

Convolutional Neural Networks

Alexey Zaytsev,

Most slides are from Evgeny Burnaev

Skoltech, Moscow, Russia

- 1 Introduction
- 2 First generation of Neural Networks
- 3 Modern Neural Networks
- 4 Convolutional Neural Networks
- 5 Transfer learning
- 6 Regularizing Neural Networks

1 Introduction

2 First generation of Neural Networks

3 Modern Neural Networks

4 Convolutional Neural Networks

5 Transfer learning

6 Regularizing Neural Networks

The reasons for the breakthrough in the combination of factors!

- The increase in power and availability of hardware (GPU)
- Big volumes of open (labeled) data
- Available frameworks: Torch (Facebook, Google), Tensorflow (Google), Theano (open-source)
- The availability of research and of source code in open access
- Significant investments from big IT companies (Google, Facebook, Amazon, IBM, etc.)

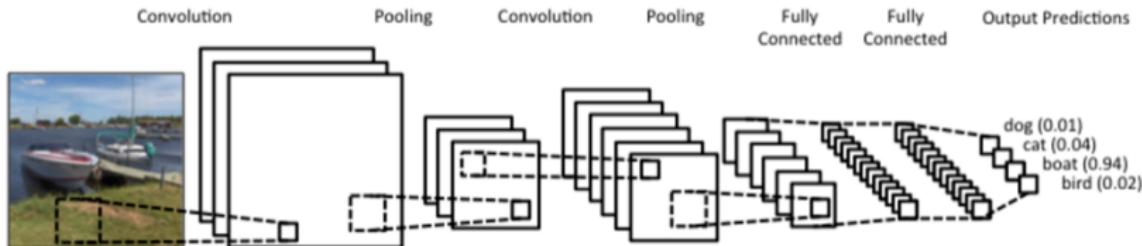
As a result, the point “focusing” of research on a “limited” topic (> 50 research groups, > 1000 researchers) with many prominent papers during recent 10 years

- Senior researchers from Princeton, Stanford, Google
- More than 14 millions of images, manually labeled using Amazon Mechanical Turk to ~ 20000 classes.

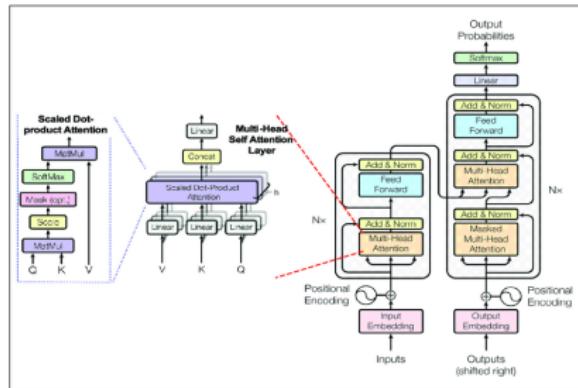


- Computer vision: ImageNet - large labeled dataset of images, JFT-300M - internal Google dataset with semi-automatic labeling
- NLP: Corpops of unlabeled NLP data scraped for the internet
- In oil&gas industry: no large high quality datasets. So, hard to apply ML.

Convolutional neural network



Transformer neural network



- PyTorch is an open framework for deep neural networks
- Also see JAX, Tensorflow. There were more.
- More than 15 Google engineers develop Tensorflow full-time
- ~ 700 thousands of code, ~ 600 participants of external open-source development
- Example: playground.tensorflow.org

1 Introduction

2 First generation of Neural Networks

3 Modern Neural Networks

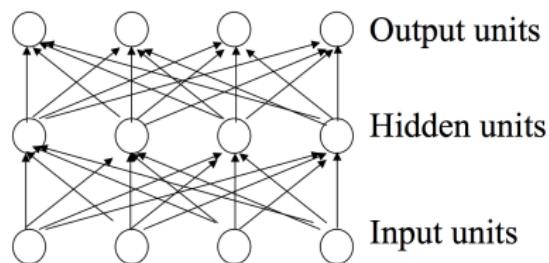
4 Convolutional Neural Networks

5 Transfer learning

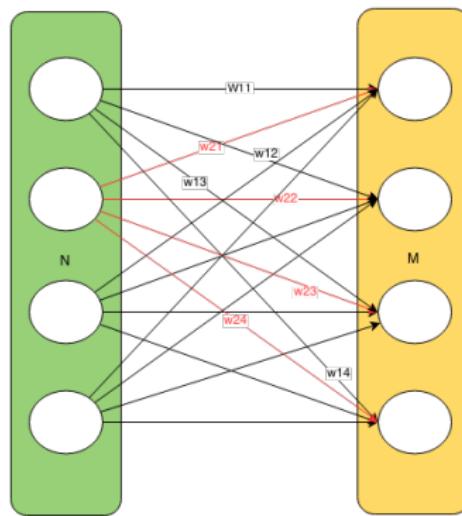
6 Regularizing Neural Networks

- Initially the most popular networks were fully-connected layered feed-forward networks

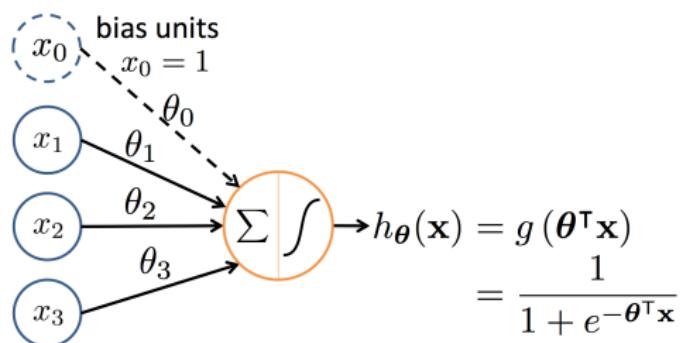
Fully-connected layered feed-forward network



