

More on CNN

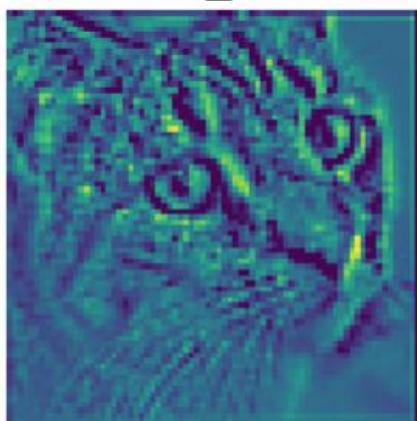
Biplab Banerjee

CNN feature maps

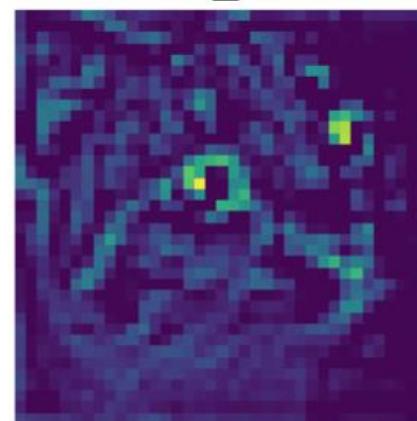
block1_conv1



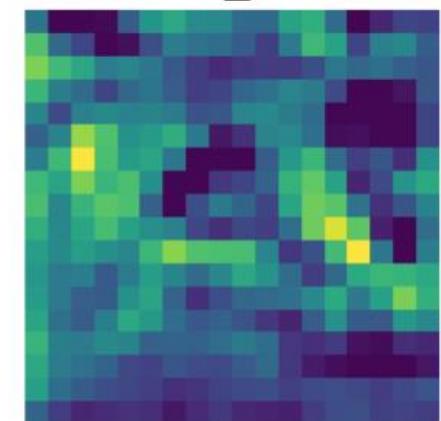
block2_conv1



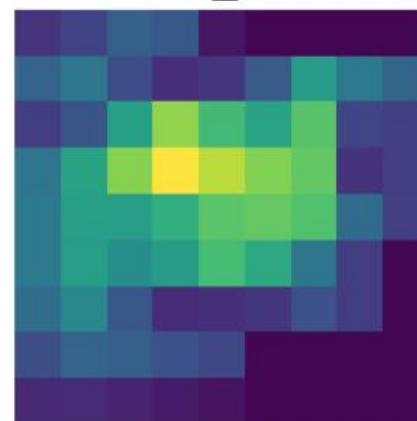
block3_conv1



block4_conv1



block5_conv1



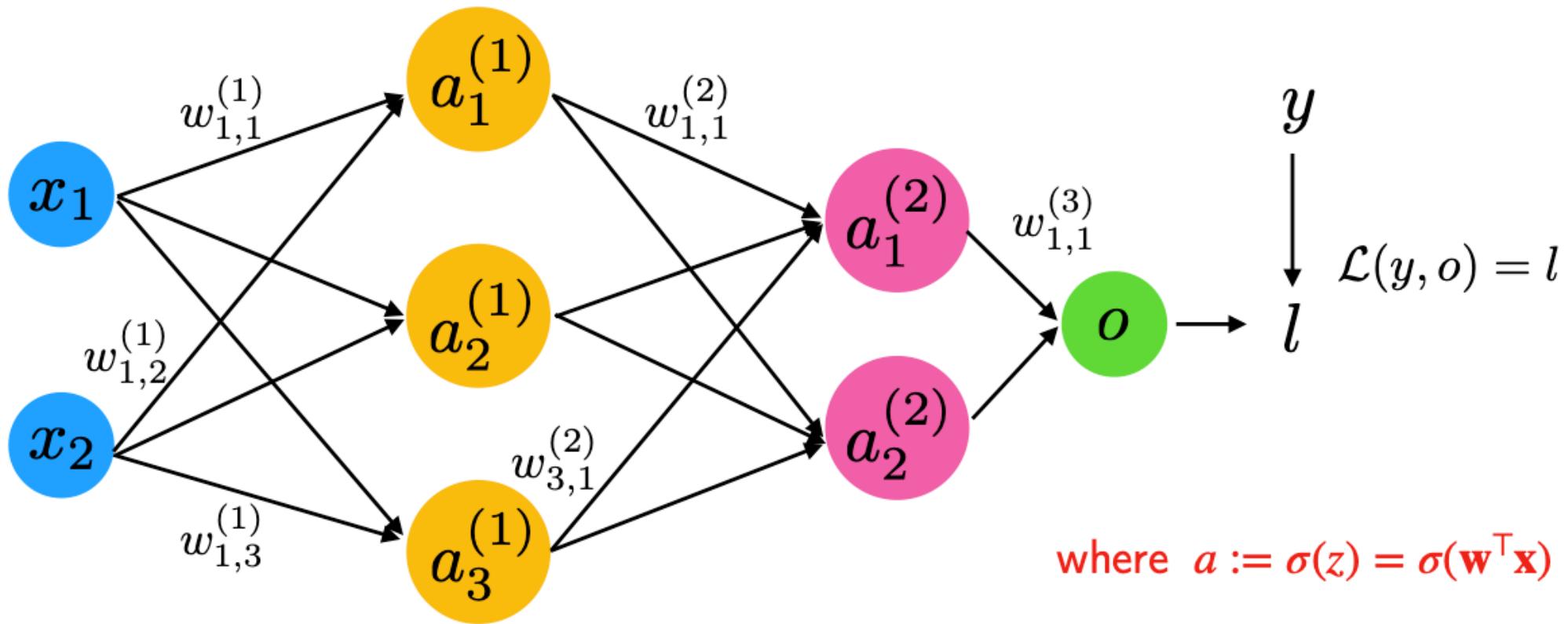
Some insights

- Translation invariance vs equivariance
- Feature hierarchy learning in CNN
- Requires loads of images
- What happens if we do not train the model properly and still do imagenet classification?

Issues in CNN

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

Vanishing and exploding gradients

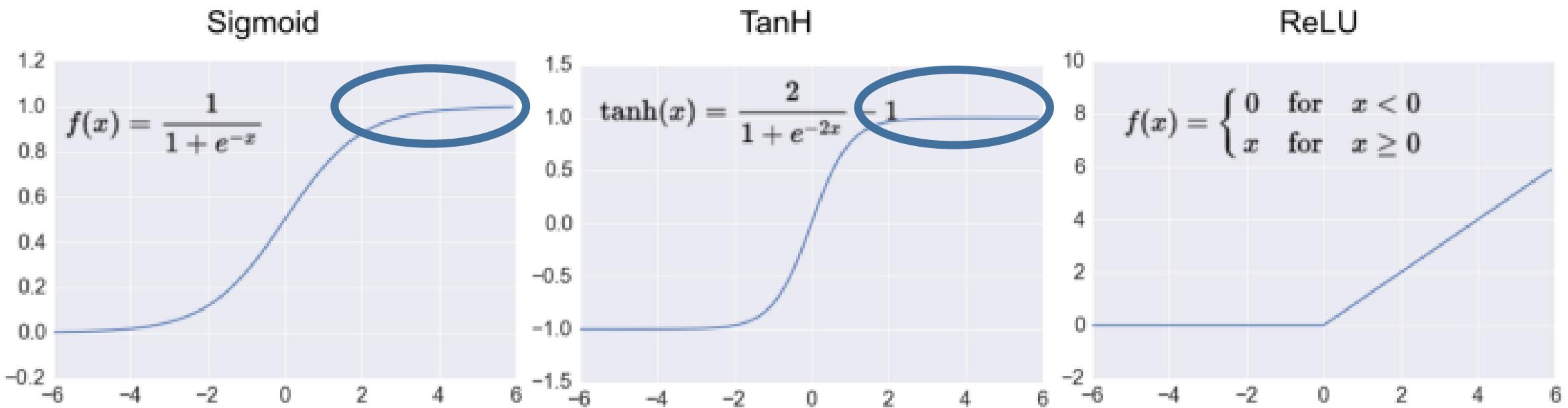


$$\frac{\partial l}{\partial w_{1,1}^{(1)}} = \frac{\partial l}{\partial o} \cdot \frac{\partial o}{\partial a_1^{(2)}} \cdot \frac{\partial a_1^{(2)}}{\partial a_1^{(1)}} \cdot \frac{\partial a_1^{(1)}}{\partial w_{1,1}^{(1)}}$$

Ways to deal with them

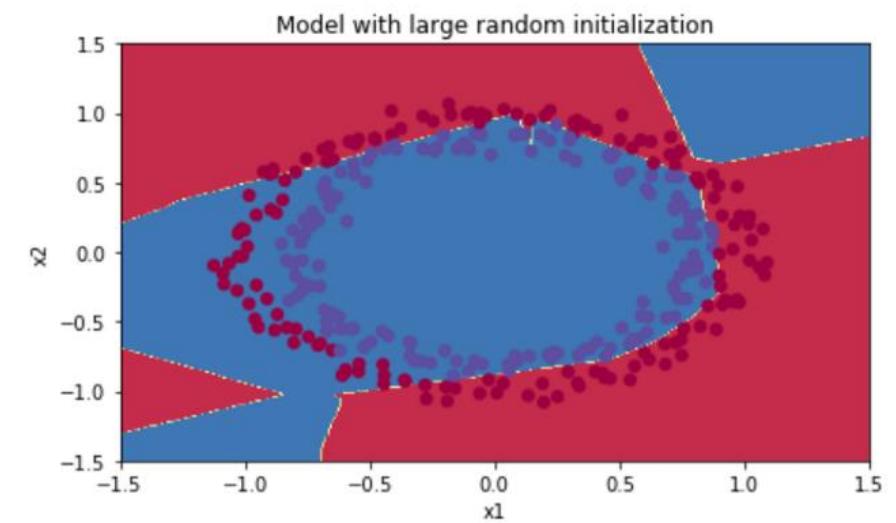
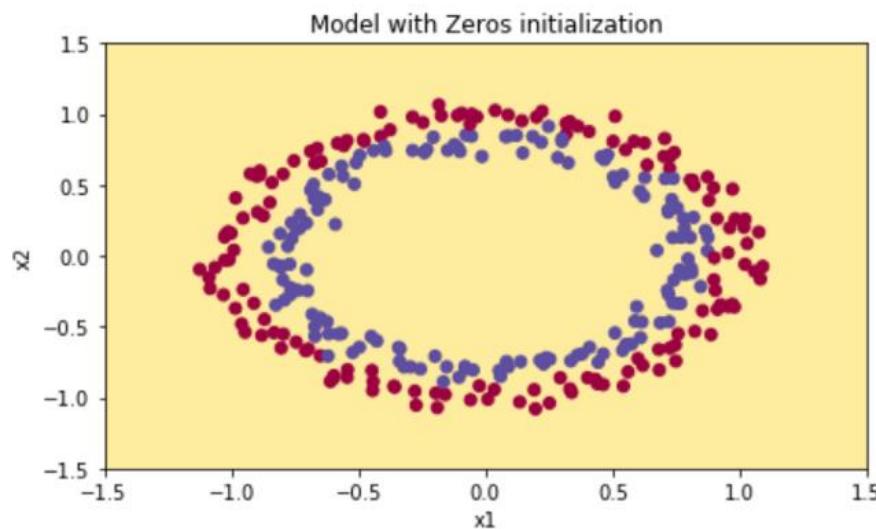
- Vanishing gradients
 - Use proper activations
 - Smaller network (if using sigmoid/tanh type activations)
 - **Proper initialization**
- Exploding gradients
 - Weight clipping
 - **Proper initialization**
 - Model engineering

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?

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

$$Var(X_iW_i) = [E(X_i)]^2Var(W_i) + [E(W_i)]^2Var(X_i) + Var(X_i)Var(W_i)$$

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

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

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

Assuming
each product
term is
independent!



Xavier initialization

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

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

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

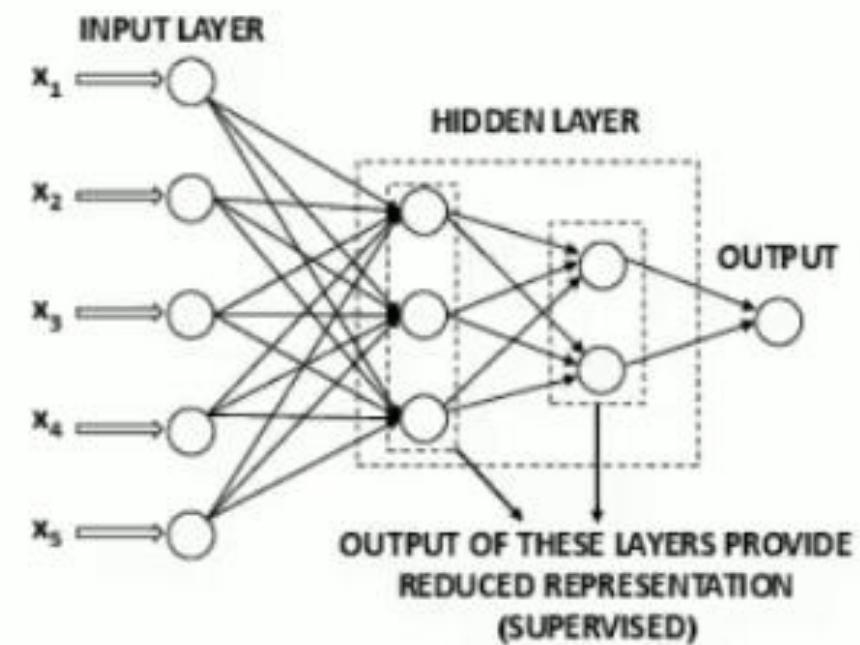
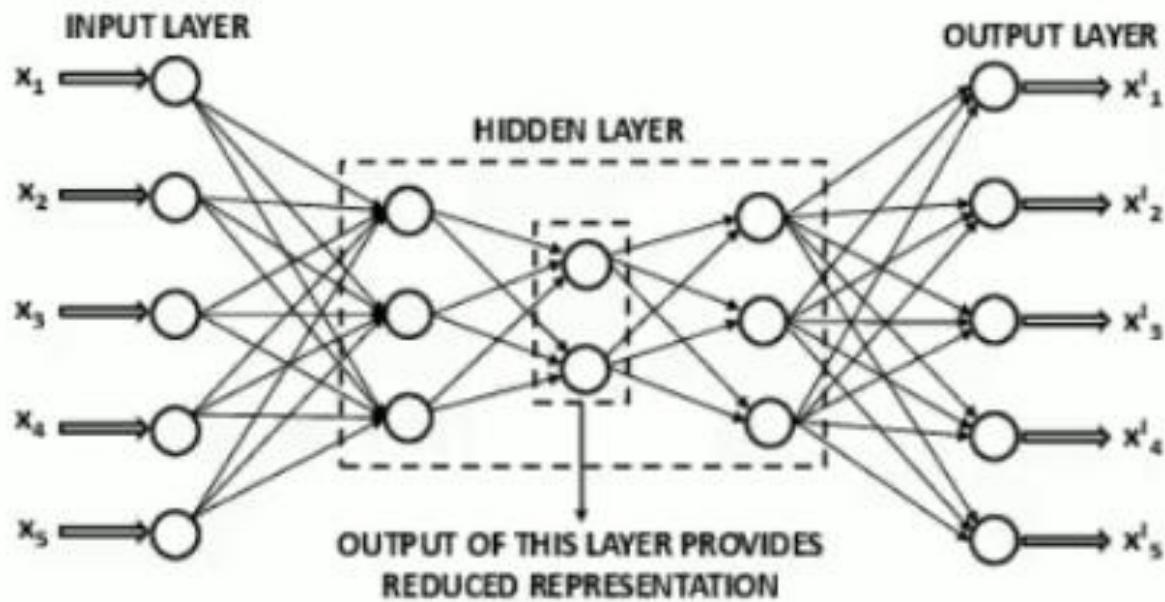
For forward and backward passes

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

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

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.



Suggested reading

Understanding the difficulty of training deep feedforward neural networks

Xavier Glorot

DIRO, Université de Montréal, Montréal, Québec, Canada

Yoshua Bengio

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

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

Repeat {

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

For every $j = 0 \dots n$

}

Stochastic gradient descent

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

Repeat {
    For i=1 to m{
        θj = θj - (learning rate) * Σ( hθ(x(i)) - y(i))xj(i)
        For every j =0 ...n
    }
}
```

Mini-batch gradient descent

```
Repeat {  
    For i=1,11, 21,...,91
```

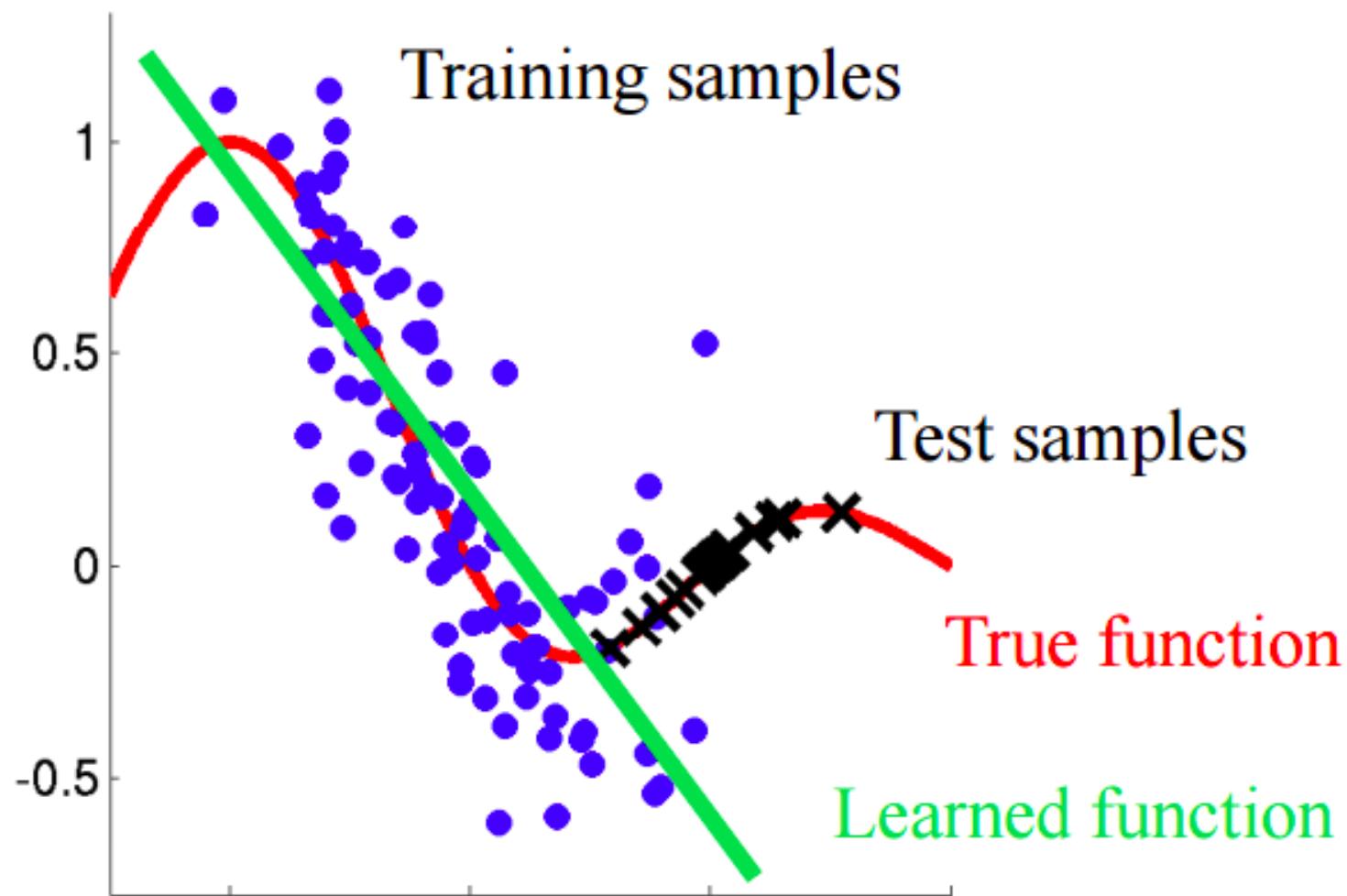
Let Σ be the summation from i to $i+9$ represented by k .

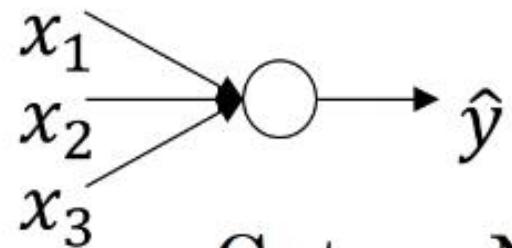
$$\theta_j = \theta_j - (\text{learning rate}/\text{size of } (b)) * \Sigma(h_{\theta}(x^{(k)}) - y^{(k)})x_j^{(k)}$$

For every $j = 0 \dots n$

