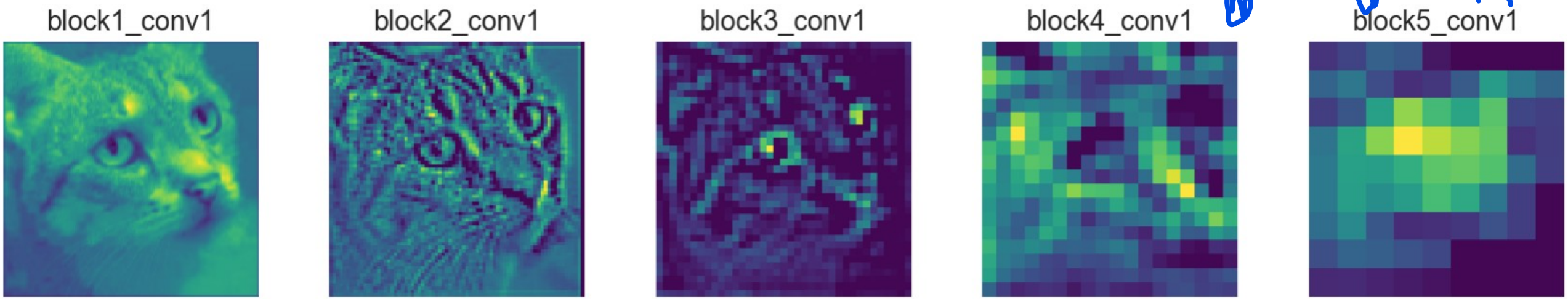
More on CNN

Biplab Banerjee

Thanks to the authors, towards data science and medium

CNN feature maps

abstract for us to describe but meaningful



Some insights

$f(g(x)) = g(f(x))$

Conv Layer

pooling

- Translation invariance vs equivariance
- Feature hierarchy learning in CNN

feature complexity

- Requires loads of images
- What happens if we do not train the model properly and still do imagenet classification?

$\approx 50\%$ classification

(cause DS very large

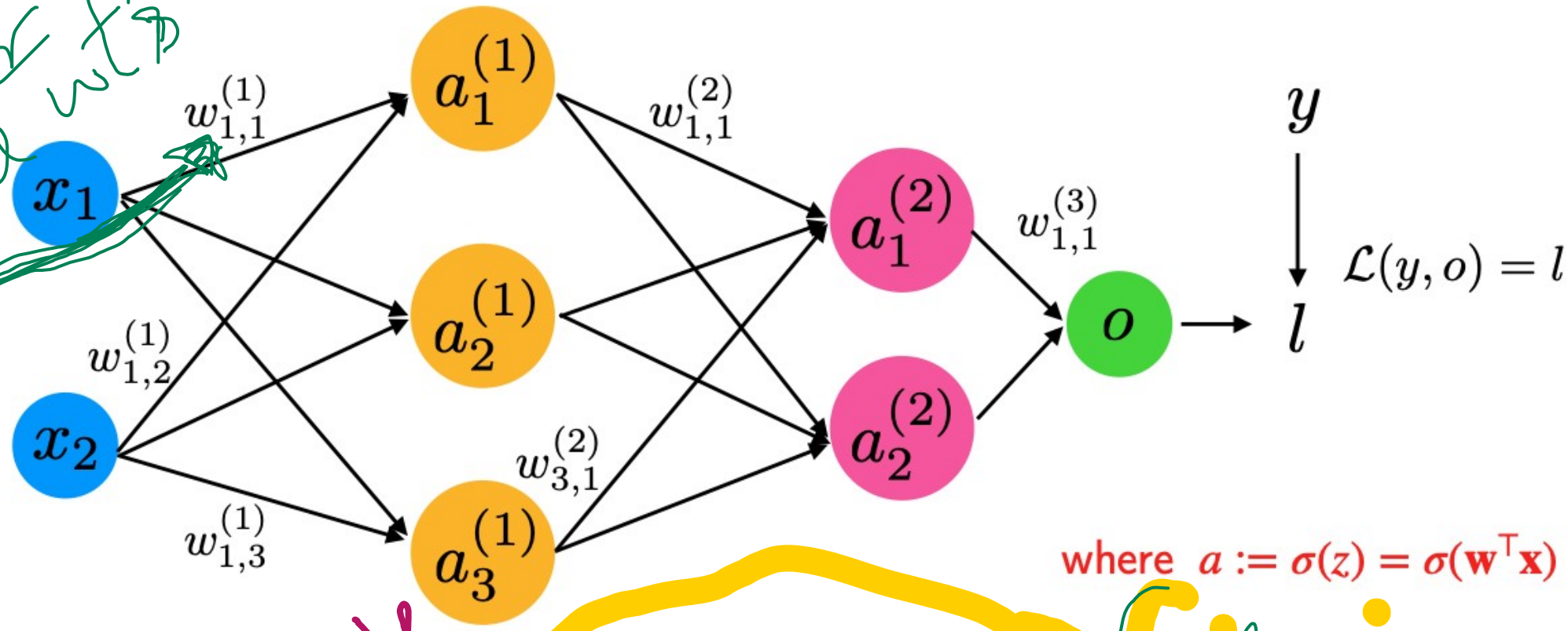
Issues in CNN

- Weight initialization
- Normalization
- Regularization
- Some intuitive CNN architectures

CNN RNN

Vanishing and exploding gradients

for initial wts



where $a := \sigma(z) = \sigma(\mathbf{w}^\top \mathbf{x})$

$$W_{11}' = W_{11} - \eta \frac{\partial l}{\partial w_{11}}$$

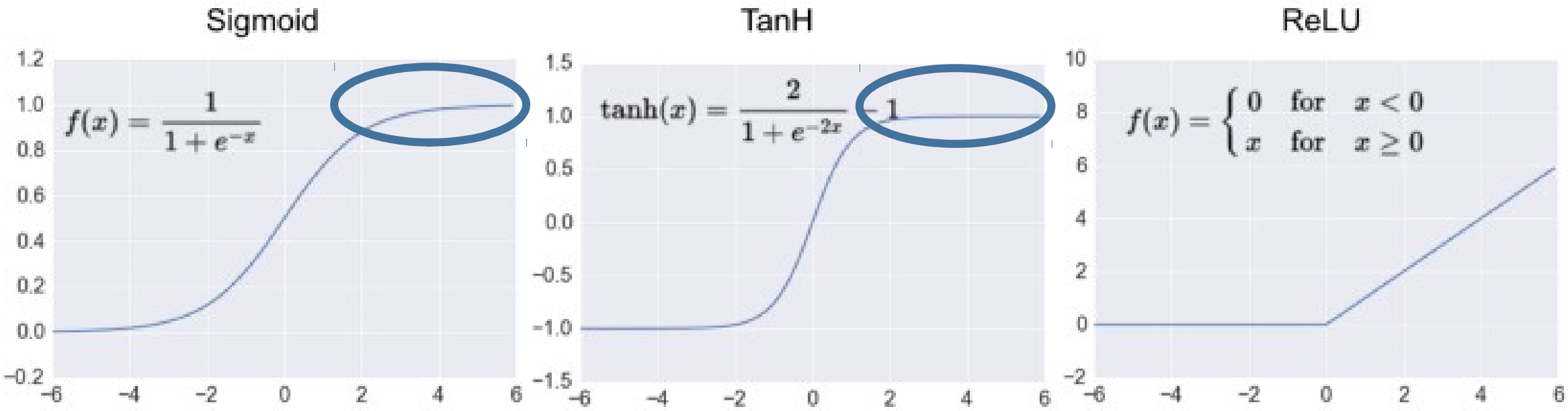
$$\frac{\partial l}{\partial w_{1,1}^{(1)}} = \frac{\partial l}{\partial o} \cdot \frac{\partial o}{\partial a_1^{(2)}} \cdot \frac{\partial a_1^{(2)}}{\partial a_1^{(1)}} \cdot \frac{\partial a_1^{(1)}}{\partial w_{1,1}^{(1)}}$$

Chain Rule

Ways to deal with them

- Vanishing gradients
  - Use proper activations
  - Smaller network (if using sigmoid/tanh type activations)
  - **Proper initialization** of wts
- Exploding gradients
  - Weight clipping
  - **Proper initialization**
  - Model engineering

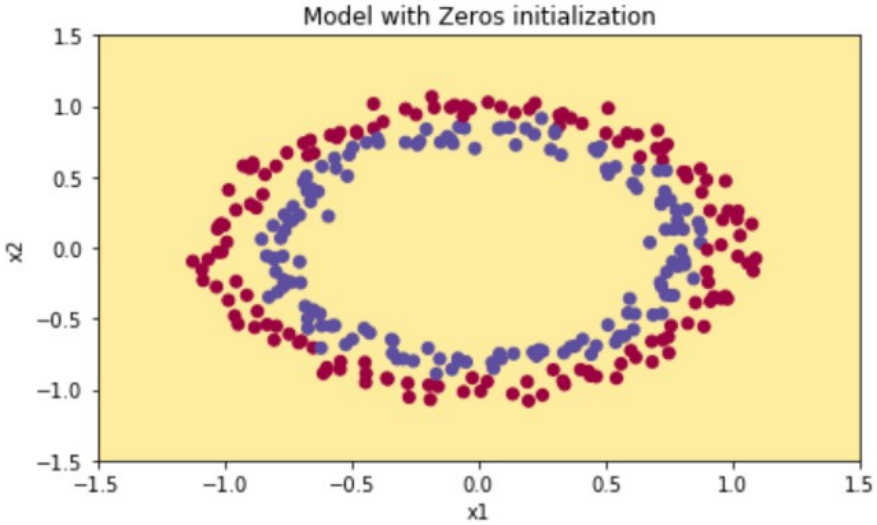
Saturation problem with activations

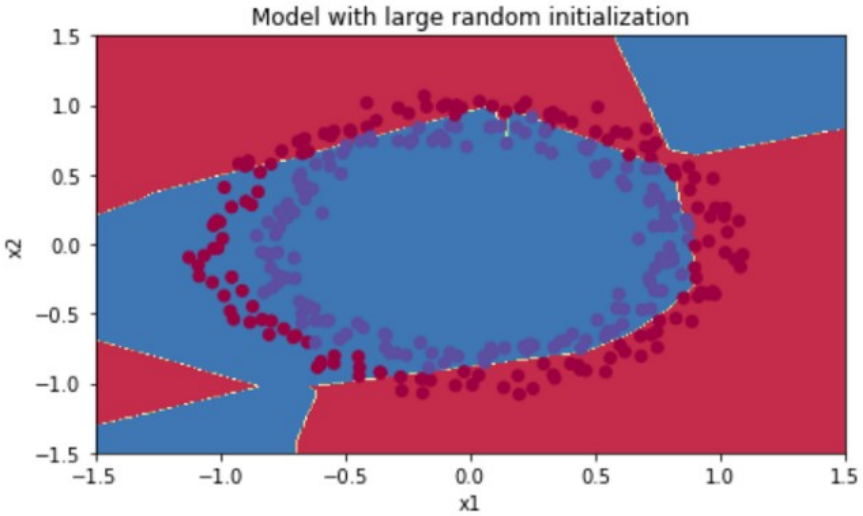


Problems with standard initializations

- Zero initialization – The model behaves like a linear model

- Random initialization – Saturation problem for sigmoid/tanh





- We need some initialization technique that will ensure that the input to the activation function lies within the feasible range to avoid any saturation

- Should take the number of nodes in a layer into account

- How about ensuring that the inputs to the activation follow a certain distribution? (e.g standard normal)

Does weight initialization have any effect at all?

$$in = X_1W_1 + X_2W_2 + X_3W_3 + X_4W_4 + \dots$$

$$\boxed{Var(X_iW_i) = [E(X_i)]^2 Var(W_i) + [E(W_i)]^2 Var(X_i) + Var(X_i)Var(W_i)}$$

Assuming each product term is independent!

nce weight and inputs are normalized to zero mean and unit variance

$$Var(X_iW_i) = 0 \times 1 + 0 \times 1 + 1 \times 1 = 1$$

$$Var(in) = \sum_{i=0}^{n} Var(X_iW_i) = n \times 1 = n$$

Xavier initialization

$$Var(Y) = n_{in} Var(W_i) Var(X_i)$$

How can we make the variance of the layers largely identical?

$$Var(W_i) = \frac{1}{n_{in}}$$

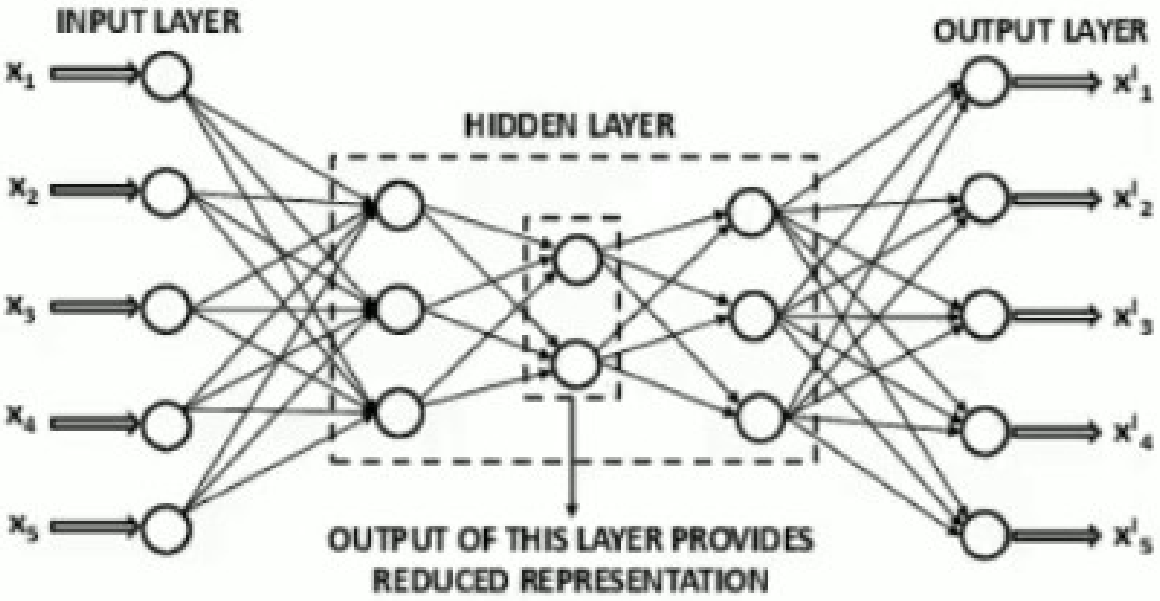
For forward and backward passes

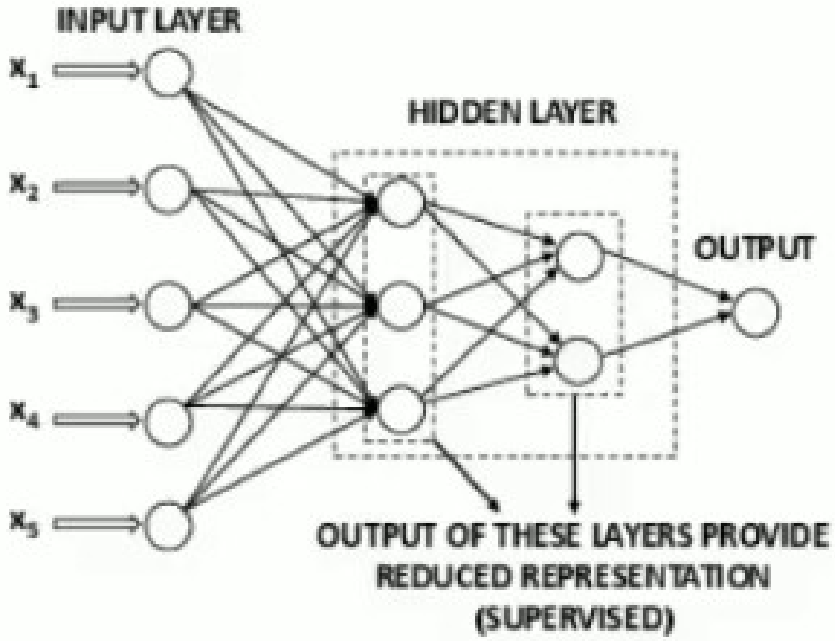
$$Var(W_i) = \frac{1}{n_{out}}$$

$$Var(W_i) = \frac{1}{n_{avg}} = \frac{2}{n_{in} + n_{out}}$$

pre-training

- *Unsupervised pretraining:* Use training data without labels for initialization.
  - Improves convergence behavior.
  - Regularization effect.
- *Supervised pretraining:* Use training data with labels for initialization.
  - Improves convergence but might overfit.
- Focus on unsupervised pretraining.


INPUT LAYER
HIDDEN LAYER
OUTPUT LAYER
OUTPUT OF THIS LAYER PROVIDES
REDUCED REPRESENTATION


INPUT LAYER
HIDDEN LAYER
OUTPUT
OUTPUT OF THESE LAYERS PROVIDE
REDUCED REPRESENTATION
(SUPERVISED)

Suggested reading

Understanding the difficulty of training deep feedforward neural networks

Xavier Glorot **Yoshua Bengio**
DIRO, Université de Montréal, Montréal, Québec, Canada

Data normalization

- Normalized data helps in stable training
- It helps in dealing with **internal covariate shift**
- Since the CNN is trained with BATCH of training samples, lets see the idea of batch gradient descent in back-propagation

Batch gradient descent

$$J_{train}(\theta) = (1/2m)\ \Sigma(\ h_\theta(x^{(i)}) \ - y^{(i)})^2$$

Repeat {

$$\theta j = \theta j - (\text{learning rate}/m) * \Sigma(\ h_\theta(x^{(i)}) \ - y^{(i)})x_j^{(i)}$$

For every j =0 …n

}

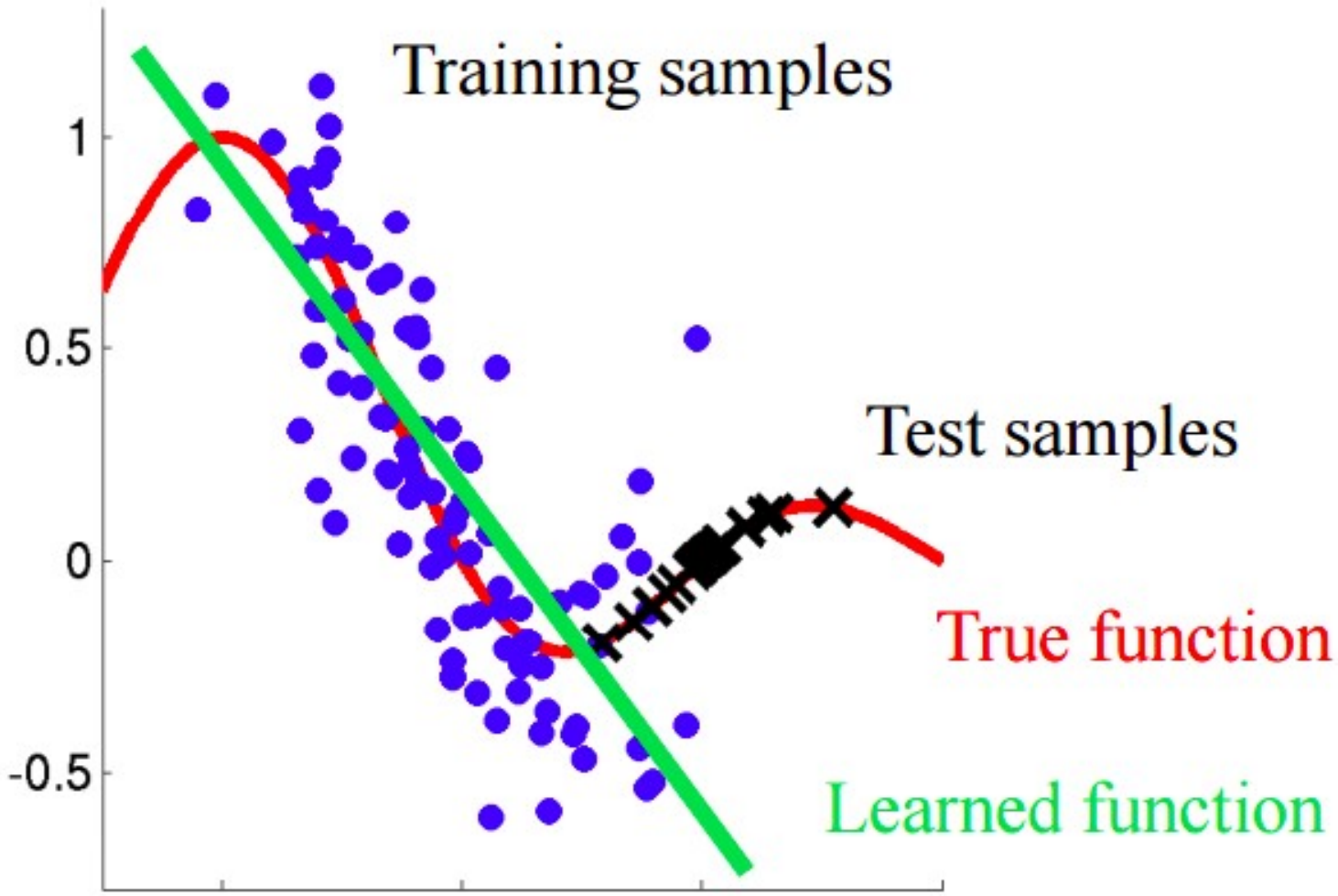
Stochastic gradient descent

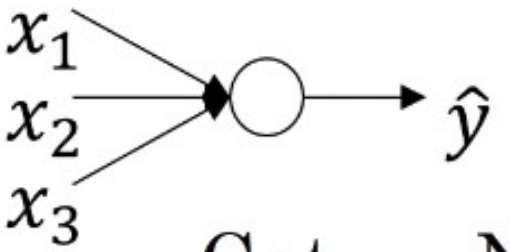
```
J_train(θ) = (1/m) Σ Cost(θ, (x^(i),y^(i)))

Repeat {

For i=1 to m{

        θ_j = θ_j – (learning rate) * Σ( h_θ(x^(i))  - y^(i))x_j^(i)
       For every j =0 …n

                }
}
```

Mini-batch gradient descent

```
Repeat {
 For i=1,11, 21,…..,91

    Let Σ be the summation from i to i+9 represented by k.

    θj = θj – (learning rate/size of (b) ) * Σ( hθ(x(k))  - y(k))xj(k)
        For every j =0 …n

}
```

The idea of dataset (covariate) shift





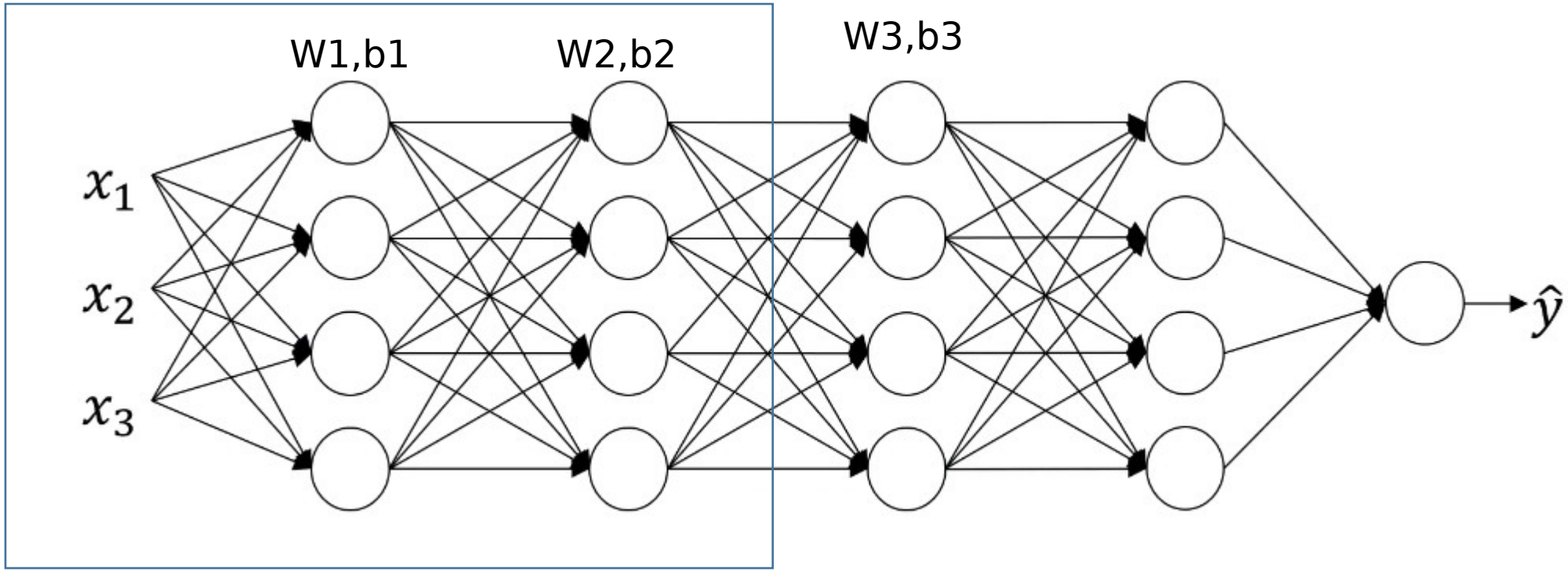
Cat $y = 1$

Non-Cat $y = 0$

$y = 1$

$y = 0$

What does each layer weight see?