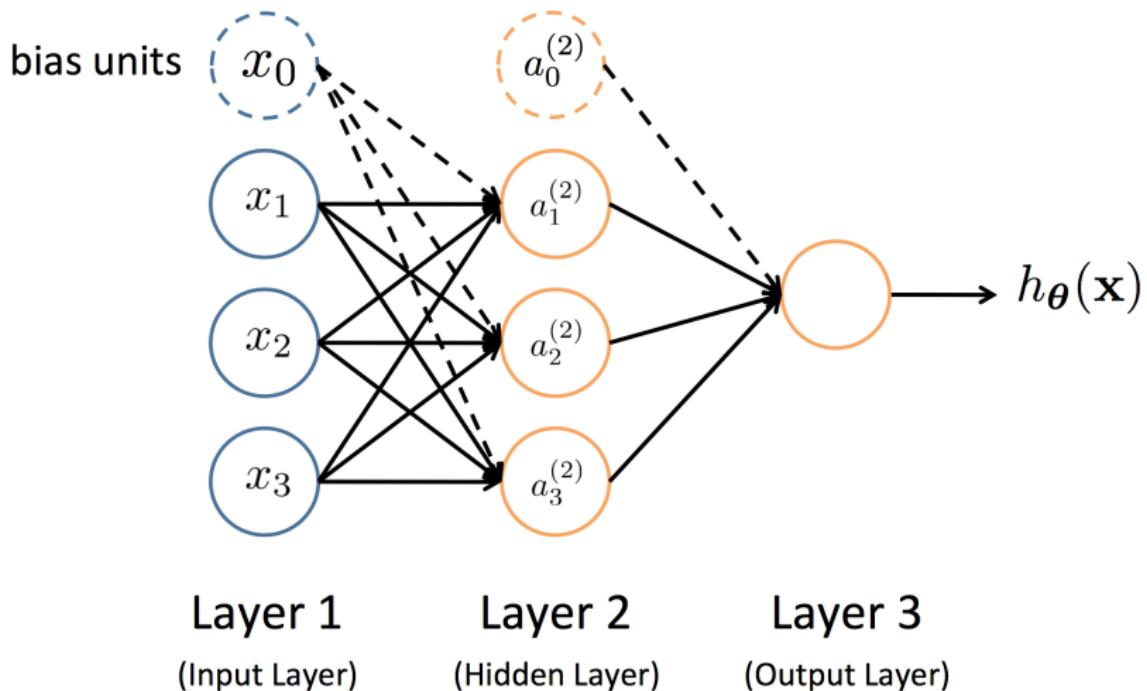
Fully connected layer



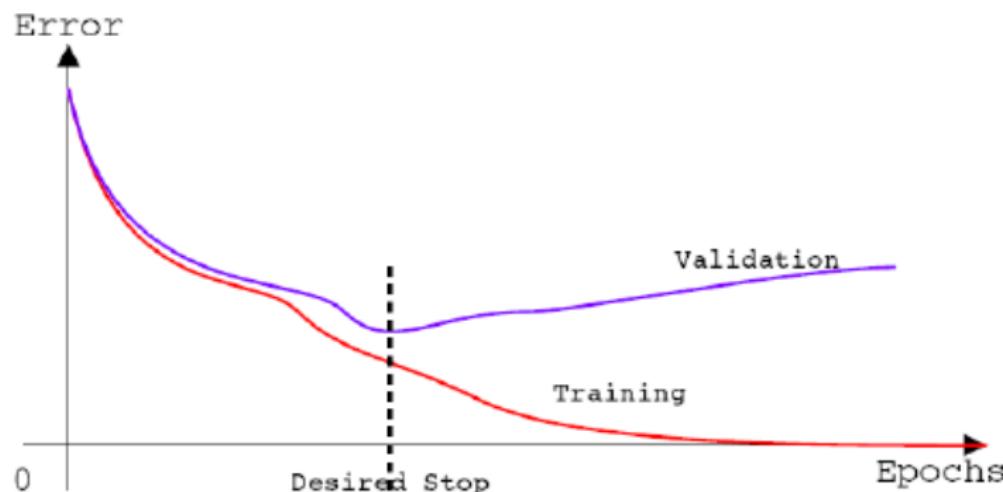
- Let us define $\mathbf{x} = \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{bmatrix}$ and $\boldsymbol{\theta} = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \\ \theta_3 \end{bmatrix}$



- Sigmoid (logistic) activation function $g(z) = \frac{1}{1+e^{-z}}$



- Backpropagation for training: gradient descent + efficient computations (based on graph-like representation)
- Local minima
- Random initialization
- Random division into train and validation sets
- Second order optimization methods
- Early stopping



1 Introduction

2 First generation of Neural Networks

3 Modern Neural Networks

4 Convolutional Neural Networks

5 Transfer learning

6 Regularizing Neural Networks

Let $\phi(\cdot)$ be a non-constant, bounded, and monotonically-increasing continuous function. Let I_m denote the m -dimensional unit hypercube $[0, 1]^m$. The space of continuous function on I_m is denoted by $C(I_m)$. Then, given any $\varepsilon > 0$ and any function $f \in C(I_m)$, there exist an integer p , real constants $v_i, b_i \in \mathbb{R}$ and real vectors $\mathbf{w}_i \in \mathbb{R}^m$, where $i = 1, \dots, p$, such that we may define:

$$\hat{f}(\mathbf{x}) = \sum_{i=1}^p v_i \phi(\mathbf{w}_i^T \mathbf{x} + b_i)$$

as an approximate realization of the function f , where f is independent of ϕ ; that is,

$$|\hat{f}(\mathbf{x}) - f(\mathbf{x})| < \varepsilon$$

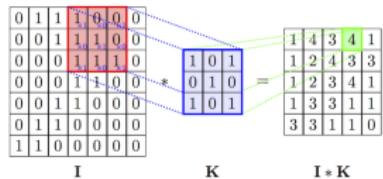
for all $\mathbf{x} \in I_m$. In other words, functions of the form $F(\mathbf{x})$ are dense in $C(I_m)$.

- Deep nets can be trained and regularized effectively
- Deep nets can achieve better performance with similar number of parameters
- Some geometrical advantages of deep architectures
- Deep architectures are more intuitive and can be interpreted as a sequence of simple data transformations

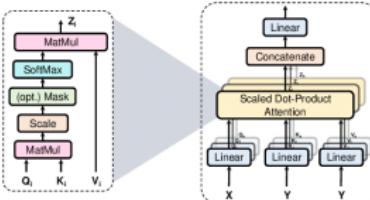
- $f_j(\mathbf{x}, \mathbf{w}_j)$ is a j -th layer of the network
- $v_j(\mathbf{x}) = f_j(v_{j-1}(\mathbf{x}), \mathbf{w}_j)$
- $a(\mathbf{x}, \mathbf{w}) = v_n(\mathbf{x})$ is an output
- Optimization problem

$$Q(\mathbf{w}) = \sum_{i=1}^l L(y_i, a(\mathbf{x}_i, \mathbf{w})) \rightarrow \min_{\mathbf{w}}$$

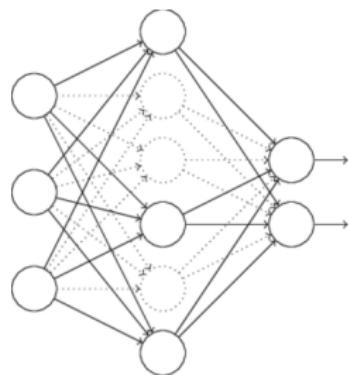
Convolutional layer



Attention layer



Dropout layer



Why specific architectures are better?