```
}
```

The idea of dataset (covariate) shift





Cat Non-Cat

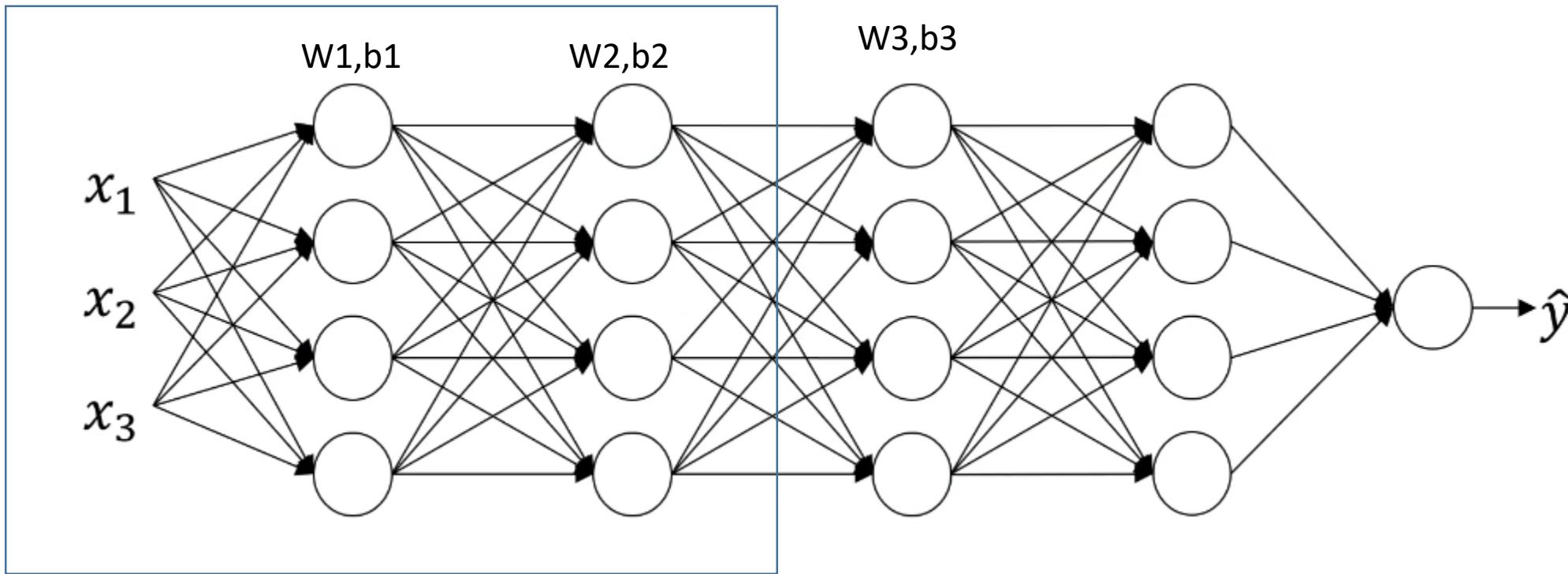
$$y = 1$$



$$y = 1 \quad y = 0$$



What does each layer weight see?



Inputs to layer 3 changes over time as $(W_{1,b1})$ and $(W_{2,b2})$ are also changing

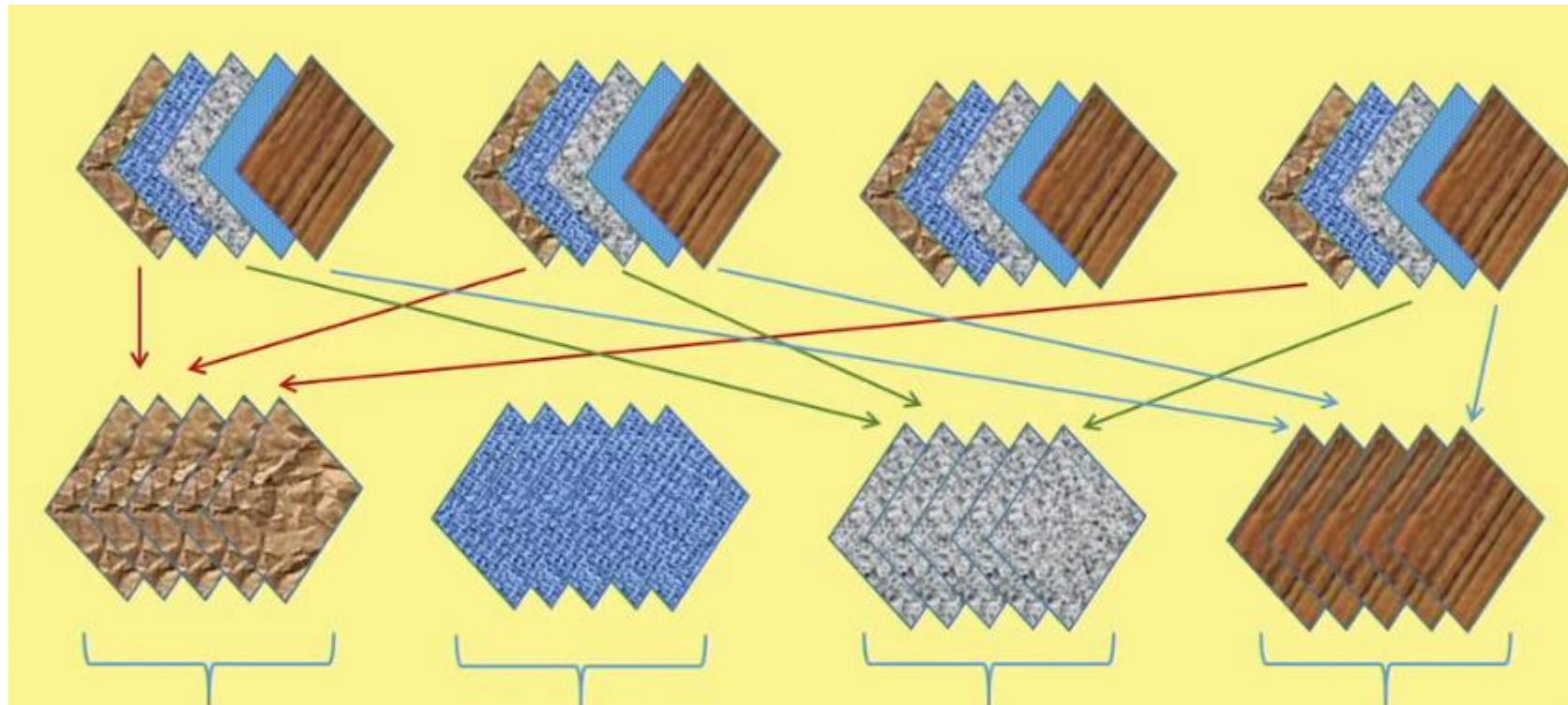
Batch-normalization



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



Input: Values of x over a mini-batch: $\mathcal{B} = \{x_1 \dots m\}$;
Parameters to be learned: γ, β

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

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

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

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

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

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

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

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

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

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

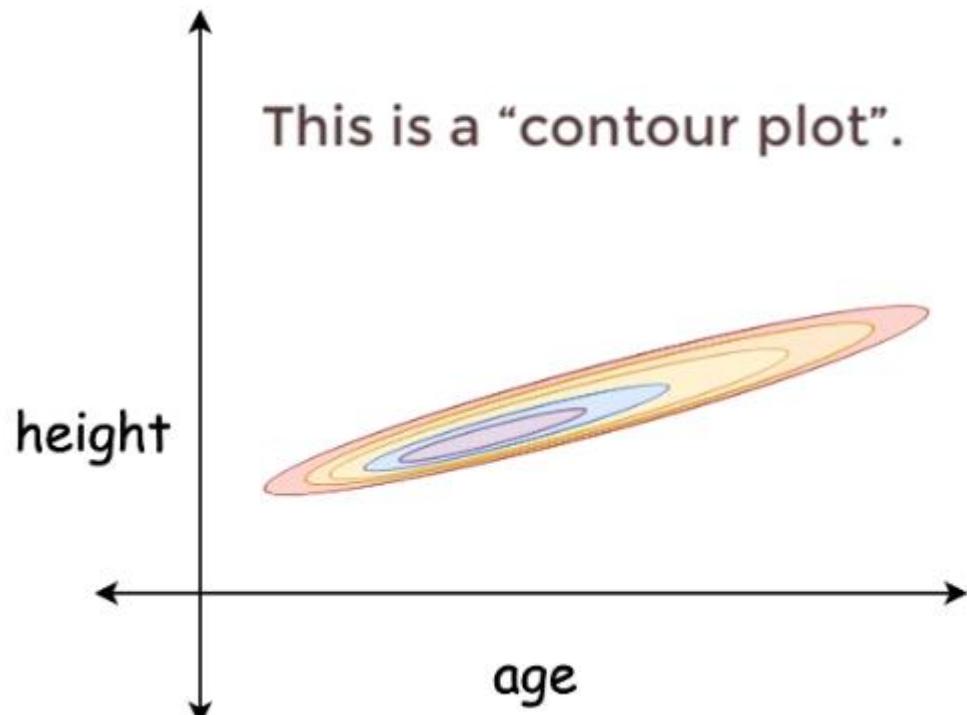
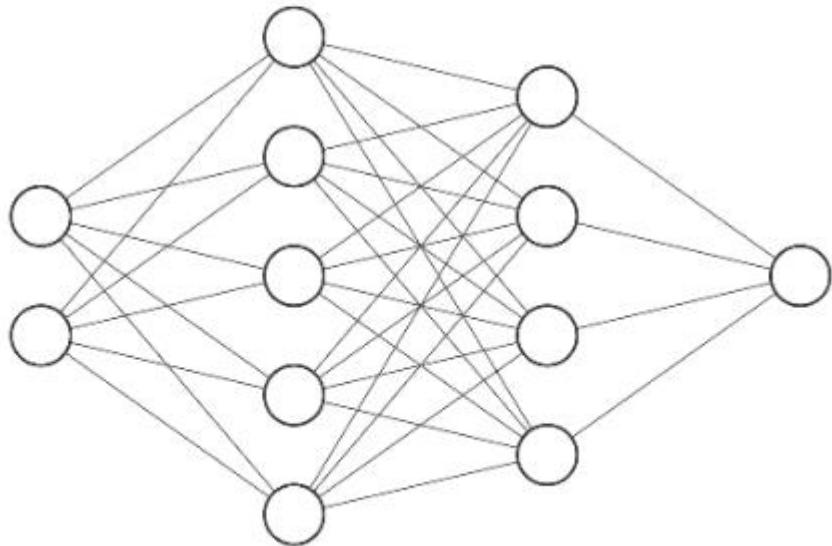
$$\frac{\partial \ell}{\partial \beta} = \sum\nolimits_{i=1}^m \frac{\partial \ell}{\partial y_i}$$

Advantages of BN

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

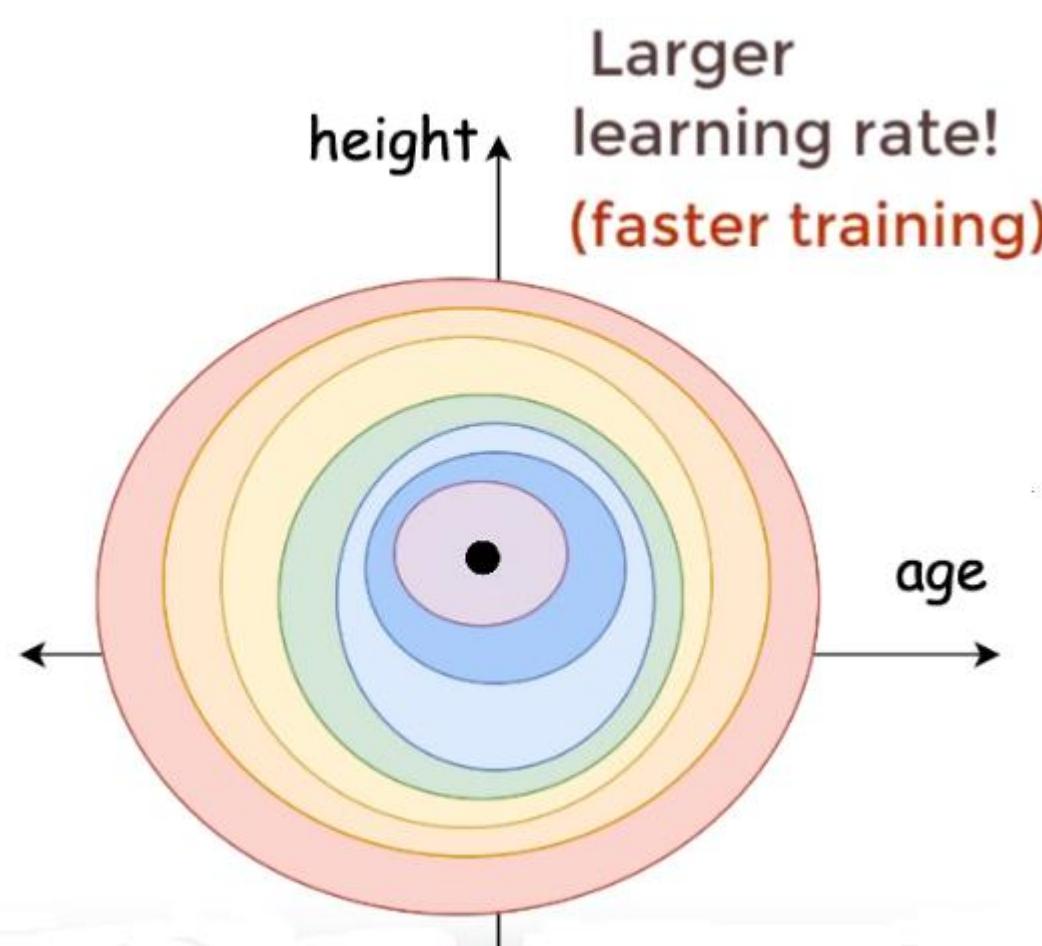
Speeds up training

1. Speeds up training

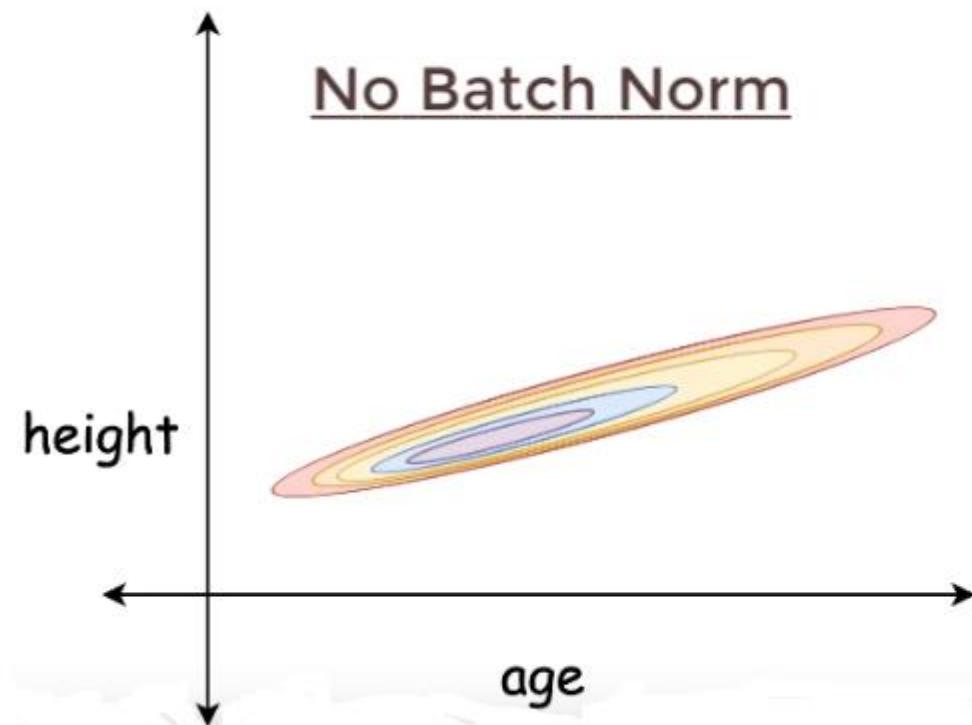


In different batch, let us consider we have young to old people

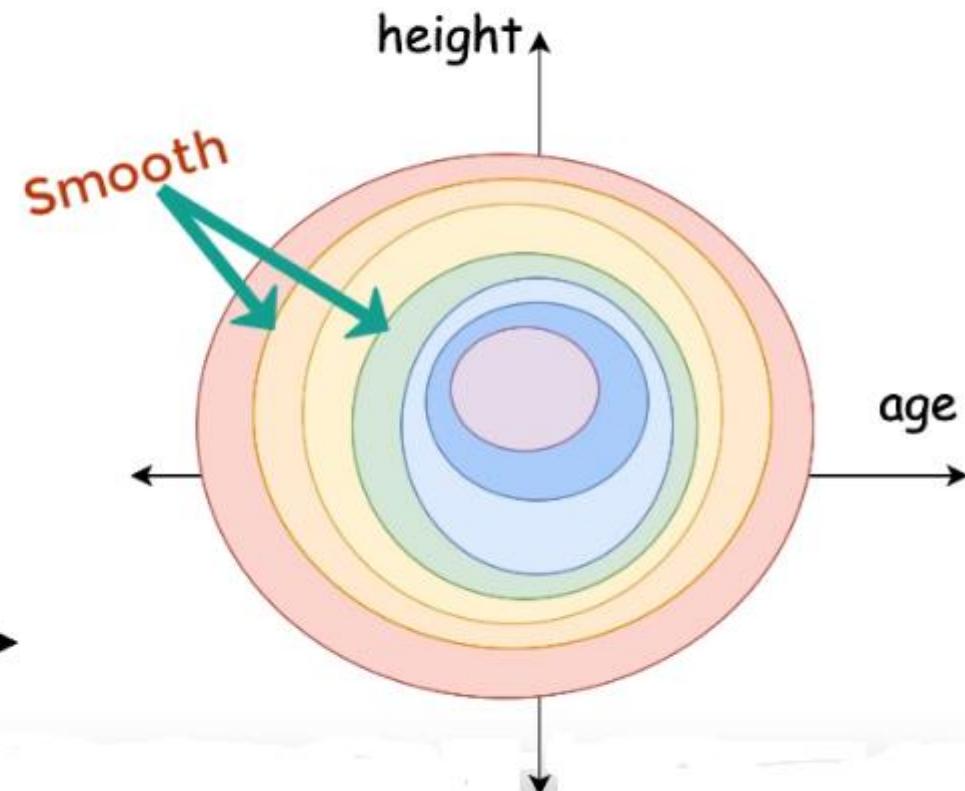
After standardization



1. Speeds up training



With Batch Norm

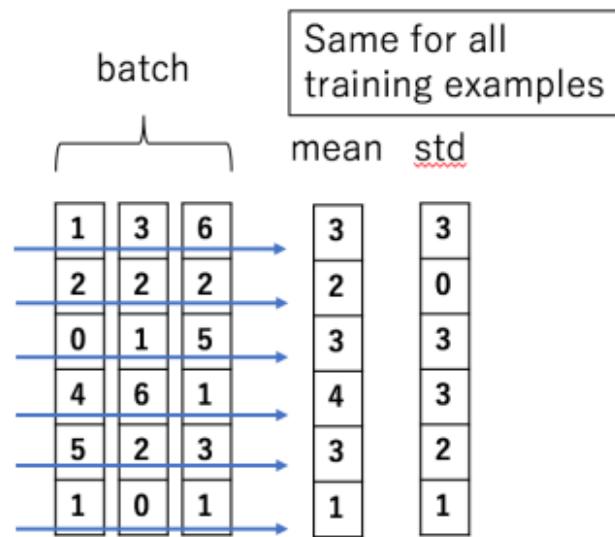


Where can the BN not be used?

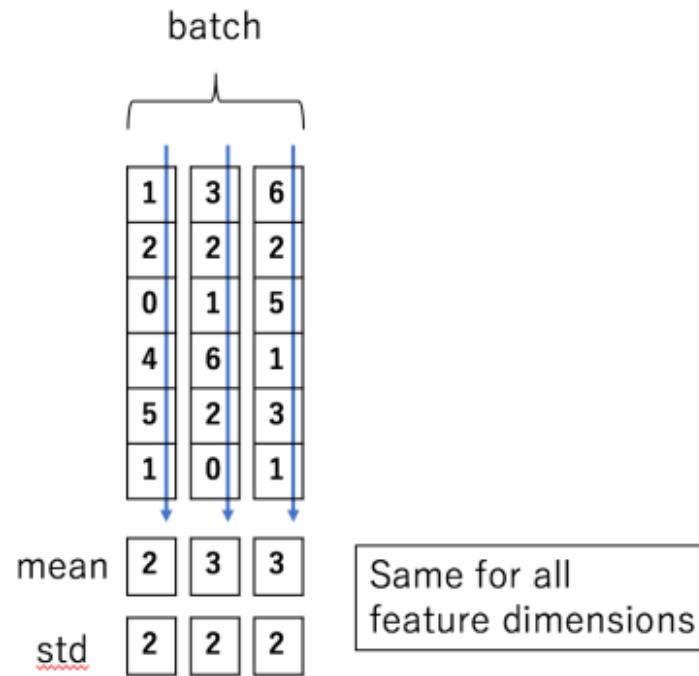
- Small batch size – high noise in calculated batch mean and std!
- Recurrent type models – separate batch-norm at each time-step!
- Solutions?
 - Weight normalization
 - Layer normalization
 - Instance normalization
 - Spectral normalization
 - Group normalization
 - ...

Layer normalization

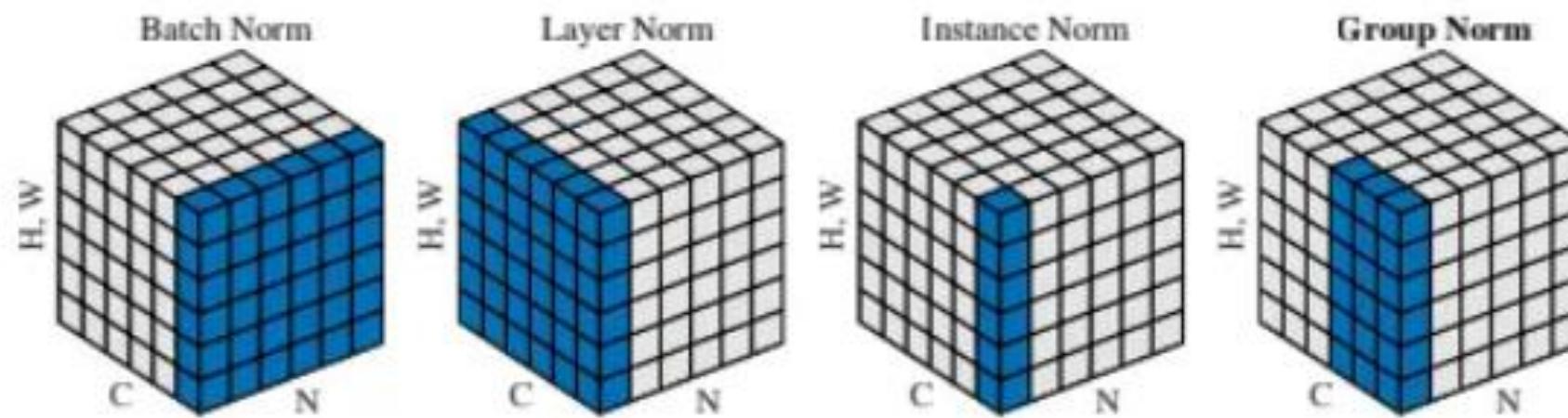
Batch Normalization



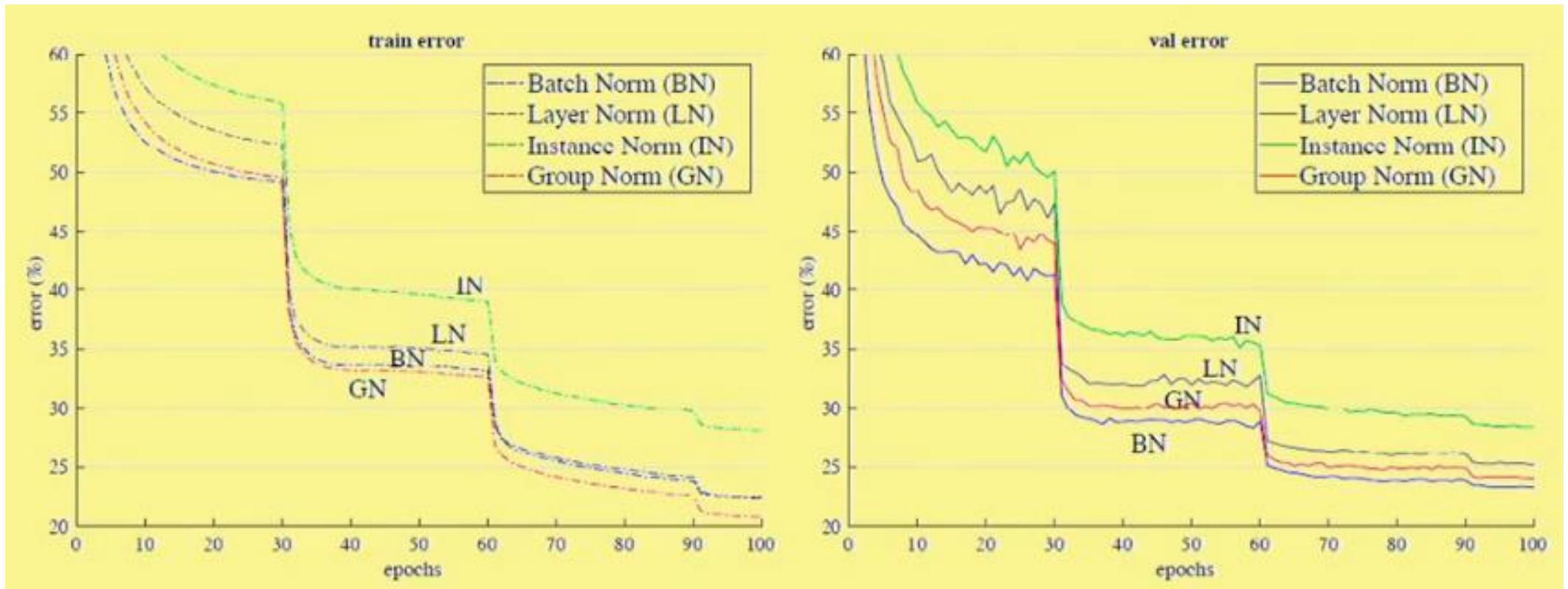
Layer Normalization



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