Inputs to layer 3 changes over time as (W1,b1) and (W2,b2) are also changing

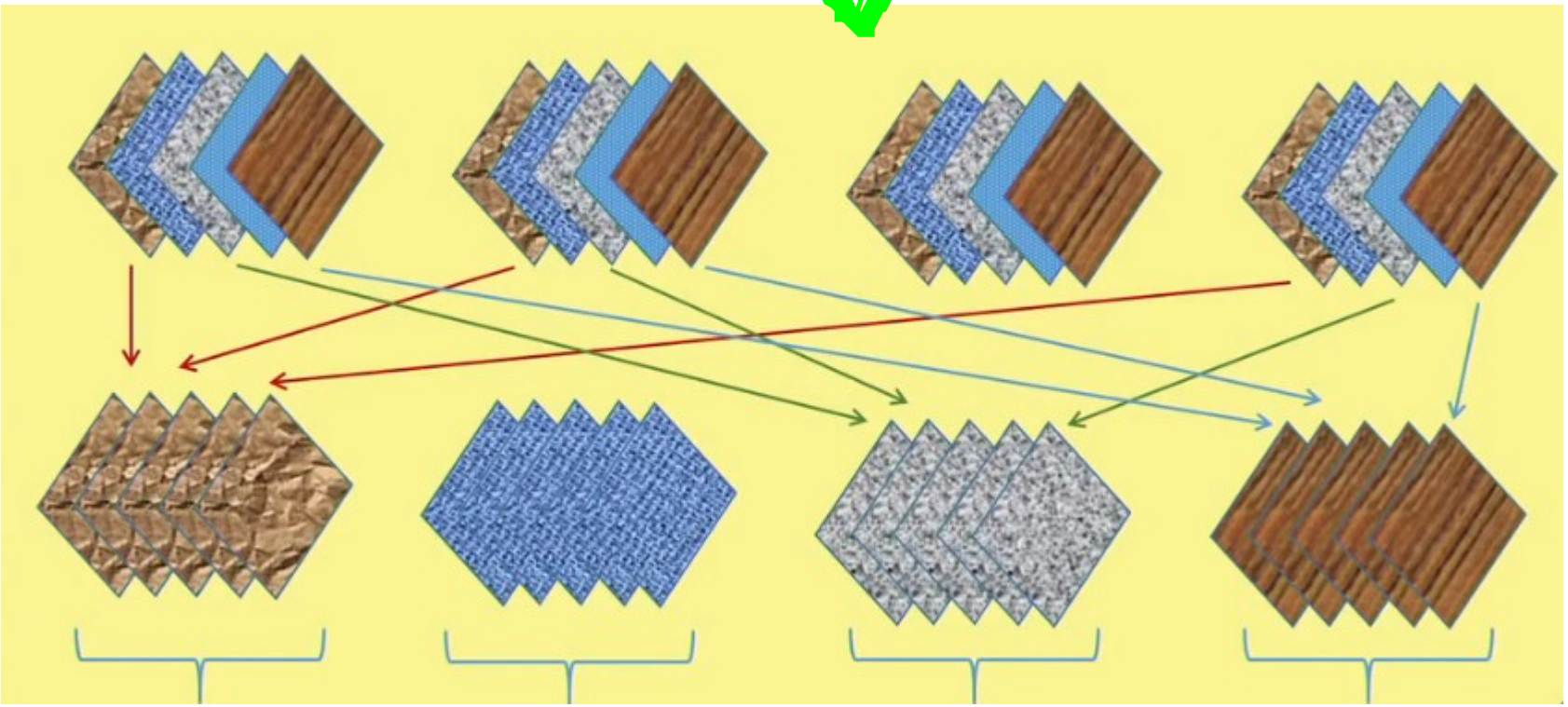
Batch-normalization

> *Training Deep Neural Networks is complicated by the fact that the distribution of each layer's inputs changes during training, as the parameters of the previous layers change. This slows down the training by requiring lower learning rates and careful parameter initialization, and makes it notoriously hard to train models with saturating nonlinearities.*

Normalizing the inputs to the layer has an effect on the training of the model, dramatically reducing the number of epochs required. It can also have a regularizing effect, reducing generalization error much like the use of activation regularization.

How BN works

we do this for every layer



μ_i σ_i

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1\ldots m}\}$;
Parameters to be learned: γ, β

Output: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{// mini-batch mean}$$

$$\sigma_{\mathcal{B}}^2 \leftarrow \frac{1}{m}\sum_{i=1}^{m} (x_i - \mu_{\mathcal{B}})^2 \qquad \text{// mini-batch variance}$$

$$\widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}} \qquad \text{// normalize}$$

$$y_i \leftarrow \gamma \widehat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i) \qquad \text{// scale and shift}$$

$$\frac{\partial \ell}{\partial \widehat{x}_i} = \frac{\partial \ell}{\partial y_i} \cdot \gamma$$

$$\frac{\partial \ell}{\partial \sigma_{\mathcal{B}}^2} = \sum_{i=1}^{m} \frac{\partial \ell}{\partial \widehat{x}_i} \cdot (x_i - \mu_{\mathcal{B}}) \cdot \frac{-1}{2}(\sigma_{\mathcal{B}}^2 + \epsilon)^{-3/2}$$

$$\frac{\partial \ell}{\partial \mu_{\mathcal{B}}} = \left(\sum_{i=1}^{m} \frac{\partial \ell}{\partial \widehat{x}_i} \cdot \frac{-1}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}} \right) + \frac{\partial \ell}{\partial \sigma_{\mathcal{B}}^2} \cdot \frac{\sum_{i=1}^{m} -2(x_i - \mu_{\mathcal{B}})}{m}$$

$$\frac{\partial \ell}{\partial x_i} = \frac{\partial \ell}{\partial \widehat{x}_i} \cdot \frac{1}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}} + \frac{\partial \ell}{\partial \sigma_{\mathcal{B}}^2} \cdot \frac{2(x_i - \mu_{\mathcal{B}})}{m} + \frac{\partial \ell}{\partial \mu_{\mathcal{B}}} \cdot \frac{1}{m}$$

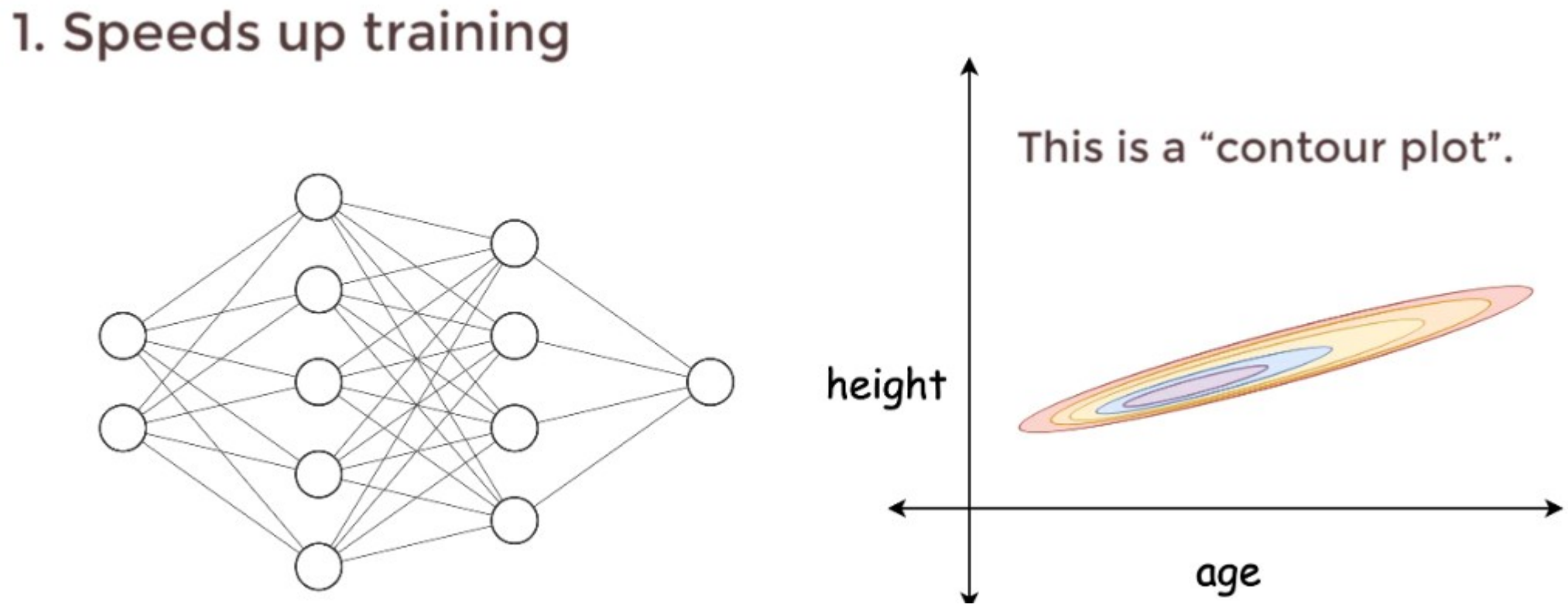
$$\frac{\partial \ell}{\partial \gamma} = \sum_{i=1}^{m} \frac{\partial \ell}{\partial y_i} \cdot \widehat{x}_i$$

$$\frac{\partial \ell}{\partial \beta} = \sum_{i=1}^{m} \frac{\partial \ell}{\partial y_i}$$

Advantages of BN

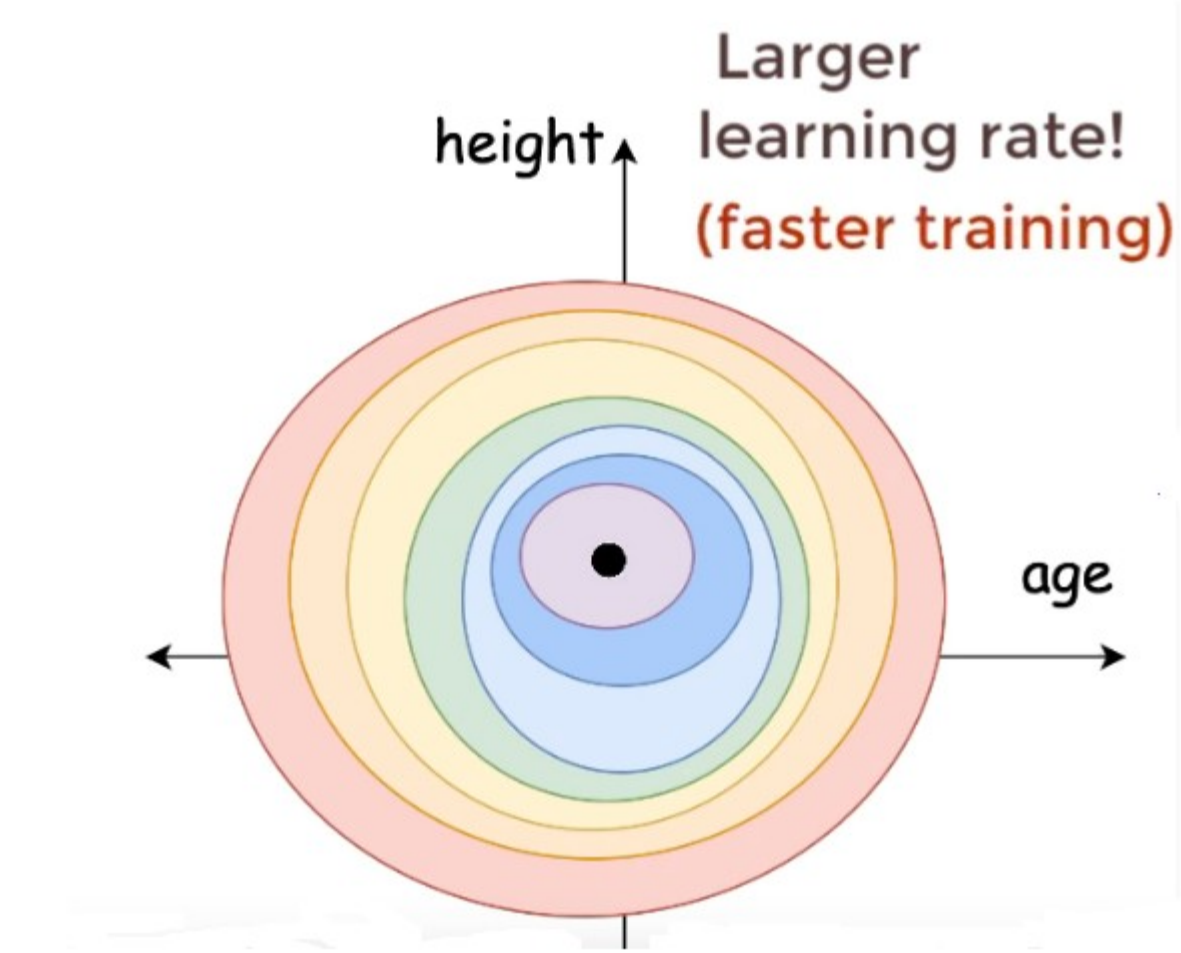
- Speeds up training
- Reduce dependence on weight initialization
- Small regularization effect

Speeds up training

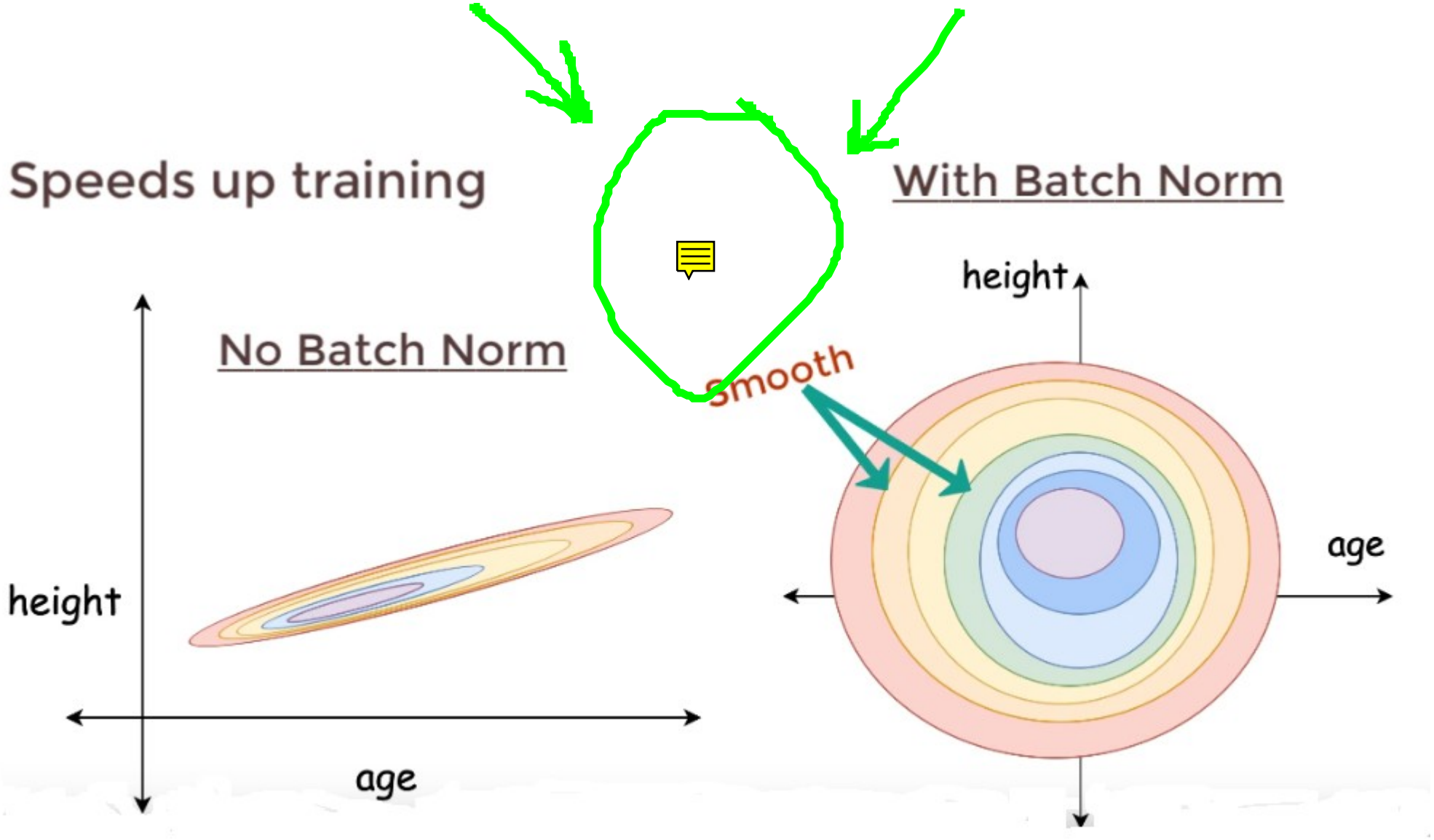


In different batch, let us consider we have young to old people

After standardization



1. Speeds up training

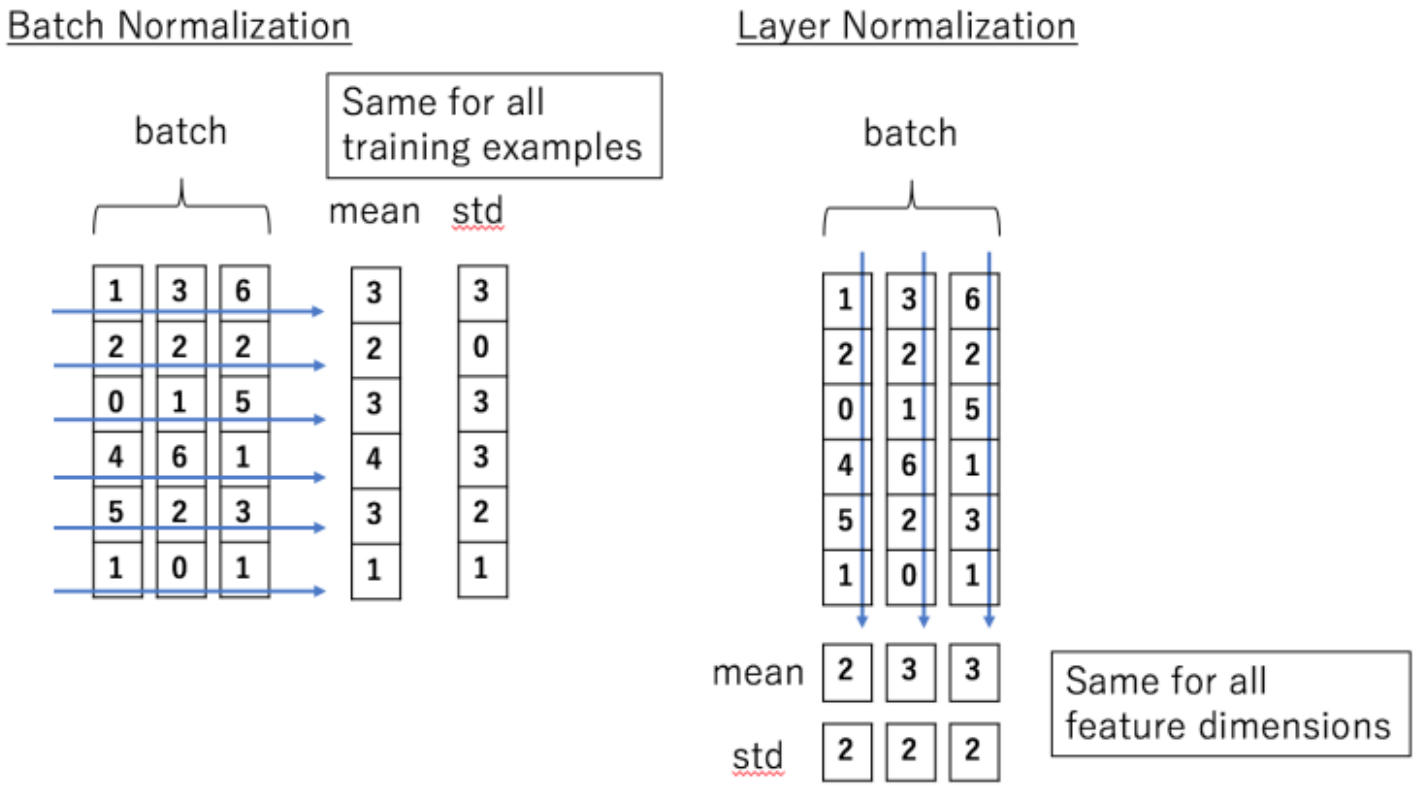


Where can the BN not be used?

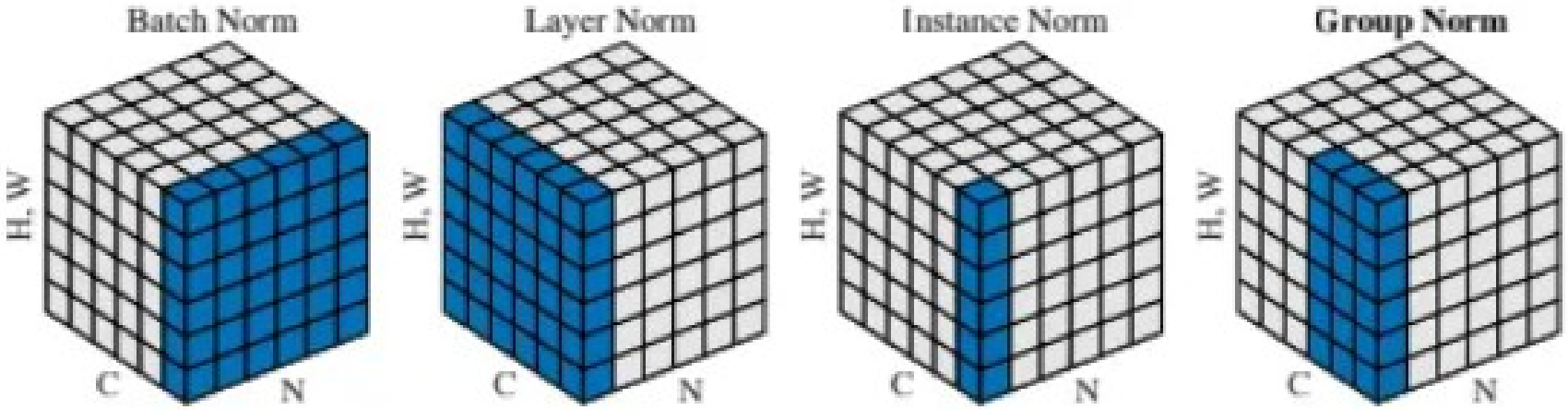
- Small batch size – high noise in calculated batch mean and std!
- Recurrent type models – separate batch-norm at each time-step!

