layers
(ILSVRC 2014)

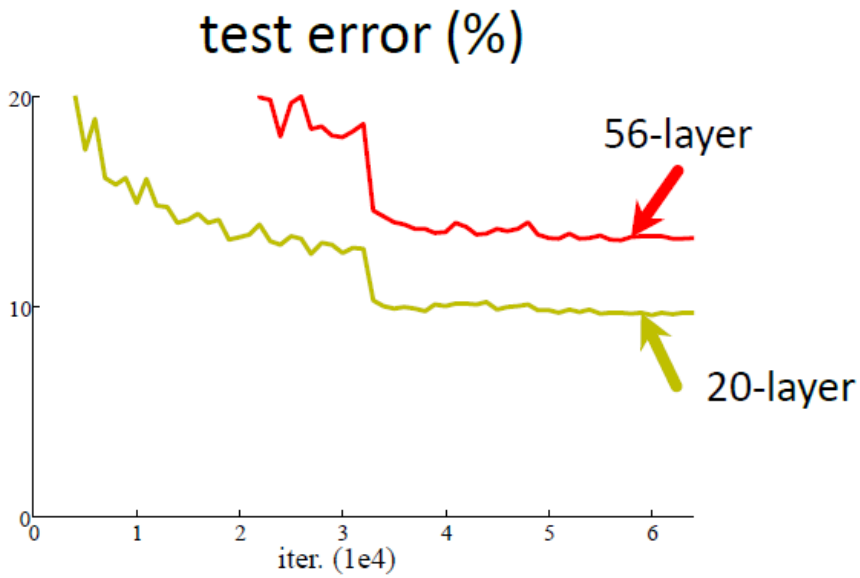
ResNet, 152 layers
(ILSVRC 2015)

ICCV15

Simply stacking layers?

CIFAR-10





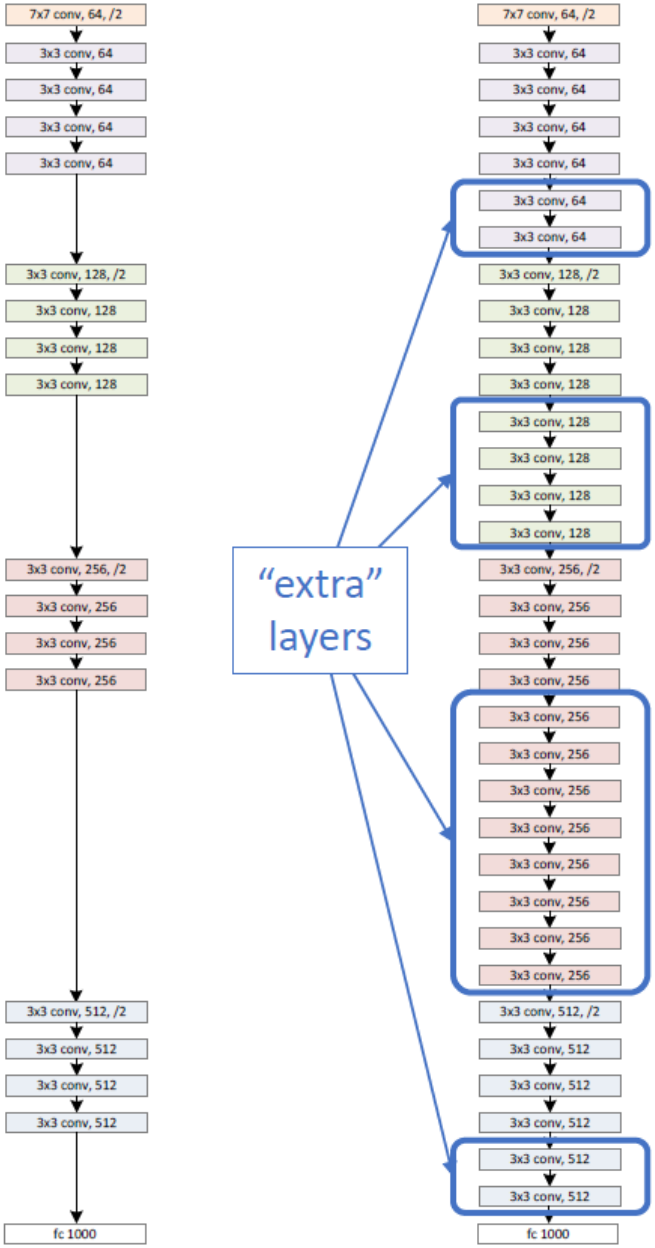
- *Plain* nets: stacking 3x3 conv layers…
- 56-layer net has **higher training error** and test error than 20-layer net

This is a general phenomenon

Why?