```
def neural_network_model(data):
    hidden_1_layer =
    {'weights':tf.Variable(tf.random_normal([784, n_nodes_hl1])),
     'biases':tf.Variable(tf.random_normal([n_nodes_hl1]))}

    hidden_2_layer =
    {'weights':tf.Variable(tf.random_normal([n_nodes_hl1,
n_nodes_hl2])),
     'biases':tf.Variable(tf.random_normal([n_nodes_hl2]))}

    hidden_3_layer =
    {'weights':tf.Variable(tf.random_normal([n_nodes_hl2,
n_nodes_hl3])),
     'biases':tf.Variable(tf.random_normal([n_nodes_hl3]))}

    output_layer =
    {'weights':tf.Variable(tf.random_normal([n_nodes_hl3,
n_classes])),
     'biases':tf.Variable(tf.random_normal([n_classes])),}

    l1 = tf.add(tf.matmul(data,hidden_1_layer['weights']),
hidden_1_layer['biases'])
    l1 = tf.nn.relu(l1)

    l2 = tf.add(tf.matmul(l1,hidden_2_layer['weights']),
hidden_2_layer['biases'])
    l2 = tf.nn.relu(l2)

    l3 = tf.add(tf.matmul(l2,hidden_3_layer['weights']),
hidden_3_layer['biases'])
    l3 = tf.nn.relu(l3)

    output = tf.matmul(l3,output_layer['weights']) +
output_layer['biases']

    return output
```

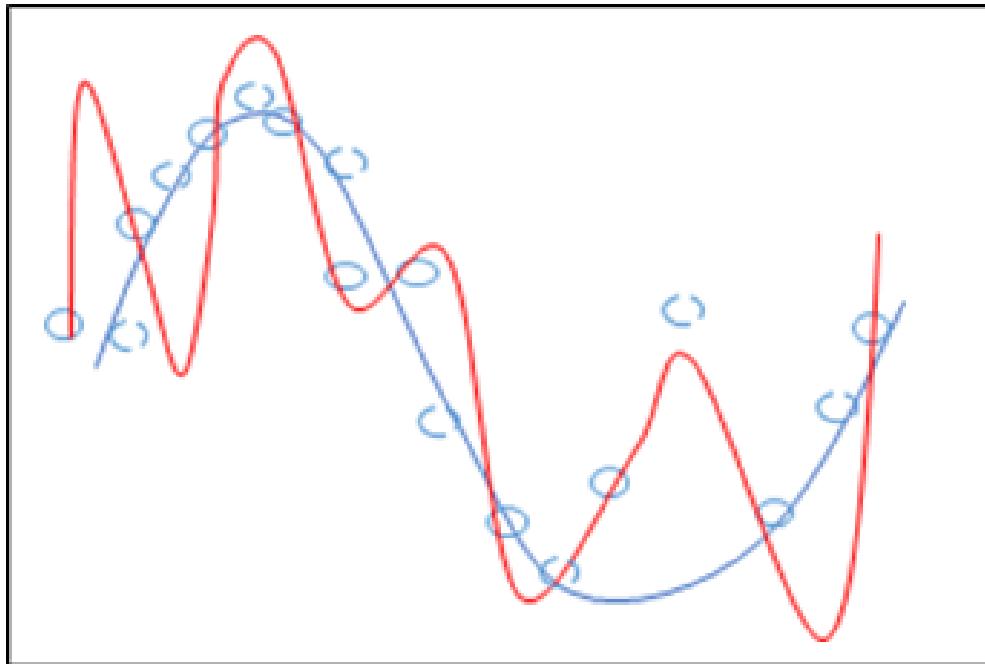
```
from keras.models import Sequential
from keras.layers import Dense, Conv2D, Flatten

#create model
model = Sequential()

#add model layers
model.add(Conv2D(64, kernel_size=3, activation='relu', input_shape=(28,28,1)))
model.add(Conv2D(32, kernel_size=3, activation='relu'))
model.add(Flatten())
model.add(Dense(10, activation='softmax'))
```