- Solutions?
  - Weight normalization
  - Layer normalization
  - Instance normalization
  - Spectral normalization
  - Group normalization
  - …

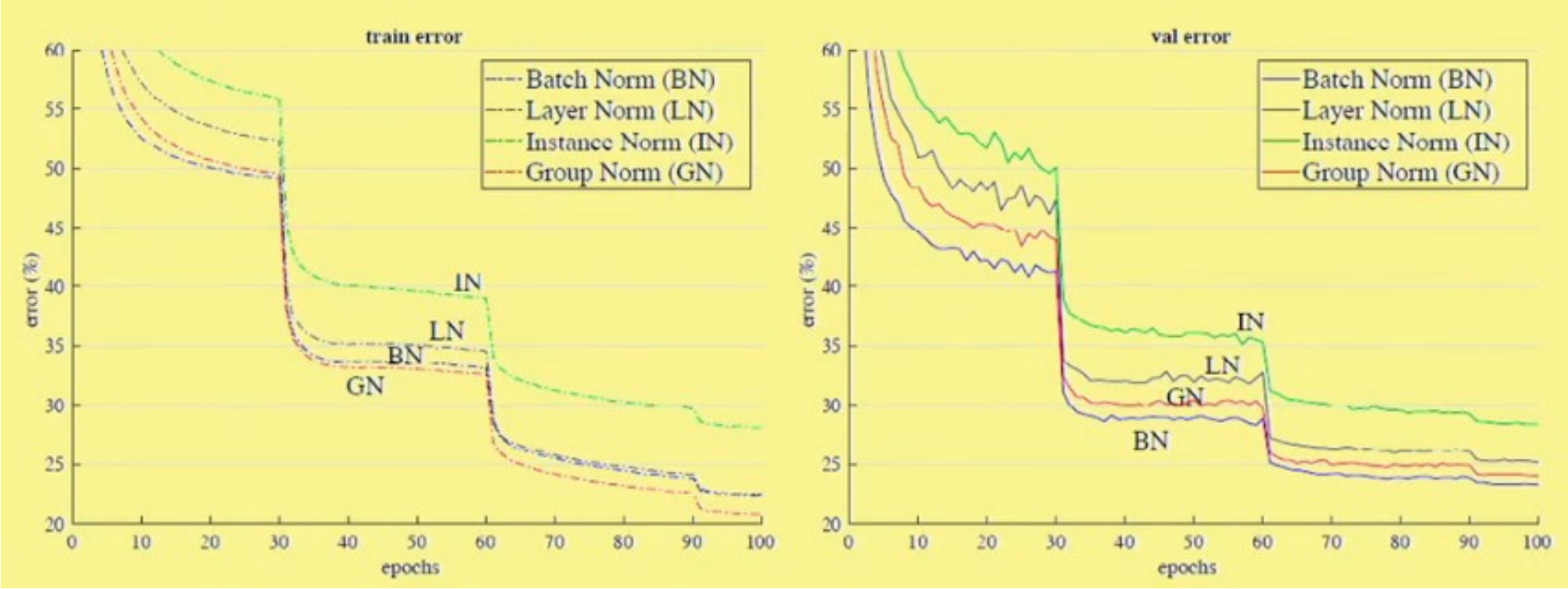
Layer normalization



All together



Comparison of different normalization (ECCV 18’)



Suggested reading

How Does Batch Normalization Help Optimization?

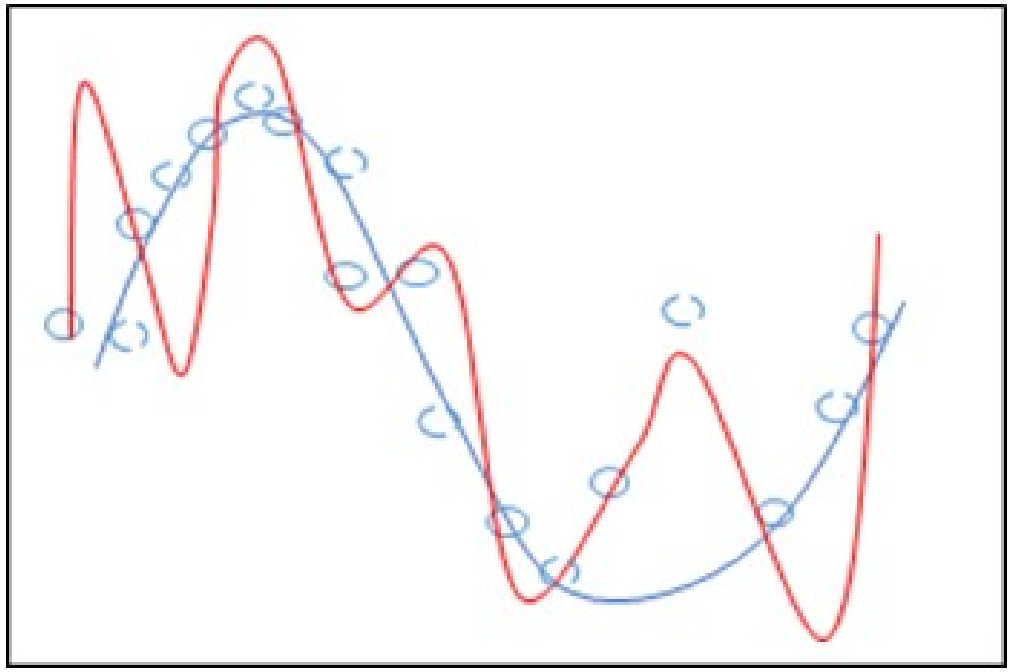
Shibani Santurkar* MIT shibani@mit.edu

Dimitris Tsipras* MIT tsipras@mit.edu

Andrew Ilyas* MIT ailyas@mit.edu

Aleksander Mądry MIT madry@mit.edu

Model regularization

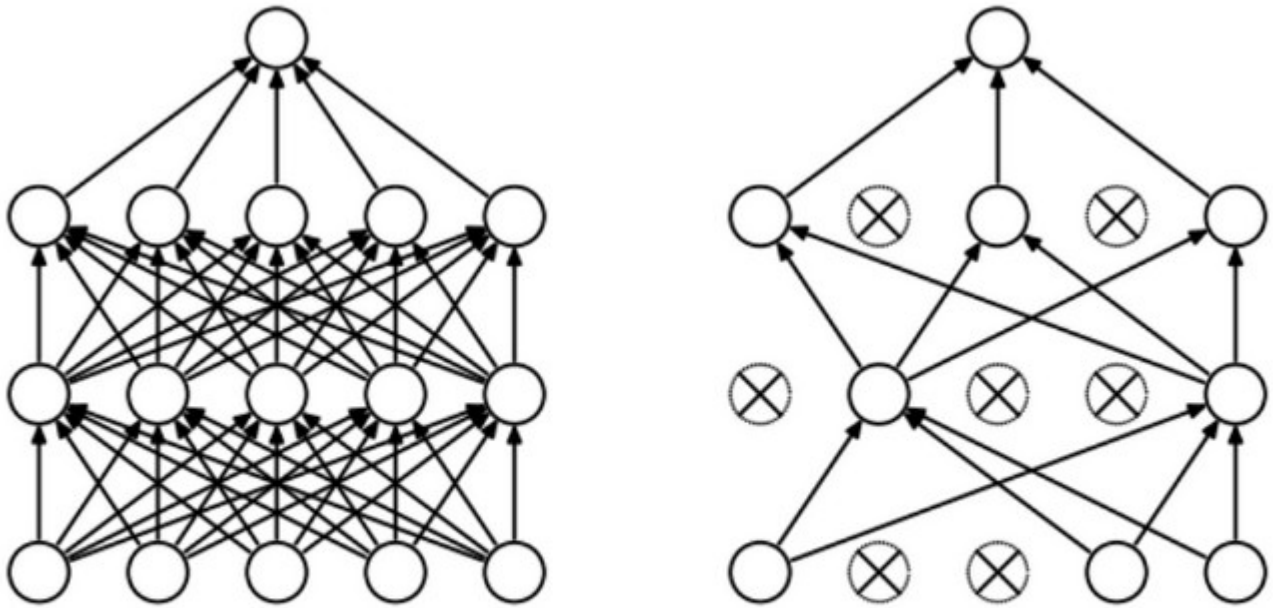


1. Less training data
2. Too many model parameters
3. Model overfitting
4. Regularization – ways to avoid overfitting

Regularization ways

- We have to ensure that the model does not overfit the training data – does not memorize the training samples
- Restrict the values the model parameters can take (lasso, ridge regressions)
- Ensemble model – mixture of experts

Dropout regularization



(a) Standard Neural Net (b) After applying dropout.

Srivastava, Nitish, et al. "Dropout: a simple way to prevent neural networks from overfitting", JMLR 2014

Given

- $h(x) = xW + b$ a linear projection of a d_i-dimensional input x in a d_h-dimensional output space.
- $a(h)$ an activation function

it's possible to model the application of Dropout, in the training phase only, to the given projection as a modified activation function:

$$f(h) = D \odot a(h)$$

Where $D = (X_1, \cdots, X_{d_h})$ is a d_h-dimensional vector of Bernoulli variables X_i.

A Bernoulli random variable has the following probability mass distribution:

$$f(k; p) = \begin{cases} p & \text{if} \quad k = 1 \\ 1 - p & \text{if} \quad k = 0 \end{cases}$$

Where k are the possible outcomes.

$$O_i = X_i a(\sum_{k=1}^{d_i} w_k x_k + b) = \begin{cases} a(\sum_{k=1}^{d_i} w_k x_k + b) & \text{if} \quad X_i = 1 \\ 0 & \text{if} \quad X_i = 0 \end{cases}$$

where $P(X_i = 0) = p$.

Since during train phase a neuron is kept on with probability q, during the testing phase we have to emulate the behavior of the ensemble of networks used in the training phase.

To do this, the authors suggest scaling the activation function by a factor of q during the test phase in order to use the expected output produced in the training phase as the single output required in the test phase. Thus:

Train phase: $O_i = X_i a(\sum_{k=1}^{d_i} w_k x_k + b)$

Test phase: $O_i = qa(\sum_{k=1}^{d_i} w_k x_k + b)$

Inverted dropout

Train phase: $O_i = \frac{1}{q} X_i a(\sum_{k=1}^{d_i} w_k x_k + b)$

Test phase: $O_i = a(\sum_{k=1}^{d_i} w_k x_k + b)$

Dropout + L2 normalization

$$\mathcal{L}(y, \hat{y}) = \mathcal{E}(y, F(W; x)) + \frac{\lambda}{2} W^2$$

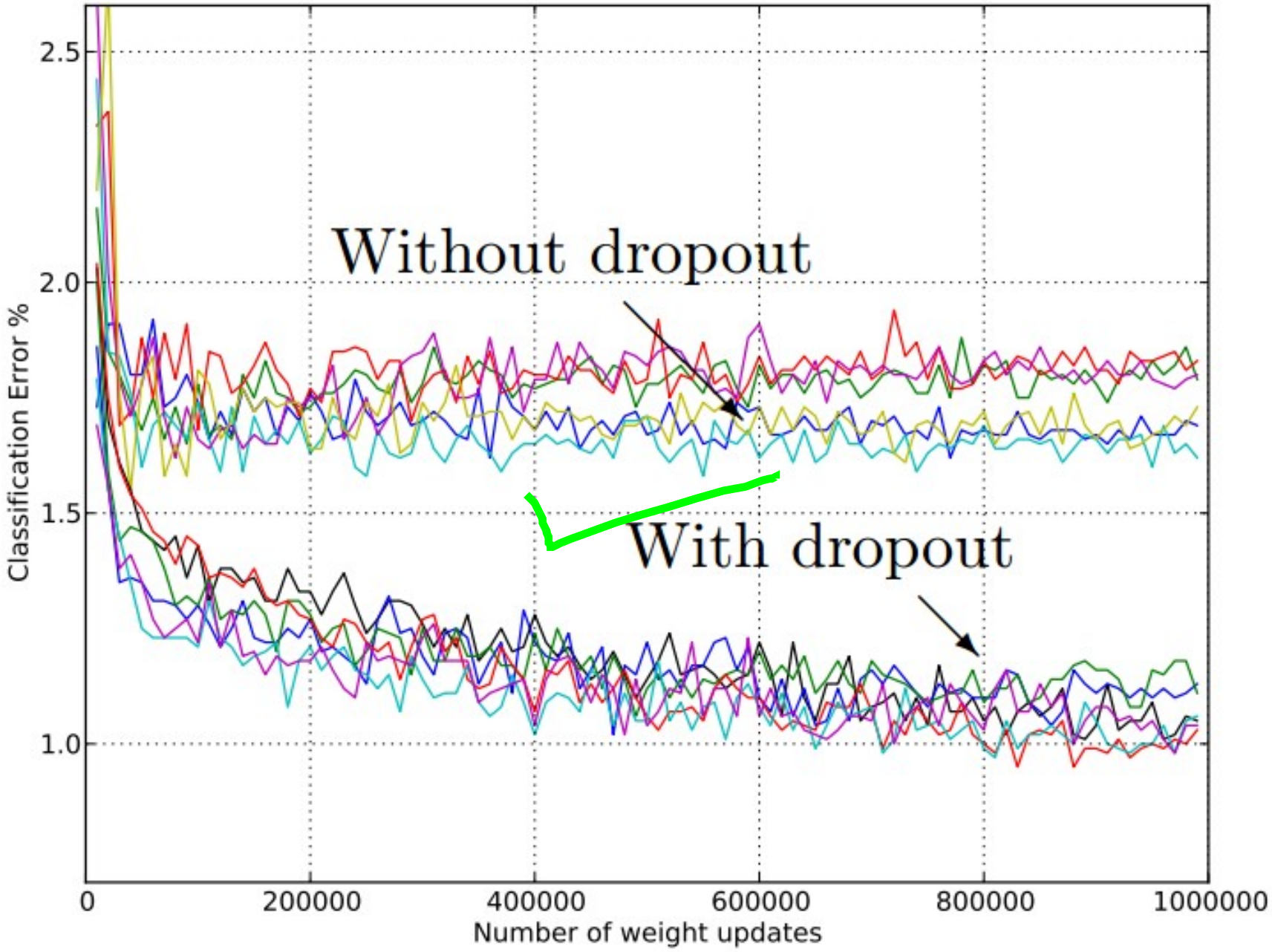
Dropout can’t control the range of values the weight variables can take!

Dropout

$$w \leftarrow w - \eta \left(\frac{\partial F(W; x)}{\partial w} + \lambda w \right)$$

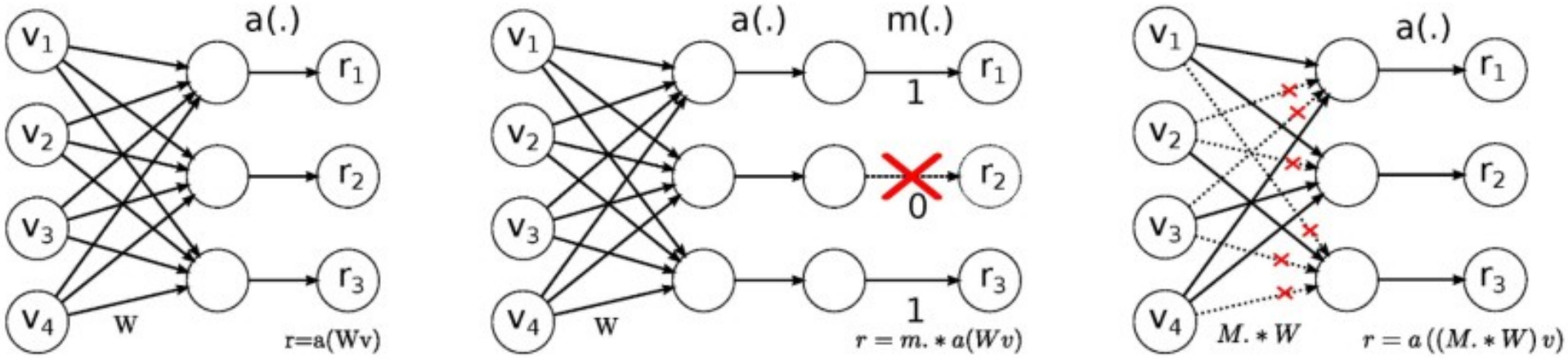
Inverted Dropout

$$w \leftarrow w - \eta \left(\frac{1}{q} \frac{\partial F(W; x)}{\partial w} + \lambda w \right)$$


Without dropout
With dropout
Classification Error %
2.5
2.0
1.5
1.0
0
200000
400000
600000
800000
1000000
Number of weight updates

DropConnect

basically putitng probability on all node (



No-Drop Network

DropOut Network

DropConnect Network

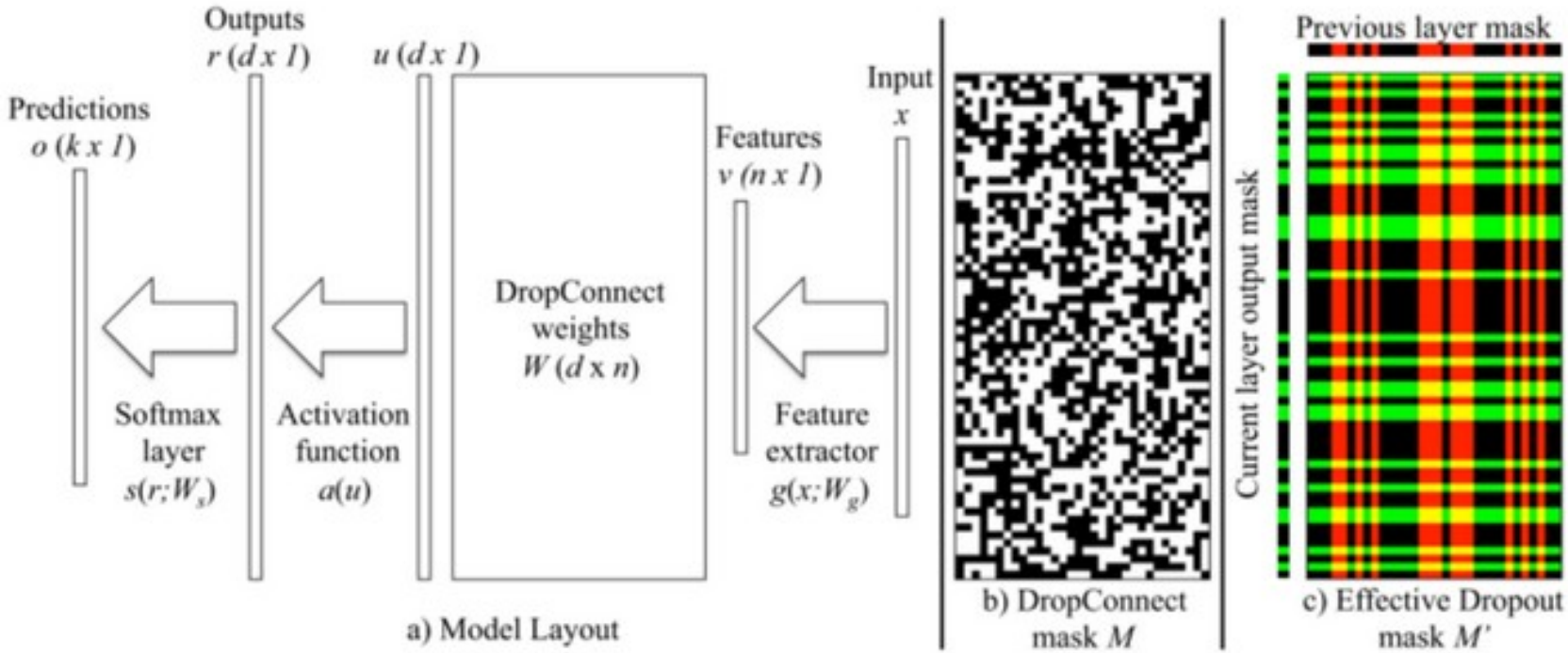


Fig 3. After Wan et al. (2013) (a): An example model layout for a single DropConnect layer. After running feature extractor g() on input x, a random instantiation of the mask M (e.g. (b)), masks out the weight matrix W. The masked weights are multiplied with this feature vector to produce u which is the input to an activation function a and a softmax layer s. For comparison, c) shows an effective weight mask for elements that Dropout uses when applied to the previous layer's output (columns) and this layer's output (rows).

Dropblock

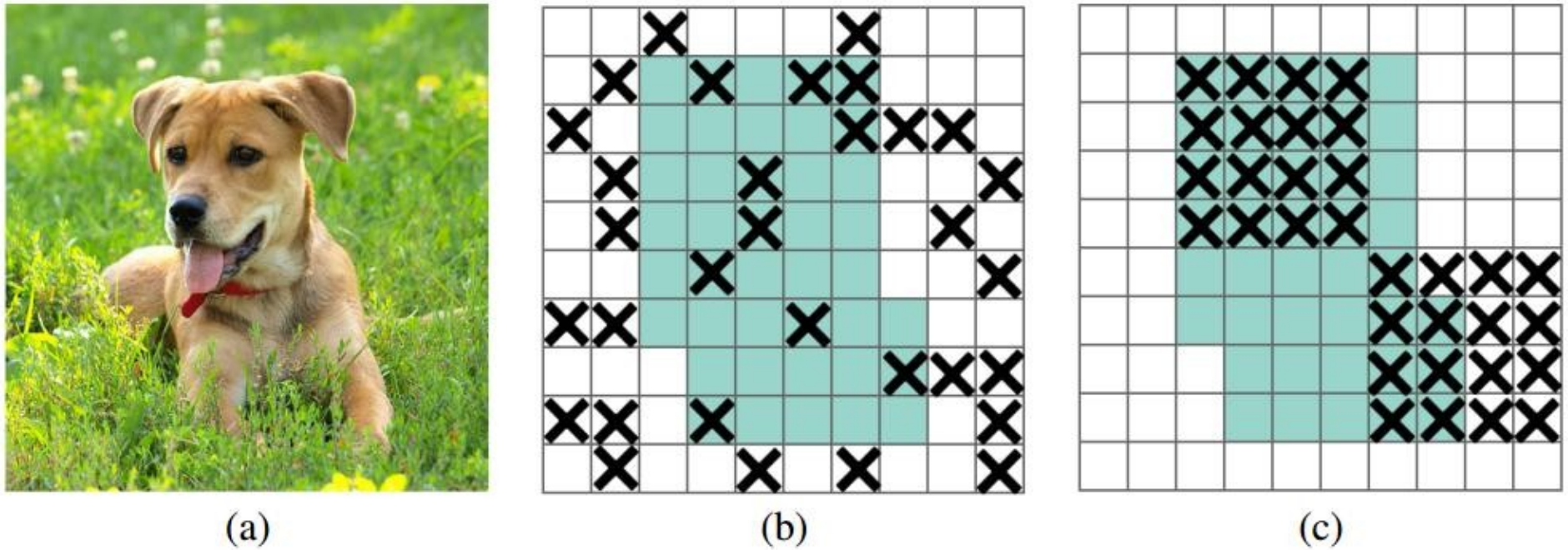
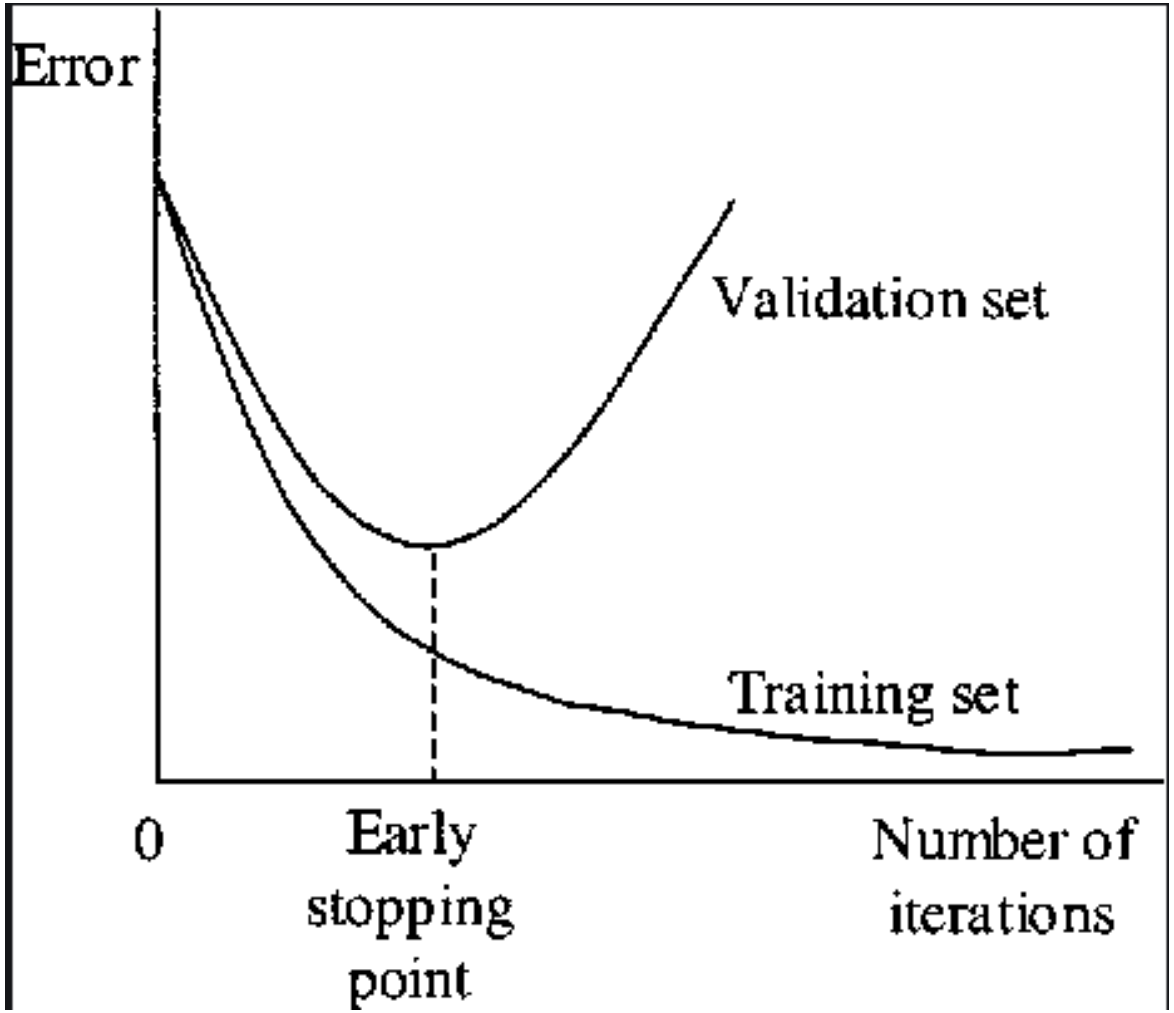


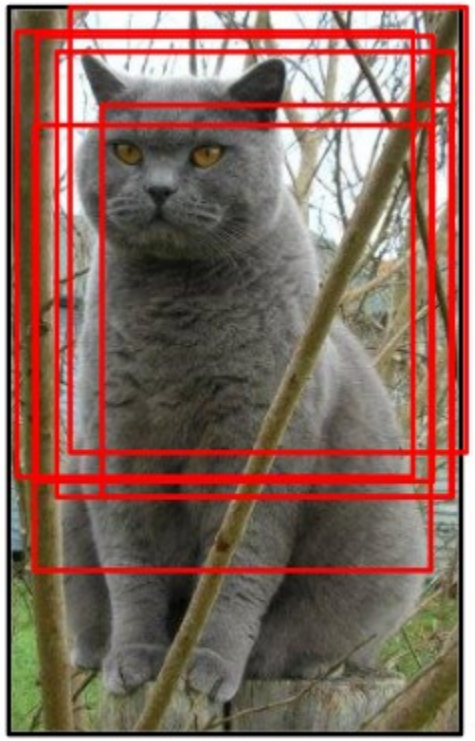
Figure 1: (a) input image to a convolutional neural network. The green regions in (b) and (c) include the activation units which contain semantic information in the input image. Dropping out activations at random is not effective in removing semantic information because nearby activations contain closely related information. Instead, dropping continuous regions can remove certain semantic information (e.g., head or feet) and consequently enforcing remaining units to learn features for classifying input image.