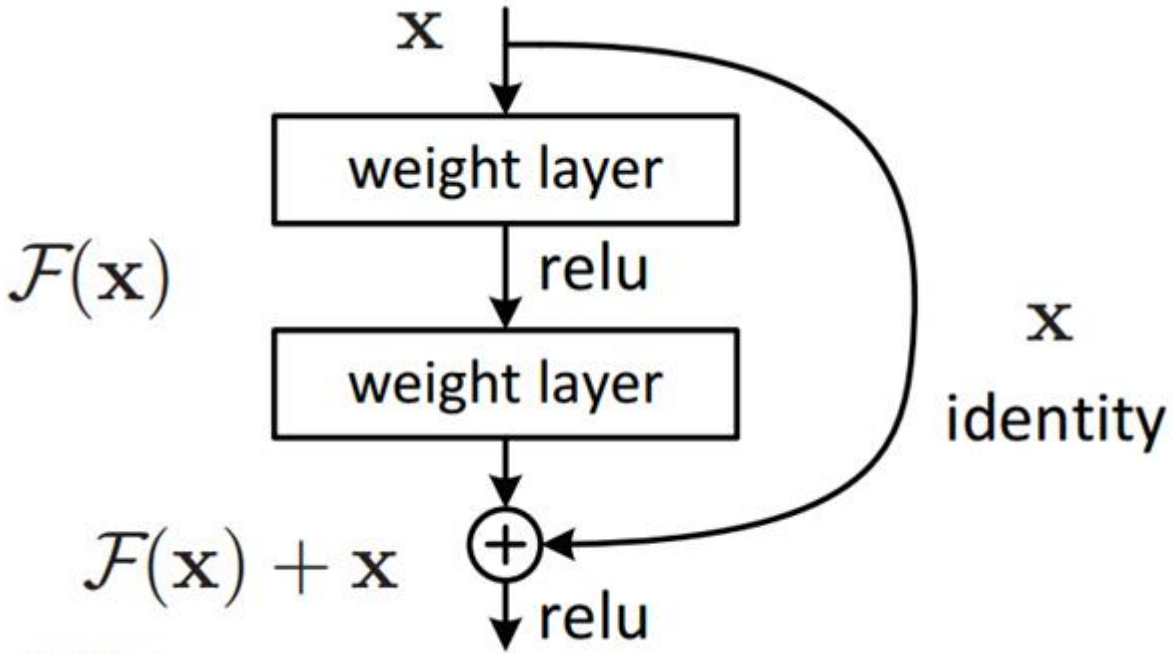
- As the network grows deeper even the convolution, dropout, RELU effects become so high that bias is creeping in.
- Add identity layers in between to ensure training info is just passed on without filtering
- 3X3 convolution, no dropout, no pooling, no full connection at the end. Just ultra deep

a shallower model (18 layers)



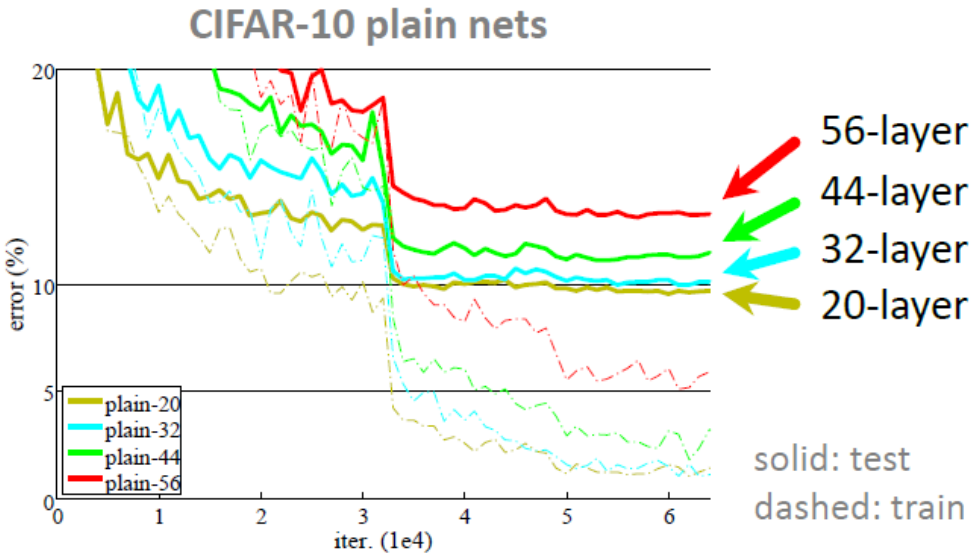
a deeper counterpart (34 layers)