```
W = tf.get_variable("W", shape=[784, 256],
                    initializer=tf.contrib.layers.xavier_initializer())
```

Model regularization

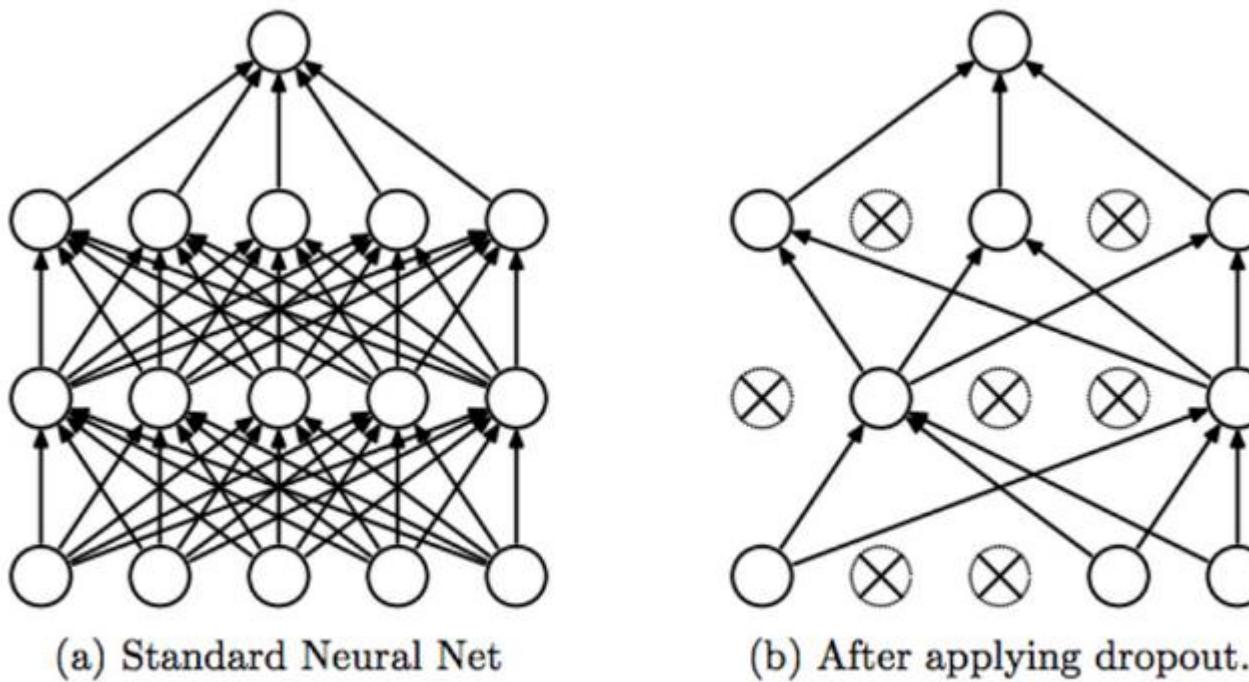


1. Less training data
2. Too many model parameters
3. Model overfitting
4. Regularization – ways to avoid overfitting

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

A Bernoulli random variable has the following probability mass distribution:

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

Where k are the possible outcomes.

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

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:

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

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

Inverted dropout

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

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

Dropout + L2 normalization

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

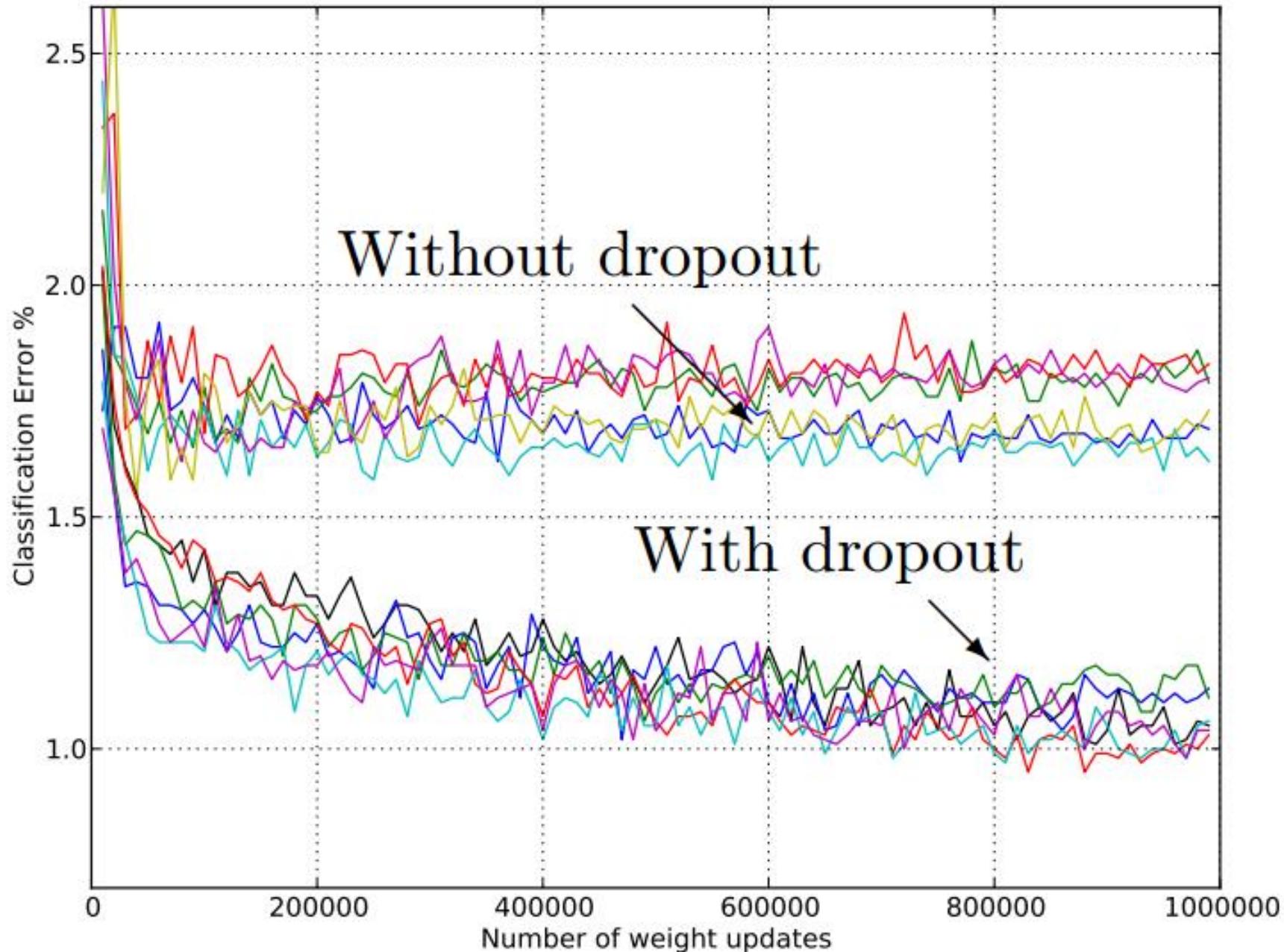
Dropout can't control the range of values the weight variables can take!

Dropout

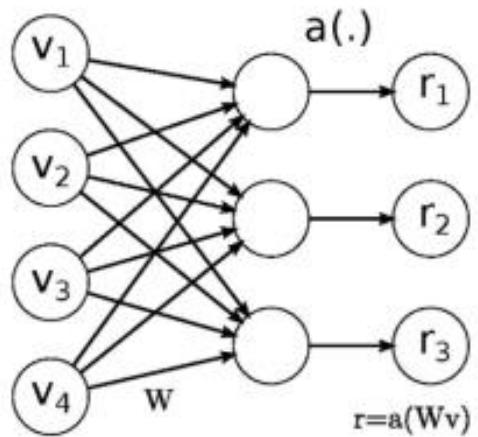
$$w \leftarrow w - \eta \left(\frac{\partial F(W; x)}{\partial w} + \lambda w \right)$$

Inverted Dropout

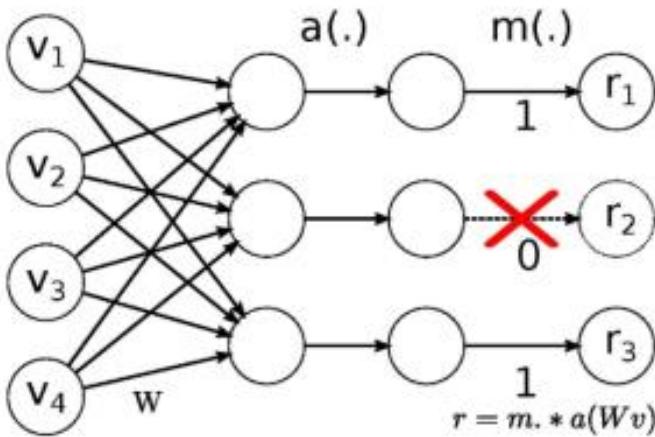
$$w \leftarrow w - \eta \left(\frac{1}{q} \frac{\partial F(W; x)}{\partial w} + \lambda w \right)$$



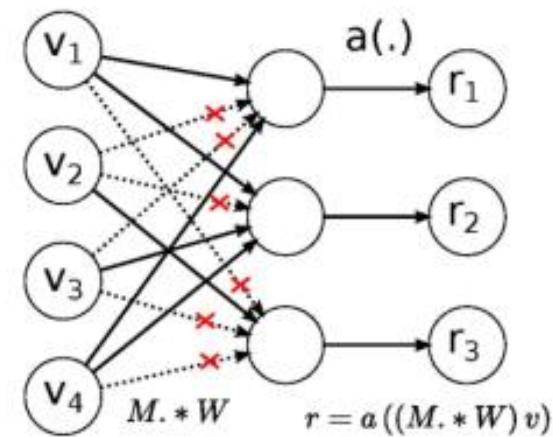
DropConnect



No-Drop Network



DropOut Network



DropConnect Network

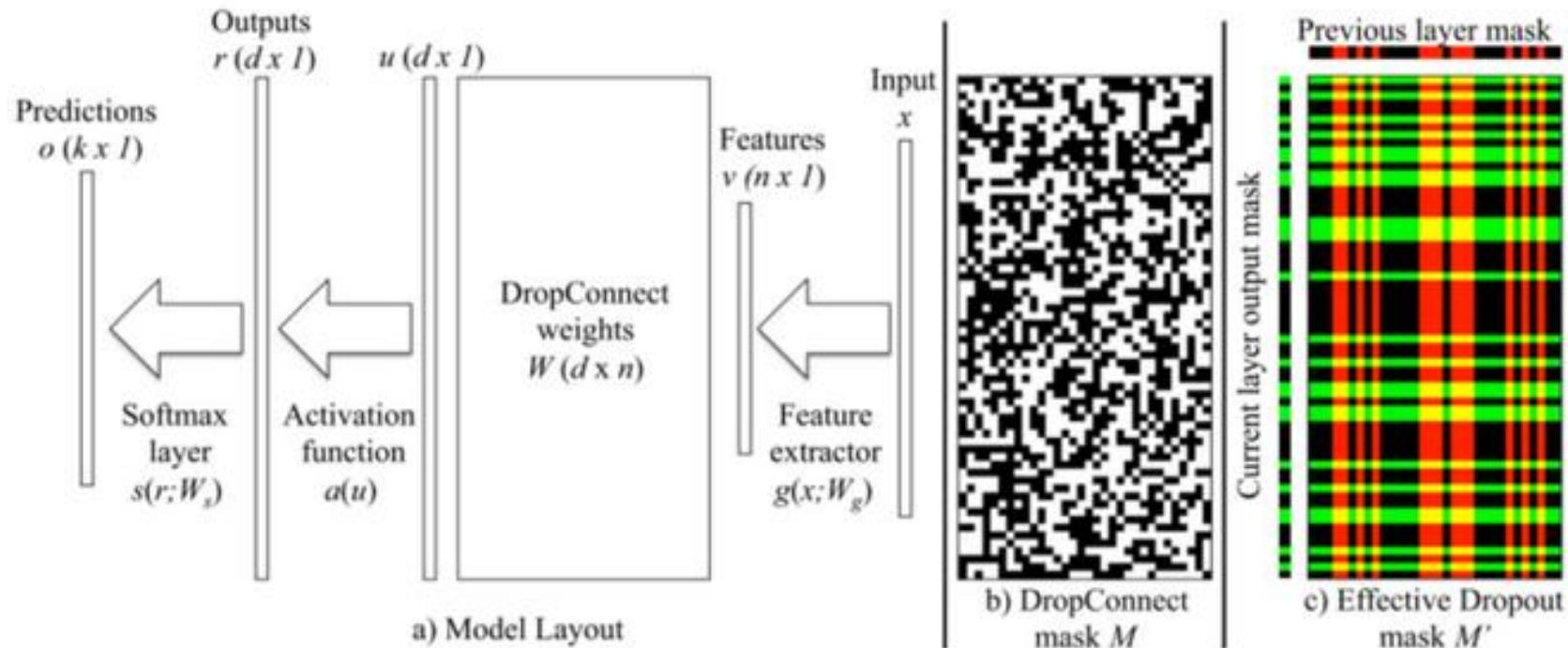
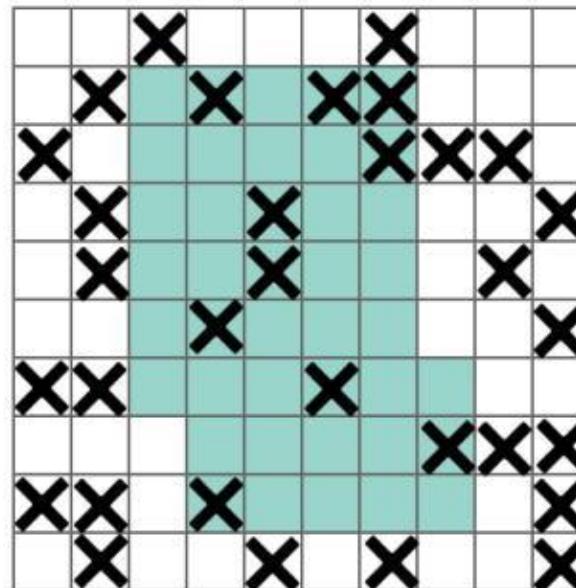


Fig 3. After Wan et al. (2013) (a): An example model layout for a single DropConnect layer. After running feature extractor $g()$ on input x , a random instantiation of the mask M (e.g. (b)), masks out the weight matrix W . The masked weights are multiplied with this feature vector to produce u which 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).

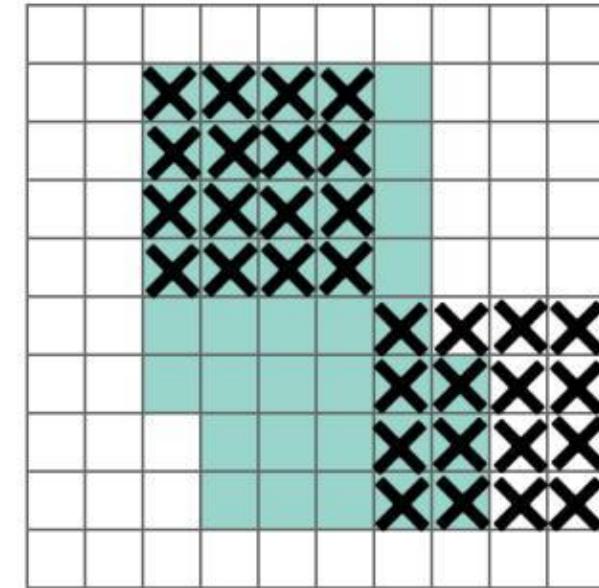
Dropblock



(a)



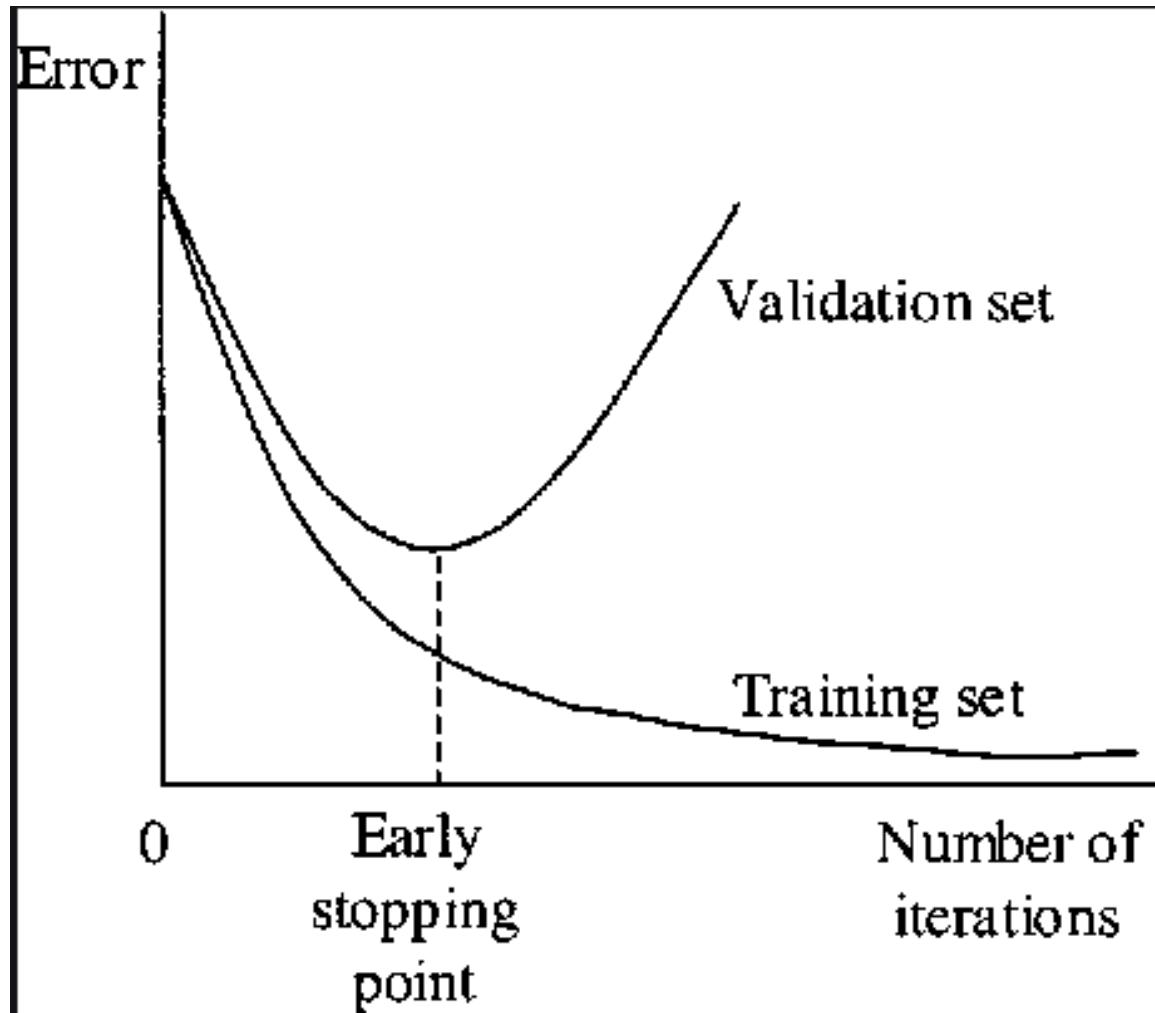
(b)



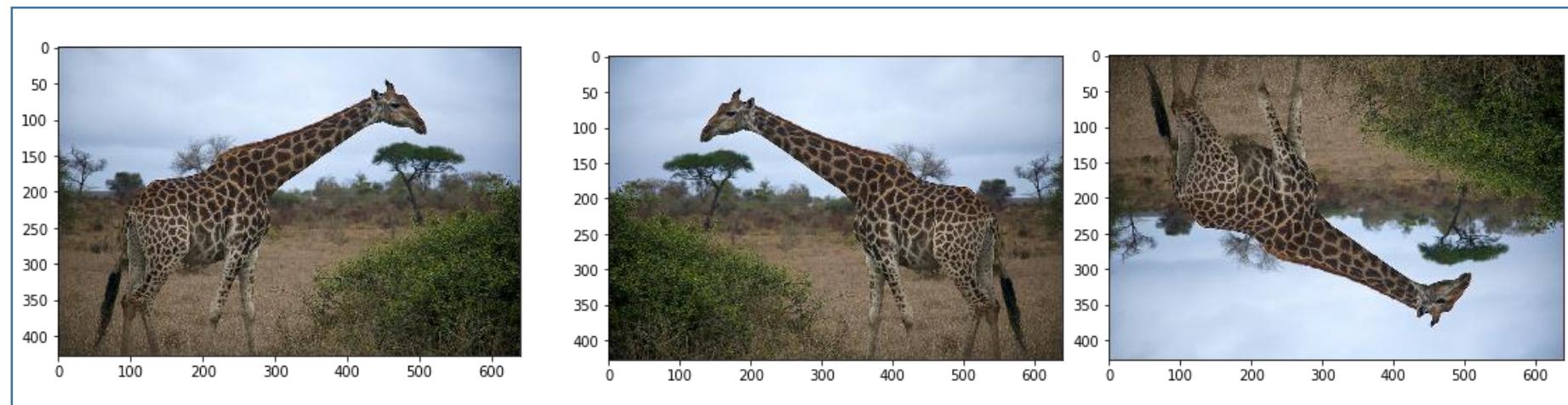
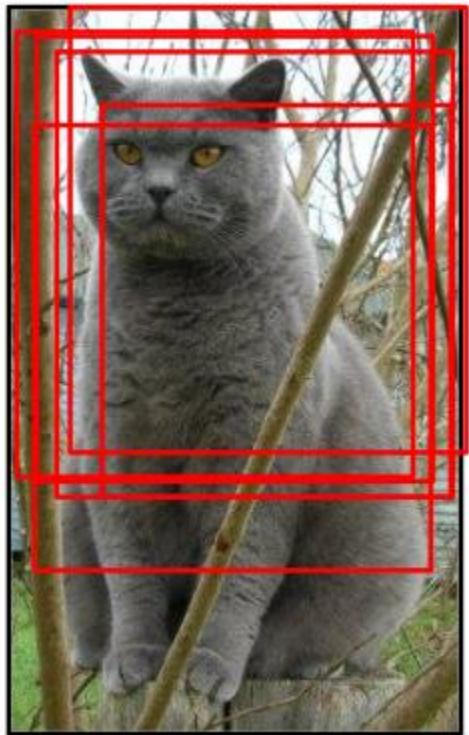