- Smaller number of parameters to estimate
- Take into account the data structure and links in it
- Many tricks on top of it

1 Introduction

2 First generation of Neural Networks

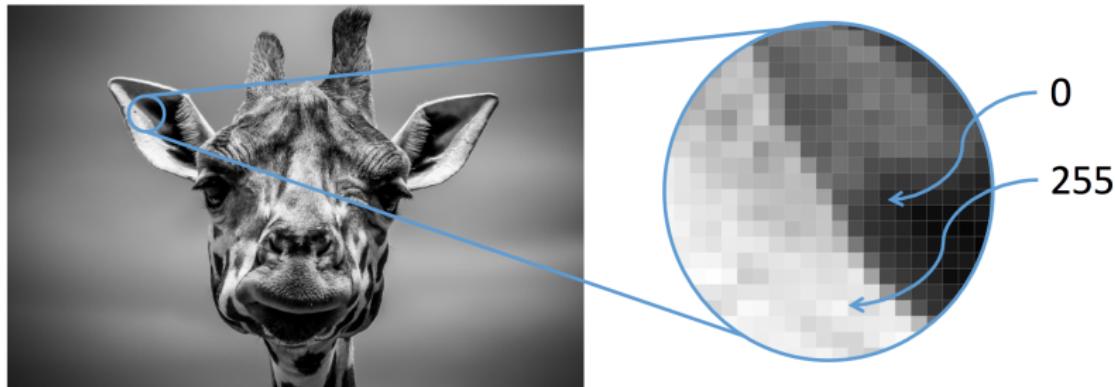
3 Modern Neural Networks

4 Convolutional Neural Networks

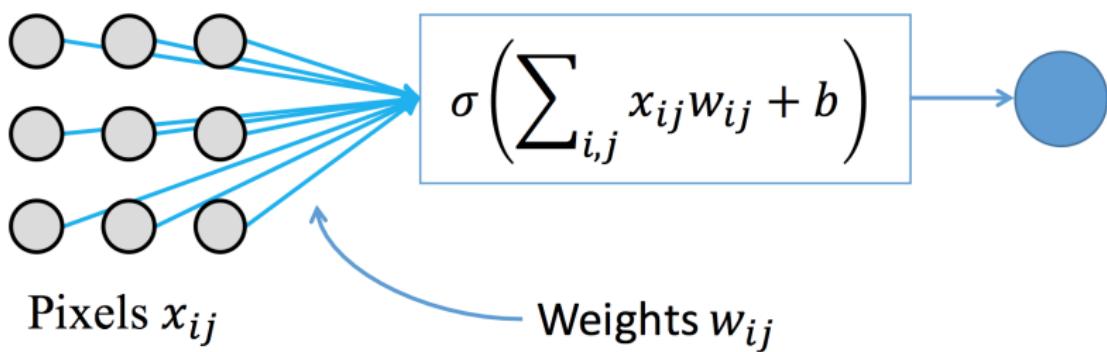
5 Transfer learning

6 Regularizing Neural Networks

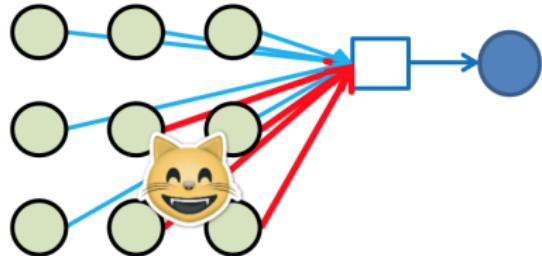
- Each pixel stores its brightness (or **intensity**) ranging from 0 to 255, 0 intensity corresponds to black color.
- Color images store pixel intensities for 3 channels: **red**, **green** and **blue**.
- ImageNet image's size is $224 \times 224 \times 3$ ($m \approx 150000$).



- Normalize input pixels: $\mathbf{x}_{\text{norm}} = \frac{\mathbf{x}}{255} - 0.5$
- Maybe MLP will work?

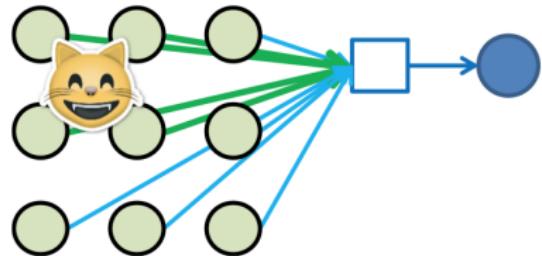


Fully-connected layered feed-forward network



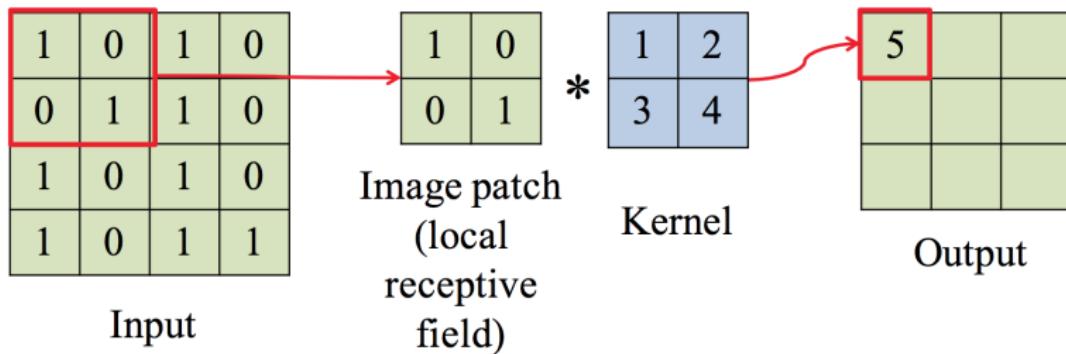
On this training image red weights w_{ij} will change a little bit to better detect a cat