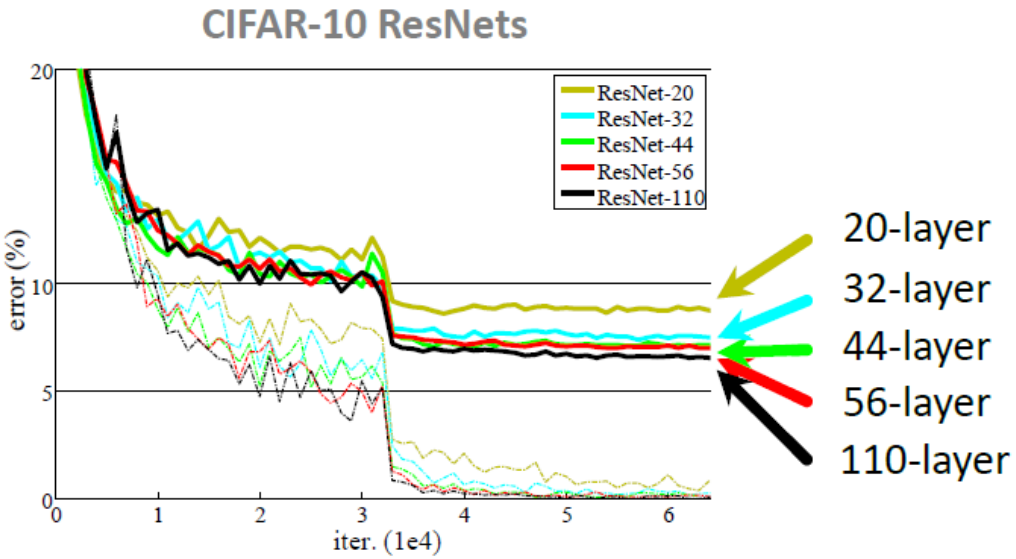
- A deeper model should not have **higher training error**
- A solution *by construction*:
  - original layers: copied from a learned shallower model
  - extra layers: set as identity
  - at least the same training error
- Optimization difficulties: solvers cannot find the solution when going deeper…



A residual block

- If identity were optimal, easy to set weights as 0
- If optimal mapping is closer to identity, easier to find small fluctuations


CIFAR-10 plain nets
error (%)
iter. (1e4)
plain-20
plain-32
plain-44
plain-56
56-layer
44-layer
32-layer
20-layer
solid: test
dashed: train


CIFAR-10 ResNets
error (%)
iter. (1e4)
ResNet-20
ResNet-32
ResNet-44
ResNet-56
ResNet-110
20-layer
32-layer
44-layer
56-layer
110-layer

Word2Vec and CNN

CNNS IN NLP

Word2Vec

word2vec and *GloVe* learn relationships between words.

The output are vectors with interesting relationships.

For example:

$$vec(king) - vec(man) + vec(woman) \approx vec(queen)$$

or

$$vec(Montreal\ Canadiens) - vec(Montreal) + vec(Toronto)$$

$$\approx vec(Toronto\ Maple\ Leafs)$$

Word2Vec

- Representation of words as vectors

[0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0]

Problem:

1. Dimensionality of the vector will be the size of vocabulary. e.g. 13 M for Google 1T and 500K for big vocab.
2. *book* [0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0]

   library [0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0]

Co-occurrence

Silence is the language of God. All else is a poor translation

counts	silence	is	the	language	of	God
silence	0	1	0	0	0	0
is	1	0	1	0	0	0
the	0	1	0	1	0	0
language	0	0	1	0	1	0
of	0	0	0	1	0	1
God	0	0	0	0	1	0

Co-occurrence matrix

- One for entire corpus
- Very large step size does not make sense
- There can be weights as the distance increases
- Predict the co-occurence

CBOW model

- Assume that $c = 1$. i.e. The length of the window is 1.
- Predict one target word, given one context word.
- This is like a bigram model.