(c)

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.

Early stopping – avoid over-training



Data augmentation



Fancy PCA



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

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

Suggested reading

Improving Deep Learning using Generic Data Augmentation

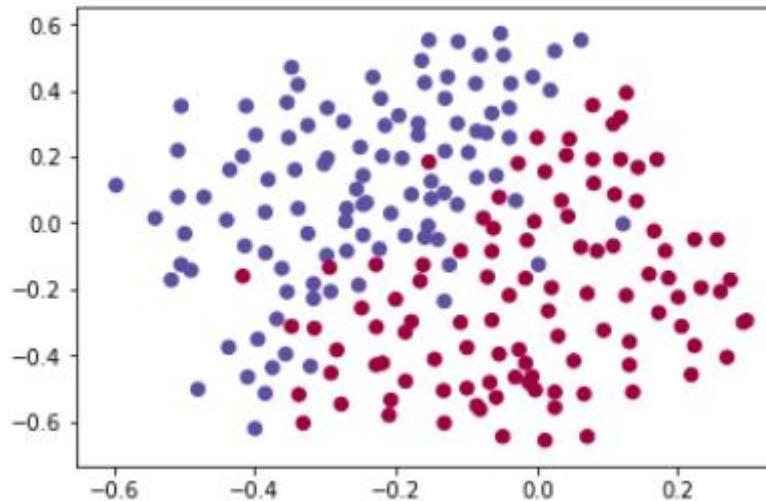
Luke Taylor

University of Cape Town
Department of Computer Science
Cape Town, South Africa
tylchr011@uct.ac.za

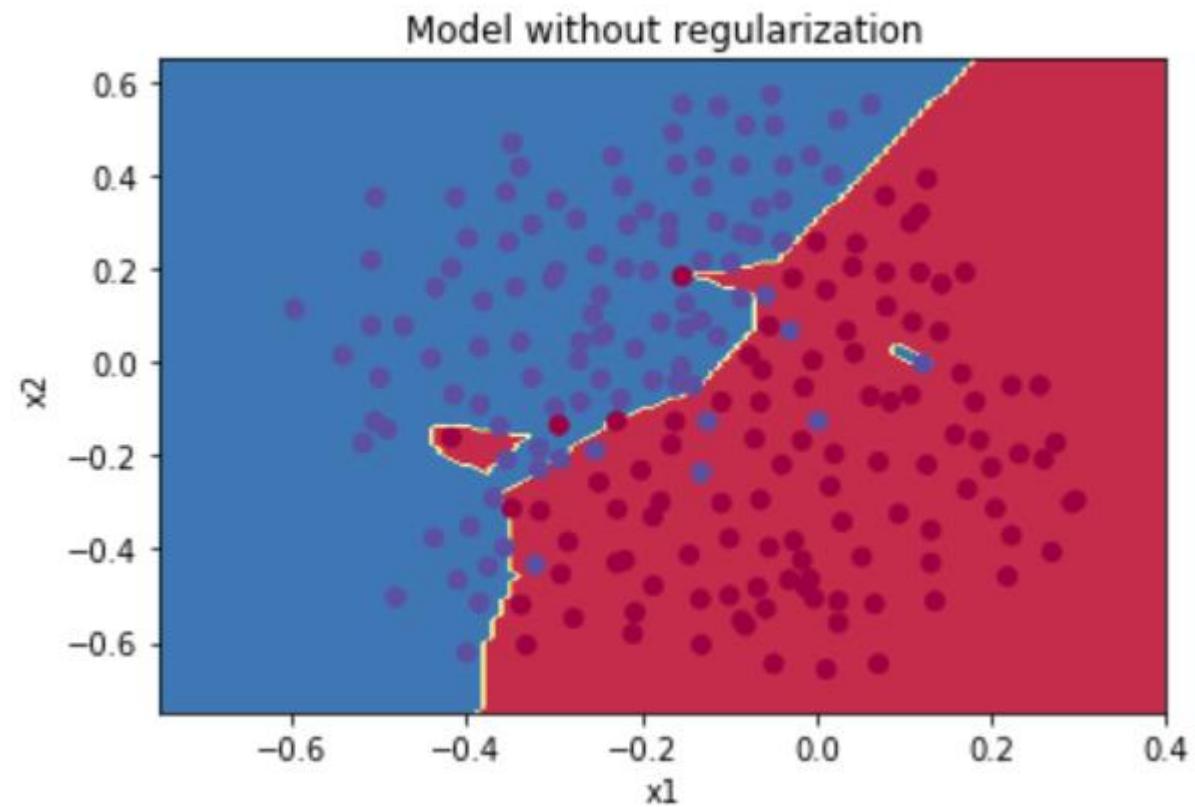
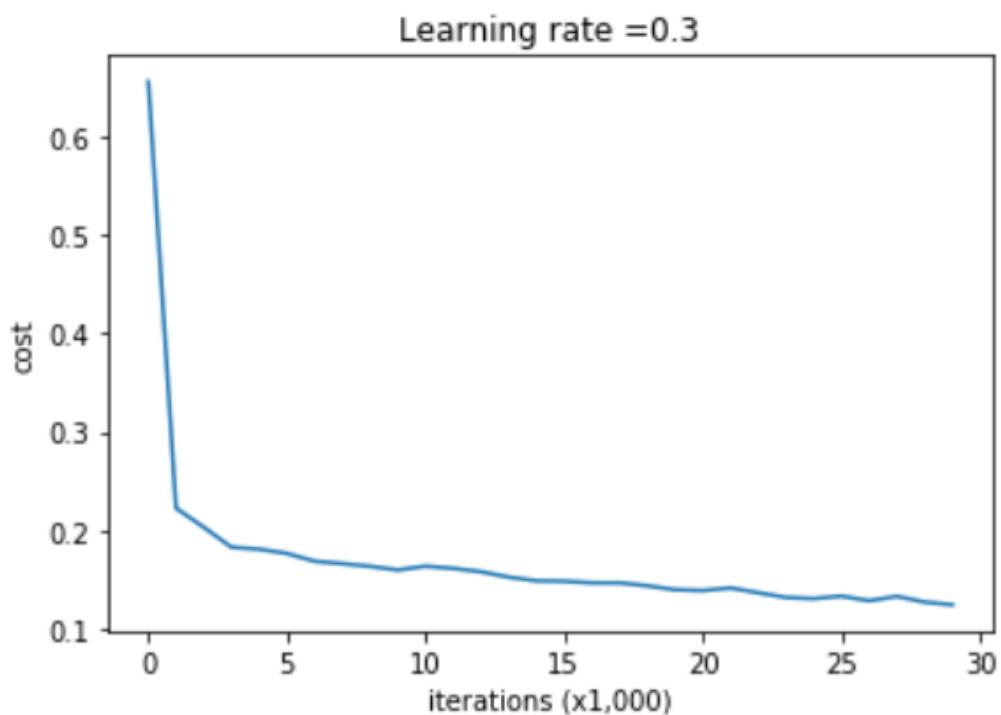
Geoff Nitschke

University of Cape Town
Department of Computer Science
Cape Town, South Africa
gnitschke@cs.uct.ac.za

Some comparison



Non-regularized version



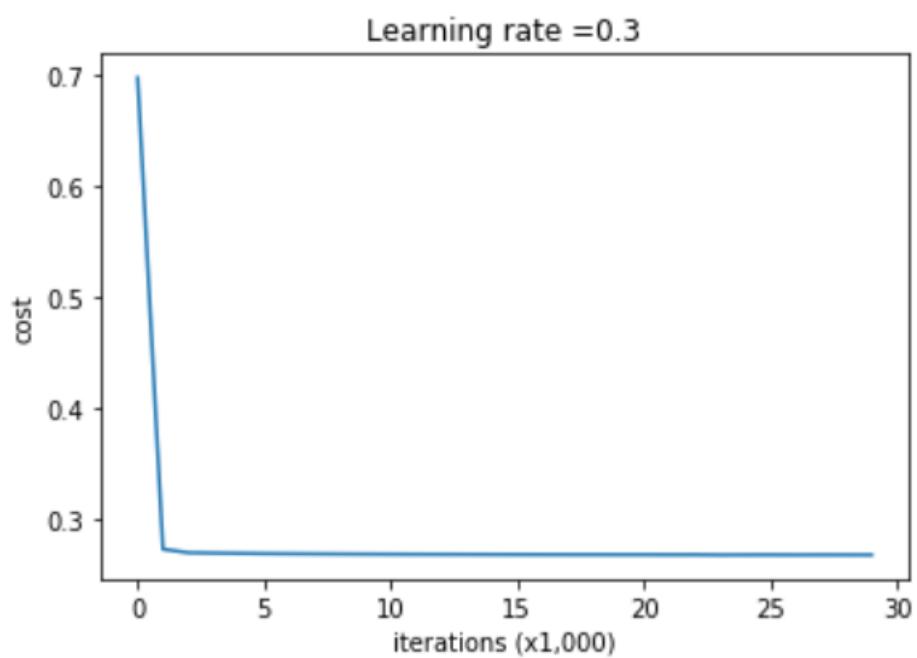
On the training set:

Accuracy: 0.9478672985781991

On the test set:

Accuracy: 0.915

L2 regularization

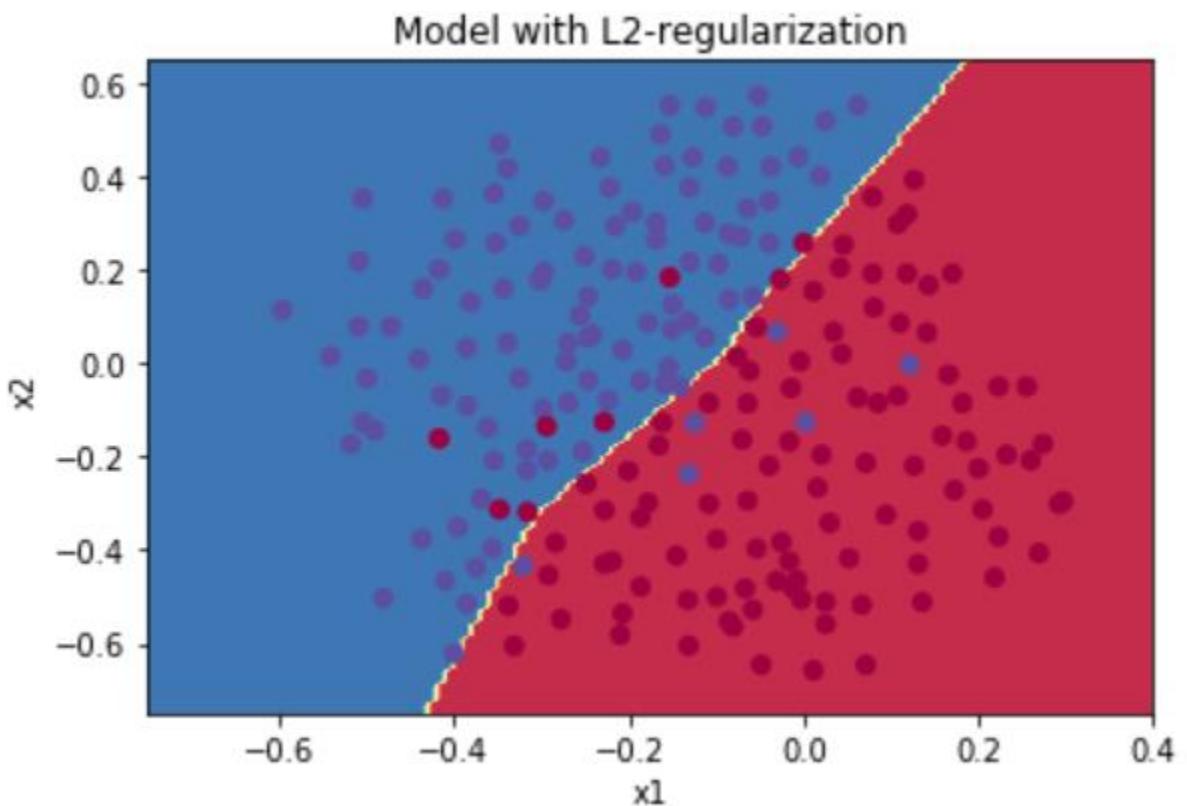


On the train set:

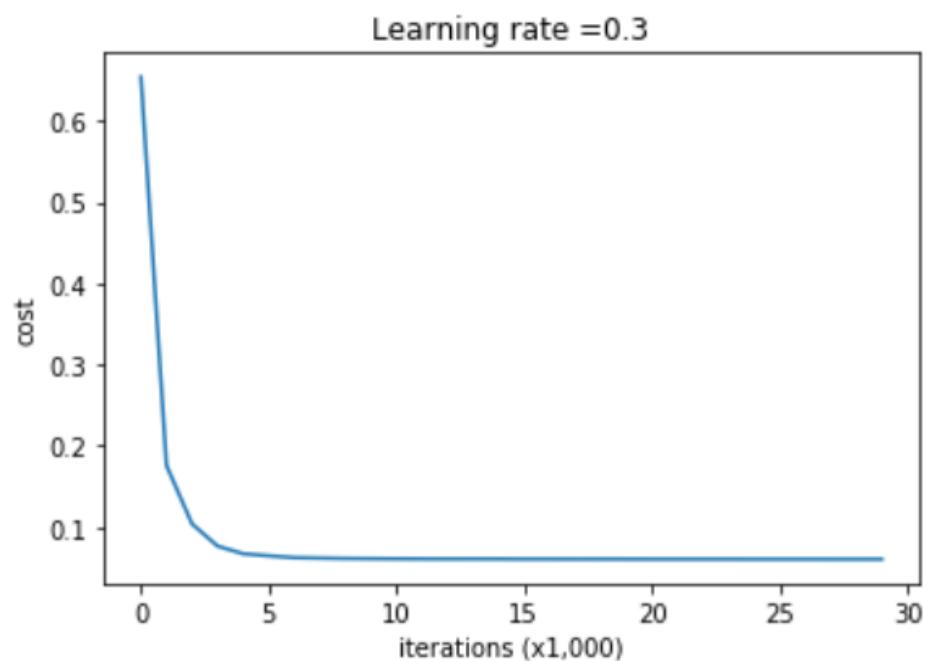
Accuracy: 0.9383886255924171

On the test set:

Accuracy: 0.93



Dropout

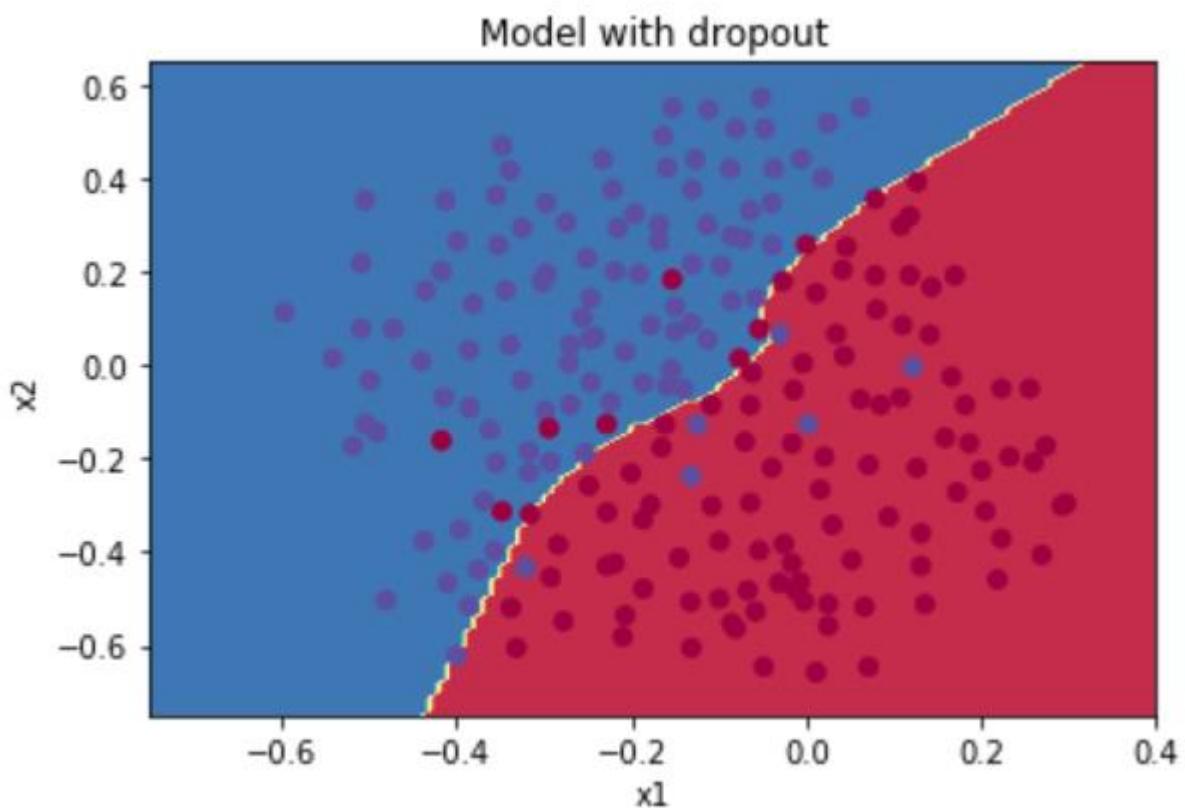


On the train set:

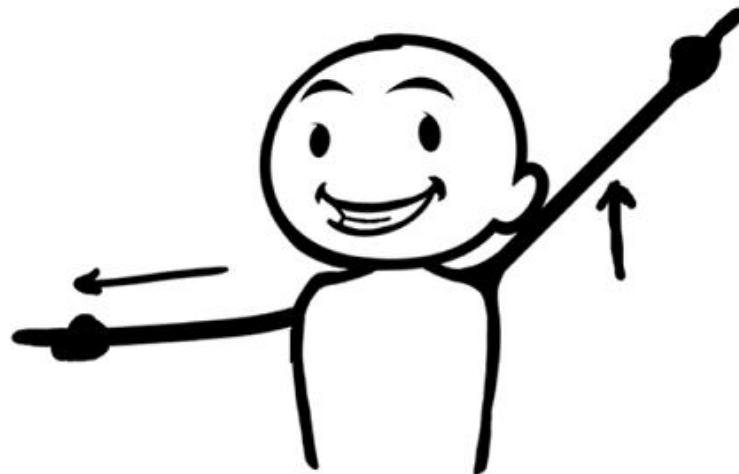
Accuracy: 0.9289099526066351

On the test set:

Accuracy: 0.95



Some notes on the ReLU function

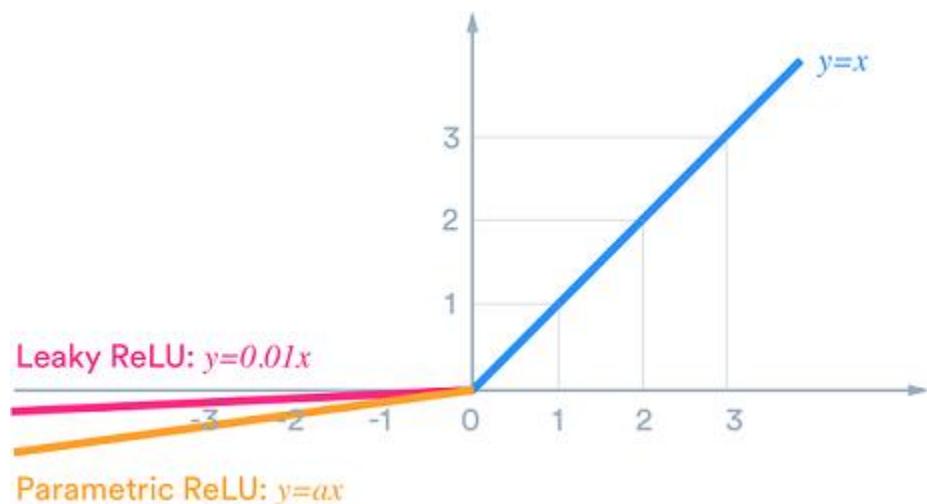


$$y = \begin{cases} 0, & x < 0 \\ x, & x \geq 0 \end{cases}$$

Idea of Dying ReLU

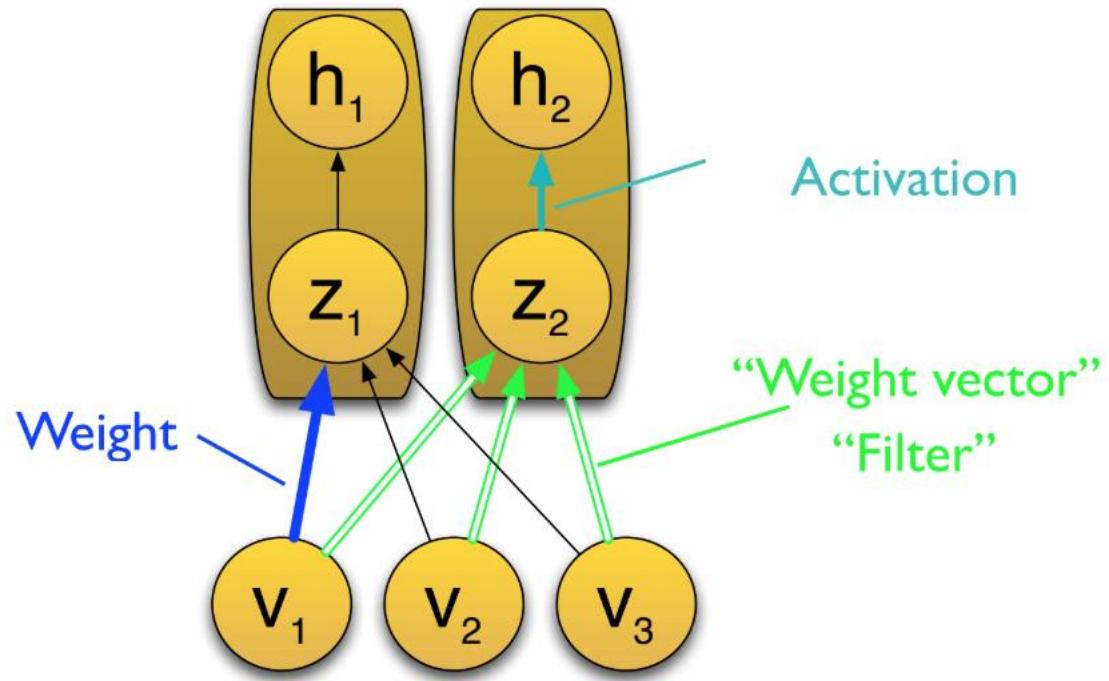
Need to ensure some gradient pass if $x < 0$

Leaky, parametric, exponential ReLU

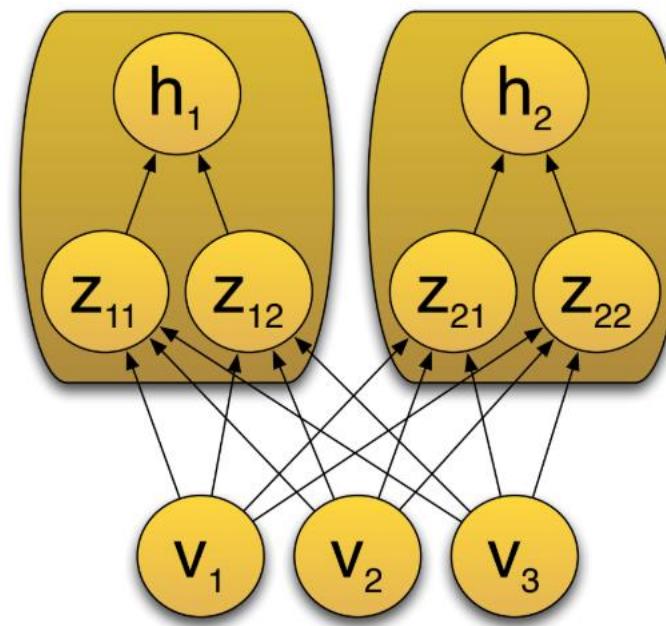


$$y = \begin{cases} \alpha(e^x - 1), & x < 0 \\ x, & x \geq 0 \end{cases}$$

Maxout activation

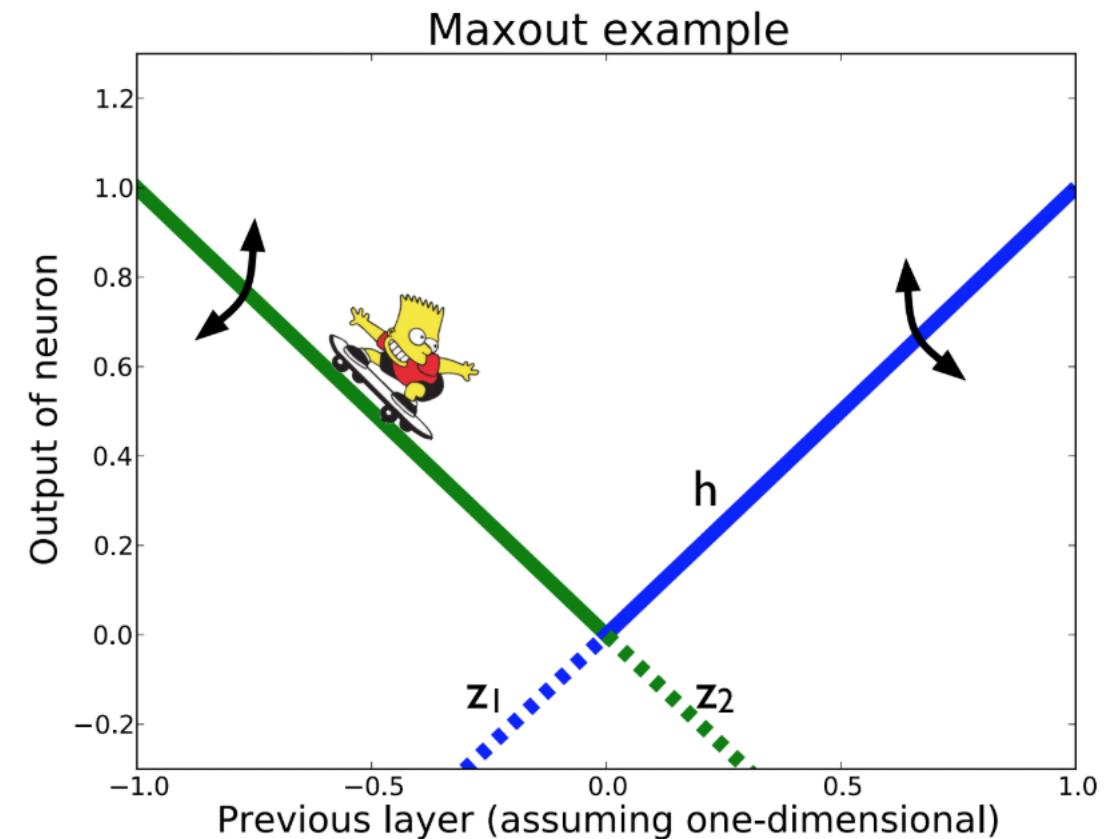
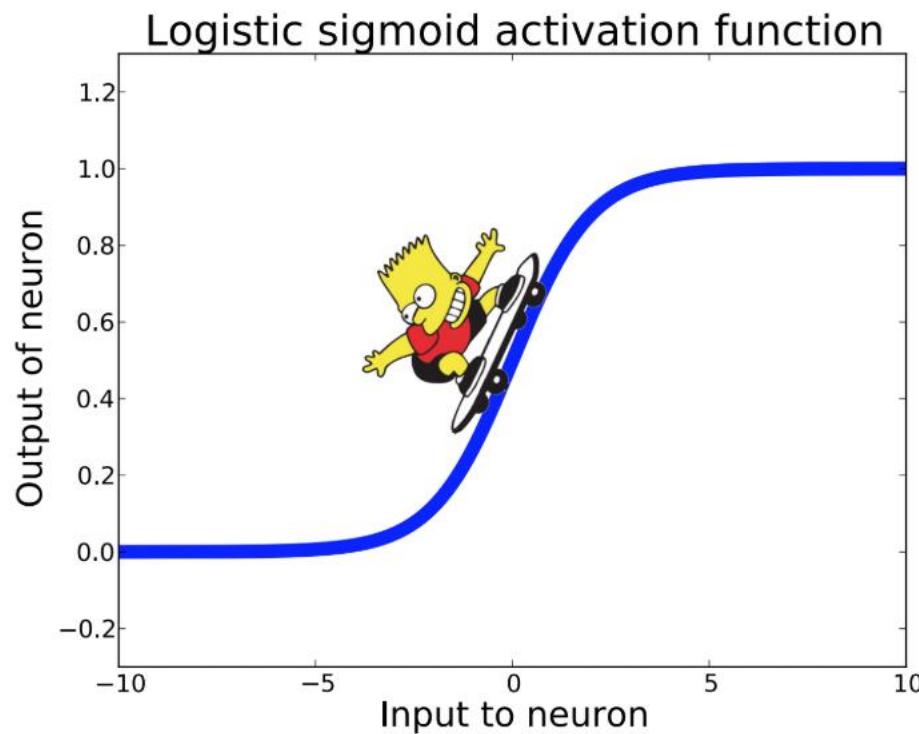


Maxout

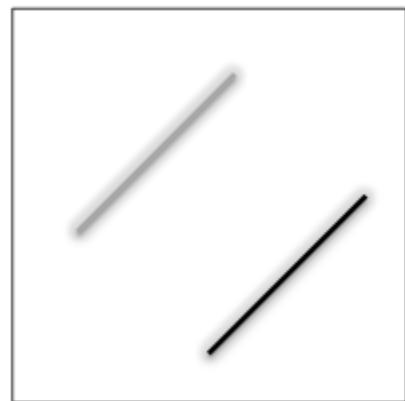


$$h_i = \max_j z_{ij}$$

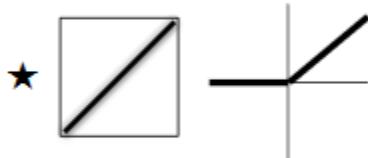
Gradients everywhere!



Stochastic pooling



a) Image



b) Filter

c) Rectified Linear

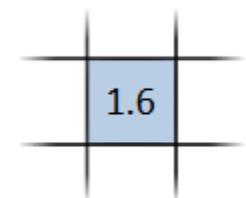
1.6	0	0	0.4	0	0
0	0	0	0	0	0
0	0	2.4	0	0	0.6

d) Activations, a_i

1.6	0	0	0.4	0	0
0	0	0	0	0	0
0	0	2.4	0	0	0.6

e) Probabilities, p_i

Sample a location