Fully-connected layered feed-forward network

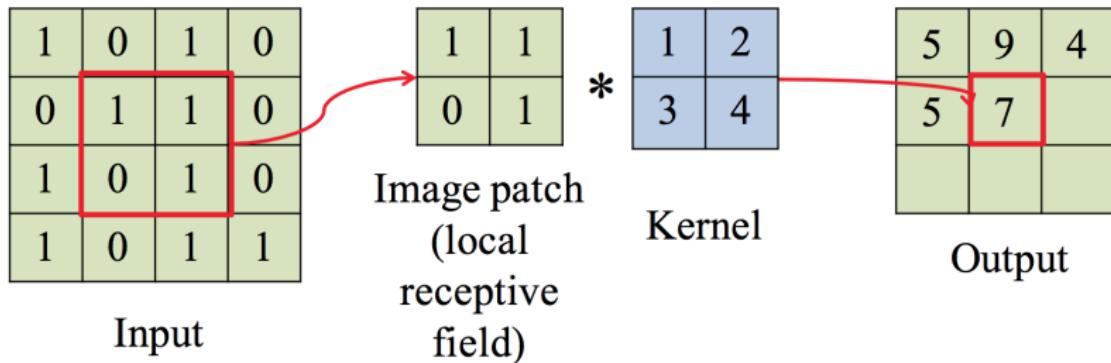


On this training image green weights w_{ij} will change a little bit to better detect a cat

- Convolution is a dot product of a **kernel** (or filter) and a patch of an image (**local receptive field**) of the same size



- Convolution is a dot product of a **kernel** (or filter) and a patch of an image (**local receptive field**) of the same size



Kernel

$$\begin{matrix} * & \begin{array}{|c|c|c|} \hline -1 & -1 & -1 \\ \hline -1 & 8 & -1 \\ \hline -1 & -1 & -1 \\ \hline \end{array} & = & \begin{array}{c} \text{Image showing edge detection results} \end{array} \end{matrix}$$

$$\begin{matrix} \begin{array}{c} \text{Original} \\ \text{image} \end{array} & * & \begin{array}{|c|c|c|} \hline 0 & -1 & 0 \\ \hline -1 & 5 & -1 \\ \hline 0 & -1 & 0 \\ \hline \end{array} & = & \begin{array}{c} \text{Image showing sharpening results} \end{array} \end{matrix}$$

$$\begin{matrix} * \frac{1}{9} & \begin{array}{|c|c|c|} \hline 1 & 1 & 1 \\ \hline 1 & 1 & 1 \\ \hline 1 & 1 & 1 \\ \hline \end{array} & = & \begin{array}{c} \text{Image showing blurring results} \end{array} \end{matrix}$$

Edge
detection

Sharpening

Blurring

Convolution is similar to correlation

0	0	0	0
0	0	0	0
0	0	1	0
0	0	0	1

Input

$$\begin{matrix} & \begin{matrix} 1 & 0 \\ 0 & 1 \end{matrix} \end{matrix} = \begin{matrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 2 \end{matrix}$$

Kernel

Output

Max = 2

0	0	0	0
0	0	0	0
0	0	0	1
0	0	1	0

Input

$$\begin{matrix} & \begin{matrix} 1 & 0 \\ 0 & 1 \end{matrix} \end{matrix} = \begin{matrix} 0 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{matrix}$$