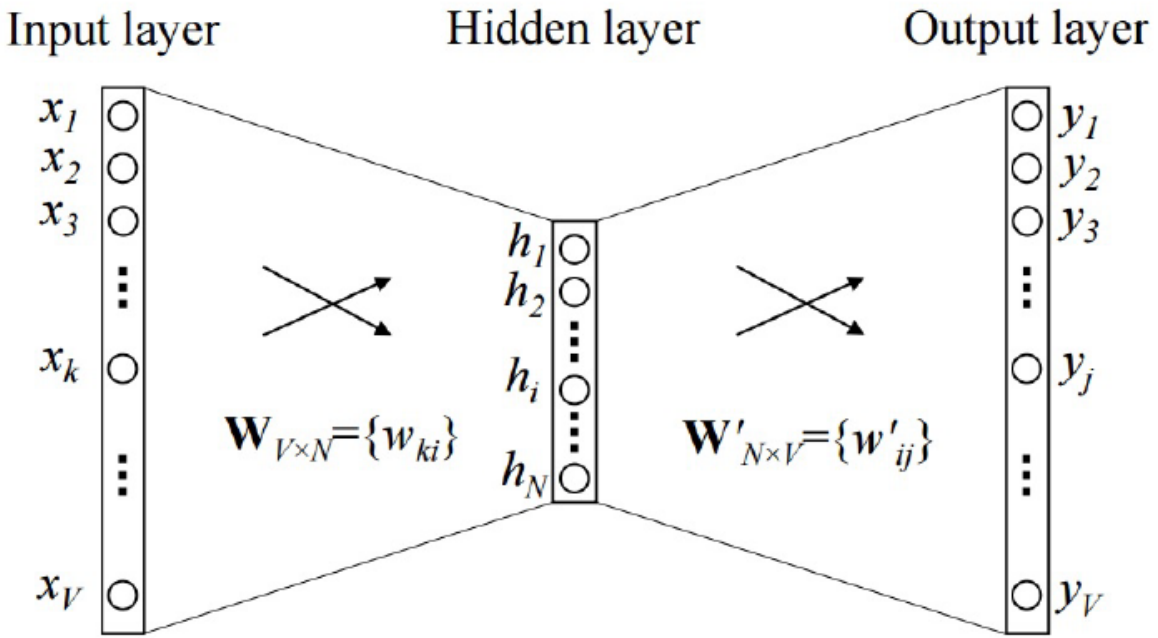
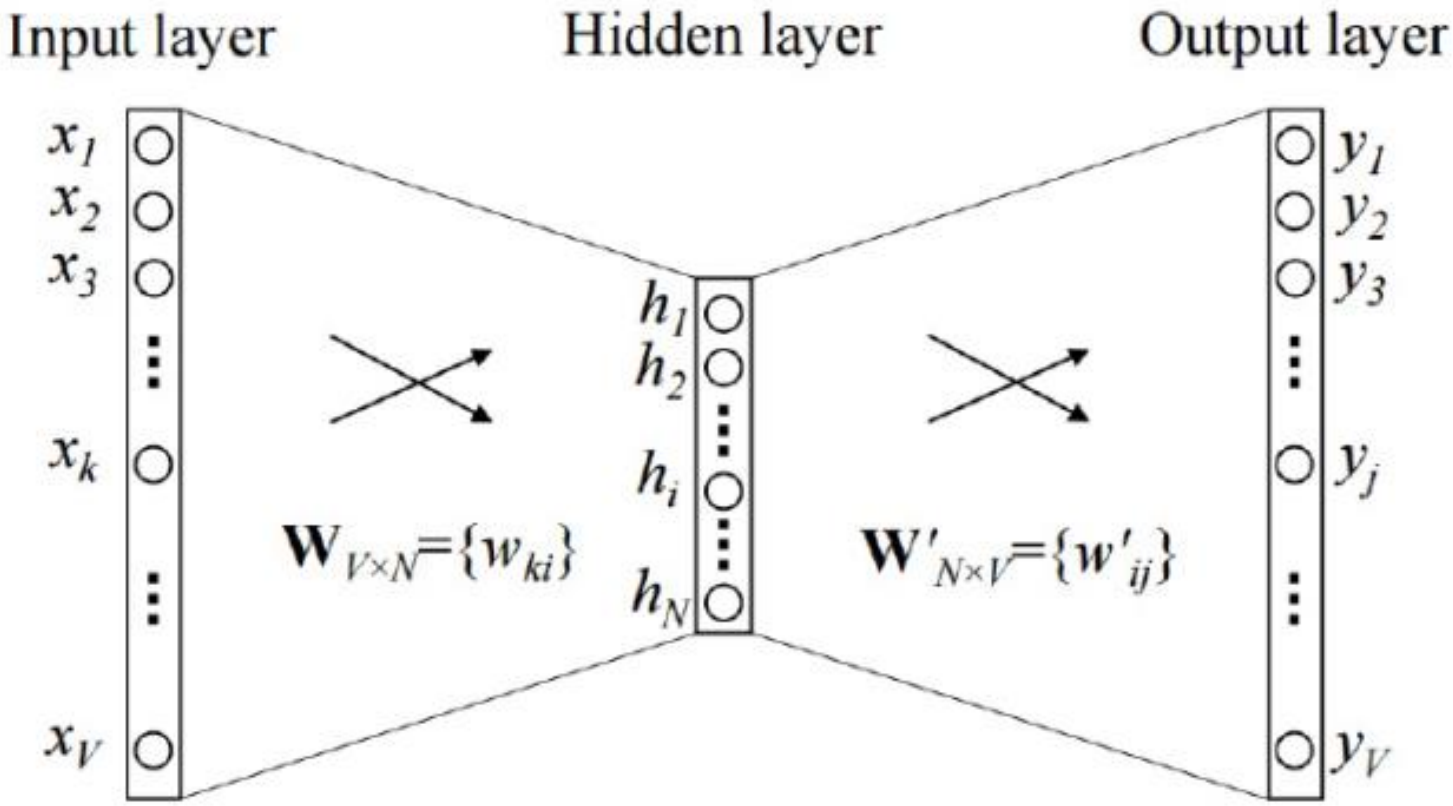


Figure: A very simple CBOW model.