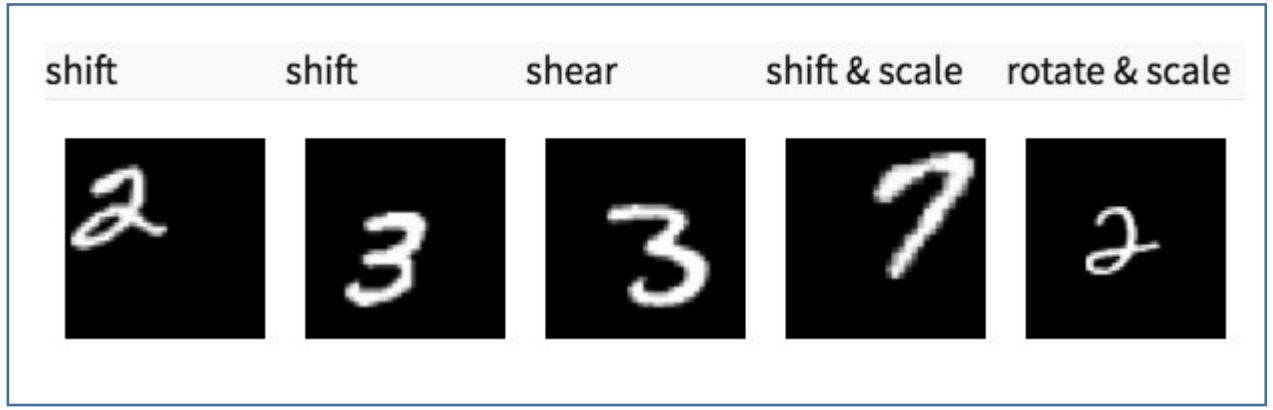
https://blog.csdn.net/Murdock_C

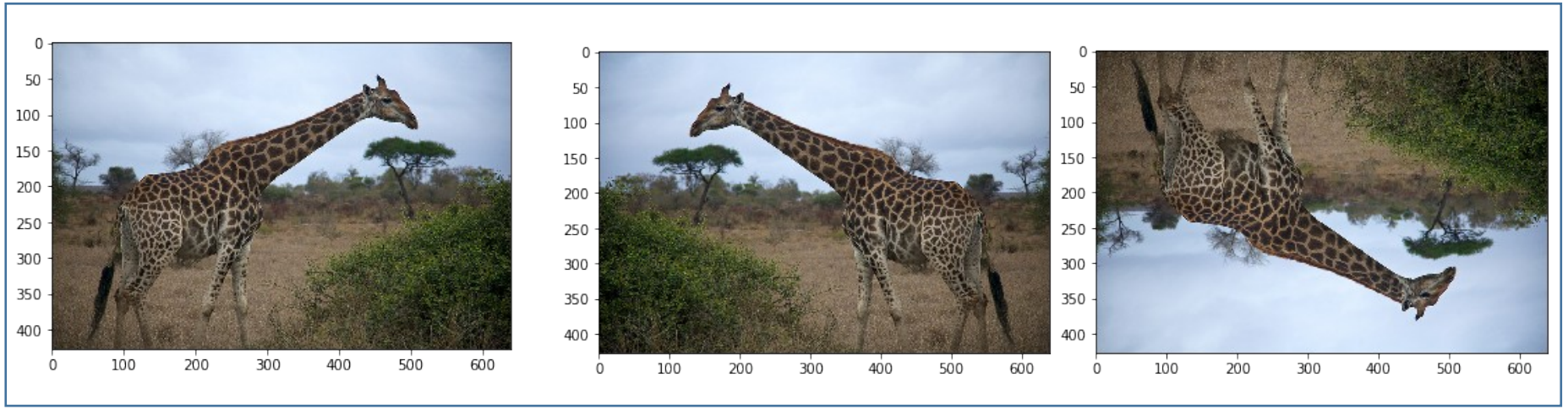
Early stopping – avoid over-training



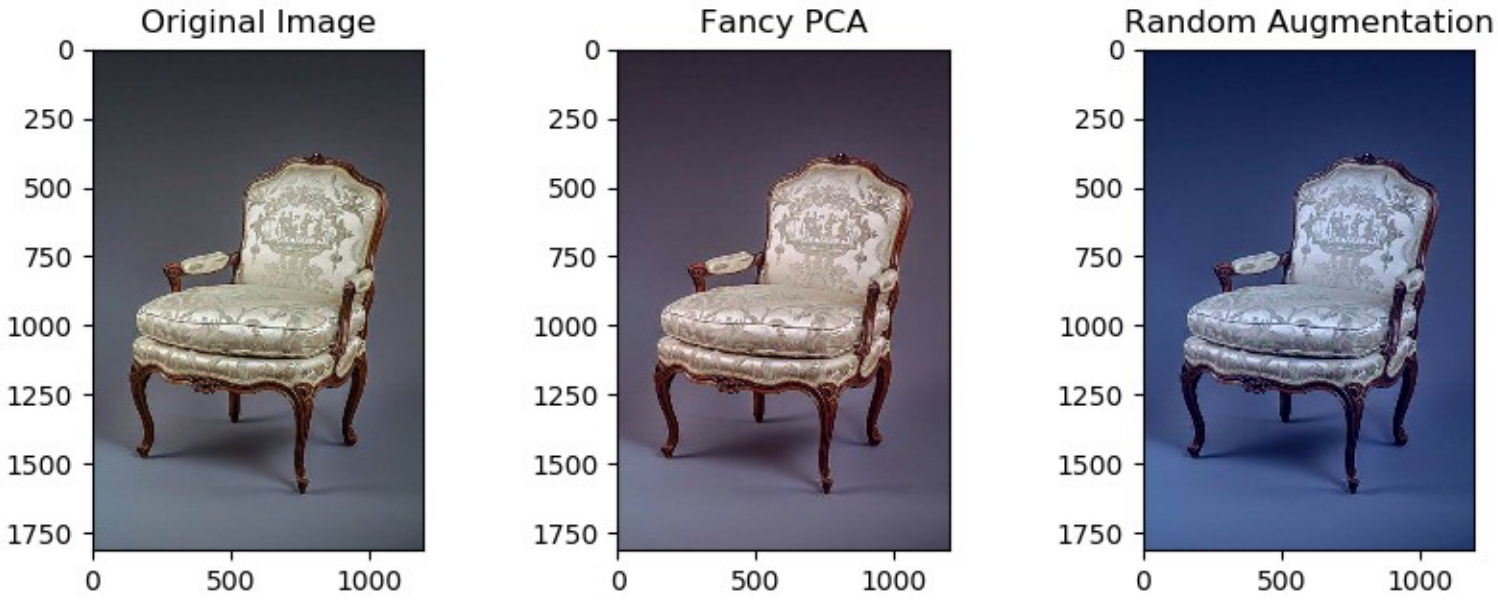
Data augmentation







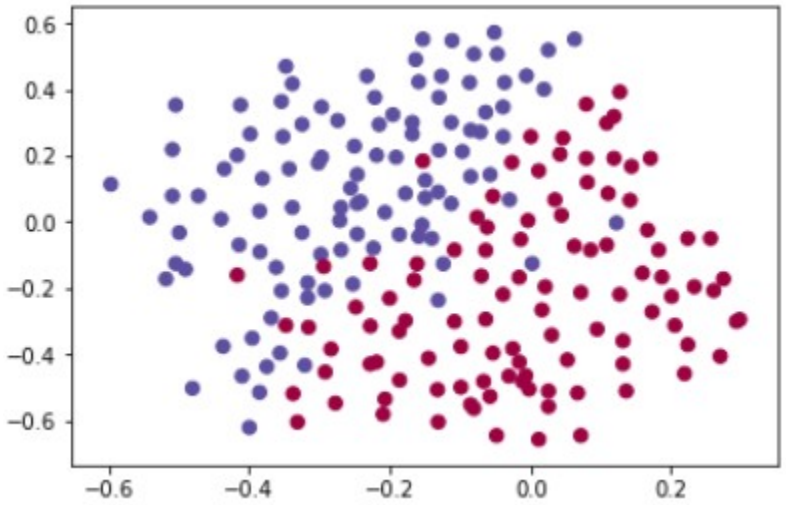
Fancy PCA



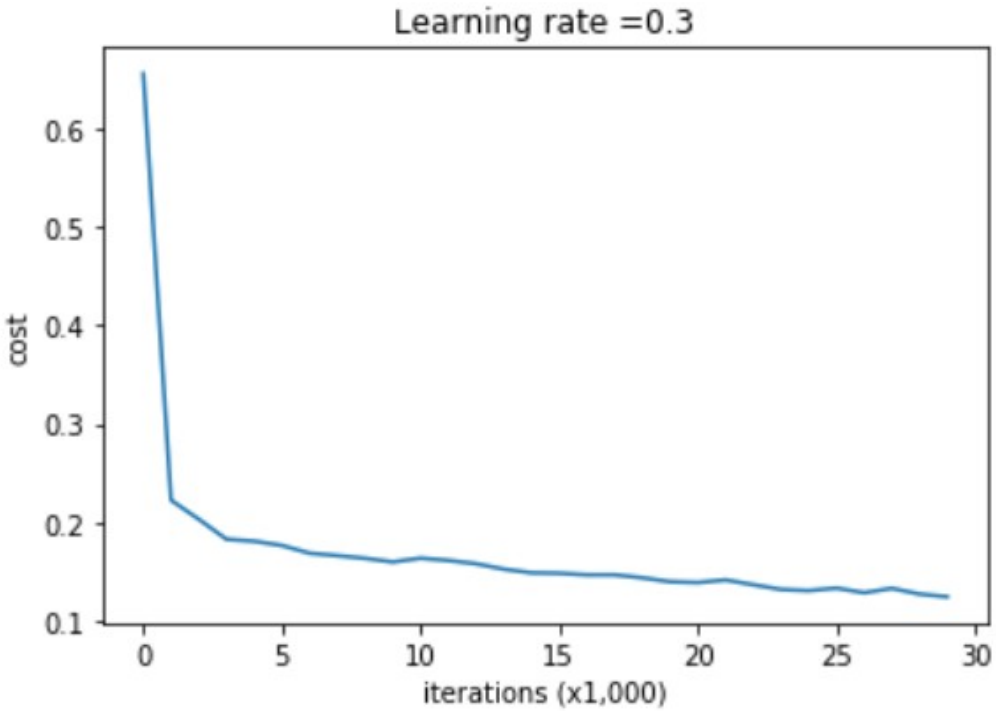
$$I_{xy} = [I_{xy}^R, I_{xy}^G, I_{xy}^B]^T + [p1, p2, p3][\alpha_1\lambda_1, \alpha_2\lambda_2, \alpha_3\lambda_3]^T$$

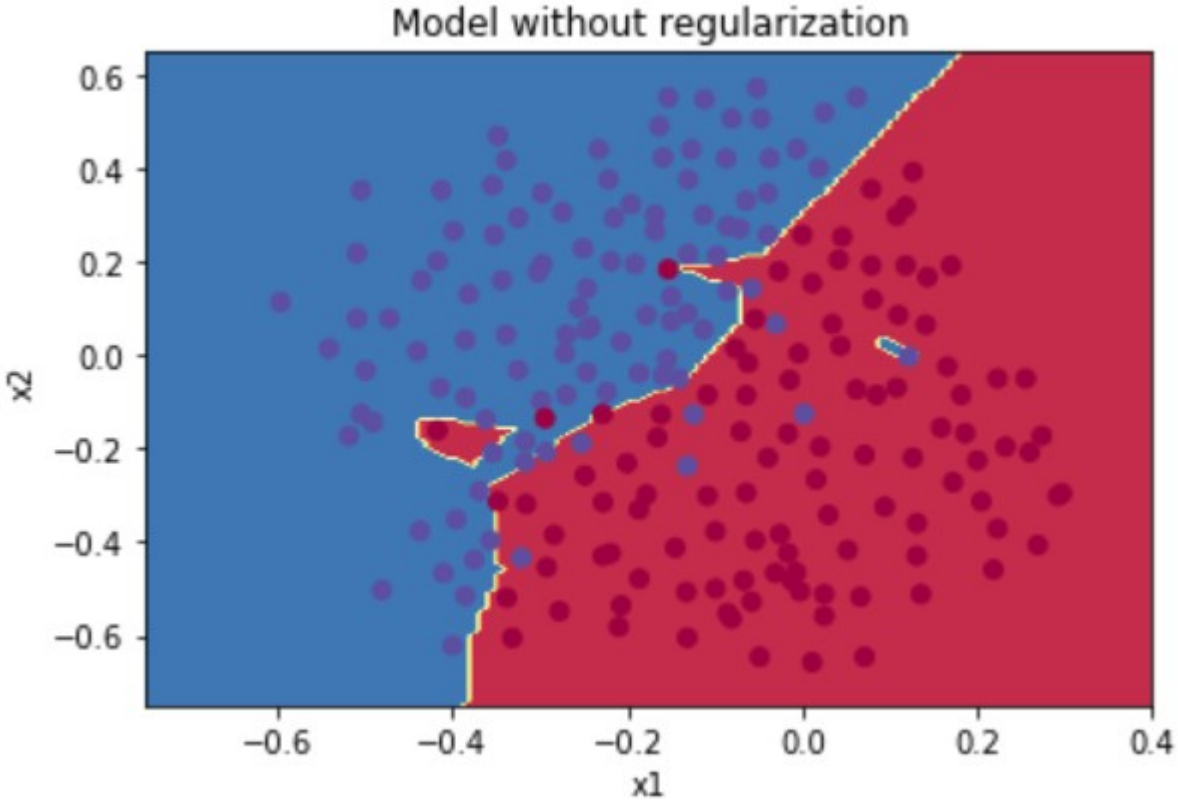
p denote the Eigenvector, \lambda Denote the eigenvalue \alpha is a random nois

Some comparison



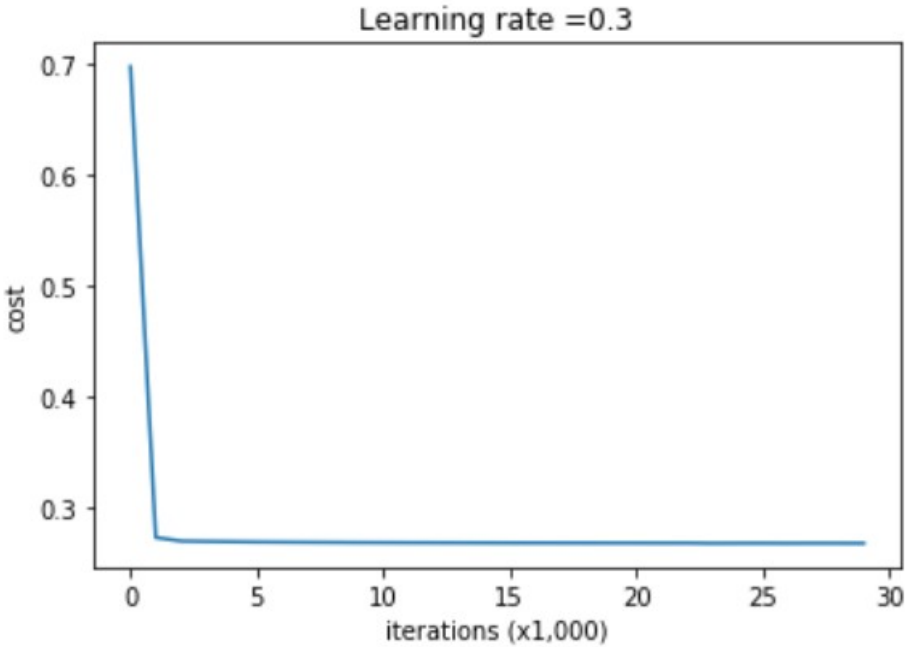
Non-regularized version

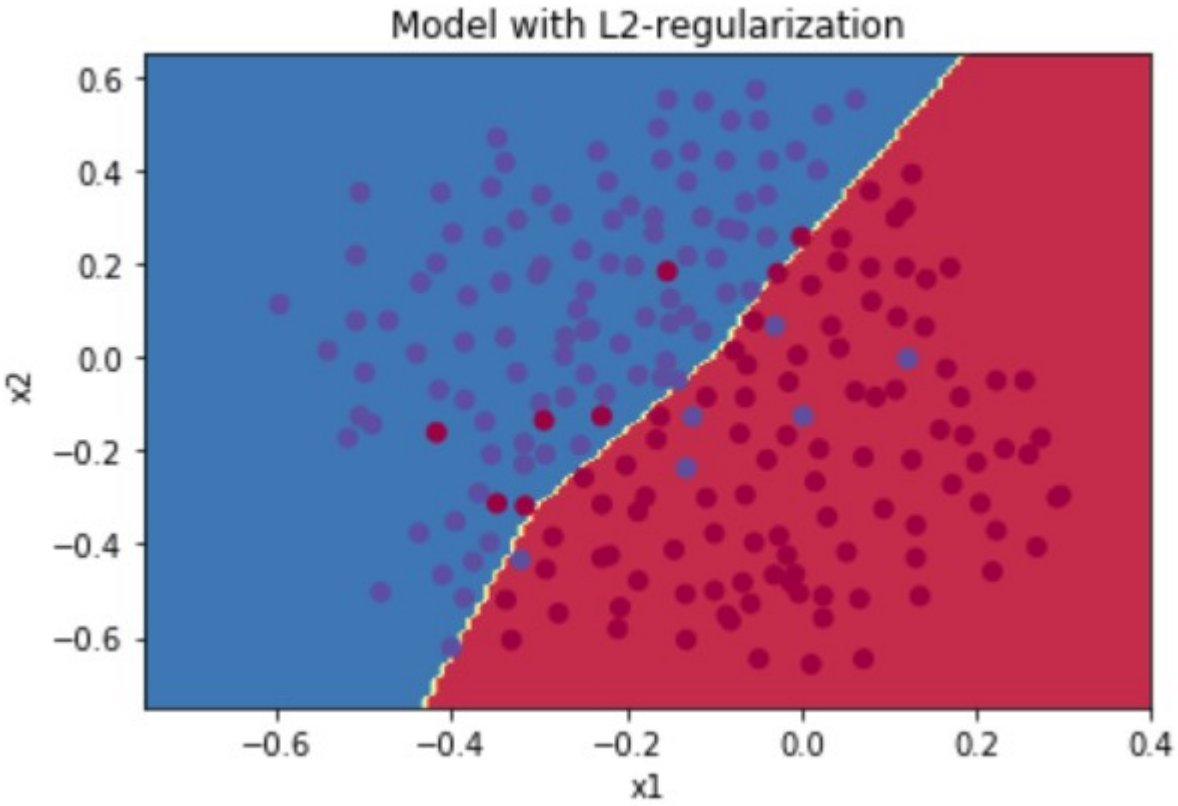





```
On the training set:
Accuracy: 0.9478672985781991
On the test set:
Accuracy: 0.915
```

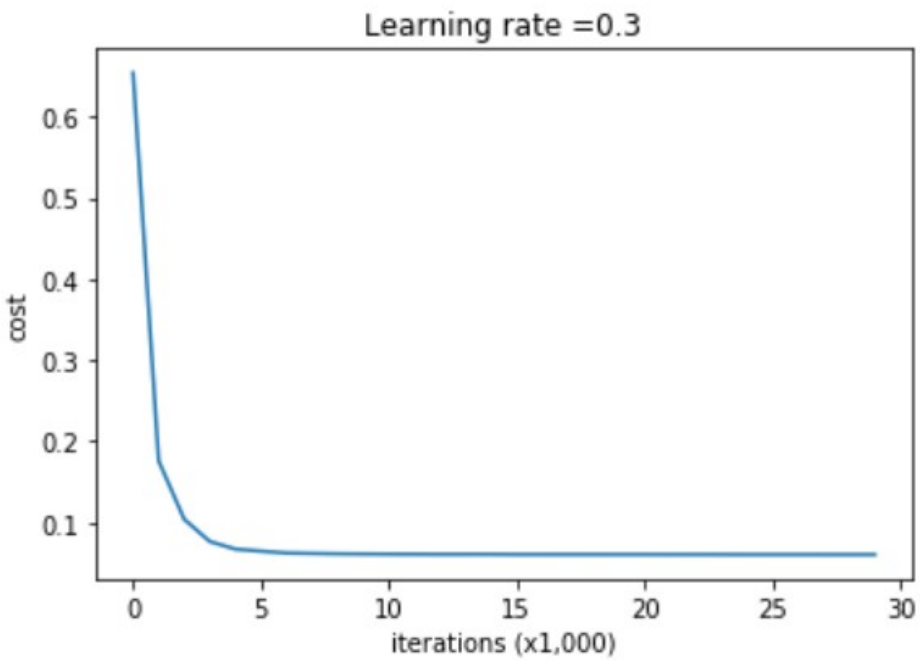
L2 regularization

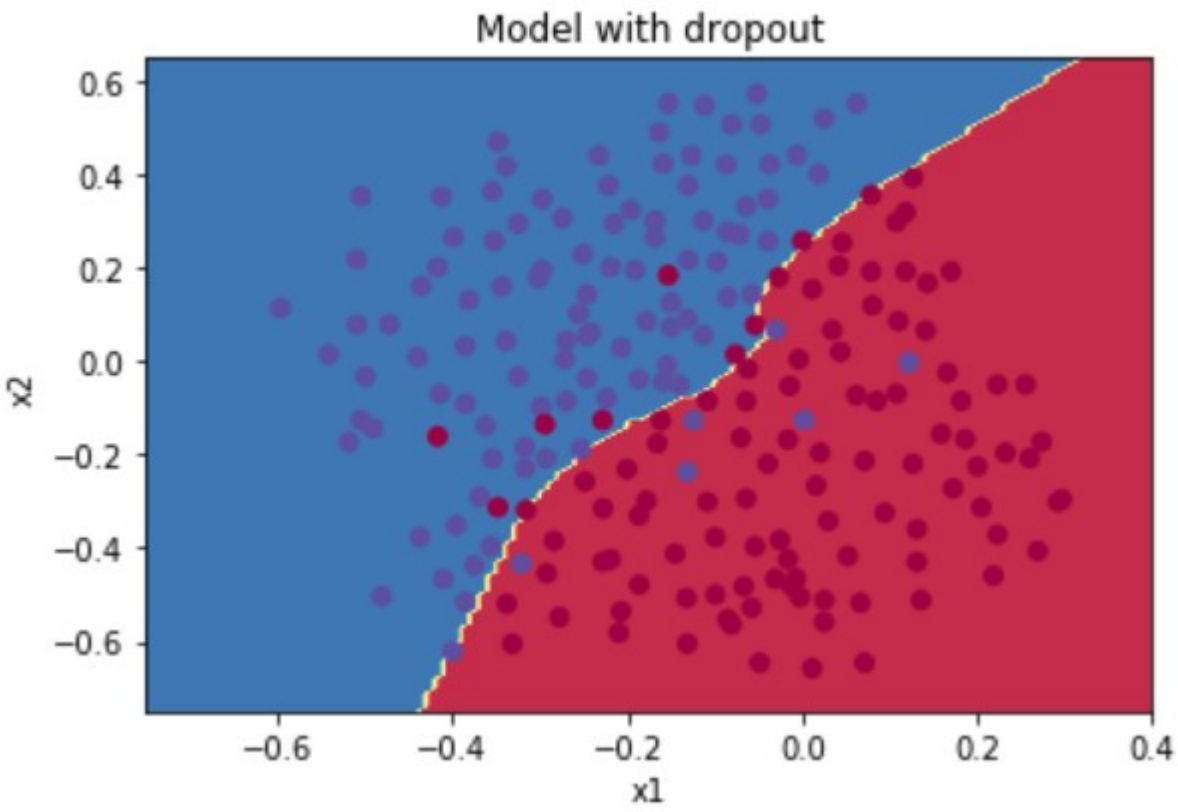





```
On the train set:
Accuracy: 0.9383886255924171
On the test set:
Accuracy: 0.93
```