Kernel

Output

Max = 1

Simple classifier

Convolution is translation invariant

0	0	0	0
0	0	0	0
0	0	1	0
0	0	0	1

Input

*

1	0
0	1

=

0	0	0
0	1	0
0	0	2

Output

Max = 2

Didn't
change

1	0	0	0
0	1	0	0
0	0	0	0
0	0	0	0

Input

*

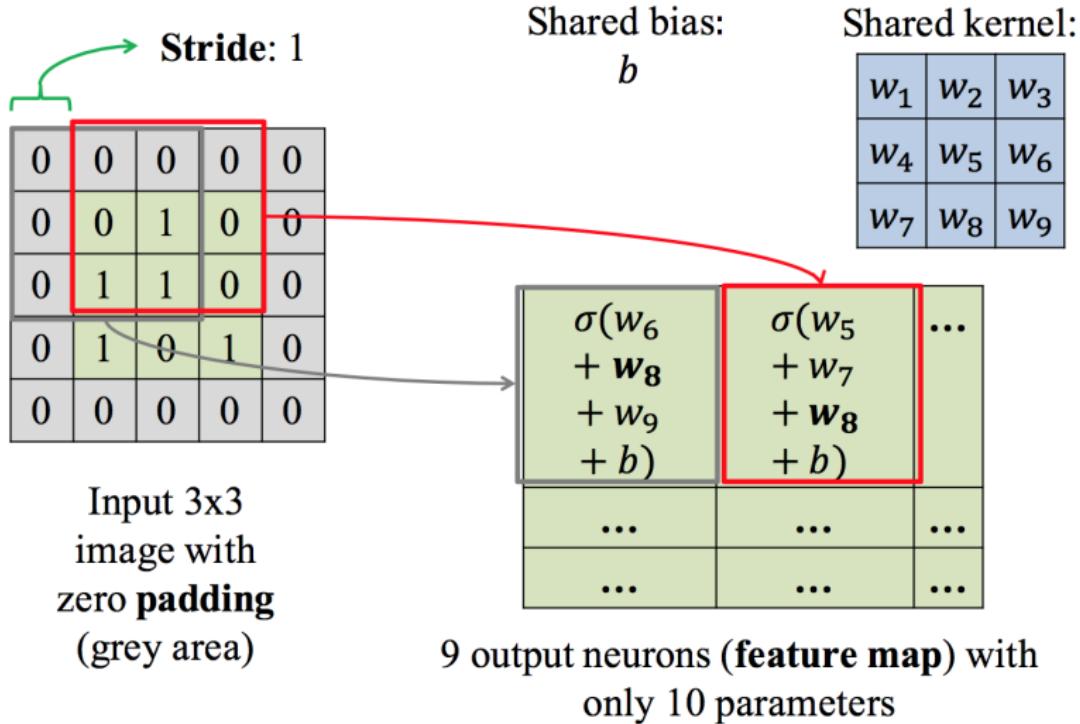
1	0
0	1

=

2	0	0
0	1	0
0	0	0

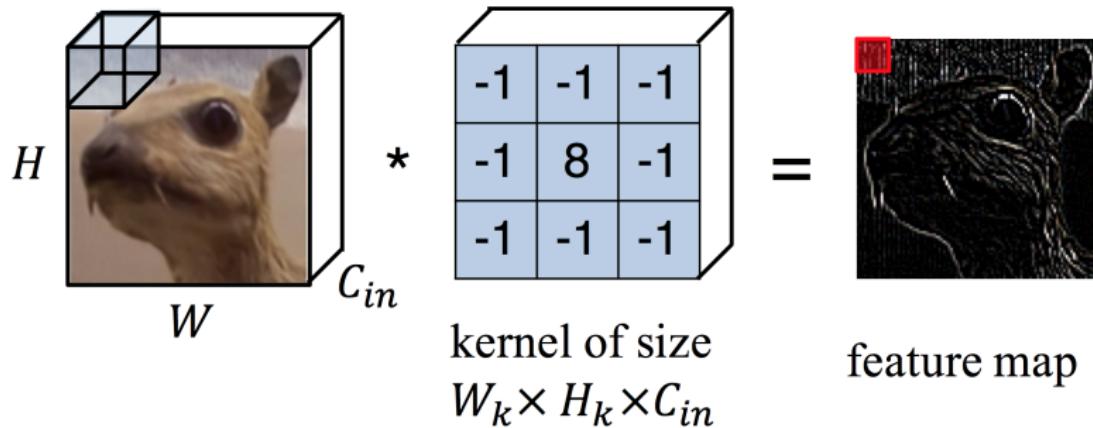
Output

Max = 2

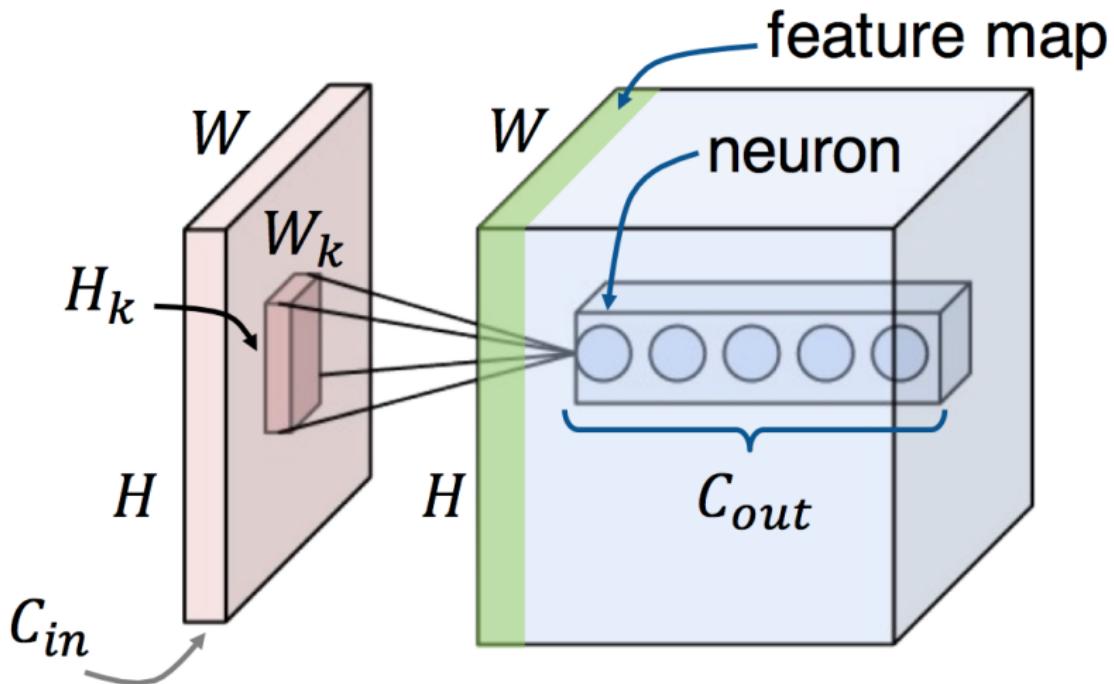


Let's say we have a color image as an input, which is $W \times H \times C_{in}$ **tensor** (multidimensional array), where

- W is an image width
- H is an image height
- C_{in} is a number of input channels (e.g. 3 **RGB** channels)

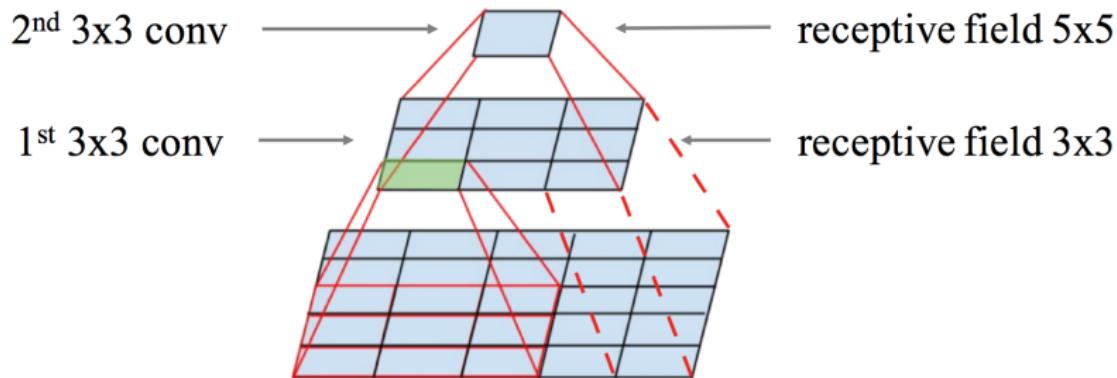


- We want to train C_{out} kernels of size $W_k \times H_k \times C_{in}$
- Having enough zero padding we can have $W \times H \times C_{out}$ output neurons



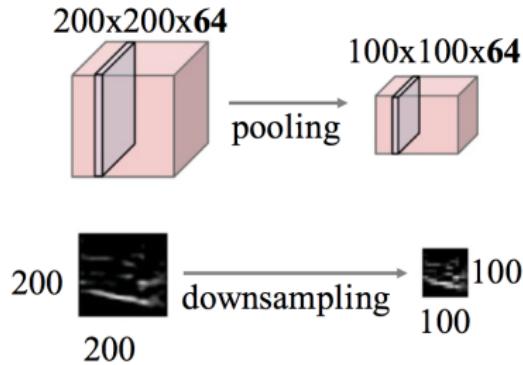
One convolutional layer is not enough!

- Let's say neurons of the 1-st convolutional layer look at the patches of the image of size 3×3
- What if an object of interest is bigger than that?
- We need a 2-nd convolutional layer on top of the 1-st!



Pooling layer

- This layer works like a convolutional layer but doesn't have kernel, instead it calculates **maximum** or **average** of input patch values



Single depth slice

1	1	1	4
2	6	5	8
3	2	1	0
1	1	3	5

2x2 **max pooling** with stride 2

- Strictly speaking: maximum is not a differentiable function!