- Assume only $\mathbf{x}_k = 1$, i.e. $\mathbf{x}^T = [0\ 0\ 0\ \ldots\ 1\ \ldots\ 0\ 0]$
- $\mathbf{h} = \mathbf{x}^T W$ is the k-th row of W, $W_{k:}$.
- Let's define $\mathbf{u}_c = \mathbf{h}$ as the vector representation of the central (input) word.



$$y_i = p(w|c);$$

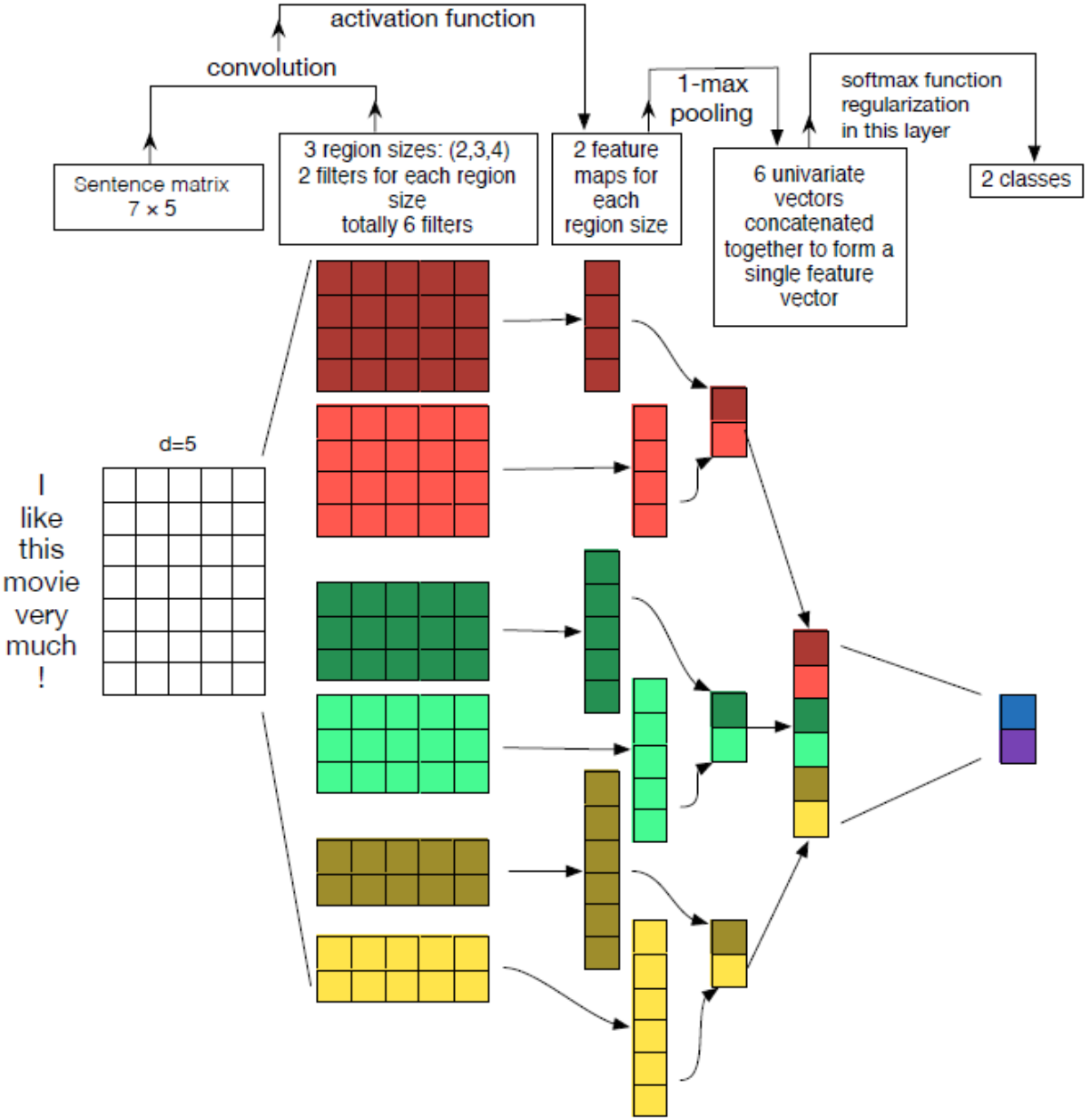
$$\sum_d y_i = 1$$

Hence use softmax transformation for the output.

$$\frac{e^y}{\sum_i e^y}$$

CNNS

Reference Paper: http://arxiv.org/pdf/1408.5882v2.pdf


activation function
convolution
1-max pooling
softmax function regularization in this layer
Sentence matrix 7 × 5
3 region sizes: (2,3,4) 2 filters for each region size totally 6 filters
2 feature maps for each region size
6 univariate vectors concatenated together to form a single feature vector
2 classes
d=5
I
like
this
movie
very
much
!

Instead of image pixels, the input to most NLP tasks are sentences or documents represented as a matrix. Each row of the matrix corresponds to one token, typically a word, but it could be a character. That is, each row is vector that represents a word.