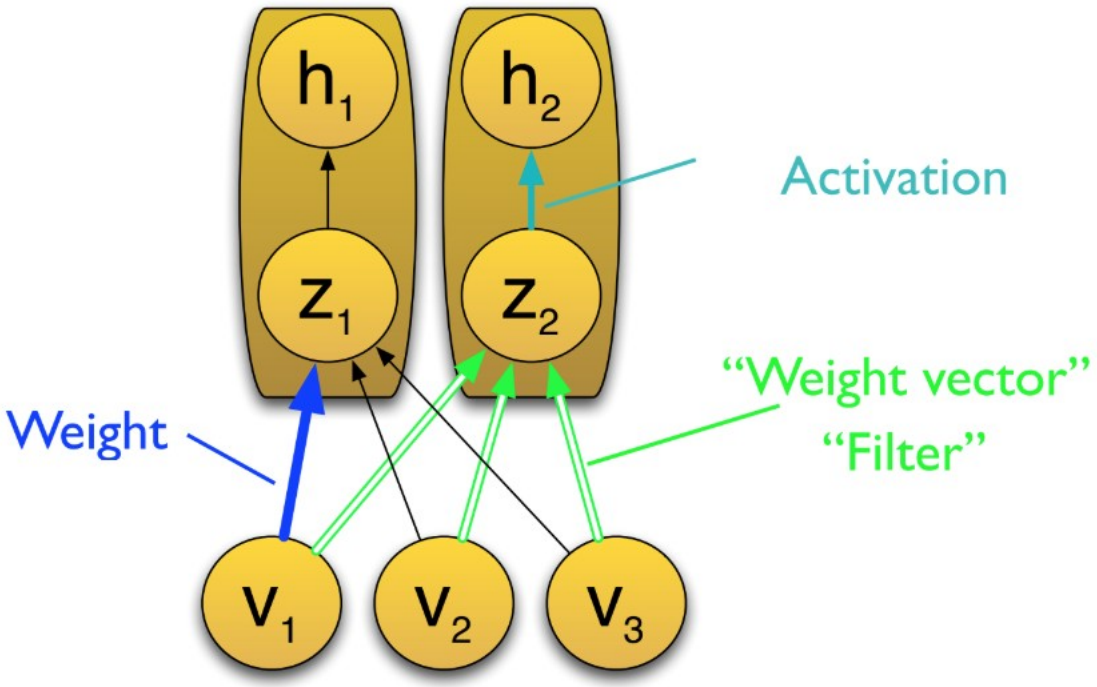
Dropout

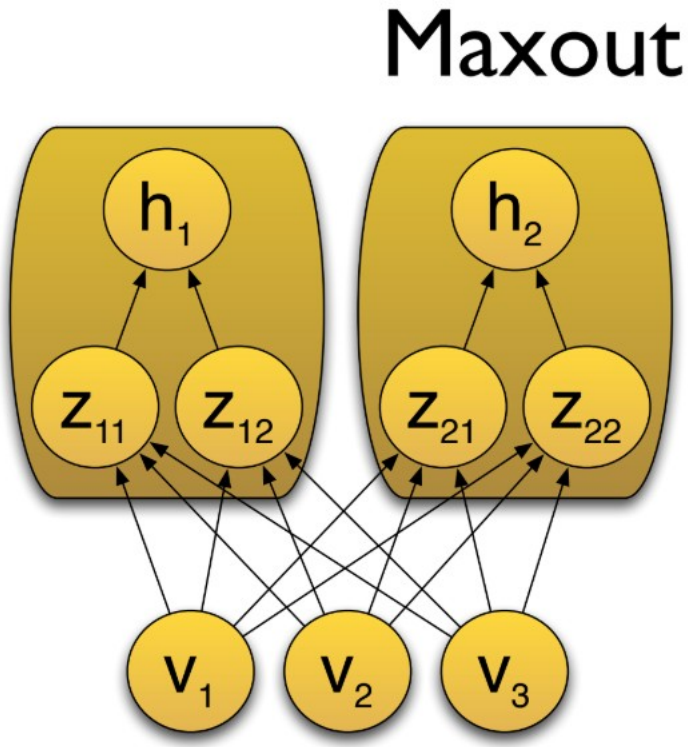





```
On the train set:
Accuracy: 0.9289099526066351
On the test set:
Accuracy: 0.95
```

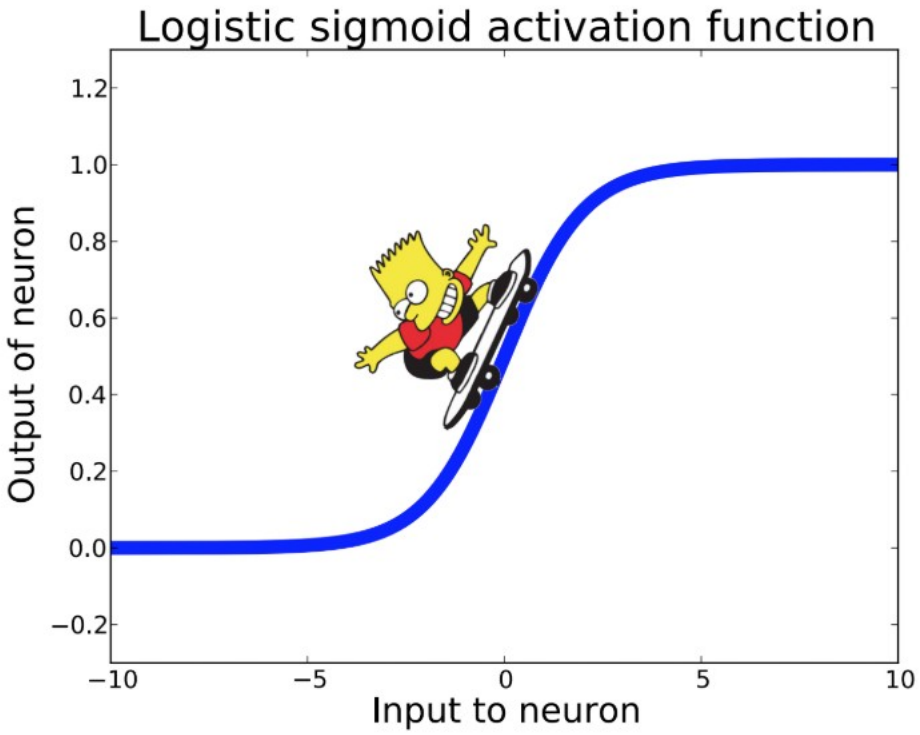
Maxout activation

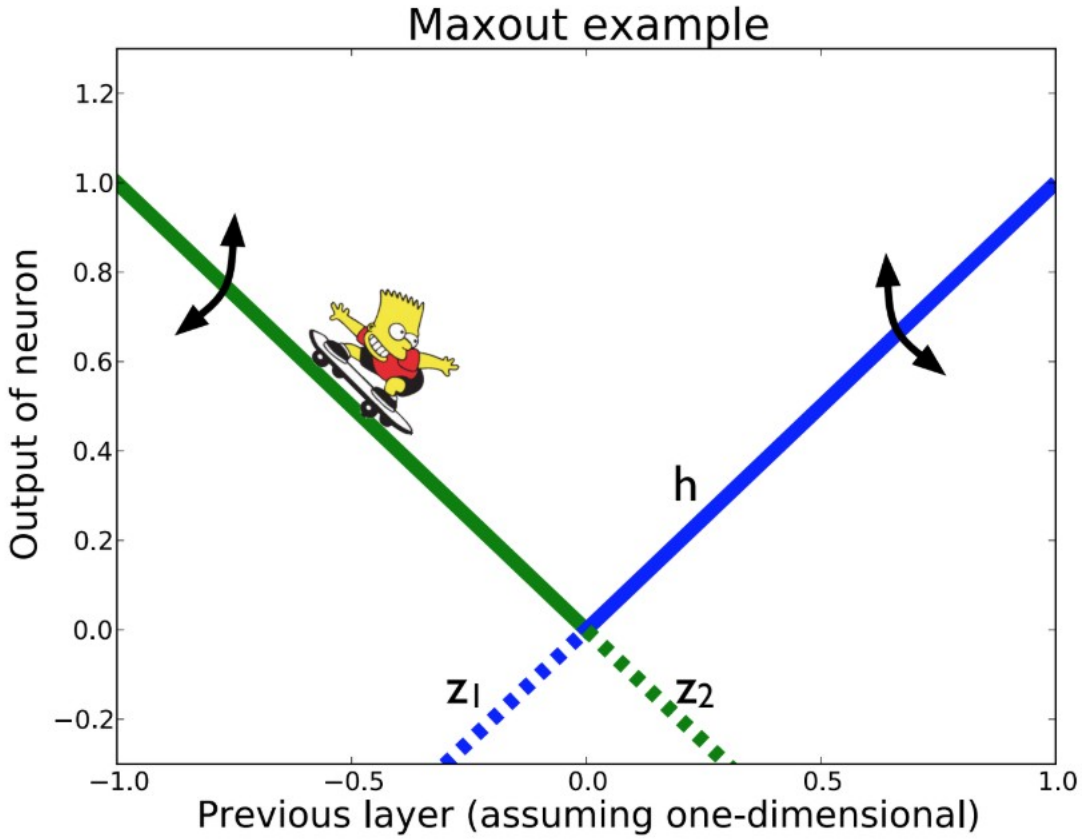




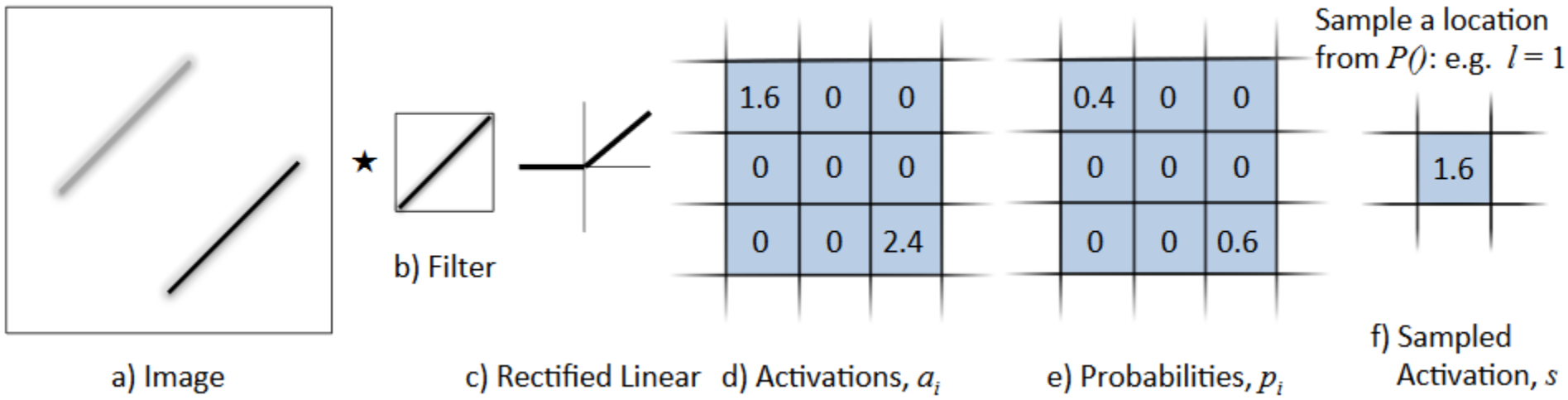
$h_i = \max_j z_{ij}$

Gradients everywhere!





Stochastic pooling

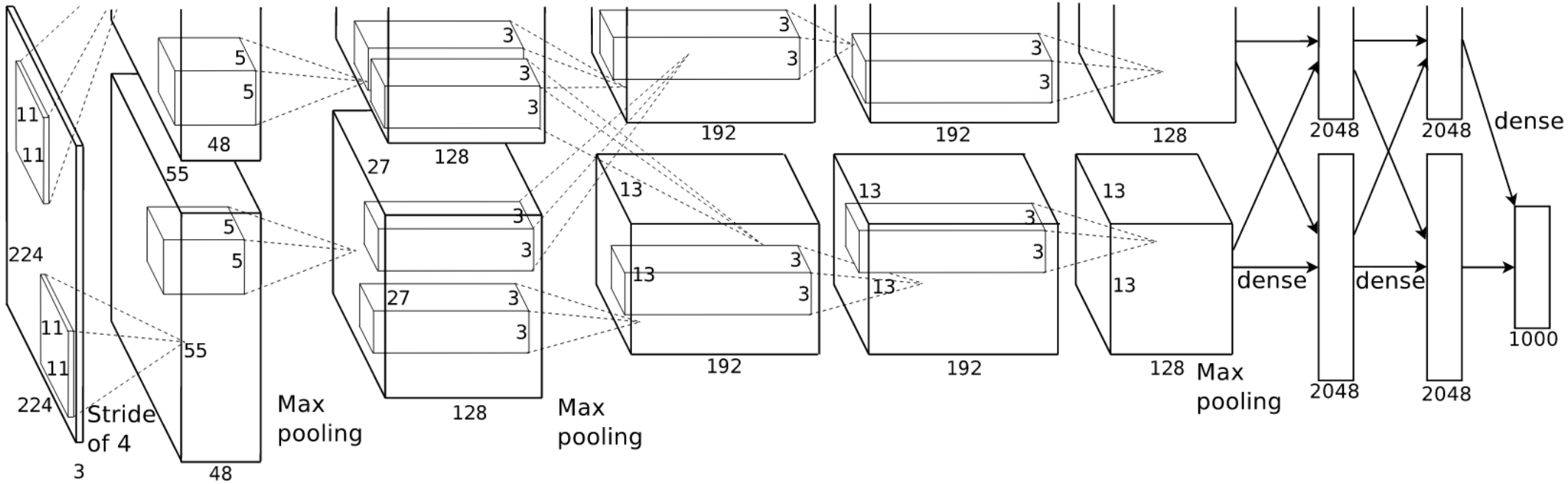


$$p_i = \frac{a_i}{\sum_{k \in R_j} a_k} \qquad s_j = a_l \quad \text{where } l \sim P(p_1, \ldots, p_{|R_j|})$$

Popular CNN architecture

- Alexnet
- Zfnet
- VGG
- Inception (v1-v4)
- Resnet
- Resnext
- ...

AlexNet


224
224
3
11
11
11
11
Stride
of 4
55
55
48
48
5
5
5
5
Max
pooling
27
27
128
128
3
3
3
3
3
3
3
3
Max
pooling
13
13
13
13
192
192
192
192
3
3
3
3
3
3
3
3
128
128
13
13
Max
pooling
dense
dense
dense
2048
2048
2048
2048
1000

Zfnet

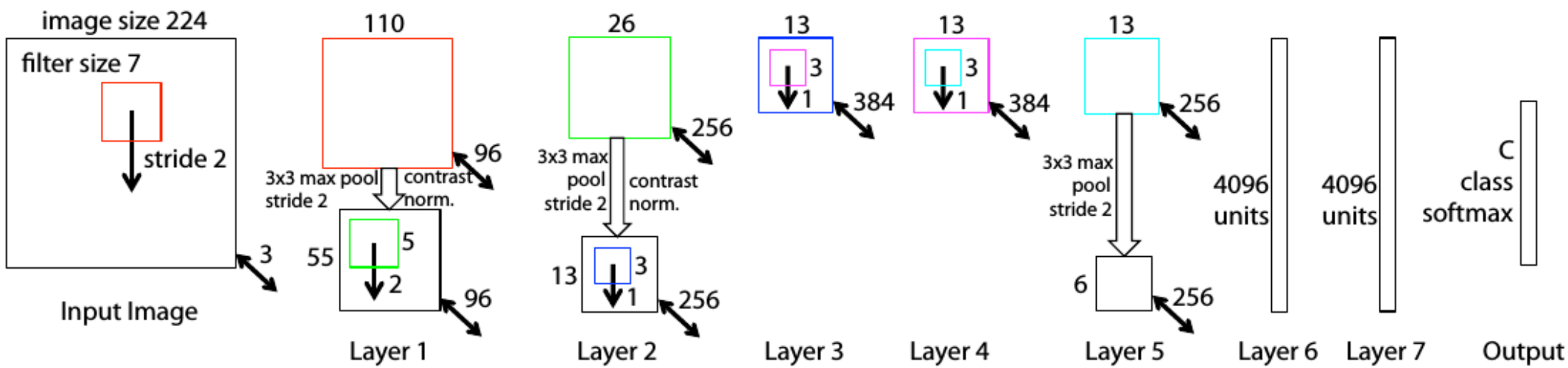


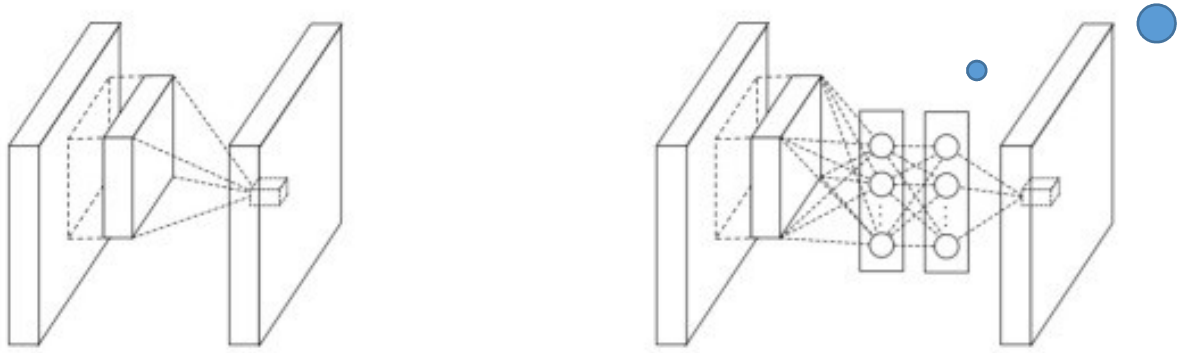
image size 224
filter size 7
stride 2
3
Input Image
110
96
3x3 max pool
stride 2
contrast
norm.
55
5
2
96
Layer 1
26
256
3x3 max
pool
stride 2
contrast
norm.
13
3
1
256
Layer 2
13
3
1
384
Layer 3
13
3
1
384
Layer 4
13
256
3x3 max
pool
stride 2
6
256
Layer 5
4096
units
Layer 6
4096
units
Layer 7
C
class
softmax
Output

VGGnet

3x3 conv, 64
3x3 conv, 64
pool/2
3x3 conv, 128
3x3 conv, 128
pool/2
3x3 conv, 256
3x3 conv, 256
3x3 conv, 256
pool/2
3x3 conv, 512
3x3 conv, 512
3x3 conv, 512
pool/2
3x3 conv, 512
3x3 conv, 512
3x3 conv, 512
pool/2
fc 4096
fc 4096
fc 4096
Size:224
Size:112
Size:56
Size:28
Size:14
Size:7

Network in network

MLP-conv layer



(a) Linear convolution layer (b) Mlpconv layer

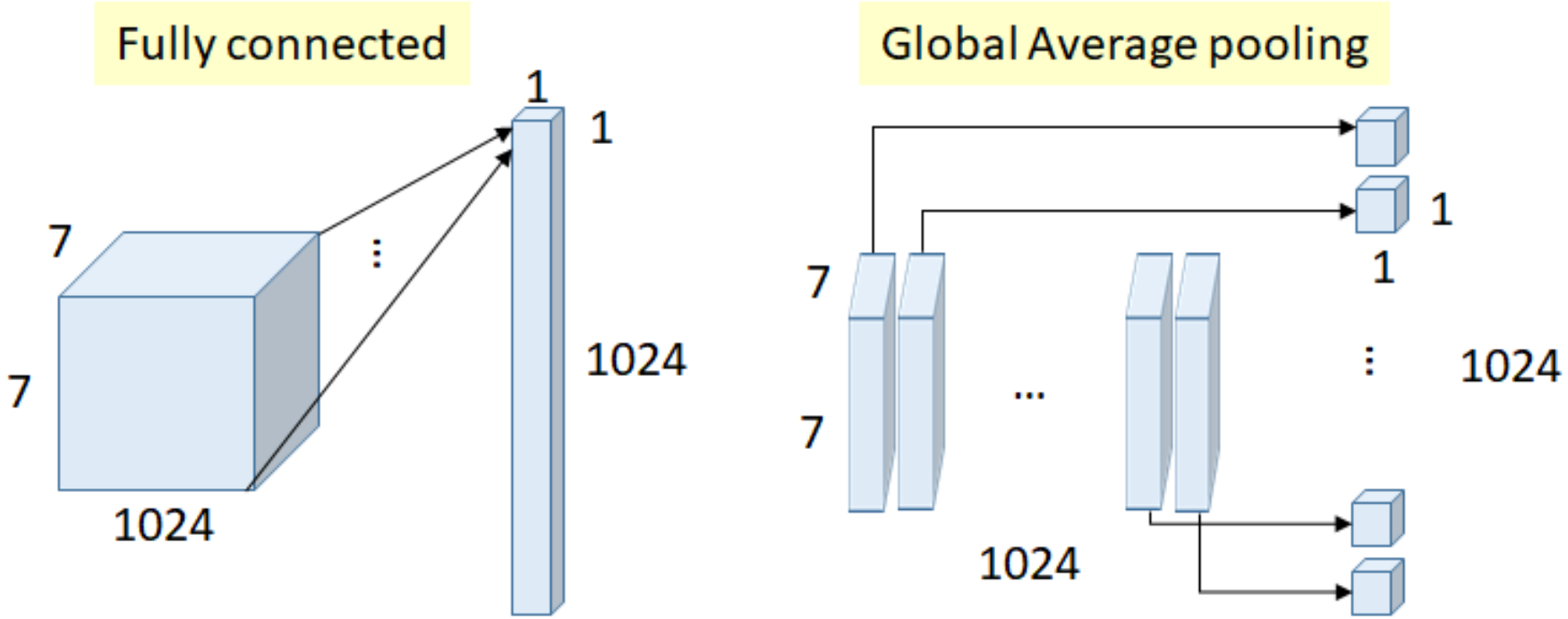
Figure 1: Comparison of linear convolution layer and mlpconv layer. The linear convolution layer includes a linear filter while the mlpconv layer includes a micro network (we choose the multilayer perceptron in this paper). Both layers map the local receptive field to a confidence value of the latent concept.

Representations that achieve good abstraction are non-linear function of the input data!