from $P(l)$: e.g. $l = 1$



f) Sampled
Activation, s

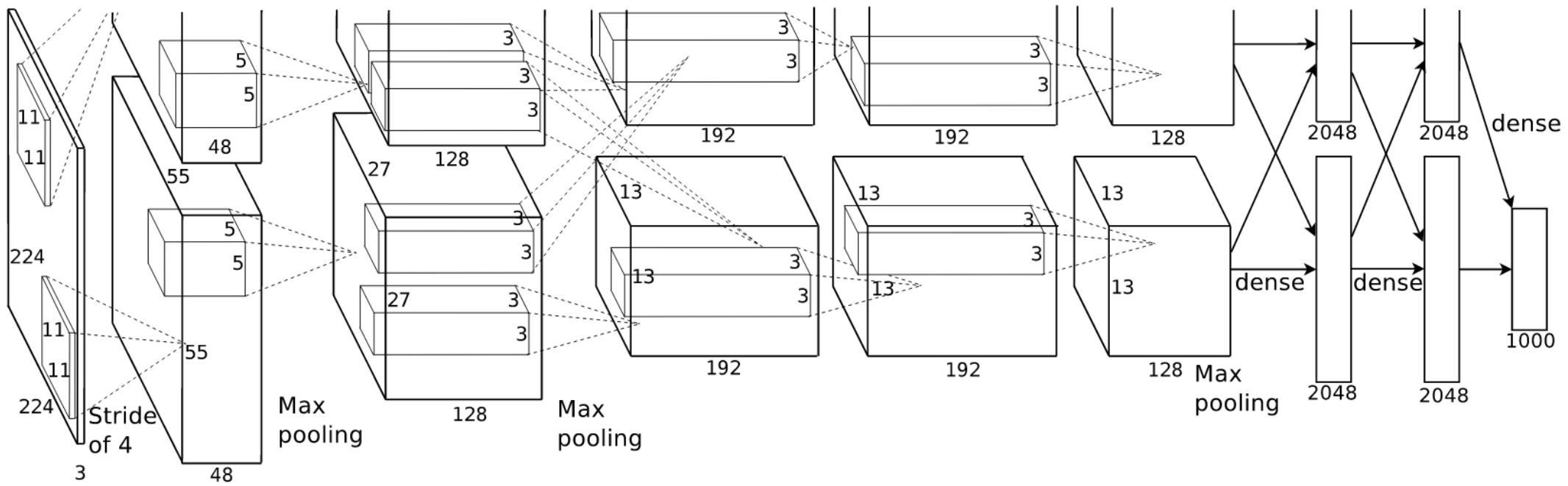
$$p_i = \frac{a_i}{\sum_{k \in R_j} a_k}$$

$$s_j = a_l \text{ where } l \sim P(p_1, \dots, p_{|R_j|})$$

Popular CNN architecture

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

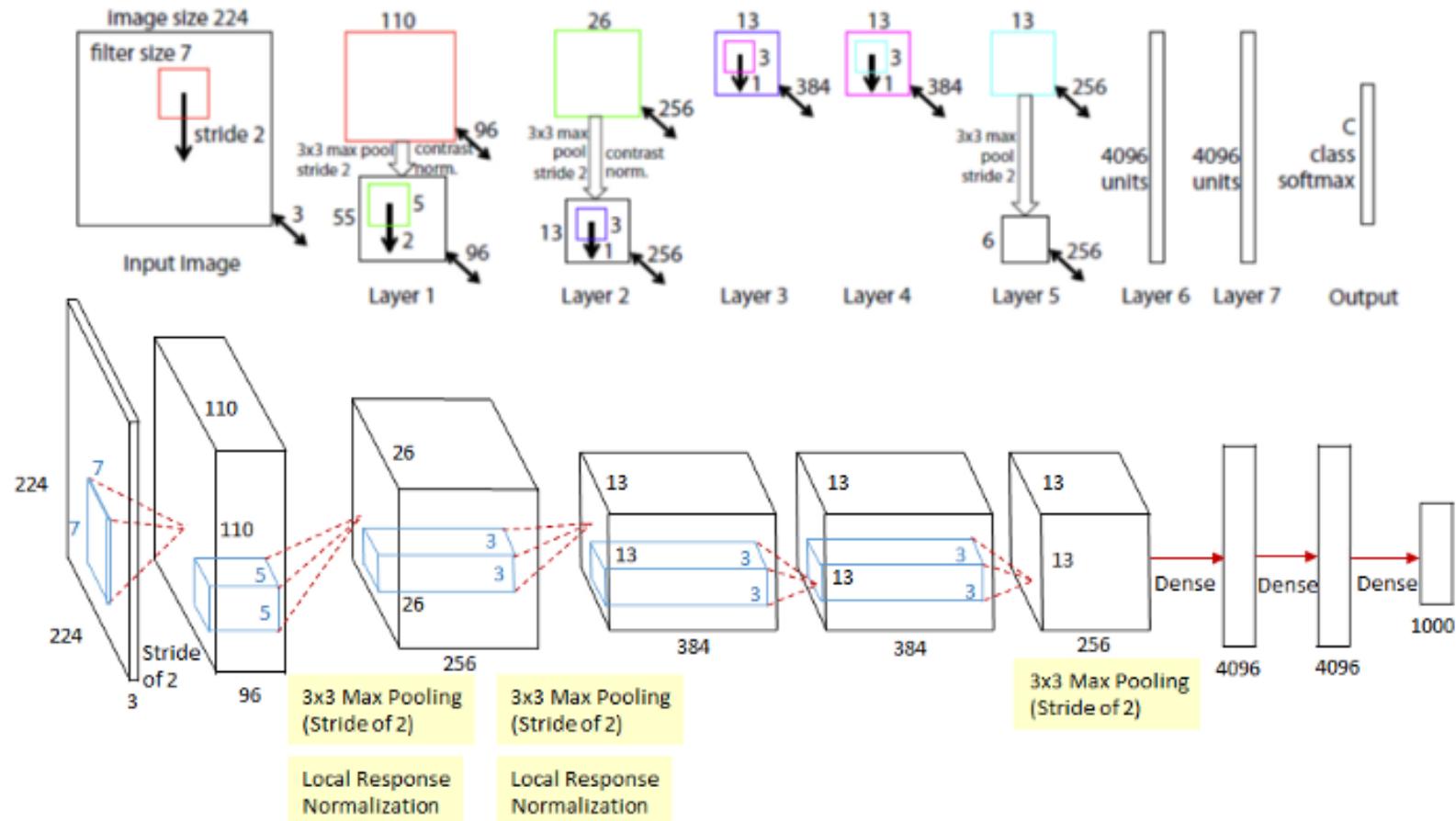
AlexNet



Highlights

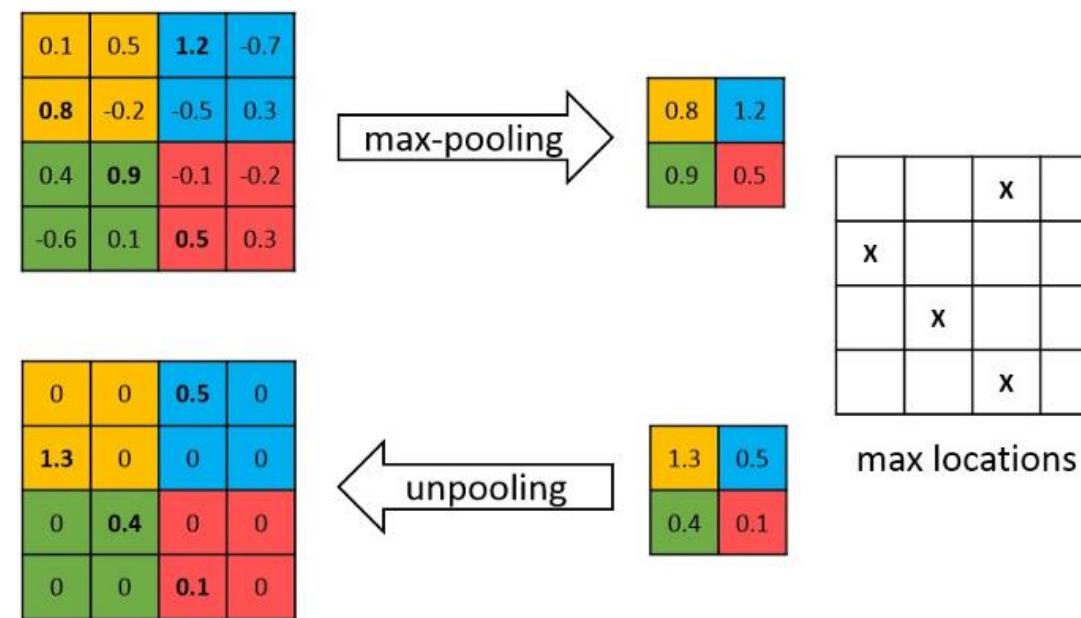
- First deep net in working mode after LeNet
- Eight layers
- ReLU non-linearity
- To avoid overfitting, some of the layers are skipped randomly during training

Zfnet



Highlights

- Mainly used for visualization by adding a symmetric decoder
- Used unpool layer for upsampling



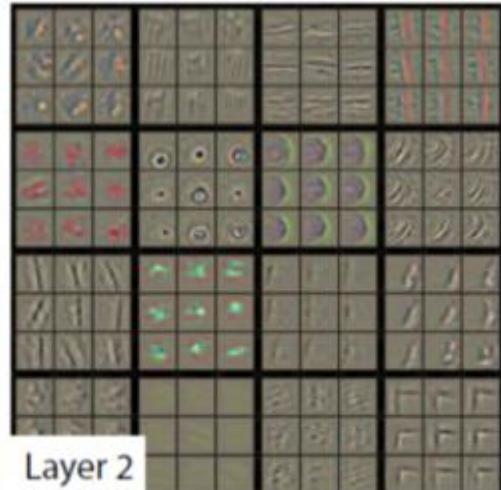
Visualization of layers



Layer 1



Extreme low and
high frequencies.
No mid frequencies.

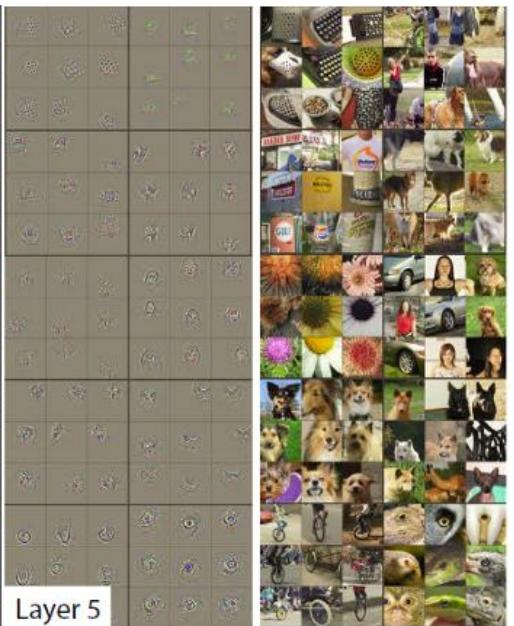


Layer 2

Aliasing problem



Layer 4



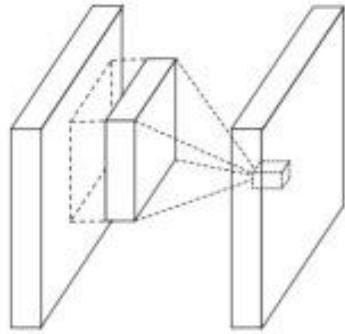
Layer 5



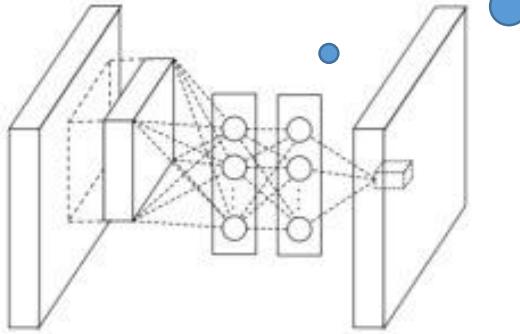
VGGnet



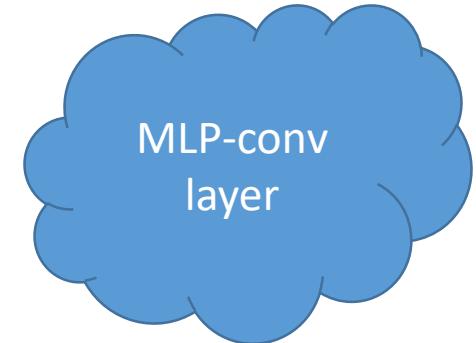
Network in network



(a) Linear convolution layer



(b) Mlpconv layer

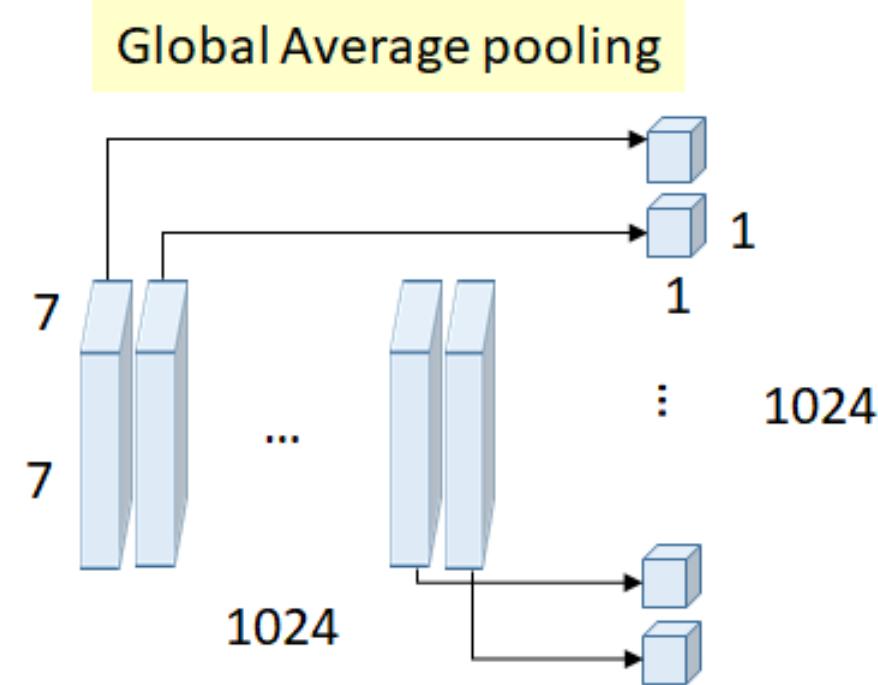
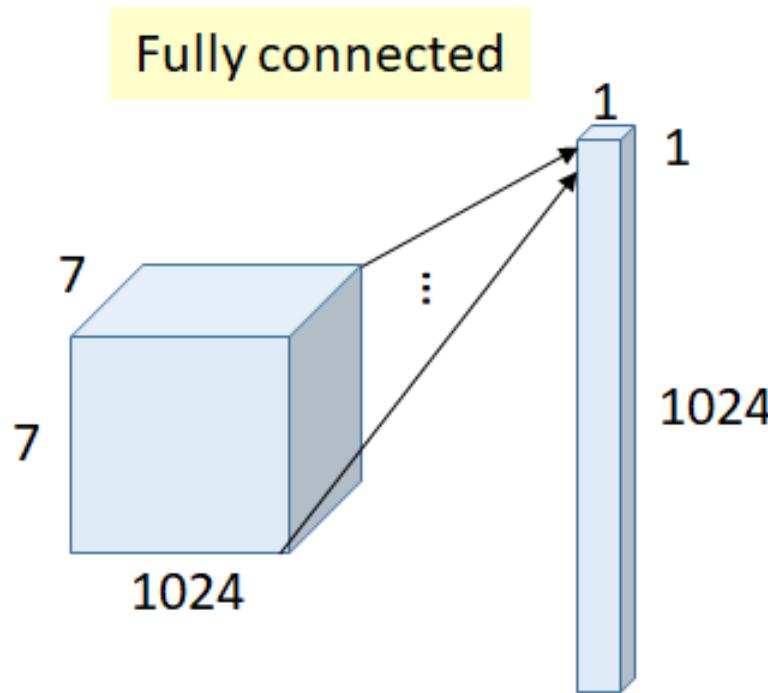


Representations that achieve good abstraction are non-linear function of the input data!

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.

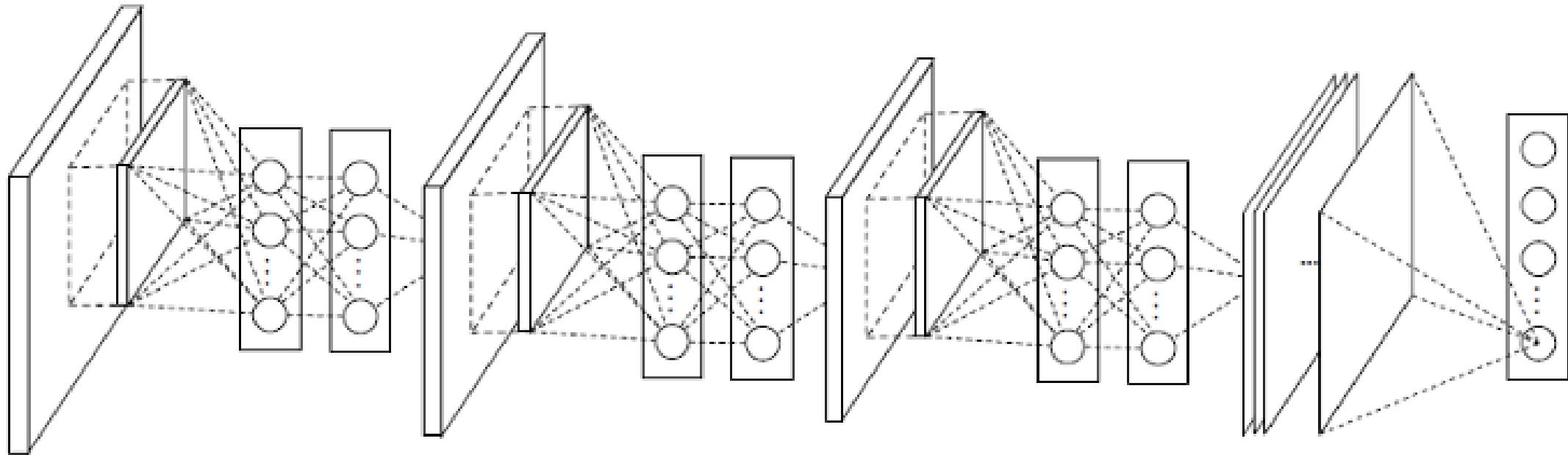
$$\begin{aligned} f_{i,j,k_1}^1 &= \max(w_{k_1}^1{}^T x_{i,j} + b_{k_1}, 0). \\ f_{i,j,k} &= \max(w_k^T x_{i,j}, 0). \\ &\vdots \\ f_{i,j,k_n}^n &= \max(w_{k_n}^n{}^T f_{i,j}^{n-1} + b_{k_n}, 0). \end{aligned}$$

Network in network

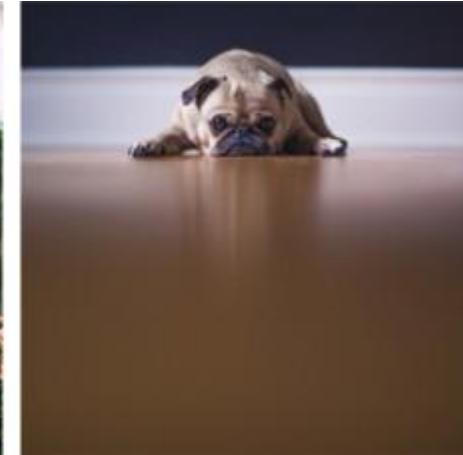


No learnable parameter!

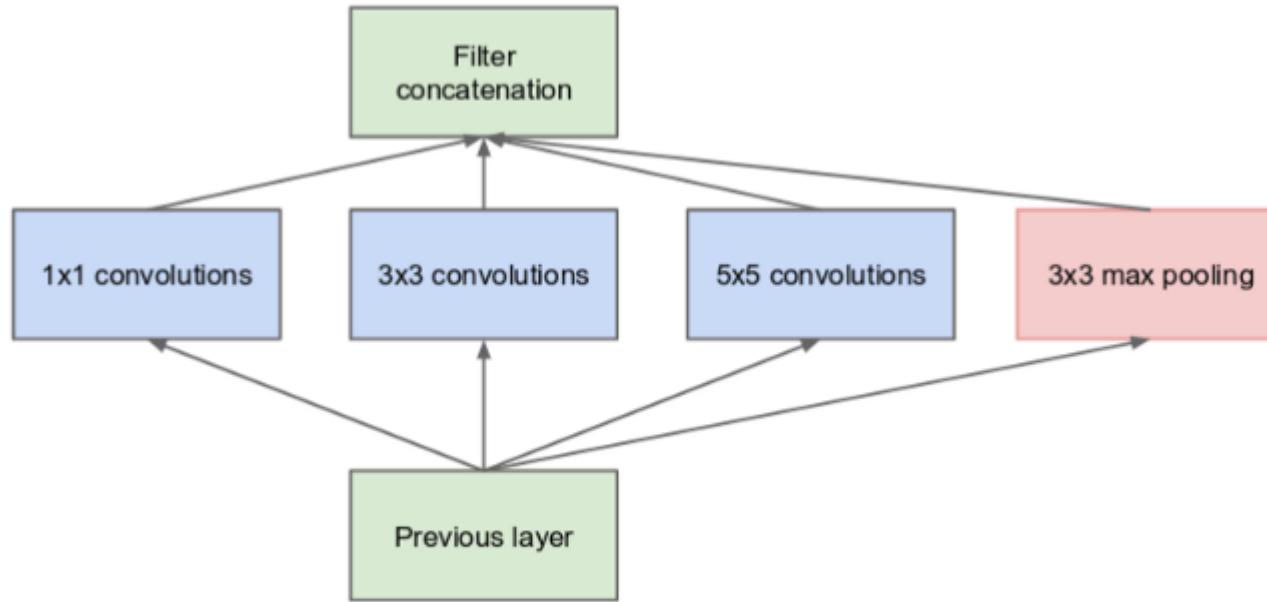
Can work with images of different resolution!



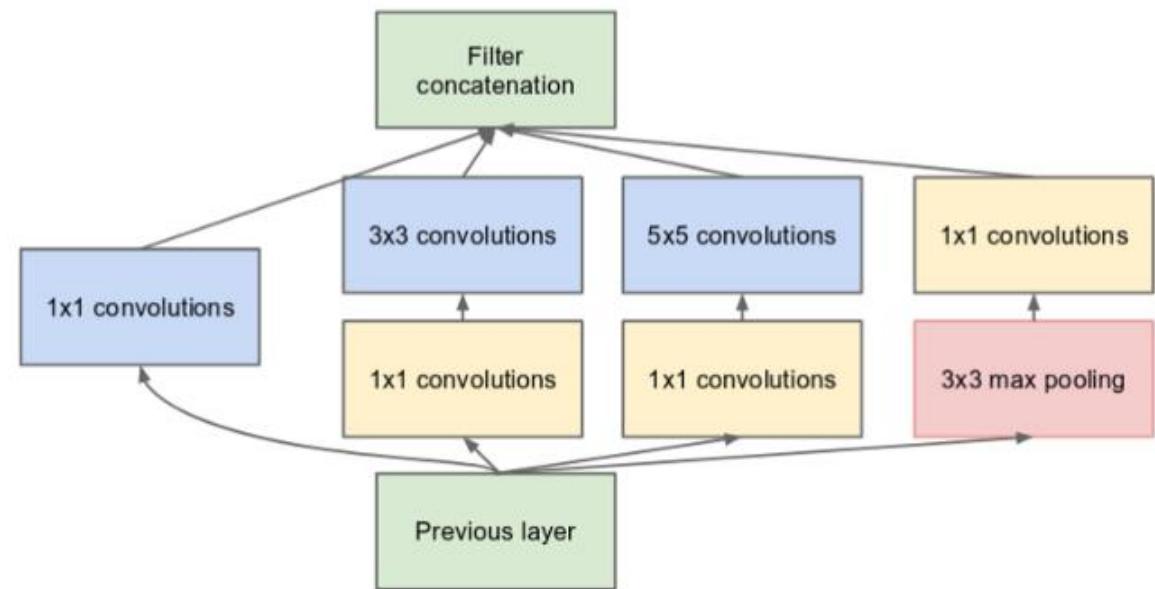
Inception architecture



What is a good kernel size for DOG feature extraction?

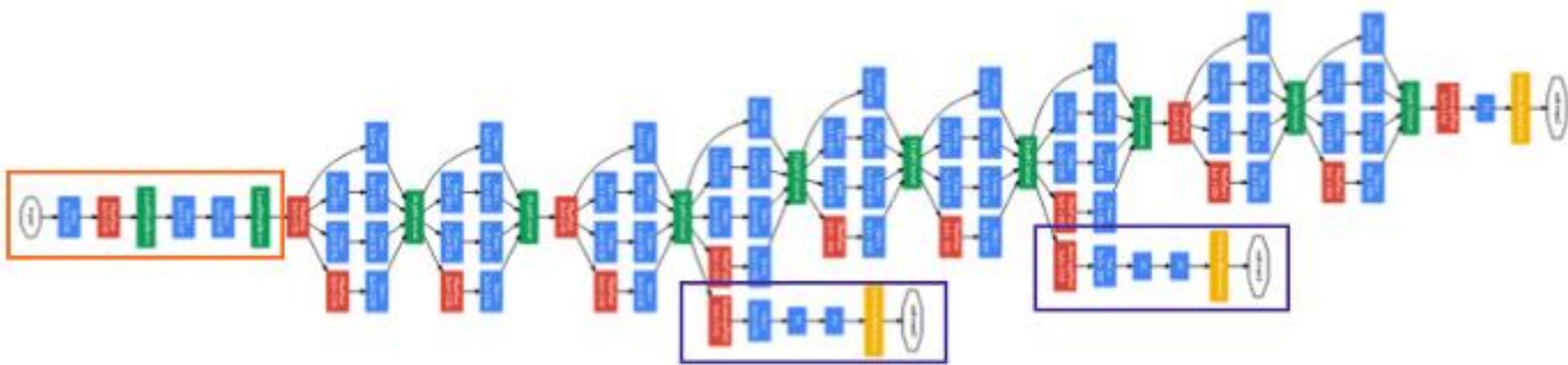


(a) Inception module, naïve version



(b) Inception module with dimension reductions

Inception v1 (Googlenet)

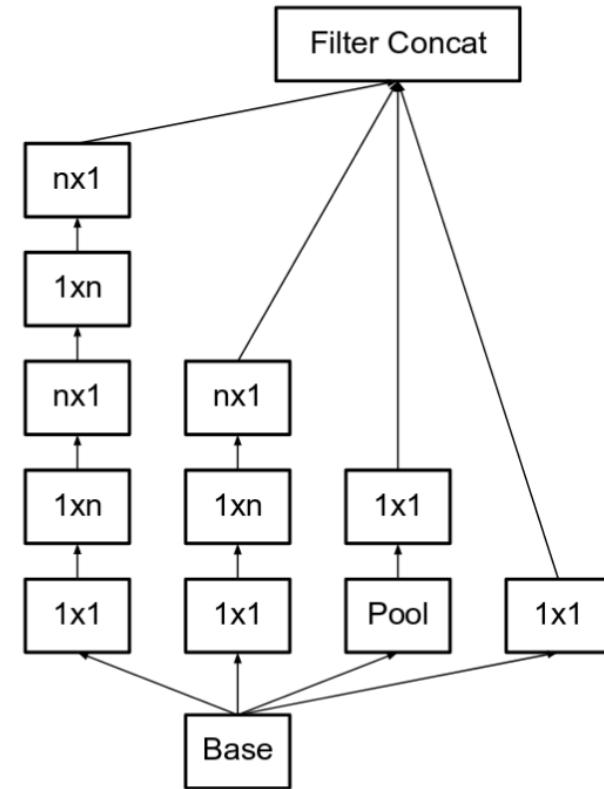
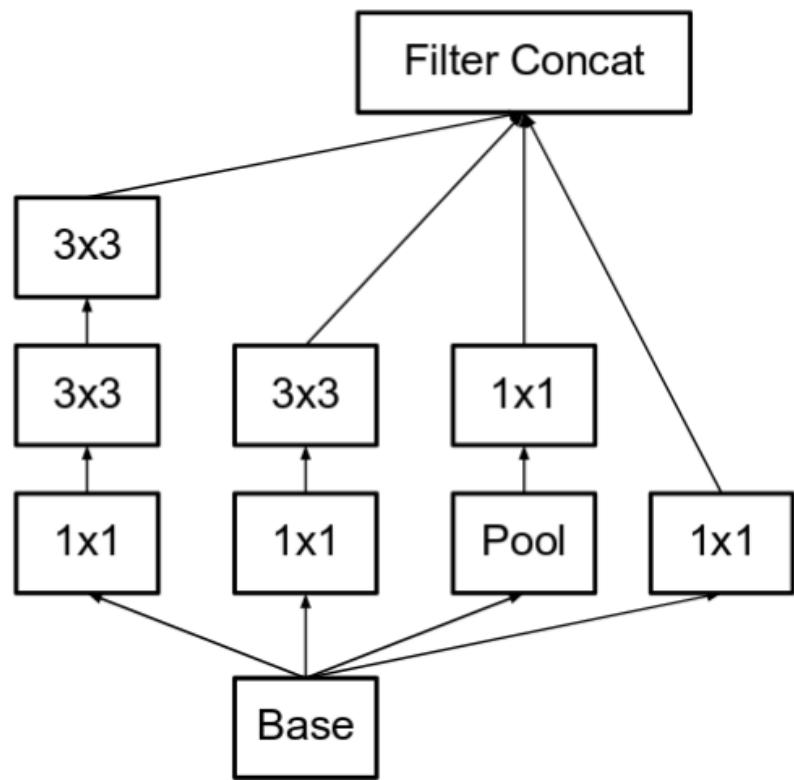