Typically, these vectors are word embeddings (low-dimensional representations) like word2vec or GloVe, but they could also be one-hot vectors that index the word into a vocabulary. For a 10 word sentence using a 100-dimensional embedding we would have a 10×100 matrix as our input. That's our "image".

If the length of a given sentence is s, then the dimensionality of the sentence matrix is s×d (where d is the word2vec dimensions).

In vision, our filters slide over local patches of an image, but in NLP we typically use filters that slide over full rows of the matrix (words). Thus, the "width" of our filters is usually the same as the width of the input matrix.

The height, or region size, may vary, but sliding windows over 1-10 words at a time is typical.

This filter is applied to each possible window of words in the sentence {x(1:h),x(2:h+1),...,x(n−h+1:n)} to produce a feature map

$$\mathbf{c} = [c_1, c_2, \ldots, c_{n-h+1}], \qquad (3)$$

with $\mathbf{c} \in \mathbb{R}^{n-h+1}$. We then apply a max-over-time pooling operation (Collobert et al., 2011) over the feature map and take the maximum value $\hat{c} = \max\{\mathbf{c}\}$ as the feature corresponding to this particular filter. The idea is to capture the most important feature—one with the highest value—for each feature map. This pooling scheme naturally deals with variable sentence lengths.

One may use multiple filters for the same region size to learn complementary features from the same regions. One may also specify multiple kinds of filters with different region sizes (i.e., ‘heights’).

The dimensionality of the feature map generated by each filter will vary as a function of the sentence length and the filter region size.

We have described the process by which one feature is extracted from one filter. The model uses multiple filters (with varying window sizes) to obtain multiple features. These features form the penultimate layer and are passed to a fully connected softmax layer whose output is the probability distribution over labels.

CNN-static: A model with pre-trained vectors from word2vec. All words— including the unknown ones that are randomly initialized—are kept static and only the other parameters of the model are learned.

CNN-non-static: Same as above but the pre-trained vectors are fine-tuned for each task.

Figure 1: Illustration of a CNN architecture for sentence classification. We depict three filter region sizes: 2, 3 and 4, each of which has 2 filters. Filters perform convolutions on the sentence matrix and generate (variable-length) feature maps; 1-max pooling is performed over each map, i.e., the largest number from each feature map is recorded. Thus a univariate feature vector is generated from all six maps, and these
6 features are concatenated to form a feature vector for the penultimate layer. The final softmax layer then receives this feature vector as input and uses it to classify the sentence; here we assume binary classification and hence depict two possible output states.

Data	c	l	N	$\|V\|$	$\|V_{pre}\|$	*Test*
MR	2	20	10662	18765	16448	CV
SST-1	5	18	11855	17836	16262	2210
SST-2	2	19	9613	16185	14838	1821
Subj	2	23	10000	21323	17913	CV
TREC	6	10	5952	9592	9125	500
CR	2	19	3775	5340	5046	CV
MPQA	2	3	10606	6246	6083	CV

Table 1: Summary statistics for the datasets after tokenization. c: Number of target classes. l: Average sentence length. N: Dataset size. $|V|$: Vocabulary size. $|V_{pre}|$: Number of words present in the set of pre-trained word vectors. *Test*: Test set size (CV means there was no standard train/test split and thus 10-fold CV was used).

More on data sets from paper

http://arxiv.org/pdf/1408.5882v2.pdf

Description	Values
input word vectors	Google word2vec
filter region size	(3,4,5)
feature maps	100
activation function	ReLU
pooling	1-max pooling
dropout rate	0.5
$l2$ norm constraint	3

Table 2: Baseline configuration. 'feature maps' refers to the number of feature maps for each filter region size. 'ReLU' refers to *rectified linear unit* (Maas et al., 2013), a commonly used activation function in CNNs.

To provide a point of reference for the CNN results, we first report the performance achieved using SVM for sentence classification.

As a baseline, we used a linear kernel SVM exploiting uni and bi-gram features.[4] We then used averaged word vectors (from Google word2vec[5] or GloVe[6]) calculated over the words comprising the sentence as features and used an RBF-kernel SVM as the classifier operating in this dense feature space.

Dataset	bowSVM	wvSVM	bowwvSVM
MR	78.24	78.53	79.67
SST-1	37.92	44.34	43.15
SST-2	80.54	81.97	83.30
Subj	89.13	90.94	91.74
TREC	87.95	83.61	87.33
CR	80.21	80.79	81.31
MPQA	85.38	89.27	89.70
Opi	61.81	62.46	62.25
Irony	65.74	65.58	66.74

Table 1: Accuracy (AUC for Irony) achieved by SVM with different feature sets.

bowSVM: uni- and bi-gram features.

wvSVM: a naïve word2vec-based representation, i.e., the average (300-dimensional) word vector for each sentence.

bowwvSVM: concatenates bow vectors with the average word2vec representations.