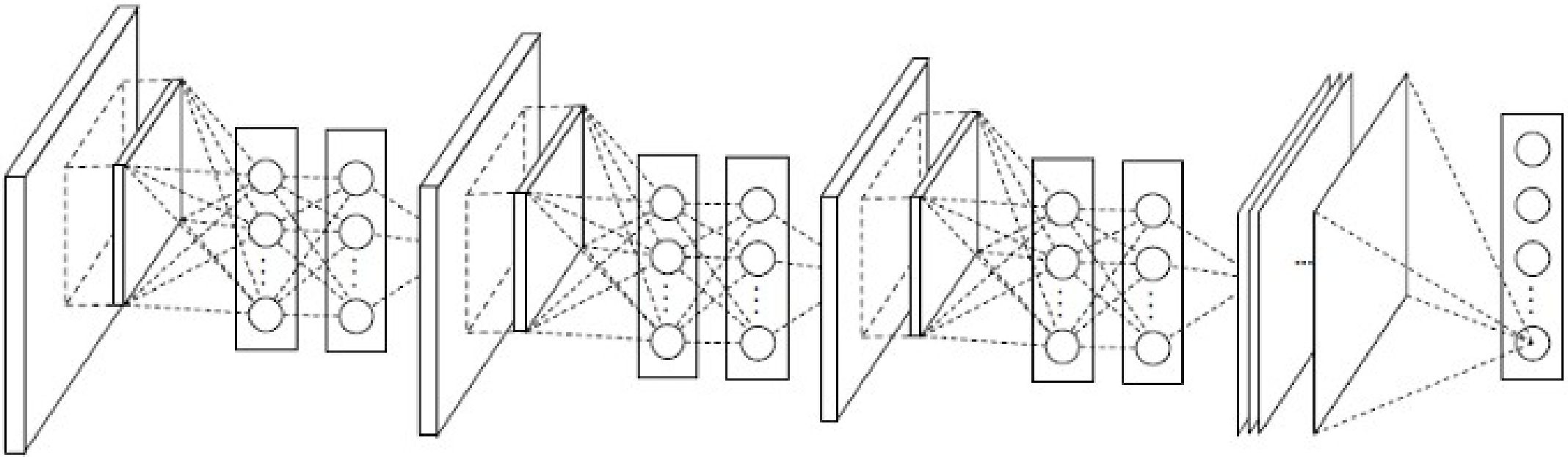
$$f_{i,j,k} = \max(w_k^T x_{i,j}, 0).$$

$$\begin{aligned} f^1_{i,j,k_1} &= \max({w^1_{k_1}}^T x_{i,j} + b_{k_1}, 0). \\ &\vdots \\ f^n_{i,j,k_n} &= \max({w^n_{k_n}}^T f^{n-1}_{i,j} + b_{k_n}, 0). \end{aligned}$$

Network in network

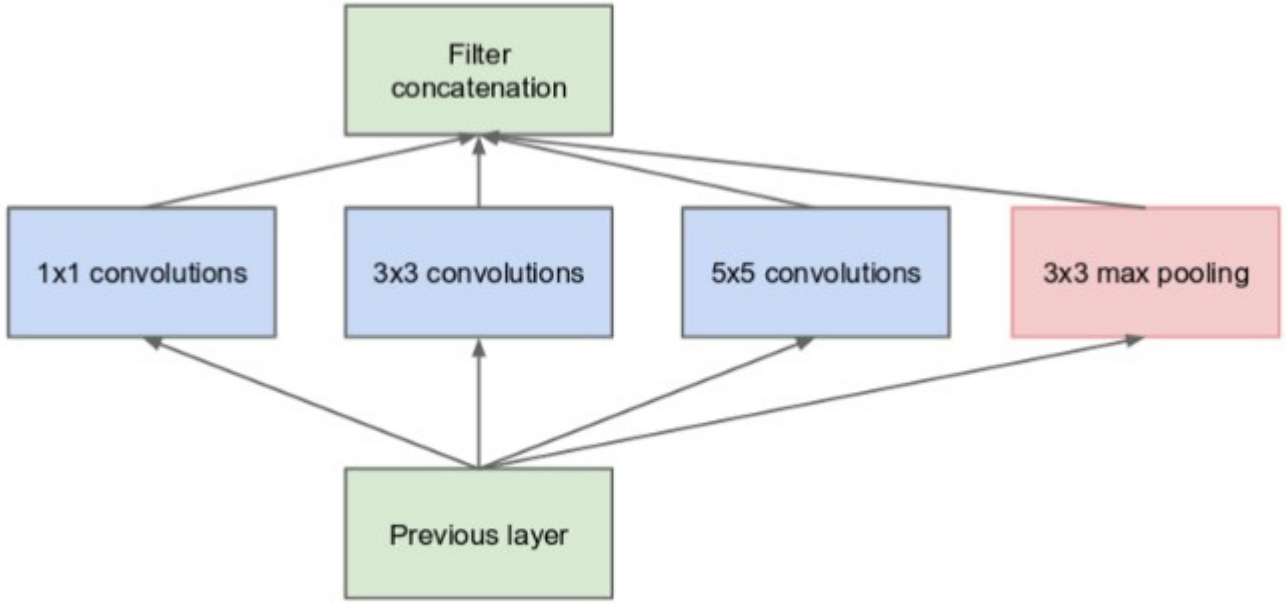


No learnable parameter!
Can work with images of different resolution!

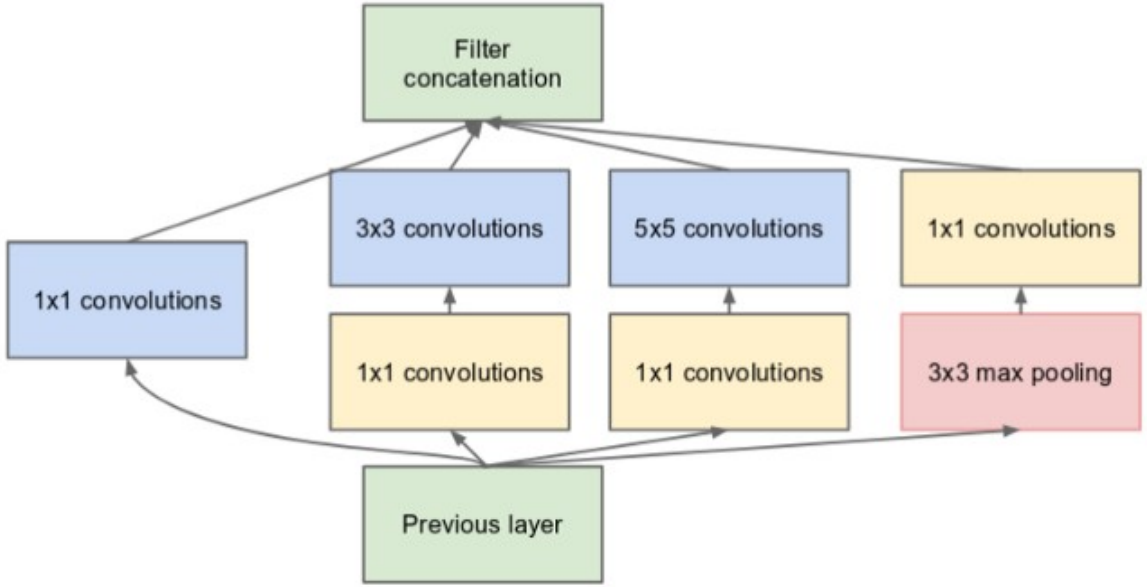


Inception architecture

What is a good kernel size for DOG feature extraction?

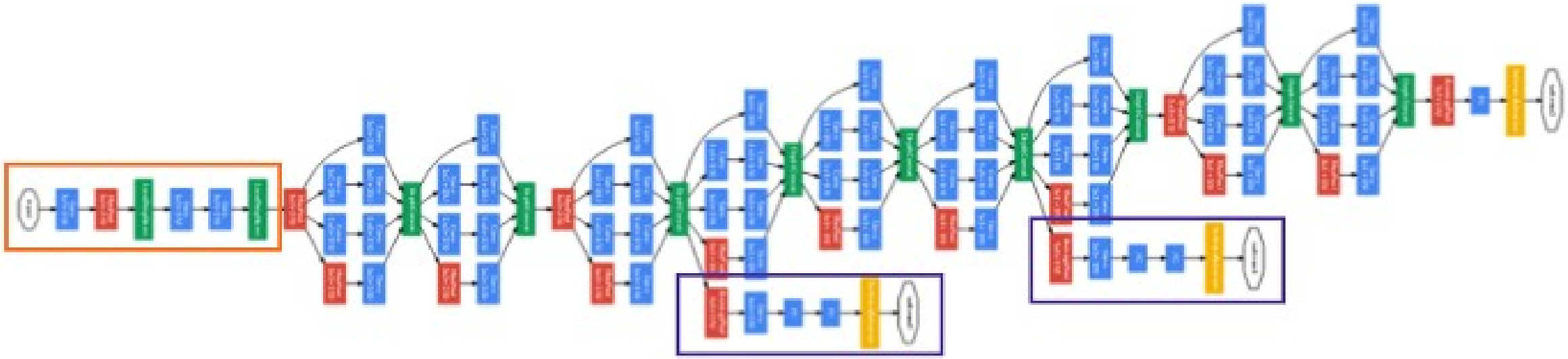


(a) Inception module, naïve version



(b) Inception module with dimension reductions

Inception v1 (Googlenet)

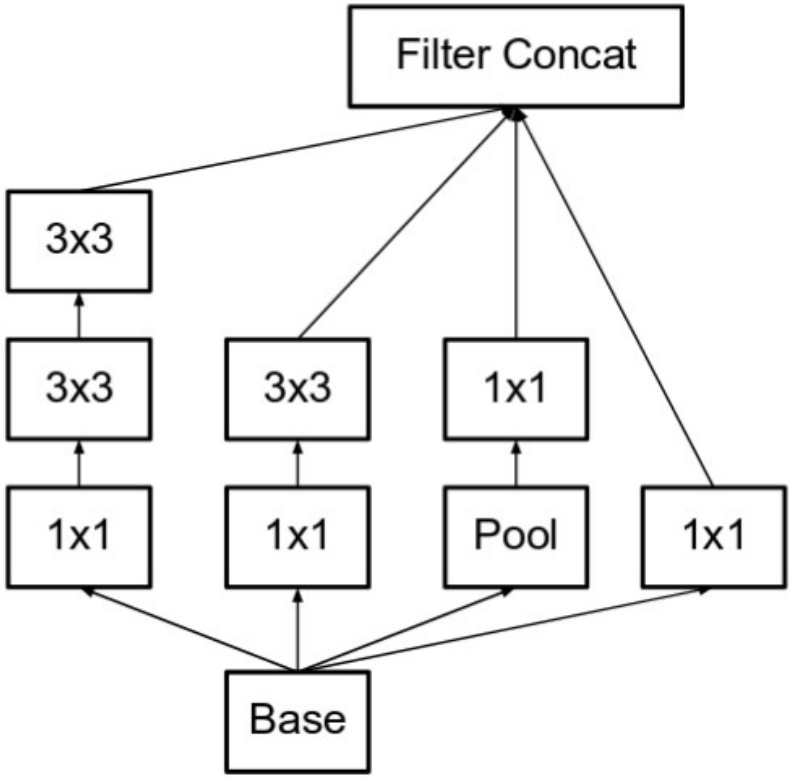


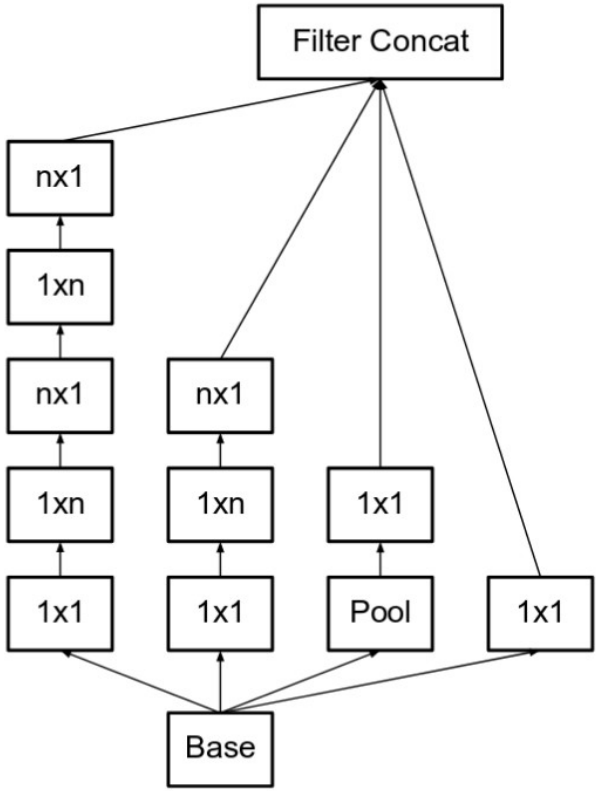
```
# The total loss used by the inception net during training.
total_loss = real_loss + 0.3 * aux_loss_1 + 0.3 * aux_loss_2
```

9 inception module, 22 layers, global average pooling at the end, 2 intermediate classifier to take care
Of the ‘dying out’ effect

Inception v2

- Filter separability and small filter size





Inception v3

- The auxiliary classifiers do not contribute much except during the end of the training phase!

1. RMSProp Optimizer.
2. Factorized 7x7 convolutions.
3. BatchNorm in the Auxillary Classifiers.
4. Label Smoothing (A type of regularizing component added to the loss formula that prevents the network from becoming too confident about a class. Prevents over fitting).

Label smoothing

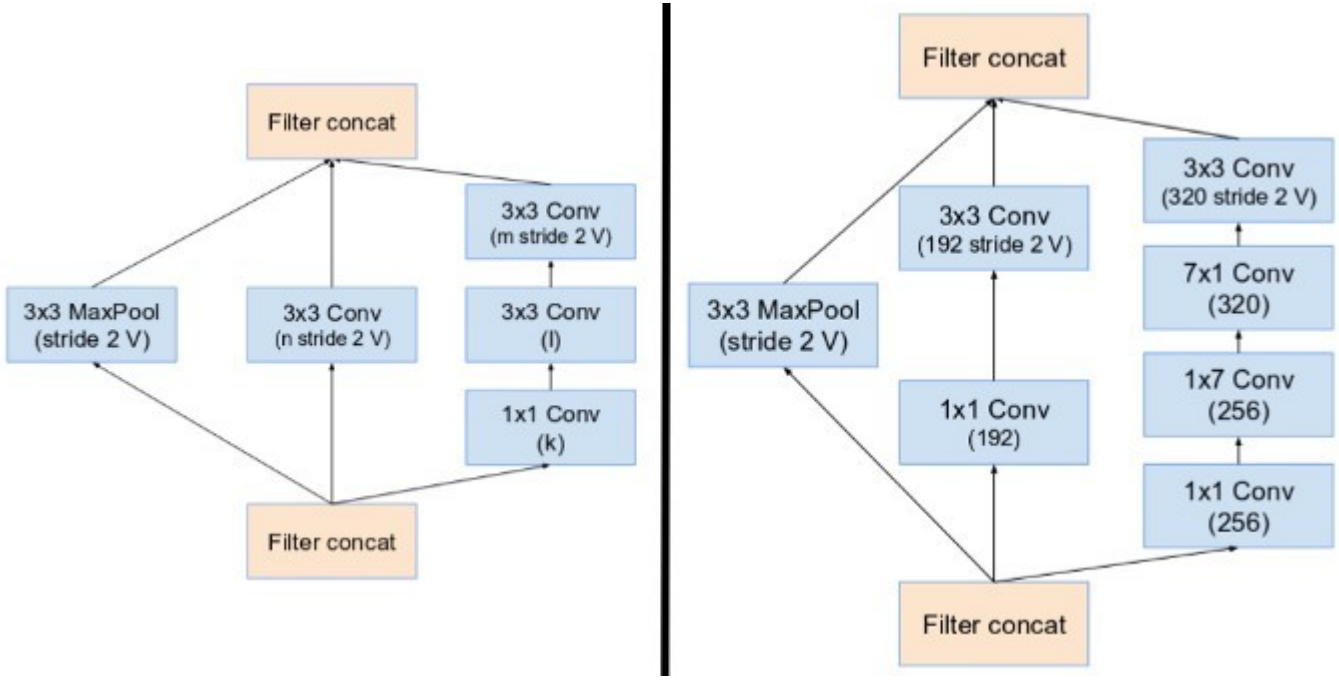
Usually, we have output vectors provided to us as
$y_{label} = [1, 0, 0, 0...0]$.

Softmax output is usually of the form
$y_{out} = [0.87, 0.001, 0.04, 0.1, ...0.03]$.

Maximum likelihood learning with a softmax classifier and hard targets may actually never converge, the softmax can never predict a probability of exactly 0 or exactly 1, so it will continue to learn larger and larger weights, making more extreme predictions. forever

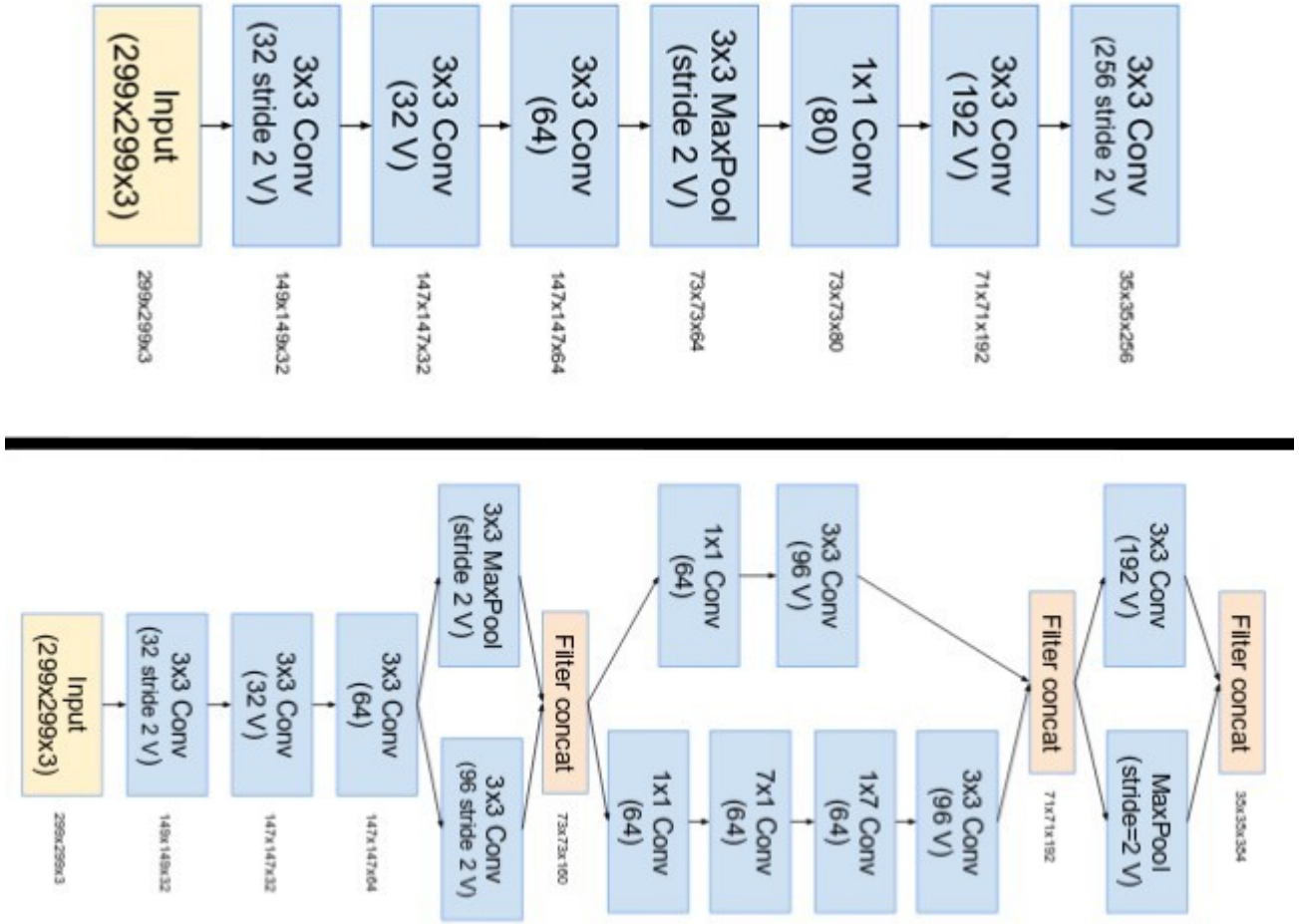
$$\boxed{y_{label} = [1 - \epsilon, \frac{\epsilon}{K-1}, \frac{\epsilon}{K-1}, \frac{\epsilon}{K-1} \cdots \frac{\epsilon}{K-1}].}$$

Inception v4

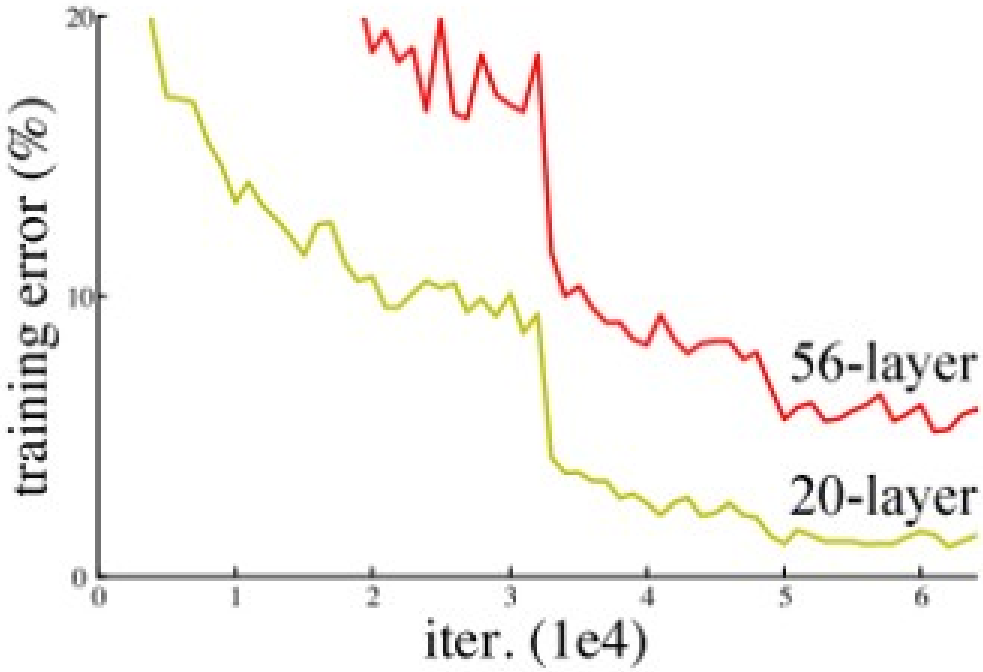


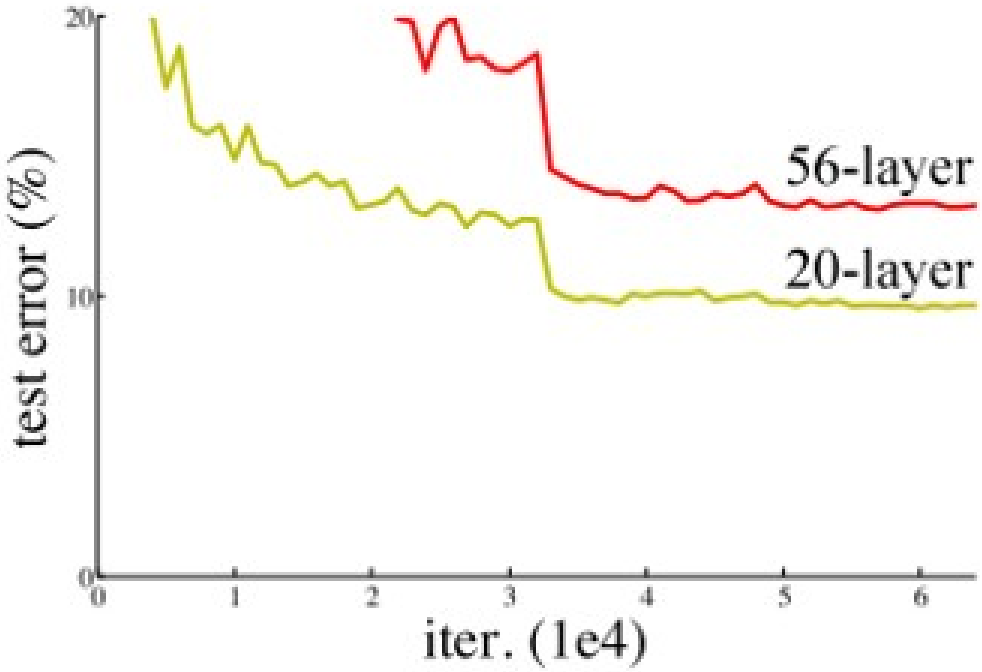
Reduce spatial dimensions!