```
# The total loss used by the inception net during training.  
total_loss = real_loss + 0.3 * aux_loss_1 + 0.3 * aux_loss_2
```

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

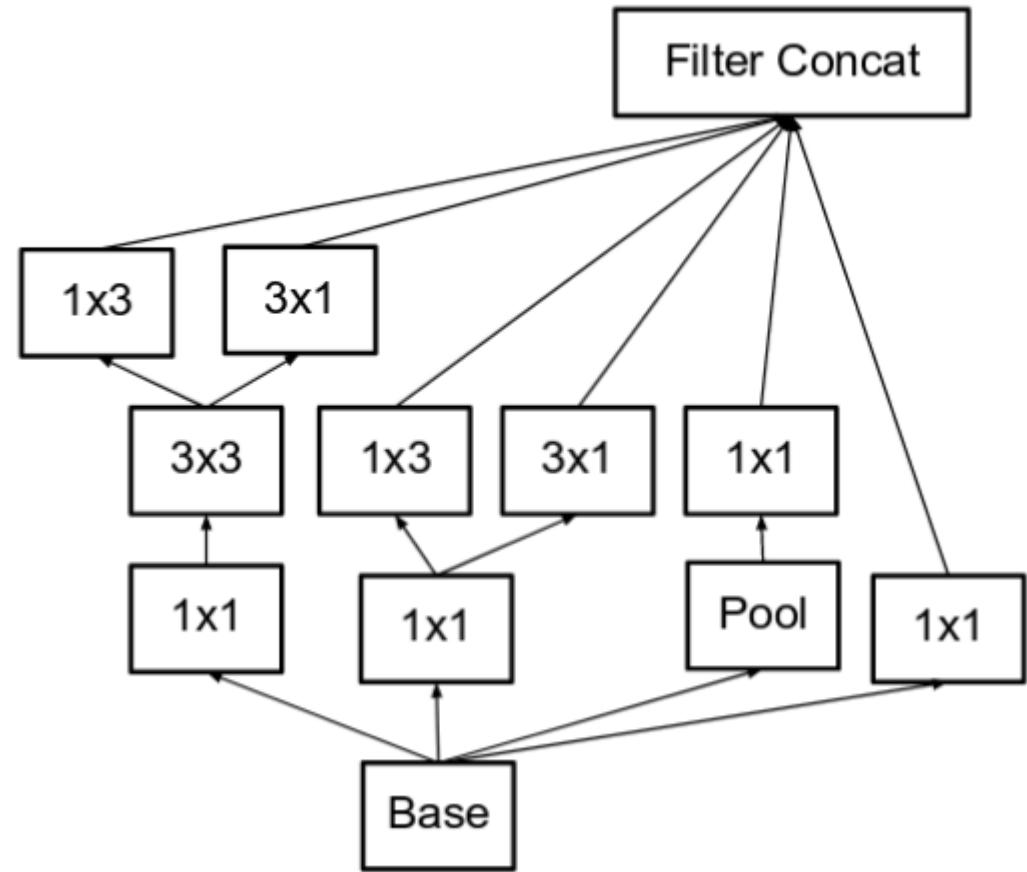
Inception v2

- Filter separability and small filter size



Representational bottleneck

- Wider network than deeper network
- Sudden reduction in depth causes information loss
- Loss – Rep Bottleneck



type	patch size/stride or remarks	input size
conv	$3 \times 3 / 2$	$299 \times 299 \times 3$
conv	$3 \times 3 / 1$	$149 \times 149 \times 32$
conv padded	$3 \times 3 / 1$	$147 \times 147 \times 32$
pool	$3 \times 3 / 2$	$147 \times 147 \times 64$
conv	$3 \times 3 / 1$	$73 \times 73 \times 64$
conv	$3 \times 3 / 2$	$71 \times 71 \times 80$
conv	$3 \times 3 / 1$	$35 \times 35 \times 192$
$3 \times$ Inception	As in figure 5	$35 \times 35 \times 288$
$5 \times$ Inception	As in figure 6	$17 \times 17 \times 768$
$2 \times$ Inception	As in figure 7	$8 \times 8 \times 1280$
pool	8×8	$8 \times 8 \times 2048$
linear	logits	$1 \times 1 \times 2048$
softmax	classifier	$1 \times 1 \times 1000$

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

$$y_{label} = [1, 0, 0, 0\dots 0].$$

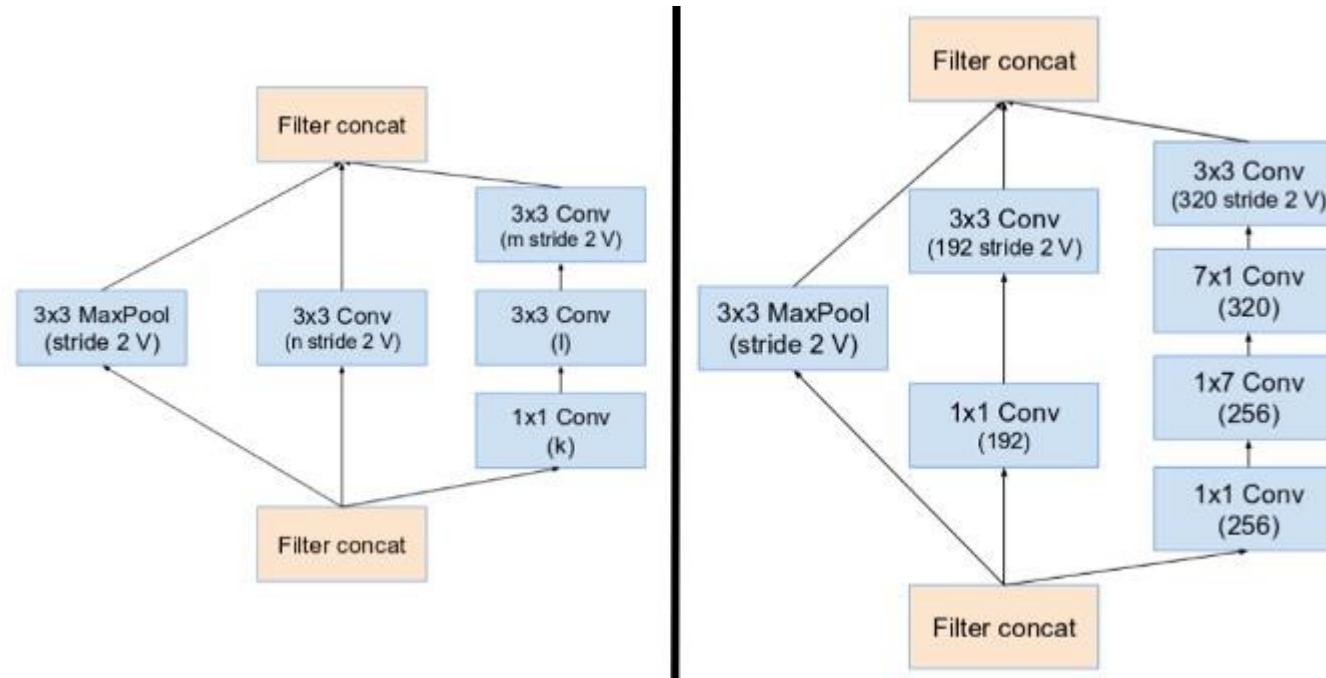
Softmax output is usually of the form

$$y_{out} = [0.87, 0.001, 0.04, 0.1, \dots, 0.03].$$

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

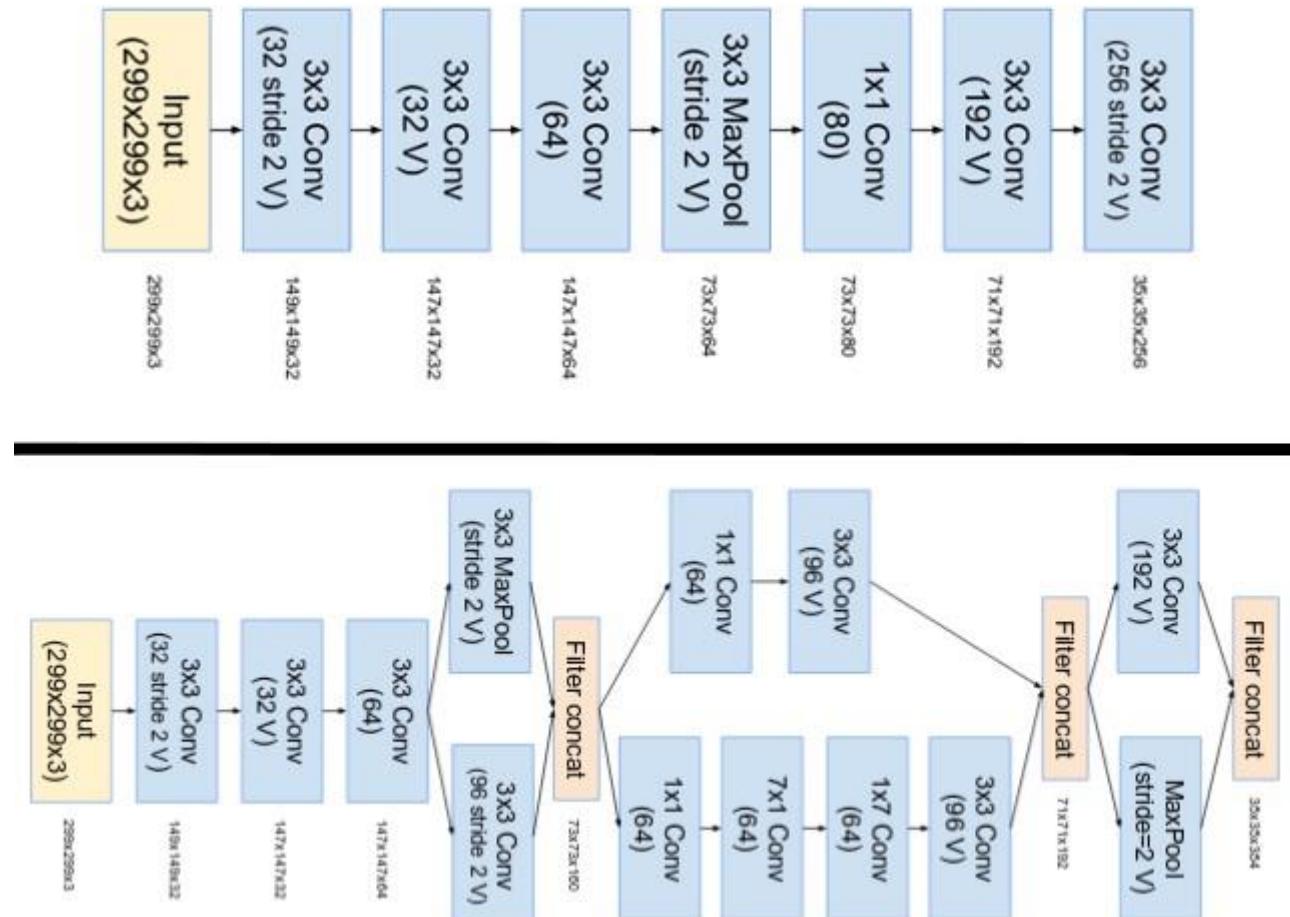
$$y_{label} = [1 - \epsilon, \frac{\epsilon}{K-1}, \frac{\epsilon}{K-1}, \frac{\epsilon}{K-1} \dots \frac{\epsilon}{K-1}].$$

Inception v4

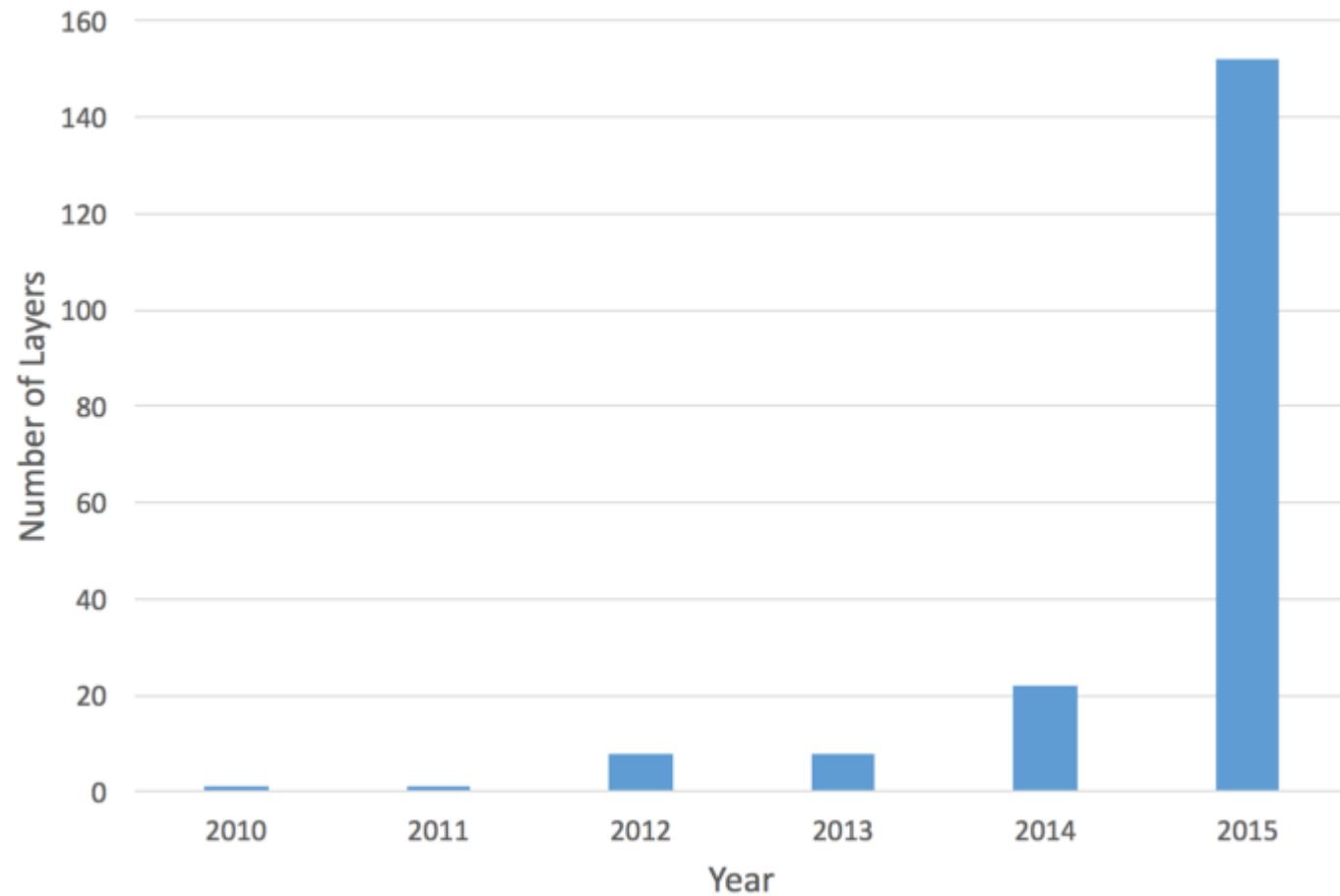


Reduce spatial dimensions!

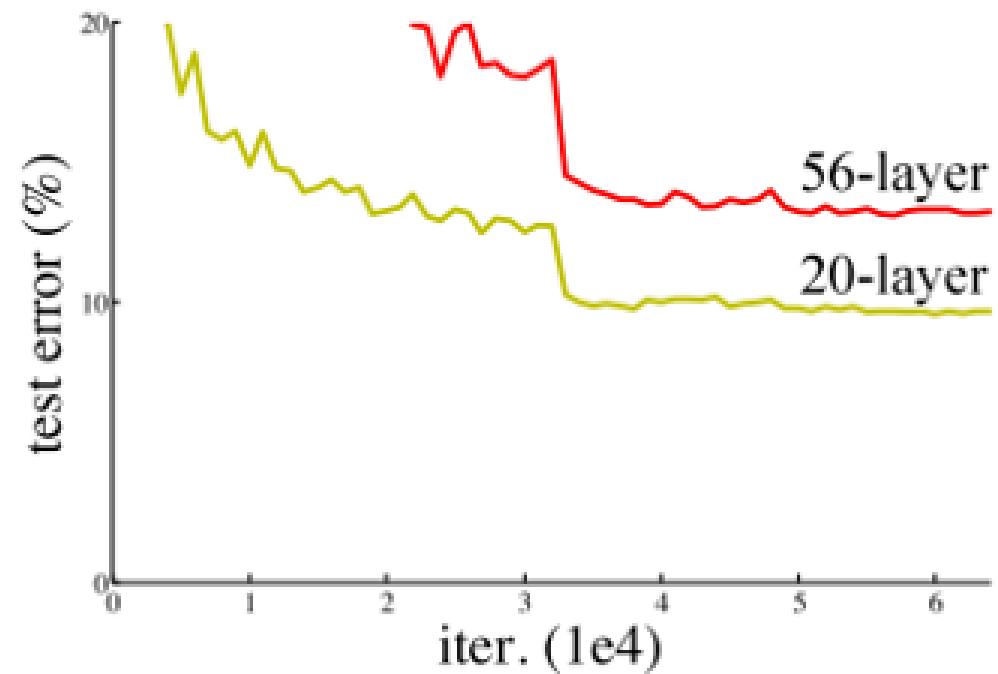
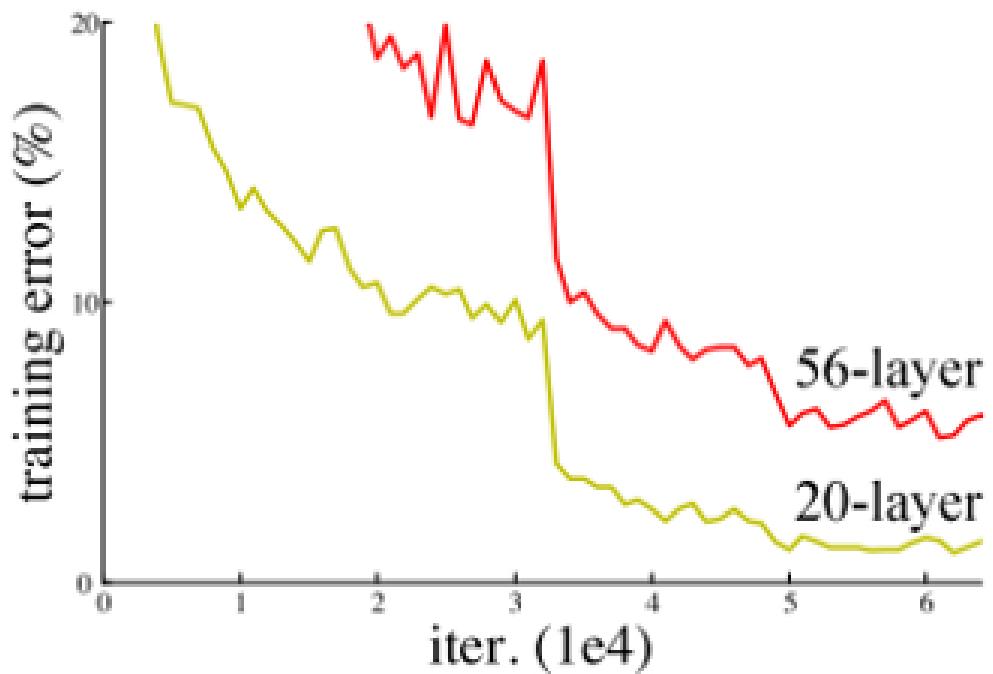
Simplify the “stem”



Network Depth of ImageNet Challenge Winner

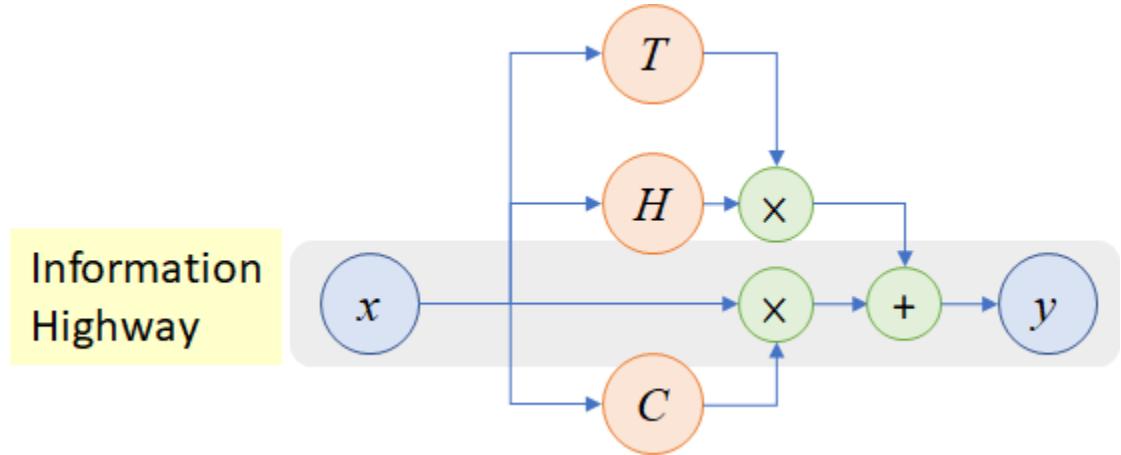


Going deeper in CNN



Highway network

In plain network



$$\mathbf{y} = H(\mathbf{x}, \mathbf{W}_H) \cdot T(\mathbf{x}, \mathbf{W}_T) + \mathbf{x} \cdot C(\mathbf{x}, \mathbf{W}_C).$$

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

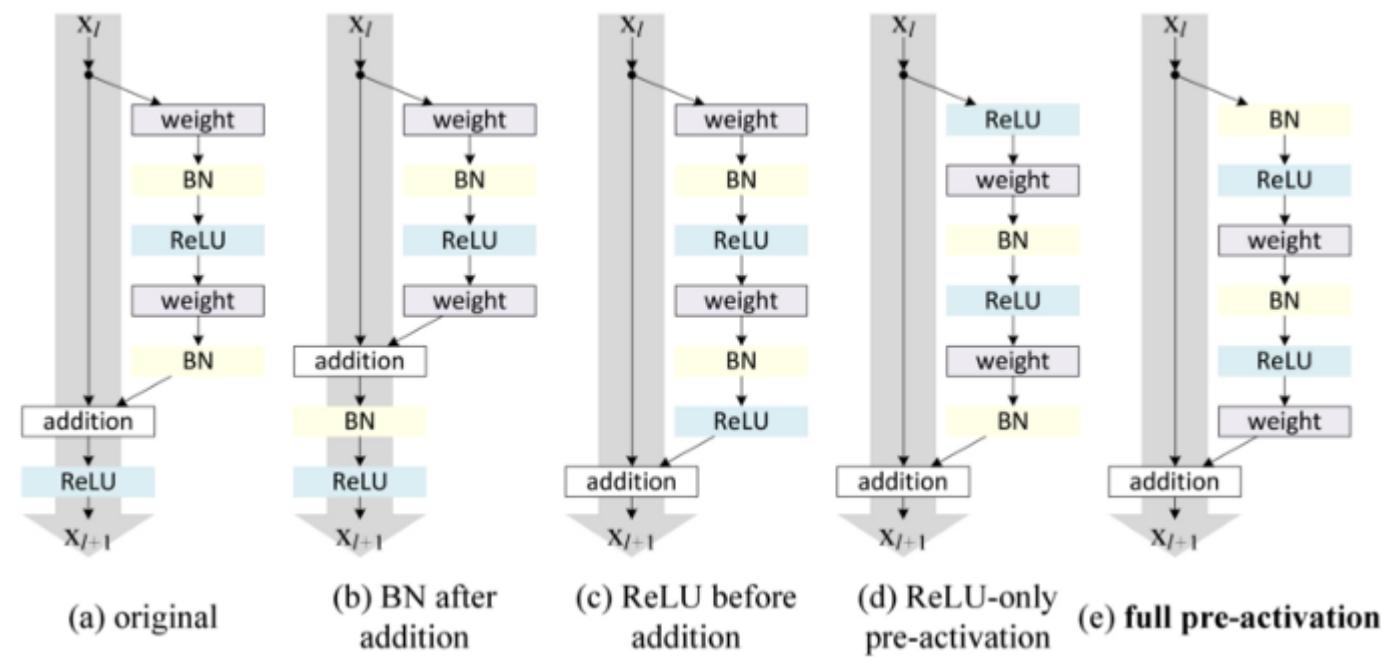
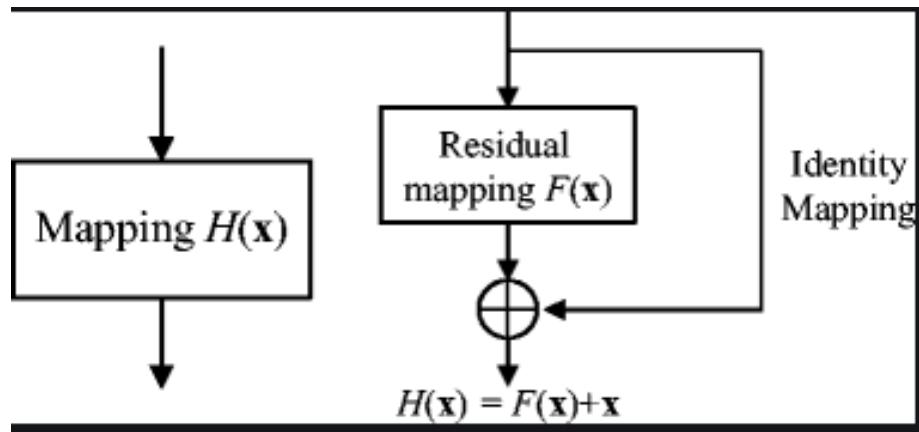
T: Transform gate

$$T(\mathbf{x}) = \sigma(\mathbf{W}_T^T \mathbf{x} + \mathbf{b}_T)$$

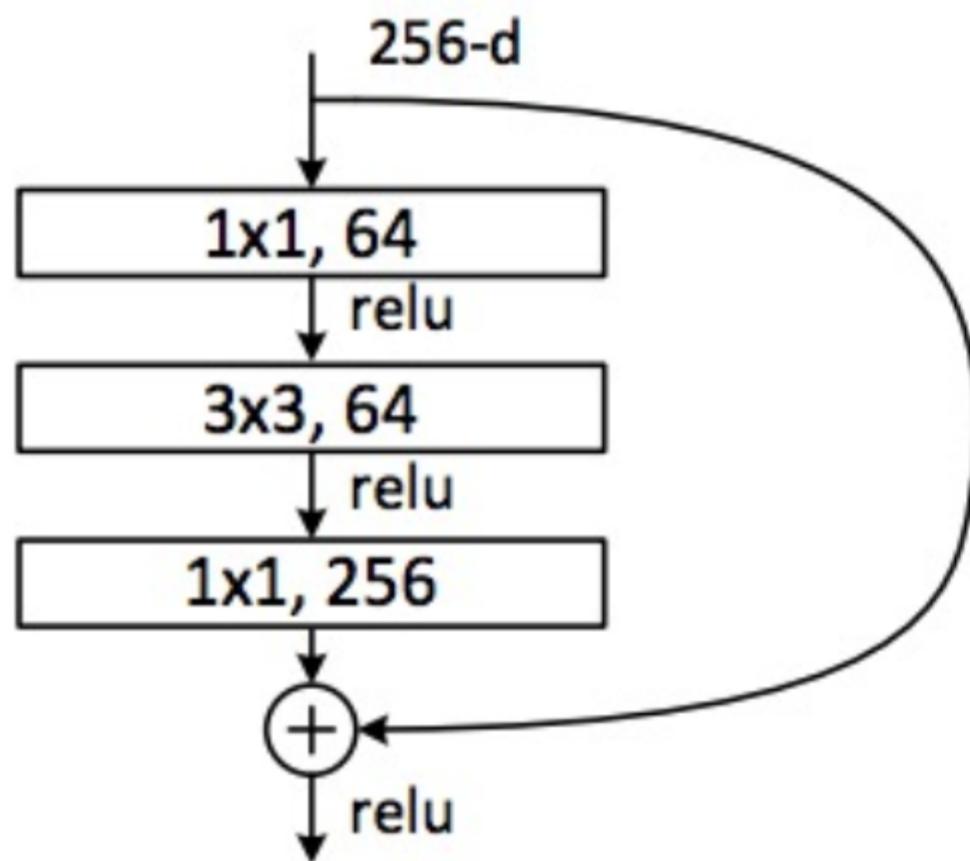
C: Carry gate, usually $C = 1 - T$

$$\mathbf{y} = \begin{cases} \mathbf{x}, & \text{if } T(\mathbf{x}, \mathbf{W}_T) = 0, \\ H(\mathbf{x}, \mathbf{W}_H), & \text{if } T(\mathbf{x}, \mathbf{W}_T) = 1. \end{cases}$$

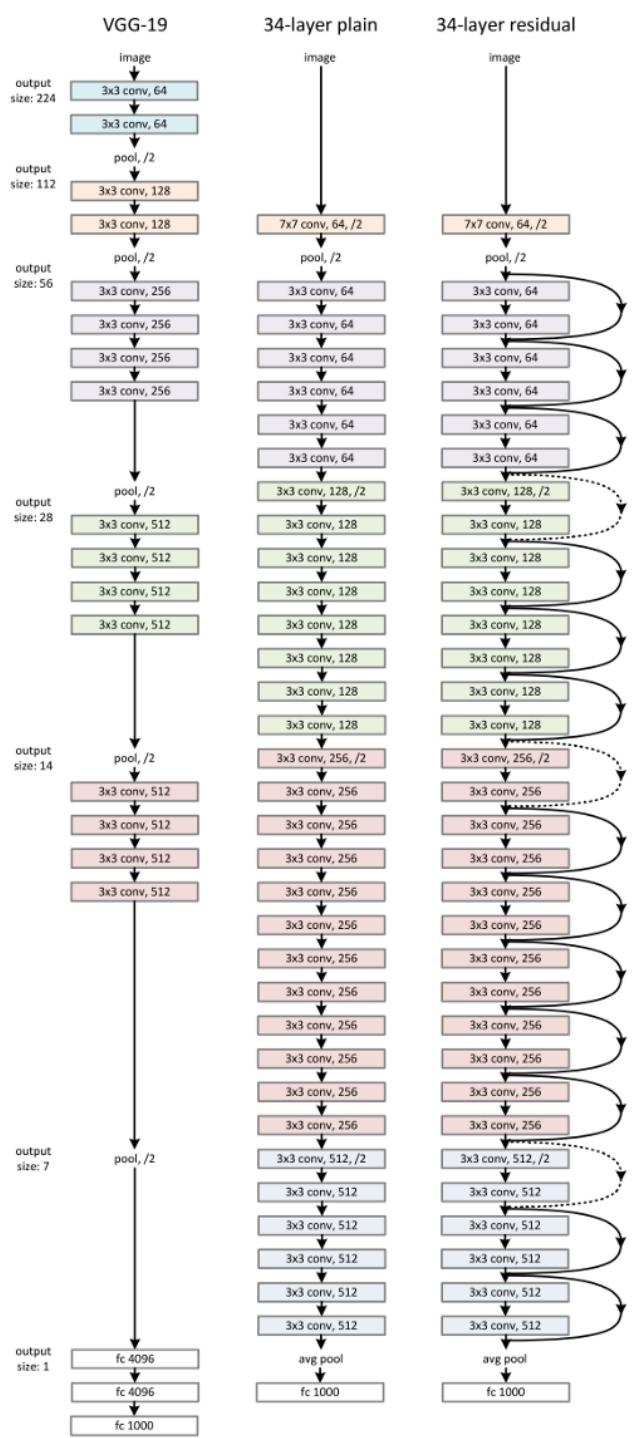
Residual networks



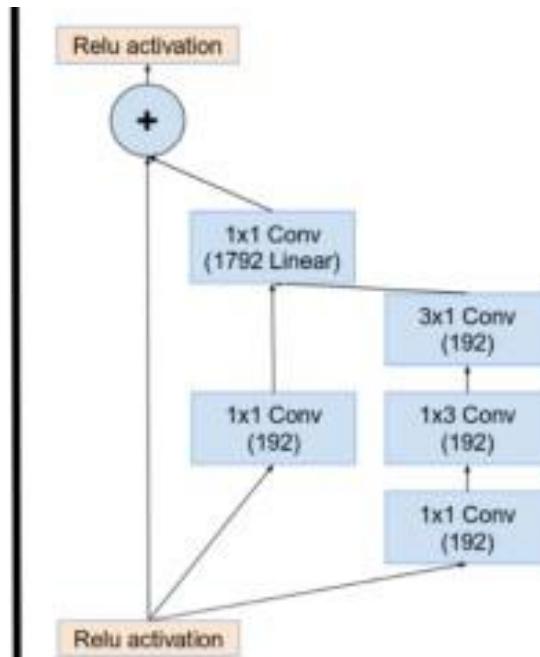
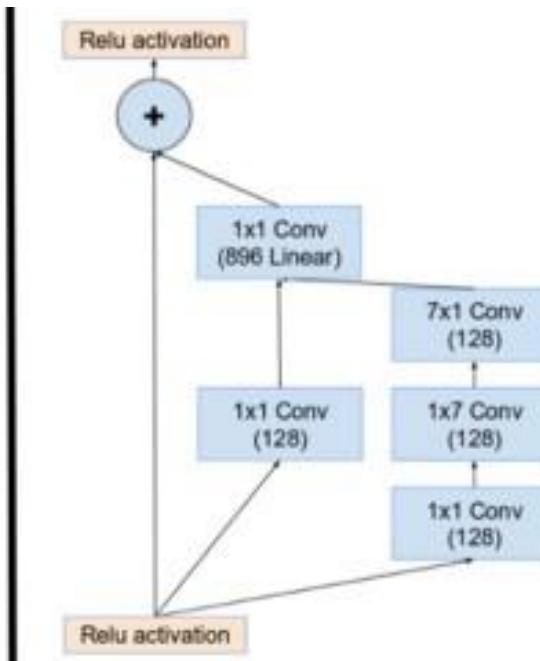
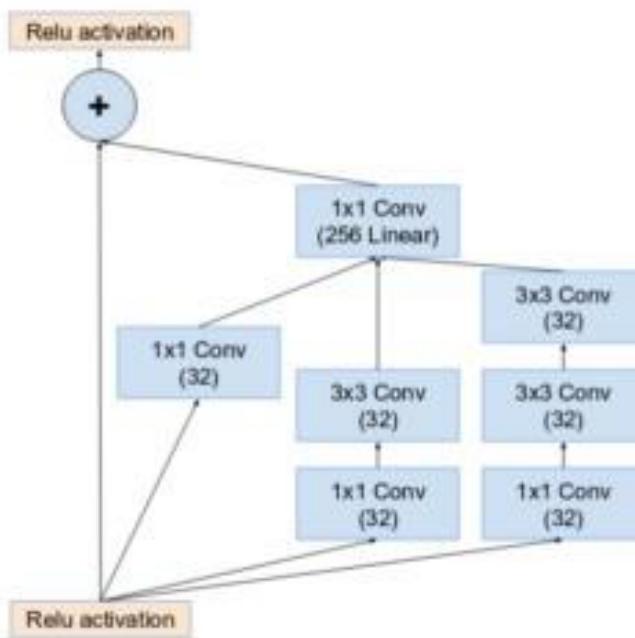
Deeper residual module (bottleneck)



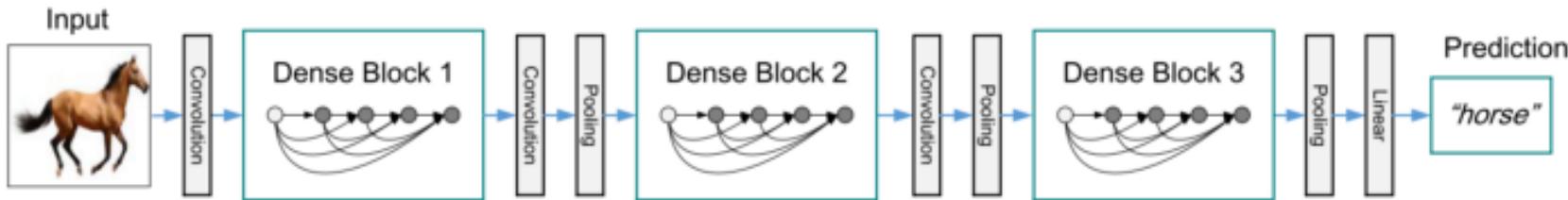
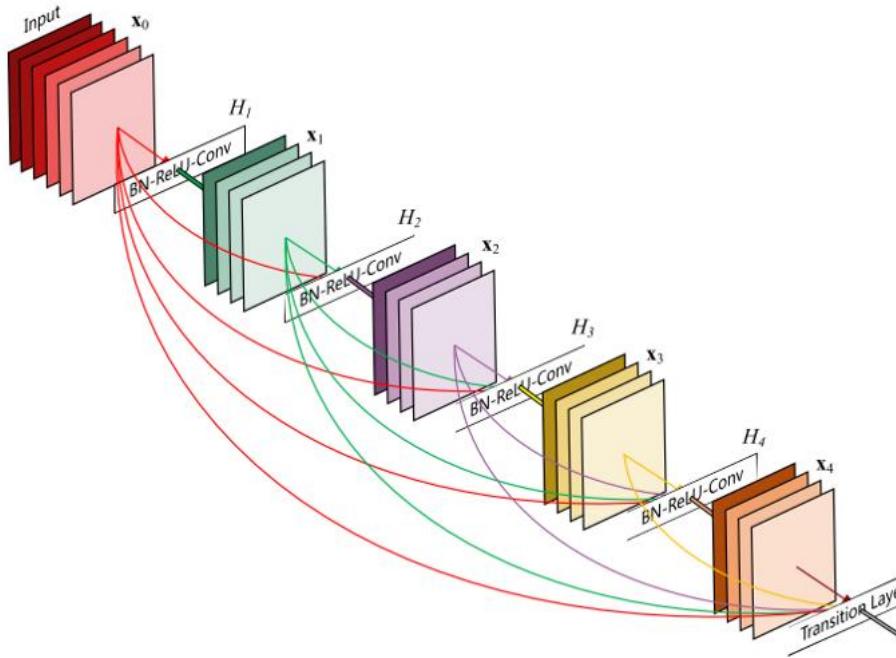
- Directly performing 3×3 convolutions with 256 feature maps at input and output:
 $256 \times 256 \times 3 \times 3 \sim 600K$ operations
- Using 1×1 convolutions to reduce 256 to 64 feature maps, followed by 3×3 convolutions, followed by 1×1 convolutions to expand back to 256 maps:
 $256 \times 64 \times 1 \times 1 \sim 16K$
 $64 \times 64 \times 3 \times 3 \sim 36K$
 $64 \times 256 \times 1 \times 1 \sim 16K$
Total: $\sim 70K$



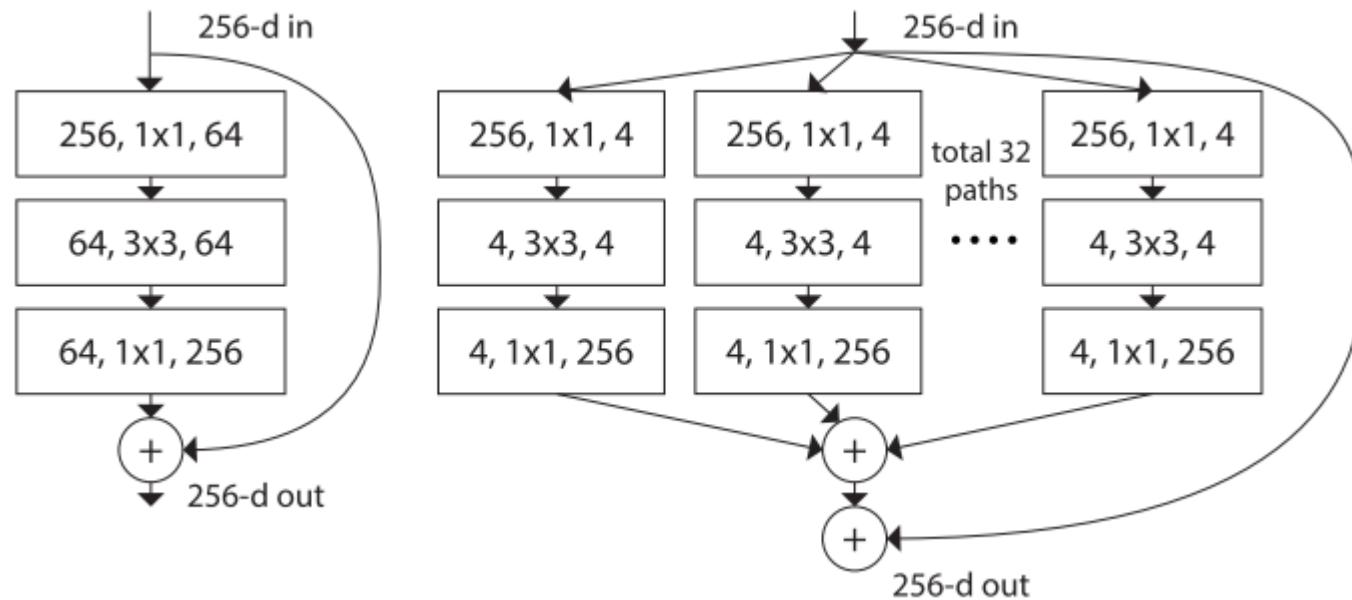
Inception resnet



Densely connected CNN



ResNext

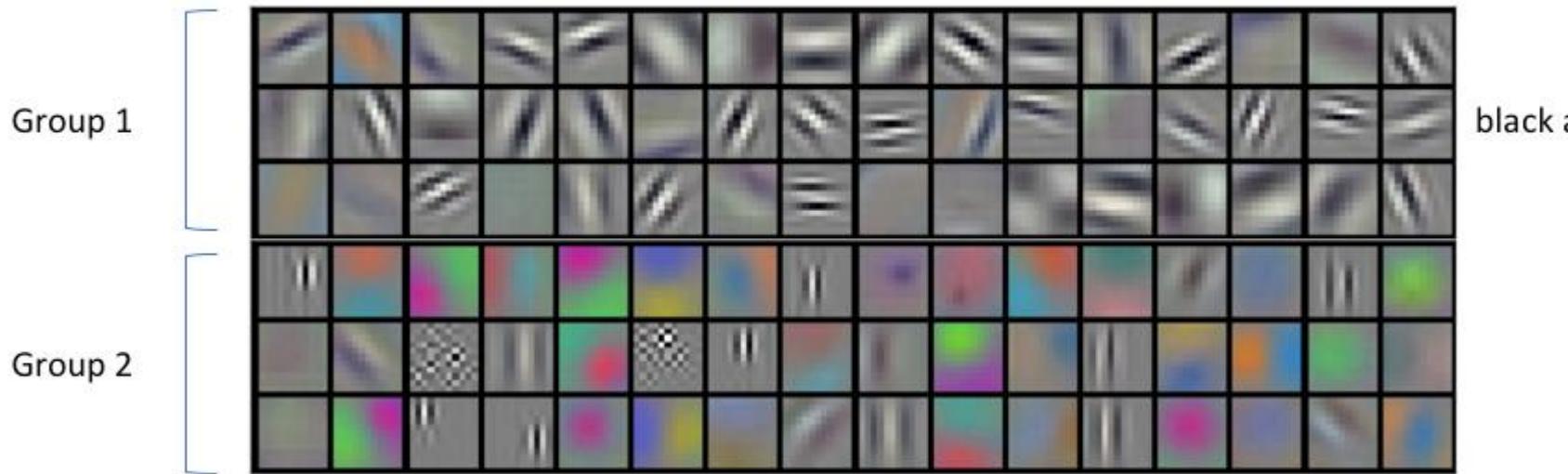
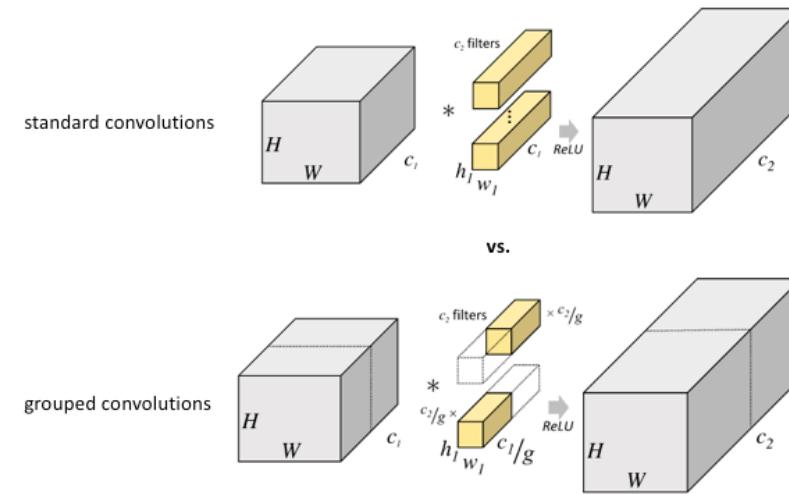


Resnext block with cardinality 32

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 [1×1, 64 3×3, 64 1×1, 256] ×3	3×3 max pool, stride 2 [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.		25.5×10⁶	25.0×10⁶