Models tried

- Random: Random initialization of vectors
- Static: W2V vectors
- Non-Static: Word2Vec is also refined
- Multi-channel: 2 sets of Word2Vec and one set is allowed to be tuned

	Most Similar Words for	
	Static Channel	Non-static Channel
bad	*good*	*terrible*
	terrible	*horrible*
	horrible	*lousy*
	lousy	*stupid*
good	*great*	*nice*
	bad	*decent*
	terrific	*solid*
	decent	*terrific*
n't	*os*	*not*
	ca	*never*
	ireland	*nothing*
	wo	*neither*
!	*2,500*	*2,500*
	entire	*lush*
	jez	*beautiful*
	changer	*terrific*
,	*decasia*	*but*
	abysmally	*dragon*
	demise	*a*
	valiant	*and*

Table 3: Top 4 neighboring words—based on cosine similarity—for vectors in the static channel (left) and fine-tuned vectors in the non-static channel (right) from the multichannel model on the SST-2 dataset after training.

The multichannel model is able to fine-tune the non-static channel to make it more specific to the task-at-hand. For example, good is most similar to bad in word2vec, presumably because they are (almost) syntactically equivalent. But for vectors in the non-static channel that were fine tuned on the SST-2 dataset, this is no longer the case (table 3). Similarly, good is arguably closer to nice than it is to great for expressing sentiment, and this is indeed reflected in the learned vectors.

Model	MR	SST-1	SST-2	Subj	TREC	CR	MPQA
CNN-rand	76.1	45.0	82.7	89.6	91.2	79.8	83.4
CNN-static	81.0	45.5	86.8	93.0	92.8	84.7	**89.6**
CNN-non-static	**81.5**	48.0	87.2	93.4	93.6	84.3	89.5
CNN-multichannel	81.1	47.4	**88.1**	93.2	92.2	**85.0**	89.4
RAE (Socher et al., 2011)	77.7	43.2	82.4	–	–	–	86.4
MV-RNN (Socher et al., 2012)	79.0	44.4	82.9	–	–	–	–
RNTN (Socher et al., 2013)	–	45.7	85.4	–	–	–	–
DCNN (Kalchbrenner et al., 2014)	–	48.5	86.8	–	93.0	–	–
Paragraph-Vec (Le and Mikolov, 2014)	–	**48.7**	87.8	–	–	–	–
CCAE (Hermann and Blunsom, 2013)	77.8	–	–	–	–	–	87.2
Sent-Parser (Dong et al., 2014)	79.5	–	–	–	–	–	86.3
NBSVM (Wang and Manning, 2012)	79.4	–	–	93.2	–	81.8	86.3
MNB (Wang and Manning, 2012)	79.0	–	–	**93.6**	–	80.0	86.3
G-Dropout (Wang and Manning, 2013)	79.0	–	–	93.4	–	82.1	86.1
F-Dropout (Wang and Manning, 2013)	79.1	–	–	**93.6**	–	81.9	86.3
Tree-CRF (Nakagawa et al., 2010)	77.3	–	–	–	–	81.4	86.1
CRF-PR (Yang and Cardie, 2014)	–	–	–	–	–	82.7	–
SVM_S (Silva et al., 2011)	–	–	–	–	**95.0**	–	–

Table 2: Results of our CNN models against other methods. **RAE**: Recursive Autoencoders with pre-trained word vectors from Wikipedia (Socher et al., 2011). **MV-RNN**: Matrix-Vector Recursive Neural Network with parse trees (Socher et al., 2012). **RNTN**: Recursive Neural Tensor Network with tensor-based feature function and parse trees (Socher et al., 2013). **DCNN**: Dynamic Convolutional Neural Network with k-max pooling (Kalchbrenner et al., 2014). **Paragraph-Vec**: Logistic regression on top of paragraph vectors (Le and Mikolov, 2014). **CCAE**: Combinatorial Category Autoencoders with combinatorial category grammar operators (Hermann and Blunsom, 2013). **Sent-Parser**: Sentiment analysis-specific parser (Dong et al., 2014). **NBSVM, MNB**: Naive Bayes SVM and Multinomial Naive Bayes with uni-bigrams from Wang and Manning (2012). **G-Dropout, F-Dropout**: Gaussian Dropout and Fast Dropout from Wang and Manning (2013). **Tree-CRF**: Dependency tree with Conditional Random Fields (Nakagawa et al., 2010). **CRF-PR**: Conditional Random Fields with Posterior Regularization (Yang and Cardie, 2014). **SVM_S**: SVM with uni-bi-trigrams, wh word, head word, POS, parser, hypernyms, and 60 hand-coded rules as features from Silva et al. (2011)

Guidelines

In practice exploring the space of possible configurations for this model is extremely expensive, for two reasons:

(1) training these models is relatively slow, even using GPUs. For example, on the SST-1 dataset (Socher et al., 2013), it takes about 1 hour to run 10-fold cross validation, using a similar configuration to that described in (Kim, 2014).[1]

(2) The space of possible model architectures and hyperparameter settings is vast. Indeed, the simple CNN architecture of one layer requires, at a minimum, specifying: input word vector representations; filter region size(s); the number of feature maps; the activation function(s); the pooling strategy; and regularization terms (dropout/l2).