Simplify the “stem”

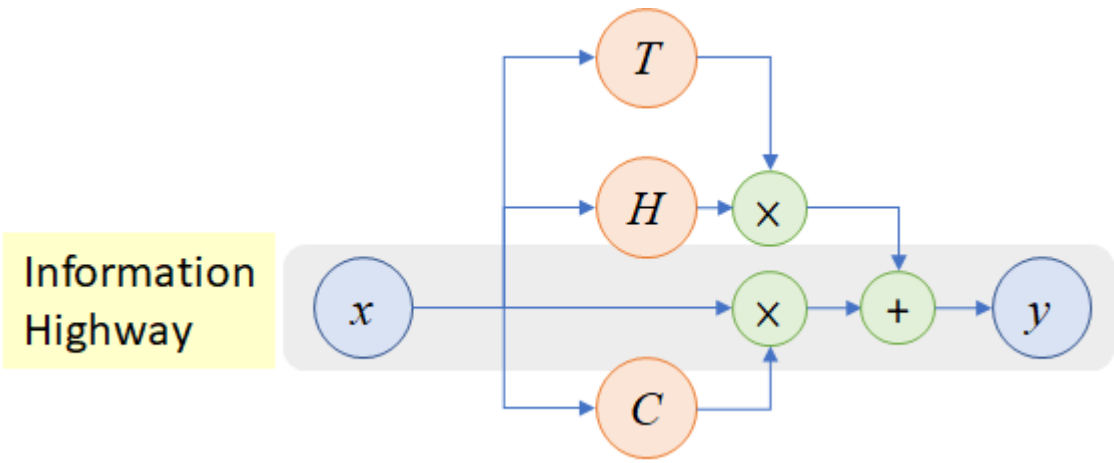


Going deeper in CNN





Highway network



In plain network

$$\mathbf{y} = H(\mathbf{x}, \mathbf{W_H}).$$

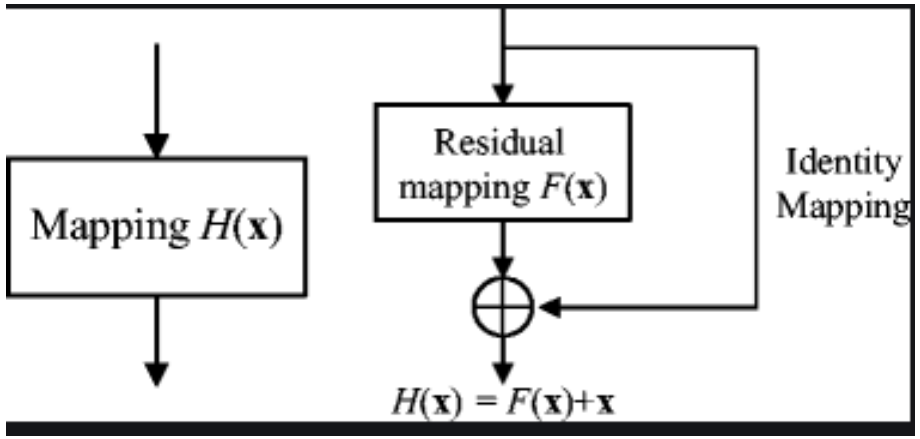
$$\mathbf{y} = H(\mathbf{x}, \mathbf{W_H}) \cdot T(\mathbf{x}, \mathbf{W_T}) + \mathbf{x} \cdot C(\mathbf{x}, \mathbf{W_C}).$$

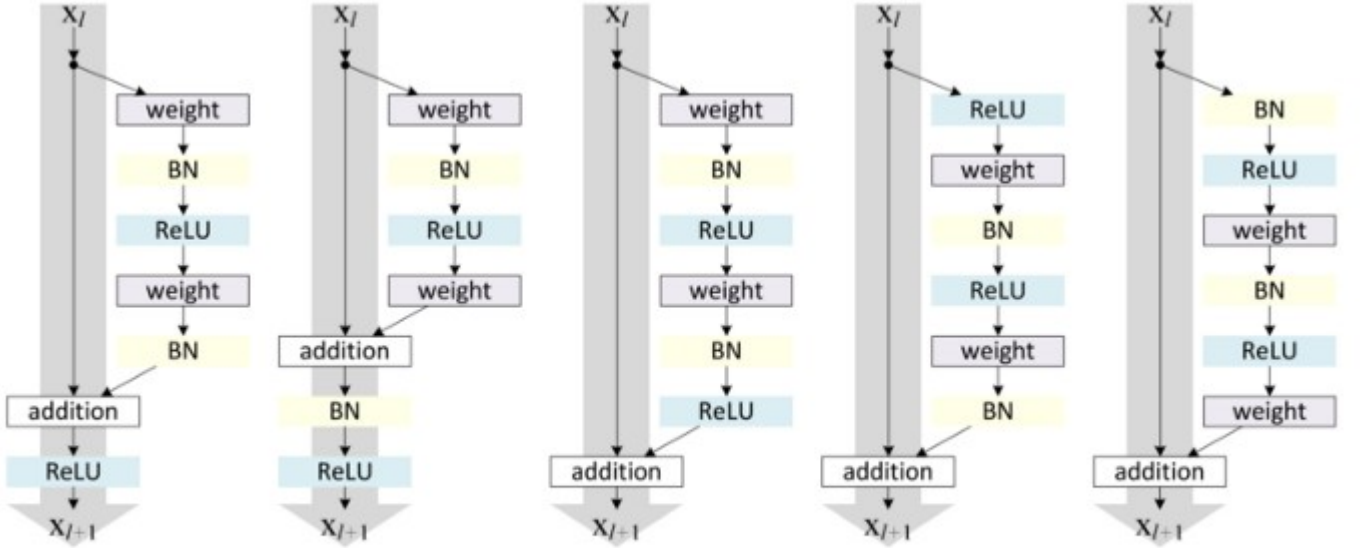
T: Transform gate $T(\mathbf{x}) = \sigma(\mathbf{W_T}^T \mathbf{x} + \mathbf{b_T})$

C: Carry gate, usually C = 1 - T

$$\boxed{\mathbf{y} = \begin{cases} \mathbf{x}, & \text{if } T(\mathbf{x}, \mathbf{W_T}) = \mathbf{0}, \\ H(\mathbf{x}, \mathbf{W_H}), & \text{if } T(\mathbf{x}, \mathbf{W_T}) = \mathbf{1}. \end{cases}}$$

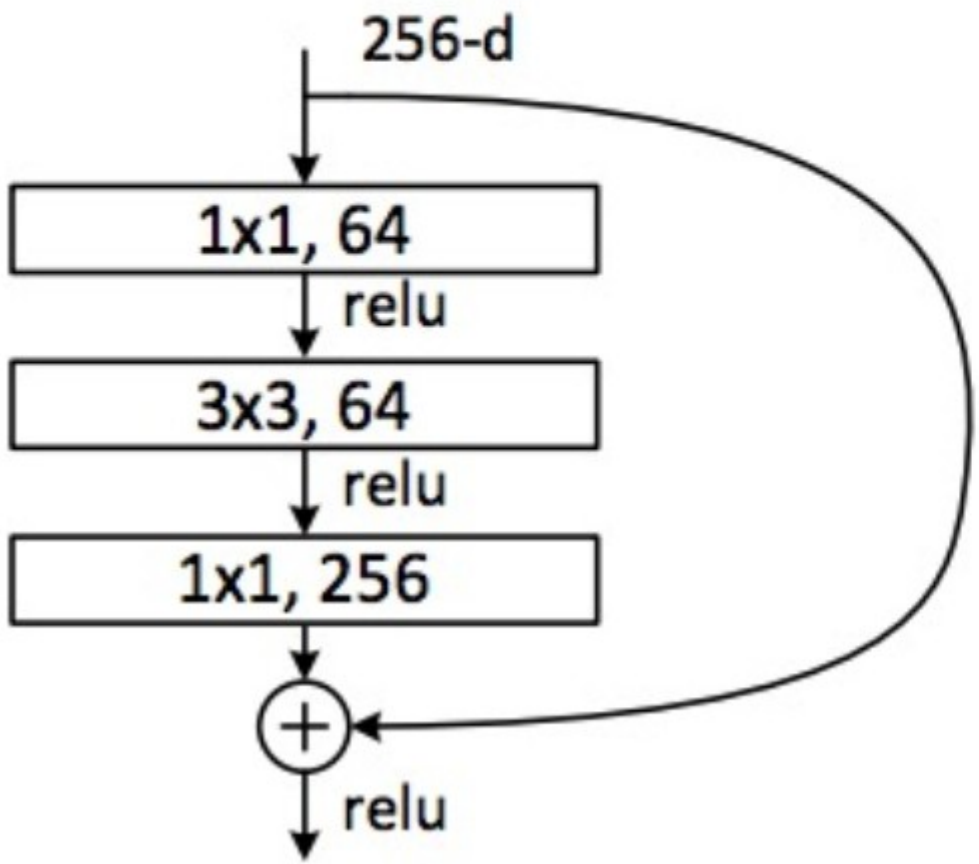
Residual networks



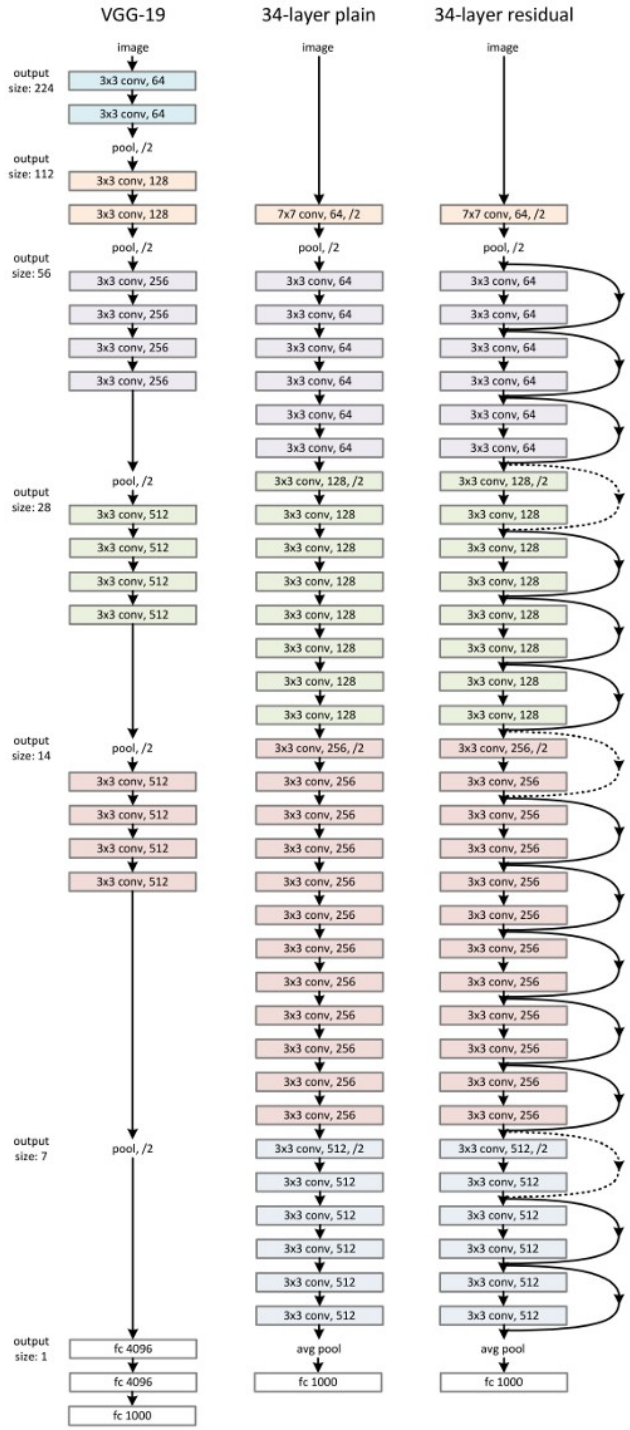


(a) original (b) BN after addition (c) ReLU before addition (d) ReLU-only pre-activation (e) **full pre-activation**

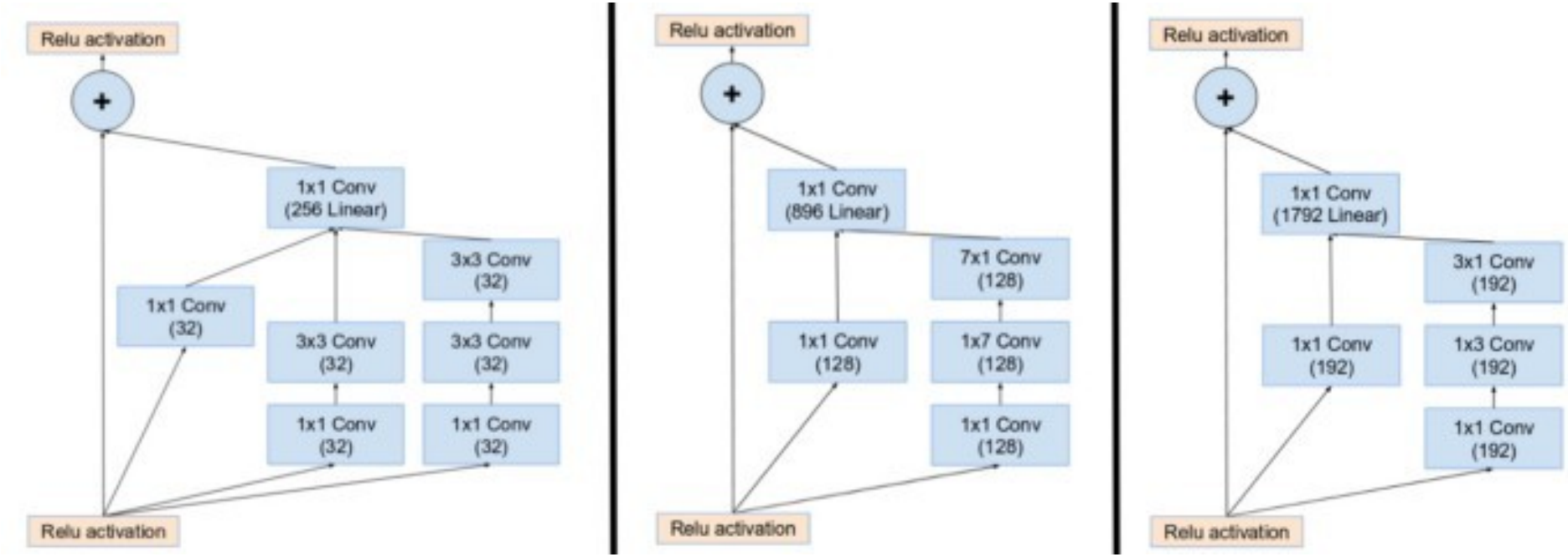
Deeper residual module (bottleneck)



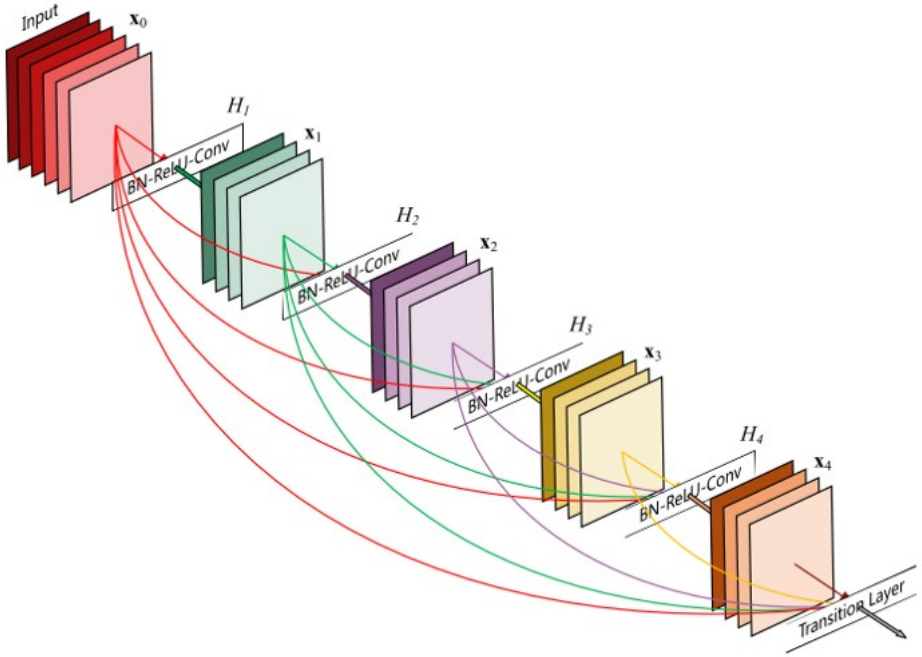
- Directly performing 3x3 convolutions with 256 feature maps at input and output:
  256 x 256 x 3 x 3 ~ 600K operations
- Using 1x1 convolutions to reduce 256 to 64 feature maps, followed by 3x3 convolutions, followed by 1x1 convolutions to expand back to 256 maps:
  256 x 64 x 1 x 1 ~ 16K
  64 x 64 x 3 x 3 ~ 36K
  64 x 256 x 1 x 1 ~ 16K
  Total: ~70K

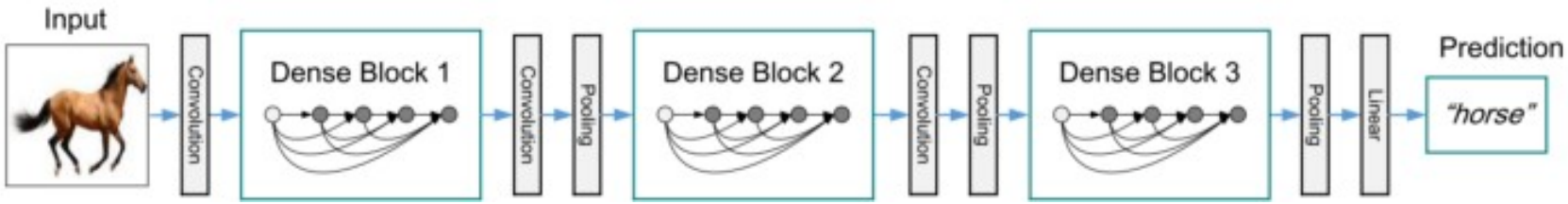

VGG-19
34-layer plain
34-layer residual
image
output size: 224
3x3 conv, 64
3x3 conv, 64
pool, /2
output size: 112
3x3 conv, 128
3x3 conv, 128
pool, /2
output size: 56
3x3 conv, 256
3x3 conv, 256
3x3 conv, 256
3x3 conv, 256
pool, /2
output size: 28
3x3 conv, 512
3x3 conv, 512
3x3 conv, 512
3x3 conv, 512
pool, /2
output size: 14
3x3 conv, 512
3x3 conv, 512
3x3 conv, 512
3x3 conv, 512
pool, /2
output size: 7
fc 4096
fc 4096
fc 1000
output size: 1
7x7 conv, 64, /2
3x3 conv, 128, /2
3x3 conv, 256, /2
3x3 conv, 512, /2
avg pool
fc 1000

Inception resnet

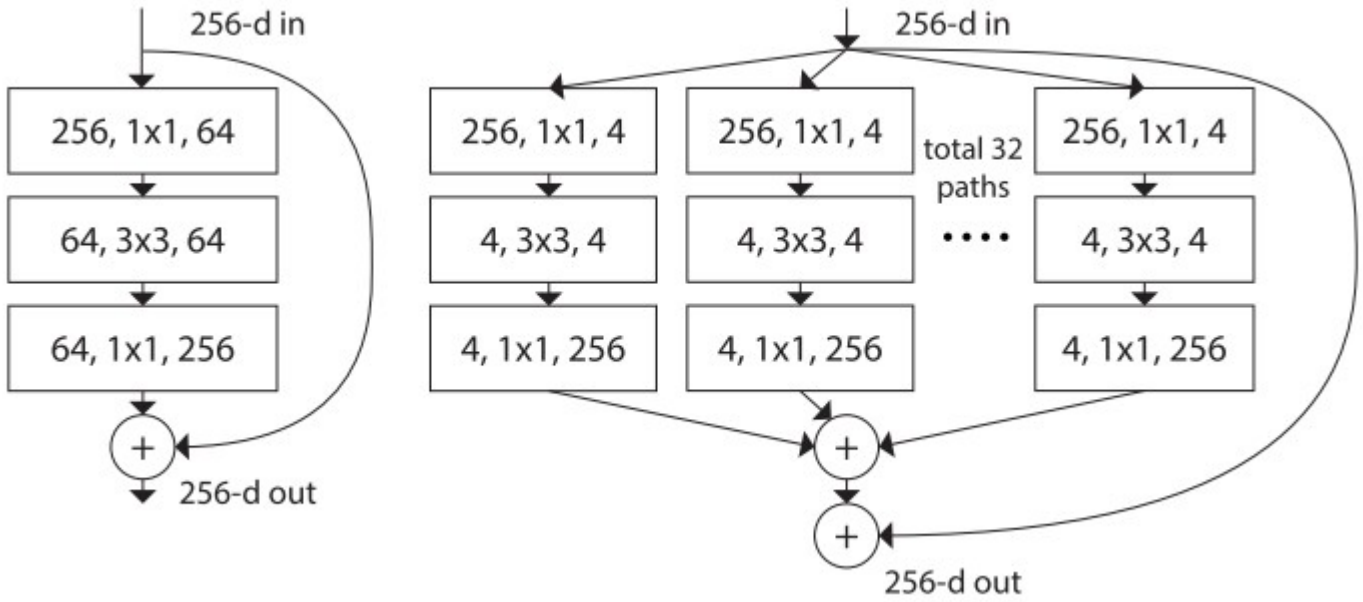


Densely connected CNN





ResNext



Resnext block with cardinality 32

stage	output	ResNet-50	**ResNeXt-50 (32×4d)**
conv1	112×112	7×7, 64, stride 2	7×7, 64, stride 2
conv2	56×56	3×3 max pool, stride 2	3×3 max pool, stride 2
		[1×1, 64 3×3, 64 1×1, 256] ×3	[1×1, 128 3×3, 128, C=32 1×1, 256] ×3
conv3	28×28	[1×1, 128 3×3, 128 1×1, 512] ×4	[1×1, 256 3×3, 256, C=32 1×1, 512] ×4
conv4	14×14	[1×1, 256 3×3, 256 1×1, 1024] ×6	[1×1, 512 3×3, 512, C=32 1×1, 1024] ×6
conv5	7×7	[1×1, 512 3×3, 512 1×1, 2048] ×3	[1×1, 1024 3×3, 1024, C=32 1×1, 2048] ×3
	1×1	global average pool 1000-d fc, softmax	global average pool 1000-d fc, softmax
# params.		$\mathbf{25.5}\times10^6$	$\mathbf{25.0}\times10^6$
FLOPs		$\mathbf{4.1}\times10^9$	$\mathbf{4.2}\times10^9$

Table 1. (**Left**) ResNet-50. (**Right**) ResNeXt-50 with a 32×4d template (using the reformulation in Fig. 3(c)). Inside the brackets are the shape of a residual block, and outside the brackets is the number of stacked blocks on a stage. "C=32" suggests grouped convolutions [24] with 32 groups. *The numbers of parameters and FLOPs are similar between these two models.*

Stochastic depth

- Can the depth of the network change during training?