6	8
3	5

Maximum = 8

7	8
3	5

Maximum = 8

6	8
3	5

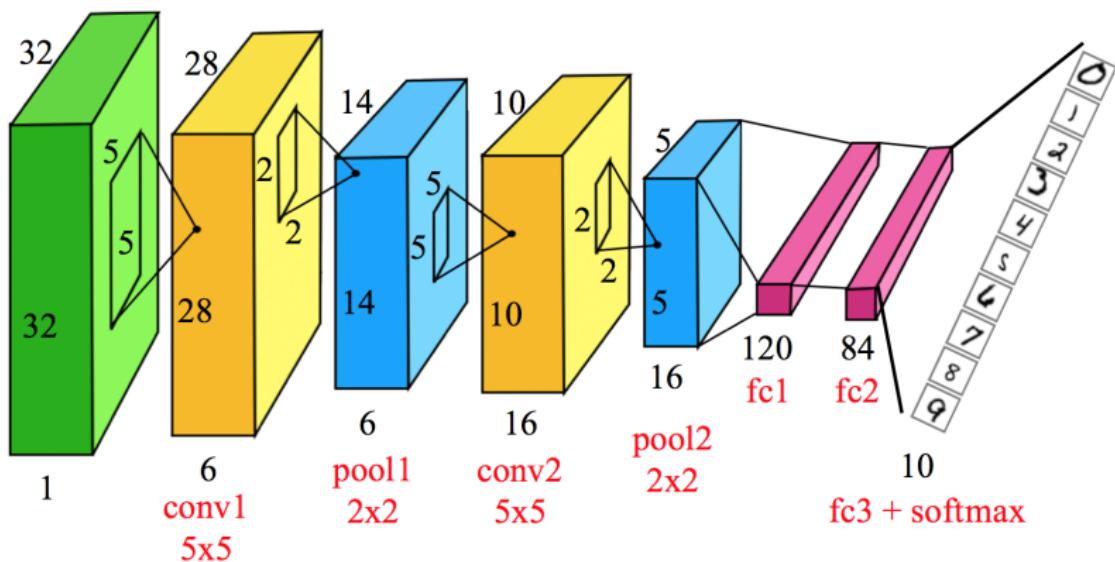
Maximum = 8

7	9
3	5

Maximum = 9

Convolutional network

We are ready to construct LeNet (1998, by Le Cun) architecture from these building blocks.



<http://yann.lecun.com/exdb/publis/pdf/lecun-98.pdf>

- The first deep convolutional neural net for ImageNet
- Significantly reduced top 5 error from 26% to 15%

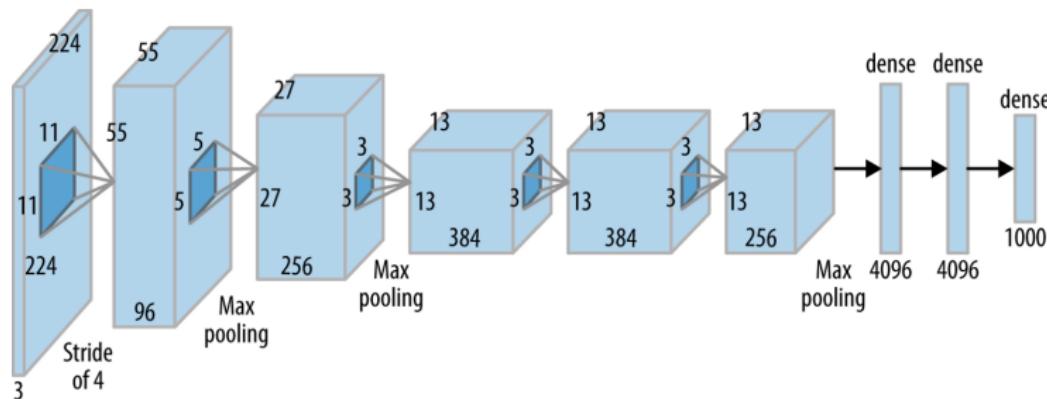
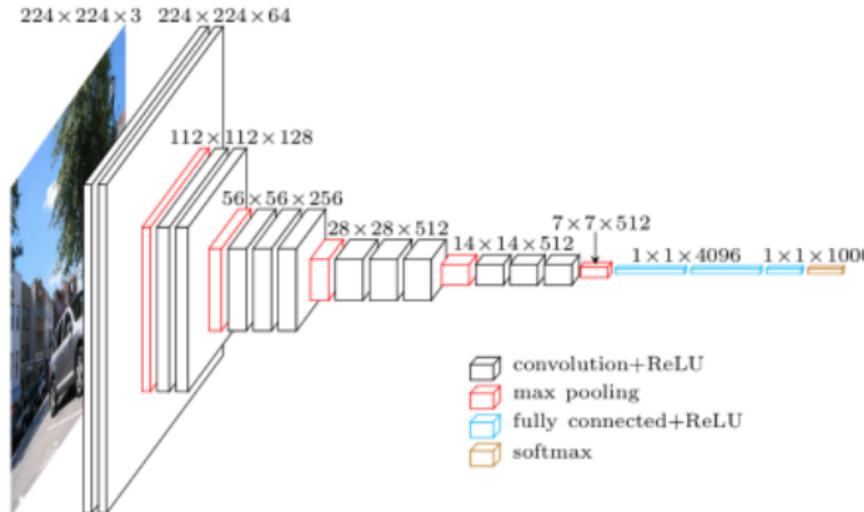


Figure – source: oreilly.com

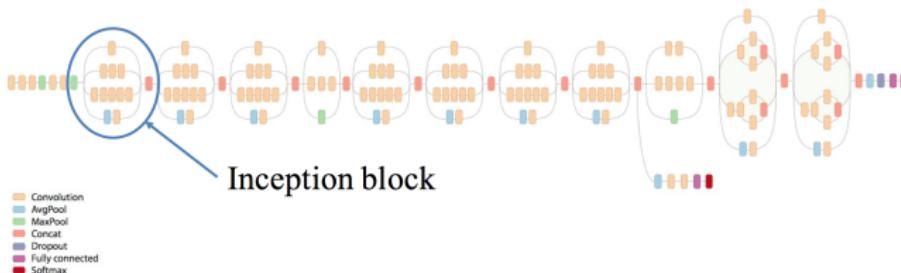
- 11×11 , 5×5 , 3×3 convolutions, max pooling, dropout, data augmentation, ReLU activations, SGD with momentum
- 60 million parameters
- Trains on 2 GPUs for 6 days

- Similar to AlexNet, only 3×3 convolutions, but lots of filters!
- ImageNet top 5 error: 8.0% for a single model