FLOPs		4.1×10⁹	4.2×10⁹

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.*

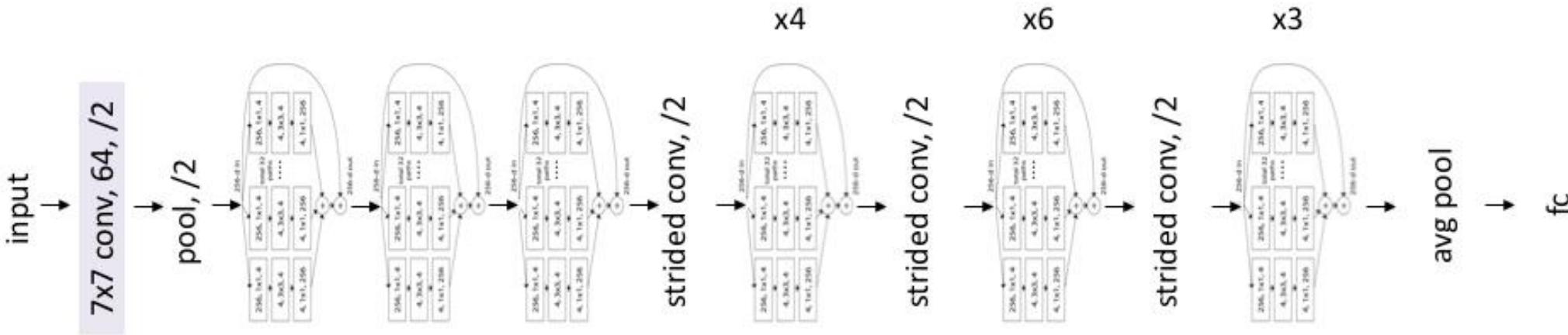
Group convolution



black and white shapes

Group 2

colors



Stochastic depth

- Can the depth of the network change during training?

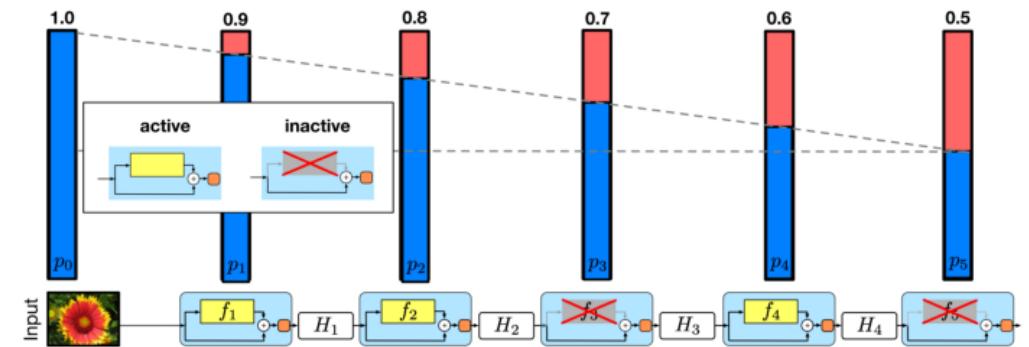
$$H_l = \text{ReLU}(b_l * f_l(H_{l-1}) + id(H_{l-1})).$$

If $b=0$

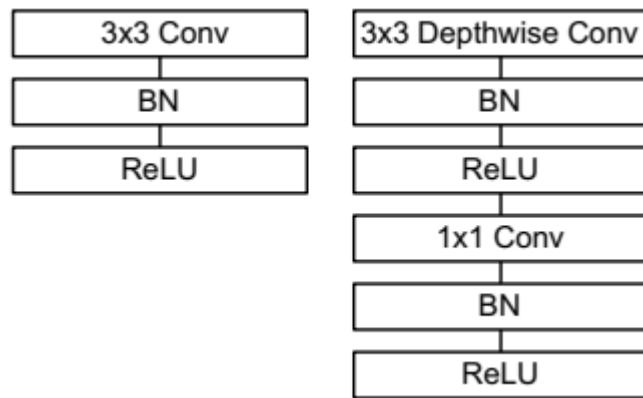
$$H_l = \text{ReLU}(id(H_{l-1})).$$

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

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



Mobilenet



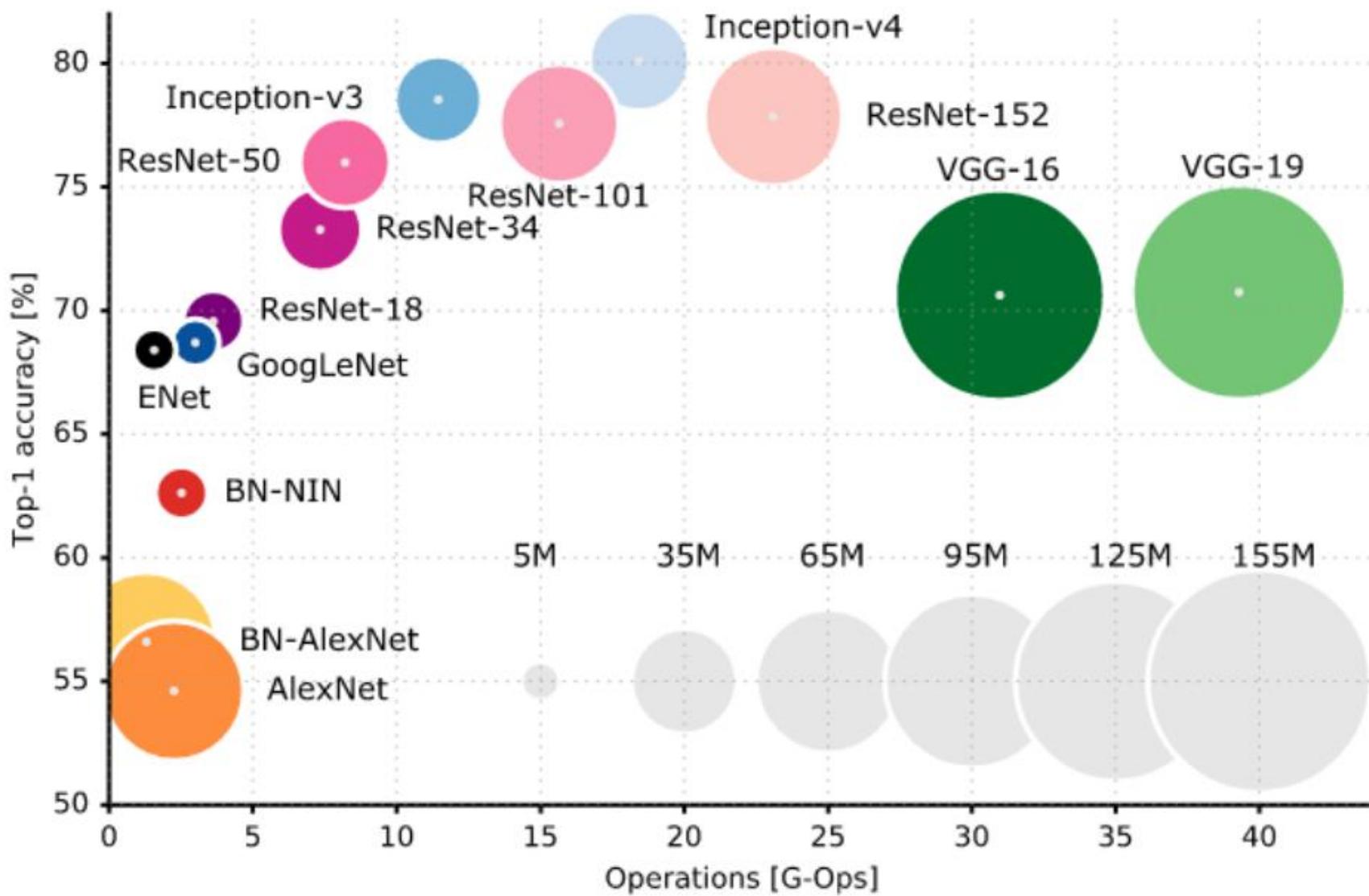
The idea of depthwise
Separable convolution

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× Conv dw / s1	$3 \times 3 \times 512$ dw	$14 \times 14 \times 512$
	$1 \times 1 \times 512 \times 512$	$14 \times 14 \times 512$
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	$1 \times 1 \times 1000$

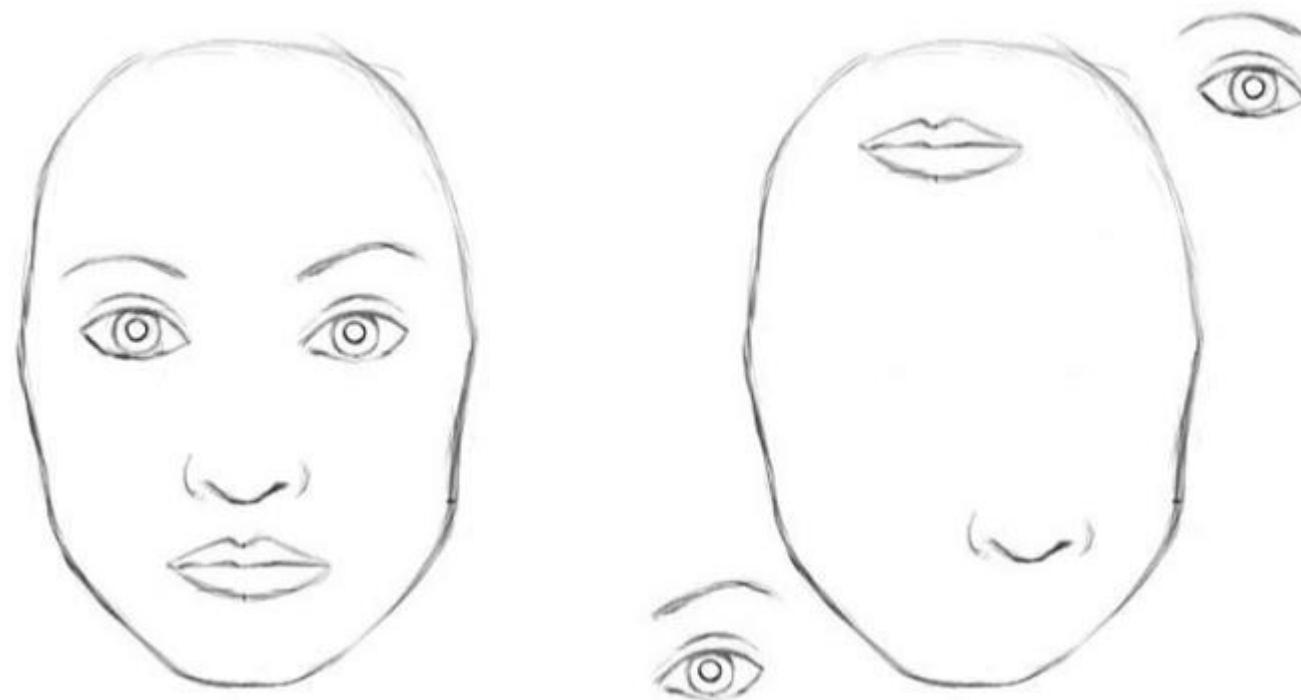
Team	Year	Place	Error (top-5)	External data
SuperVision – Toronto (AlexNet, 7 layers)	2012	-	16.4%	no
SuperVision	2012	1st	15.3%	ImageNet 22k
Clarifai – NYU (7 layers)	2013	-	11.7%	no
Clarifai	2013	1st	11.2%	ImageNet 22k
VGG – Oxford (16 layers)	2014	2nd	7.32%	no
GoogLeNet (19 layers)	2014	1st	6.67%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



What's wrong with deep CNN?

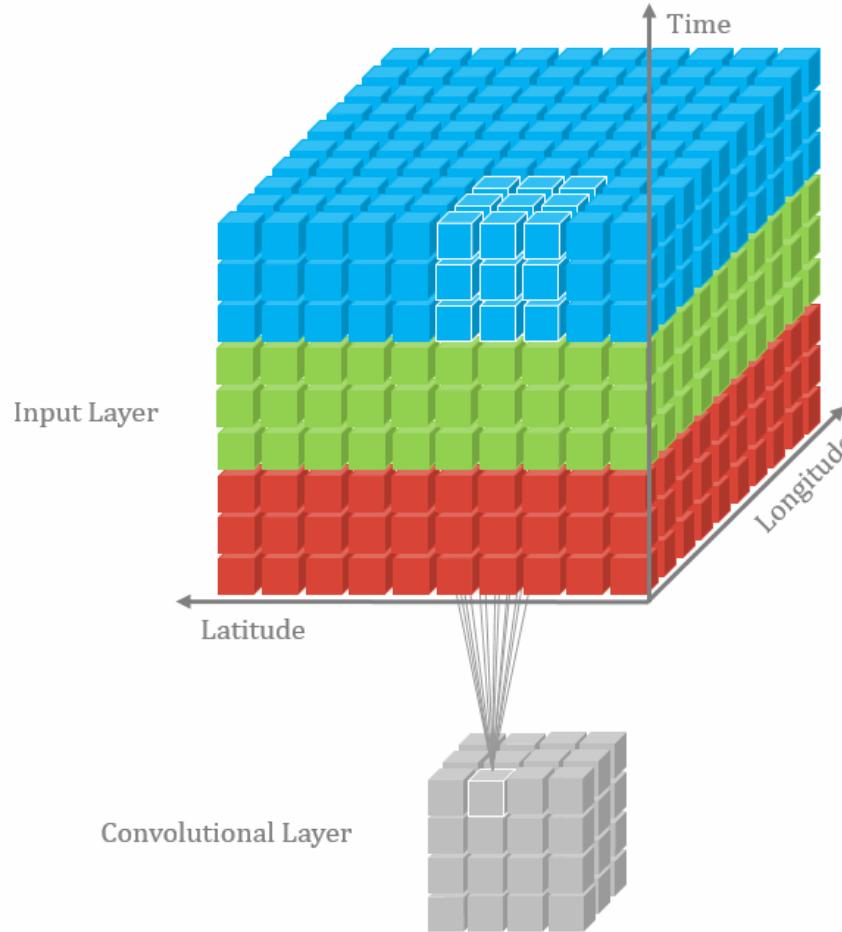
- Pooling causes much information loss



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



Must suggested reading

A Survey of the Recent Architectures of Deep Convolutional Neural Networks

[Asifullah Khan](#), [Anabia Sohail](#), [Umme Zahoor](#), [Aqsa Saeed Qureshi](#)

Deep Convolutional Neural Network (CNN) is a special type of Neural Networks, which has shown exemplary performance on several competitions related to Computer Vision and Image Processing. Some of the exciting application areas of CNN include Image Classification and Segmentation, Object Detection, Video Processing, Natural Language Processing, and Speech Recognition. The powerful learning ability of deep CNN is primarily due to the use of multiple feature extraction stages that can automatically learn representations from the data. The availability of a large amount of data and improvement in the hardware technology has accelerated the research in CNNs, and recently interesting deep CNN architectures have been reported. Several inspiring ideas to bring advancements in CNNs have been explored, such as the use of different activation and loss functions, parameter optimization, regularization, and architectural innovations. However, the significant improvement in the representational capacity of the deep CNN is achieved through architectural innovations. Notably, the ideas of exploiting spatial and channel information, depth and width of architecture, and multi-path information processing have gained substantial attention. Similarly, the idea of using a block of layers as a structural unit is also gaining popularity. This survey thus focuses on the intrinsic taxonomy present in the recently reported deep CNN architectures and, consequently, classifies the recent innovations in CNN architectures into seven different categories. These seven categories are based on spatial exploitation, depth, multi-path, width, feature-map exploitation, channel boosting, and attention. Additionally, the elementary understanding of CNN components, current challenges, and applications of CNN are also provided.

Checking codebase

- <https://github.com/fchollet/deep-learning-models>
- Written in Keras

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