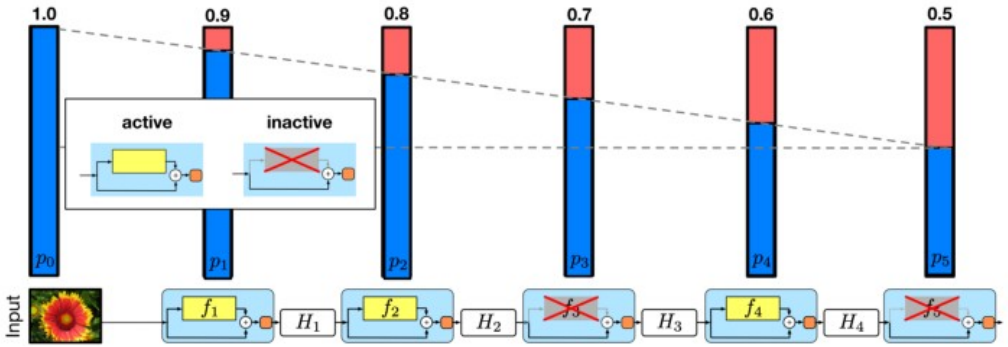
$$H_l = ReLU(b_l * f_l(H_{l-1}) + id(H_{l-1})).$$

If b=0

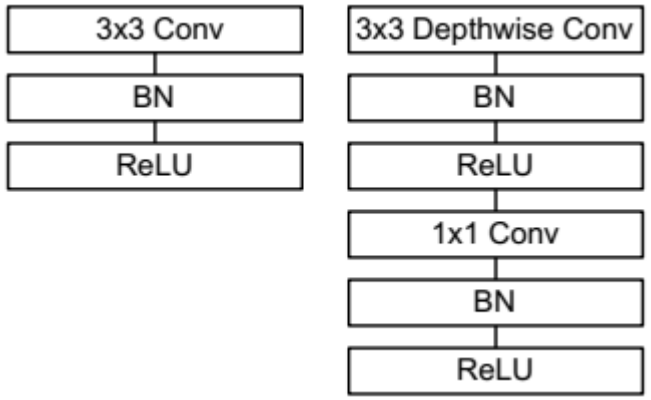
$$H_l = ReLU(id(H_{l-1})).$$

$$H_l = ReLU(p_l * f_l(H_{l-1}) + id(H_{l-1})).$$

$$p_l = 1 - \frac{l}{L}(1 - p_L).$$



Mobilenet



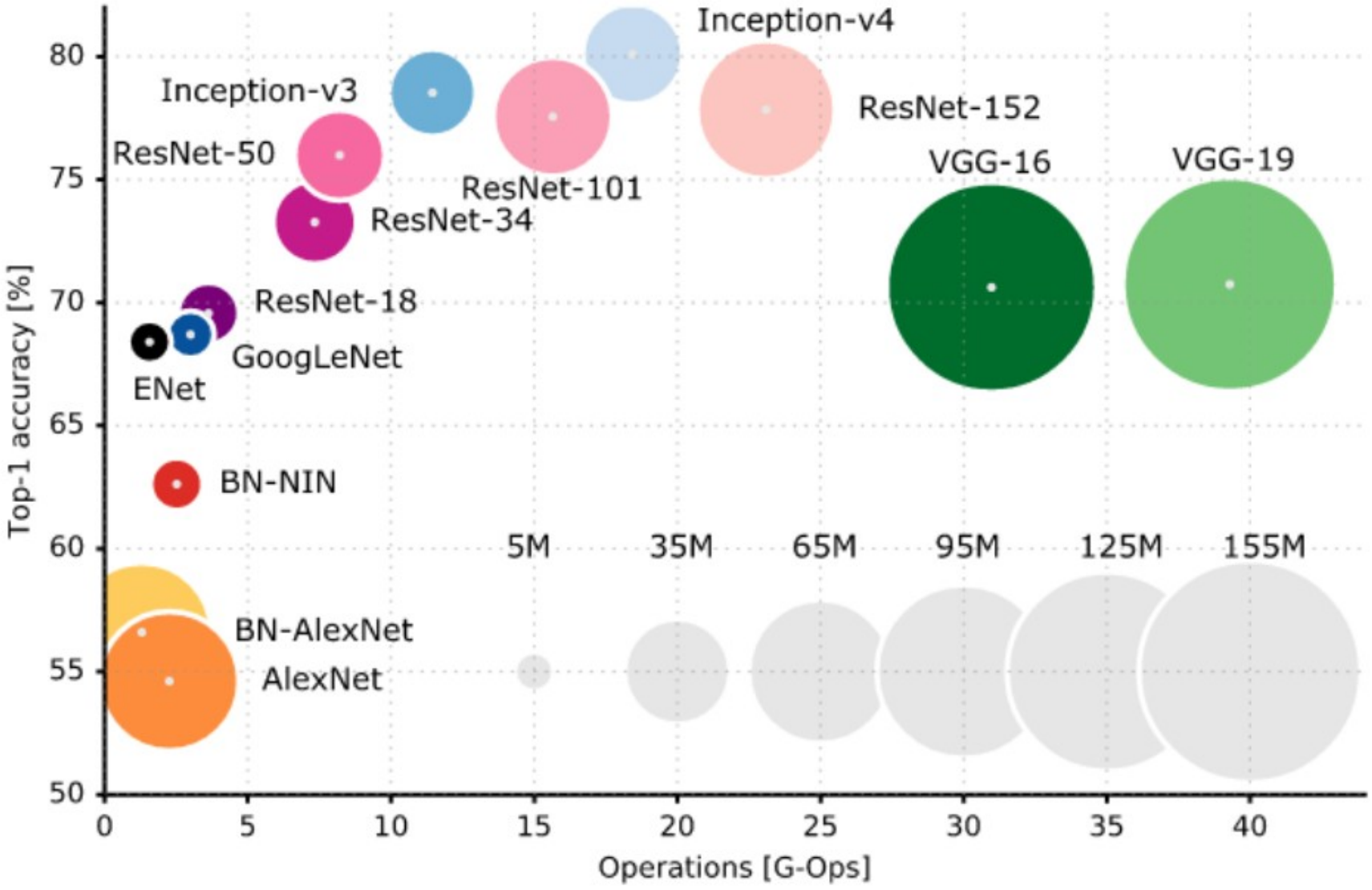
The idea of depthwise
Separable convolution

Table 1. MobileNet Body Architecture

Type / Stride	Filter Shape	Input Size
Conv / s2	$3 \times 3 \times 3 \times 32$	$224 \times 224 \times 3$
Conv dw / s1	$3 \times 3 \times 32$ dw	$112 \times 112 \times 32$
Conv / s1	$1 \times 1 \times 32 \times 64$	$112 \times 112 \times 32$
Conv dw / s2	$3 \times 3 \times 64$ dw	$112 \times 112 \times 64$
Conv / s1	$1 \times 1 \times 64 \times 128$	$56 \times 56 \times 64$
Conv dw / s1	$3 \times 3 \times 128$ dw	$56 \times 56 \times 128$
Conv / s1	$1 \times 1 \times 128 \times 128$	$56 \times 56 \times 128$
Conv dw / s2	$3 \times 3 \times 128$ dw	$56 \times 56 \times 128$
Conv / s1	$1 \times 1 \times 128 \times 256$	$28 \times 28 \times 128$
Conv dw / s1	$3 \times 3 \times 256$ dw	$28 \times 28 \times 256$
Conv / s1	$1 \times 1 \times 256 \times 256$	$28 \times 28 \times 256$
Conv dw / s2	$3 \times 3 \times 256$ dw	$28 \times 28 \times 256$
Conv / s1	$1 \times 1 \times 256 \times 512$	$14 \times 14 \times 256$
$5\times$ Conv dw / s1	$3 \times 3 \times 512$ dw	$14 \times 14 \times 512$
Conv / s1	$1 \times 1 \times 512 \times 512$	$14 \times 14 \times 512$
Conv dw / s2	$3 \times 3 \times 512$ dw	$14 \times 14 \times 512$
Conv / s1	$1 \times 1 \times 512 \times 1024$	$7 \times 7 \times 512$
Conv dw / s2	$3 \times 3 \times 1024$ dw	$7 \times 7 \times 1024$
Conv / s1	$1 \times 1 \times 1024 \times 1024$	$7 \times 7 \times 1024$
Avg Pool / s1	Pool 7×7	$7 \times 7 \times 1024$
FC / s1	1024×1000	$1 \times 1 \times 1024$
Softmax / s1	Classifier	$1 \times 1 \times 1000$

Team	Year	Place	Error (top-5)	External data
SuperVision – Toronto (AlexNet, 7 layers)	2012	-	16.4%	no
SuperVision	2012	1st	15.3%	ImageNet 22k
Clarifai – NYU (7 layers)	2013	-	11.7%	no
Clarifai	2013	1st	11.2%	ImageNet 22k
VGG – Oxford (16 layers)	2014	2nd	7.32%	no
GoogLeNet (19 layers)	2014	1st	6.67%	no
ResNet (152 layers)	2015	1st	3.57%	
Human expert*			5.1%	

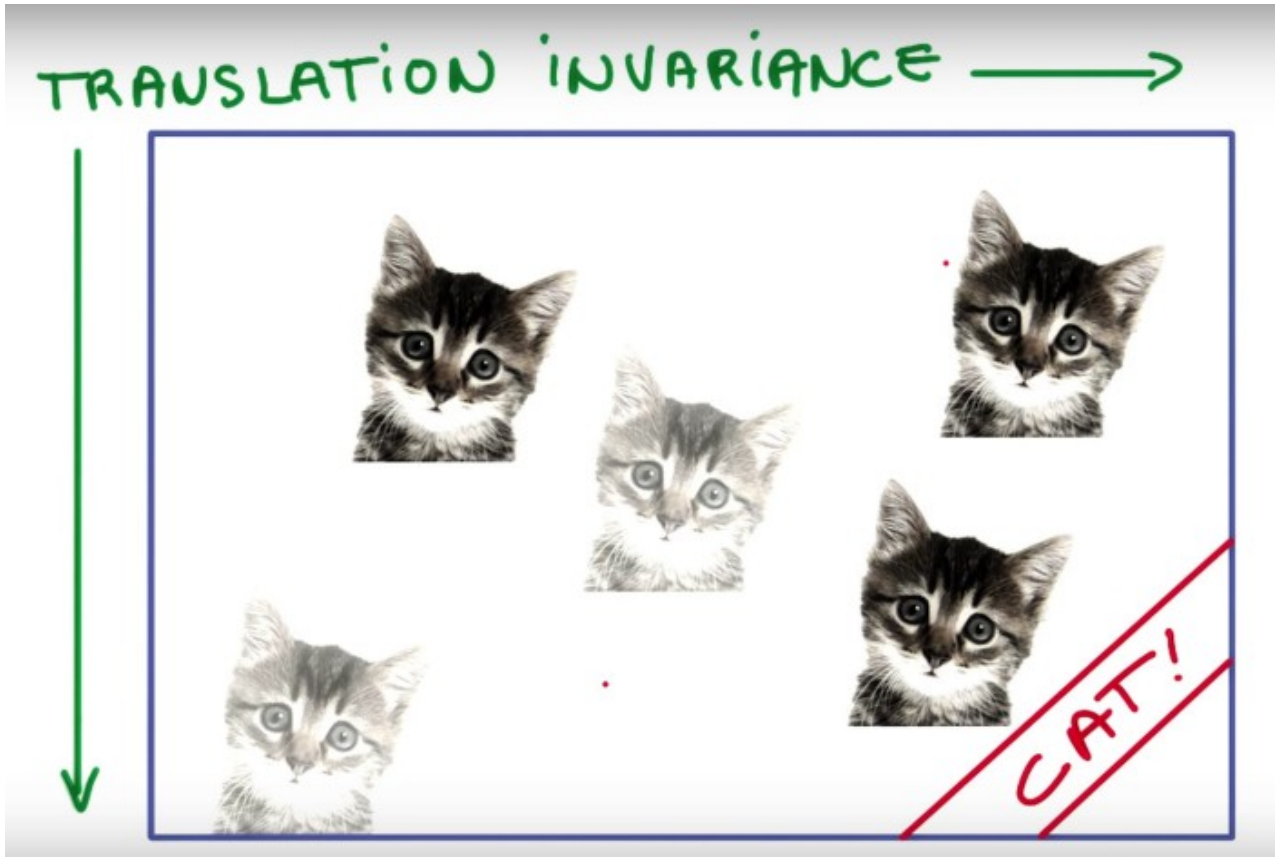
http://karpathy.github.io/2014/09/02/what-i-learned-from-competing-against-a-convnet-on-imagenet/



https://culurciello.github.io/tech/2016/06/04/nets.html

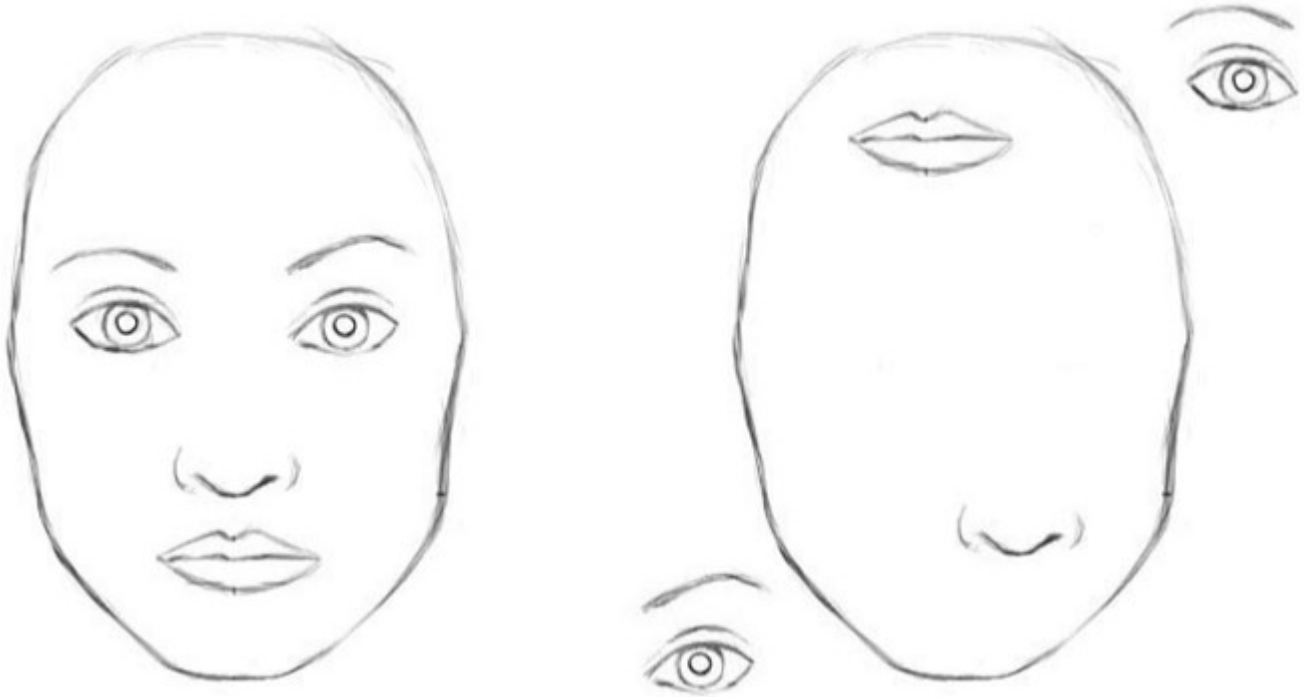
What’s wrong with deep CNN?

- Translation invariance?



What’s wrong with deep CNN?

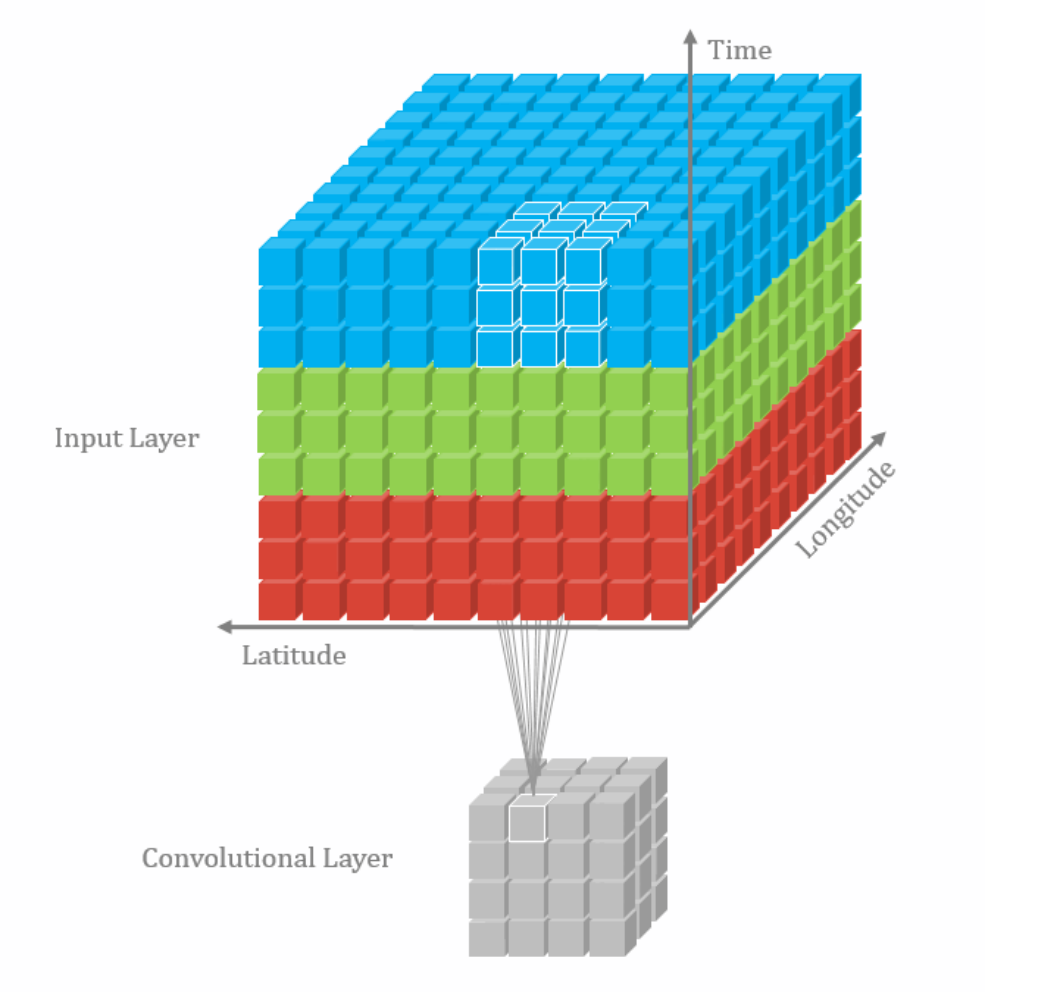
- Pooling causes much information loss



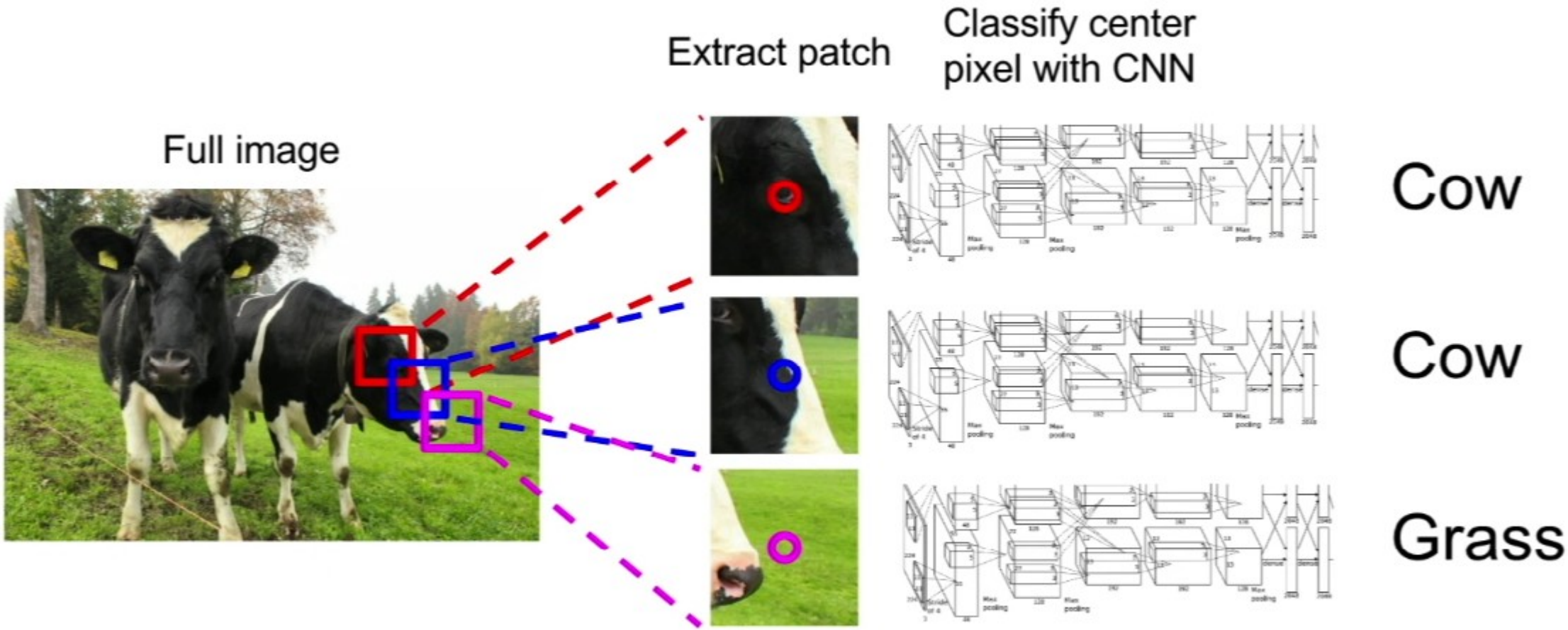
Some design tricks

- Reduce filter sizes (except possibly at the lowest layer), factorize filters aggressively
- Use 1x1 convolutions to reduce and expand the number of feature maps judiciously
- Use skip connections and/or create multiple paths through the network

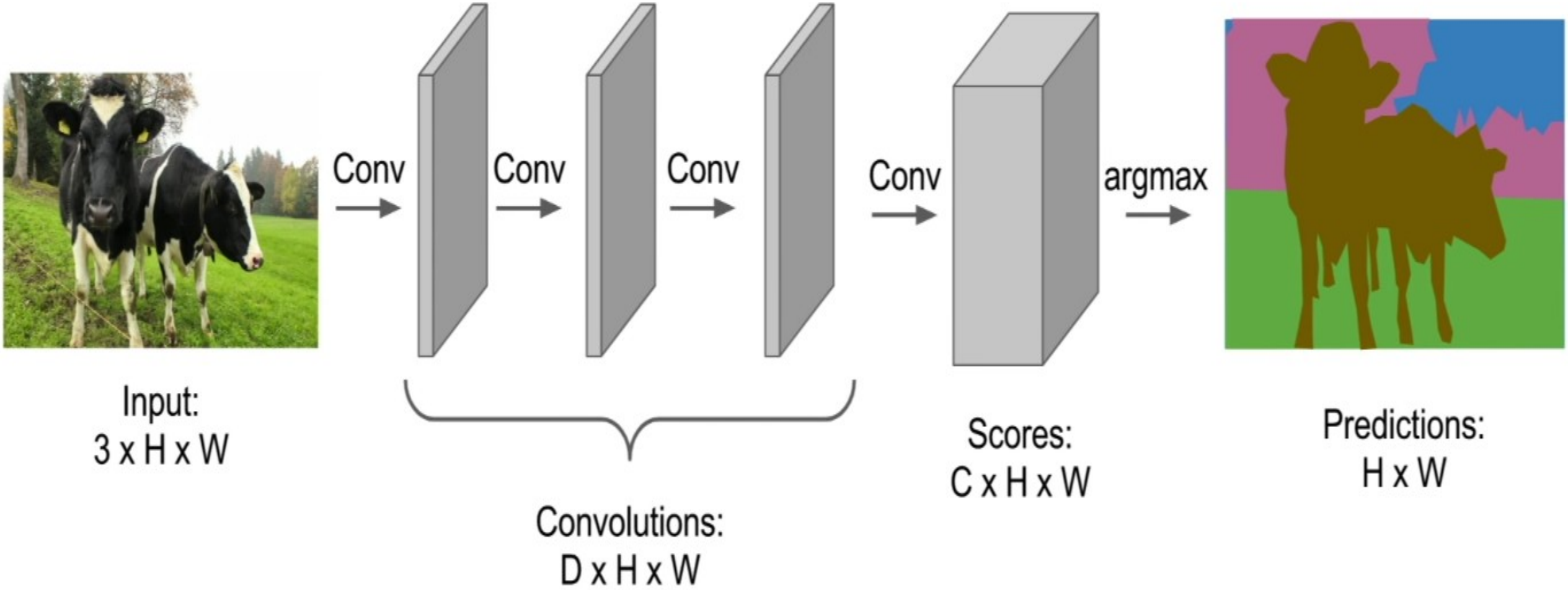
3D CNN



Going at the pixel level



Going at the pixel level



Suggested readings

- https://culurciello.github.io/tech/2016/06/04/nets.html
- Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, Gradient-based learning applied to document recognition, Proc. IEEE 86(11): 2278–2324, 1998.
- A. Krizhevsky, I. Sutskever, and G. Hinton, ImageNet Classification with Deep Convolutional Neural Networks, NIPS 2012
- M. Zeiler and R. Fergus, Visualizing and Understanding Convolutional Networks, ECCV 2014
- K. Simonyan and A. Zisserman, Very Deep Convolutional Networks for Large-Scale Image Recognition, ICLR 2015
- M. Lin, Q. Chen, and S. Yan, Network in network, ICLR 2014
- C. Szegedy et al., Going deeper with convolutions, CVPR 2015
- C. Szegedy et al., Rethinking the inception architecture for computer vision, CVPR 2016
- K. He, X. Zhang, S. Ren, and J. Sun, Deep Residual Learning for Image Recognition, CVPR 2016