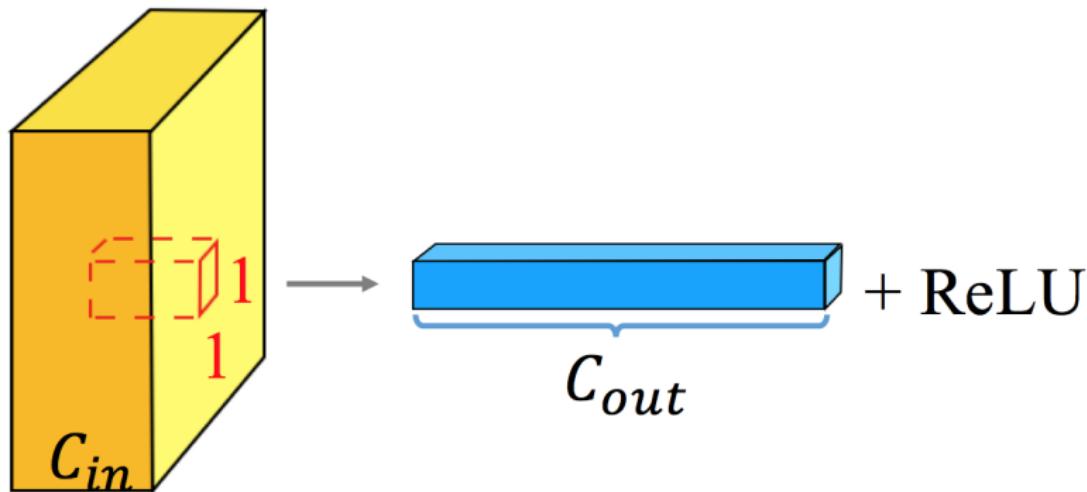
- Training similar to AlexNet with additional multi-scale cropping
- 138 million parameters
- Trains on 4 GPUs for 2 – 3 weeks

- Similar to AlexNet? Not quite, uses Inception block introduced in GoogLeNet (a.k.a. Inception V1)
- ImageNet top 5 error: 5.6% for a single model, 3.6% for an ensemble



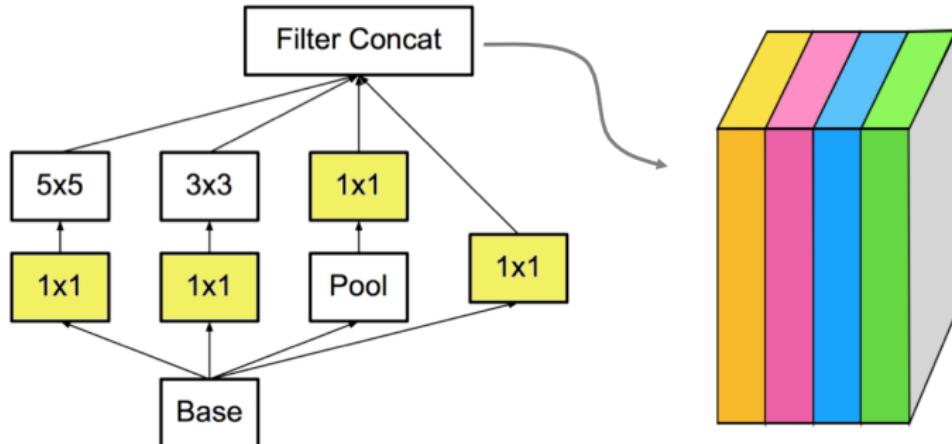
- Batch normalization, image distortions, RMSProp
- 25 million parameters!
- Trains on 8 GPUs for 2 weeks

- Such convolutions capture interactions of input channels in one “pixel” of feature map
- They can reduce the number of channels not hurting the quality of the model, because different channels can correlate
- Dimensionality reduction with added ReLU activation

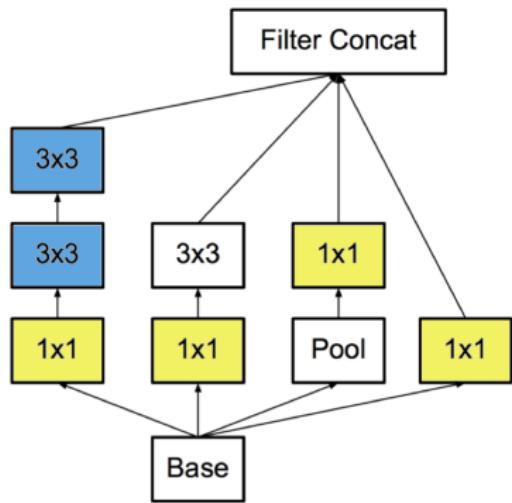


Inception block

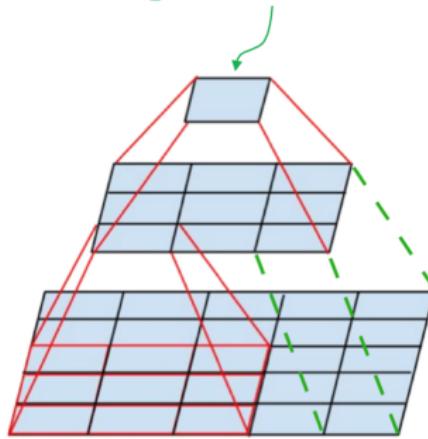
- All operations inside a block use stride 1 and enough padding to output the same spatial dimensions ($W \times H$) of feature map.
- 4 different feature maps are concatenated on depth at the end



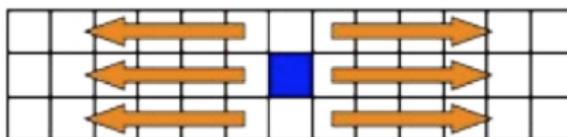
Christian Szegedy, <https://arxiv.org/pdf/1512.00567.pdf>



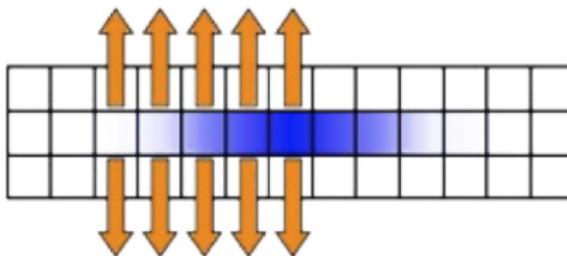
Receptive field 5×5



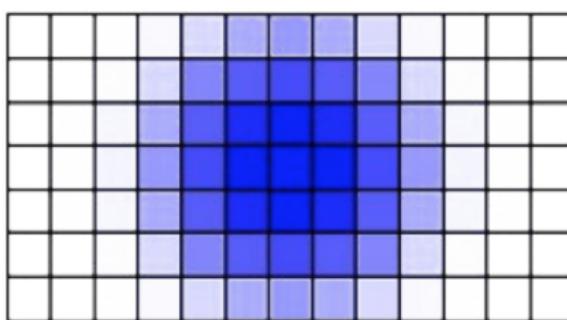
Christian Szegedy, <https://arxiv.org/pdf/1512.00567.pdf>