General advise

Consider starting with the basic configuration described using non-static word2vec or GloVe rather than one-hot vectors.

Line-search over the single filter region size to find the ‘best’ single region size. A reasonable range might be 1 to 10. However, for datasets with very long sentences like CR, it may be worth exploring larger filter region sizes. Once this ‘best’ region size is identified, it may be worth exploring combining multiple filters using regions sizes near this single best size, given that empirically multiple ‘good’ region sizes always outperformed using only the single best region size.

Alter the number of feature maps for each filter region size from 100 to 600, and when this is being explored, use a small dropout rate (0.0-0.5) and a large max norm constraint. Note that increasing the number of feature maps will increase the running time, so there is a trade-off to consider.

Other observations

Accuracy with randomly initialized words on the SST-1 dataset, increased from 37.4% to 45.0% due to more capacity (multiple filter widths and feature maps).

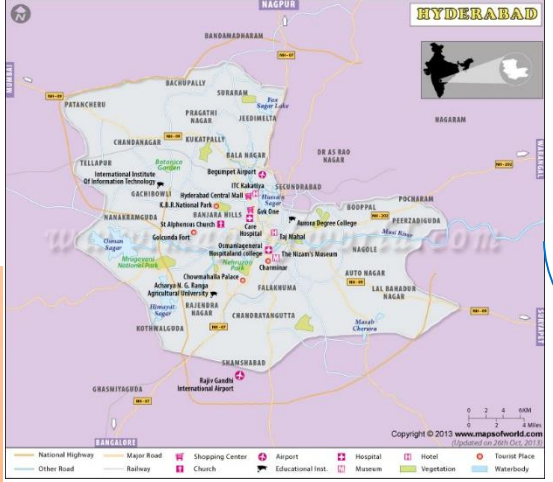
Dropout proved to be such a good regularizer that it was fine to use a larger than necessary network and simply let dropout regularize it. Dropout consistently added 2%–4% relative performance.

When randomly initializing words not in word2vec, we obtained slight improvements by sampling each dimension from U[−a,a] where a was chosen such that the randomly initialized vectors have the same variance as the pre-trained ones.

The non-static consistently outperformed static in other sets of experiments.

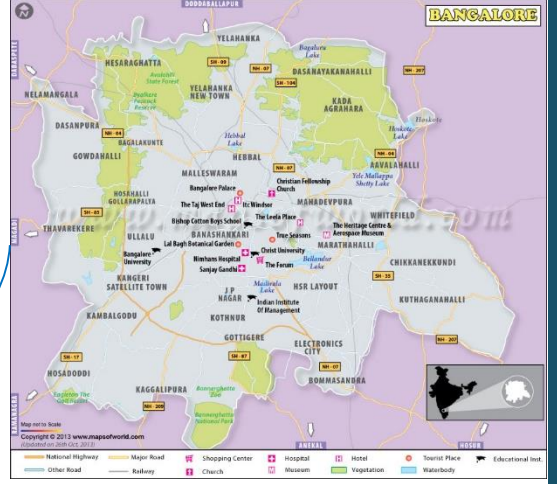
ReLU and tanh are the best overall candidates. It might also be worth trying no activation function at all for our one-layer CNN.

Use 1-max pooling; it does not seem necessary to expend resources evaluating alternative strategies.



INSOFE

Inspire...Educate...Transform.



HYDERABAD

Office and Classrooms

Plot 63/A, Floors 1&2, Road # 13, Film Nagar,
Jubilee Hills, Hyderabad - 500 033
+91-9701685511 (Individuals)
+91-9618483483 (Corporates)

BENGALURU

Office

Incubex, #728, Grace Platina, 4th Floor, CMH Road,
Indira Nagar, 1st Stage, Bengaluru – 560038
+91-9502334561 (Individuals)
+91-9502799088 (Corporates)

Classroom

KnowledgeHut Solutions Pvt. Ltd., Reliable Plaza,
Jakkasandra Main Road, Teacher's Colony, 14th Main
Road, Sector – 5, HSR Layout, Bengaluru - 560102

Social Media

Web:	http://www.insofe.edu.in
Facebook:	https://www.facebook.com/insofe
Twitter:	https://twitter.com/Insofeedu
YouTube:	http://www.youtube.com/InsofeVideos
SlideShare:	http://www.slideshare.net/INSOFE
LinkedIn:	http://www.linkedin.com/company/international-school-of-engineering

This presentation may contain references to findings of various reports available in the public domain. INSOFE makes no representation as to their accuracy or that the organization subscribes to those findings.