Blur the source horizontally

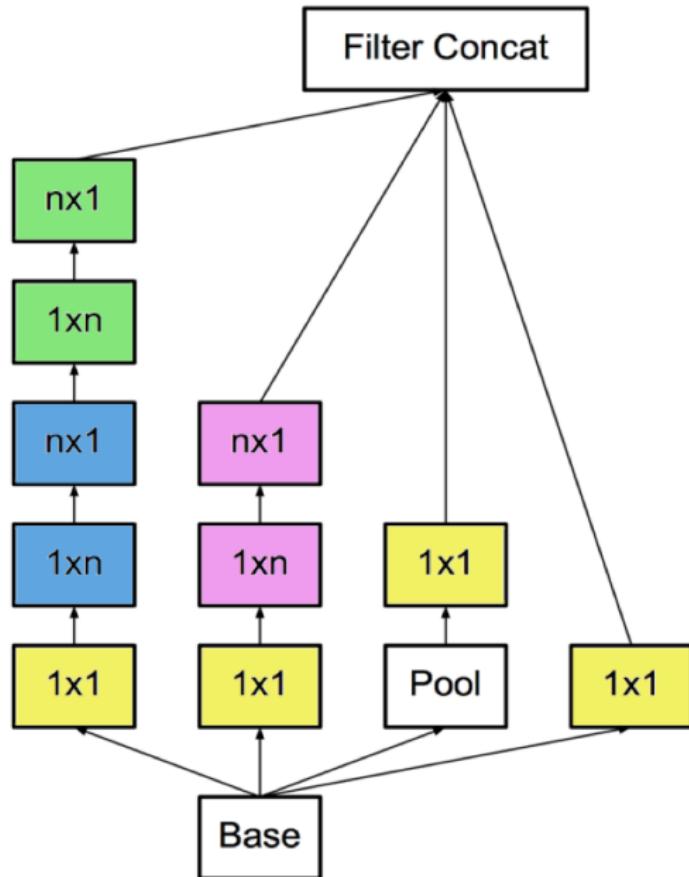


Blur the blur vertically

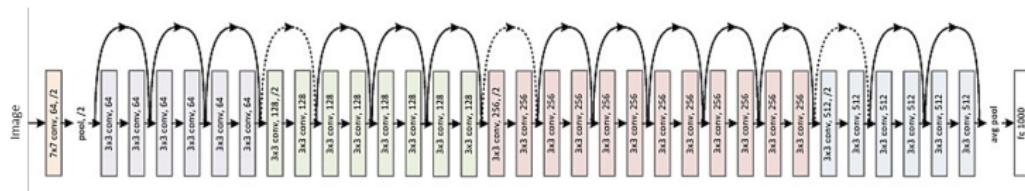


Result

Inception block

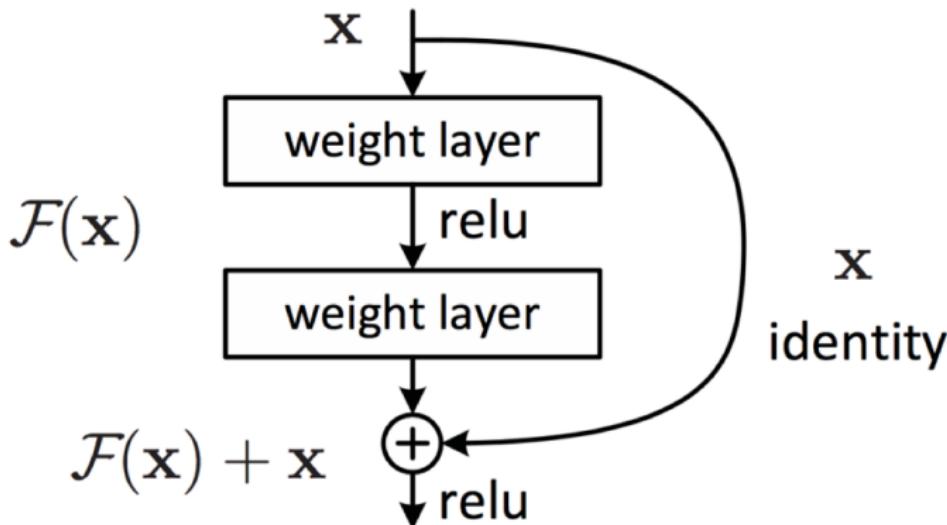


- Introduces residual connections
 - ImageNet top 5 error: 4.5% for a single model, 3.5% for an ensemble



- 152 layers, few 7×7 convolutional layers, the rest are 3×3 , batch normalization, max and average pooling
 - 60 million parameters
 - Trains on 8 GPUs for 2 – 3 weeks

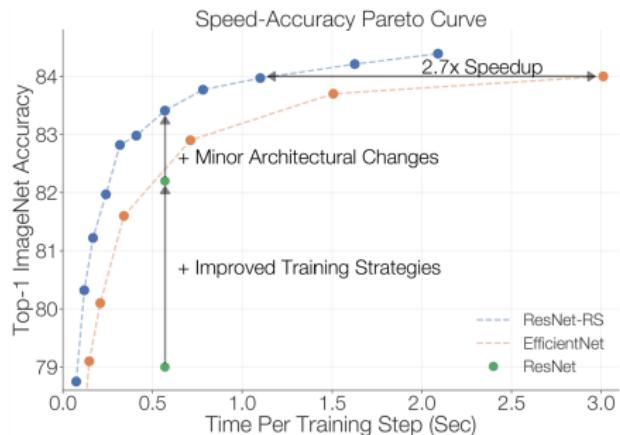
- We create output channels adding a small delta $F(x)$ to original input channels x



- This way we can stack thousands of layers and gradients do not vanish thanks to residual connections

Attack points:

- Training methods
- Regularization methods
- Architecture improvement



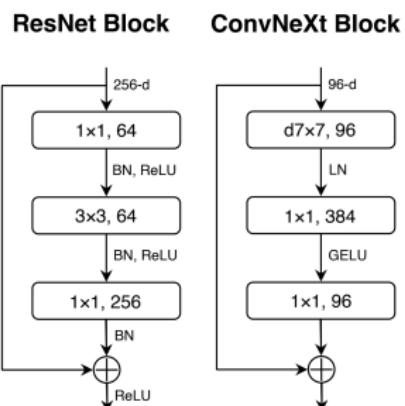
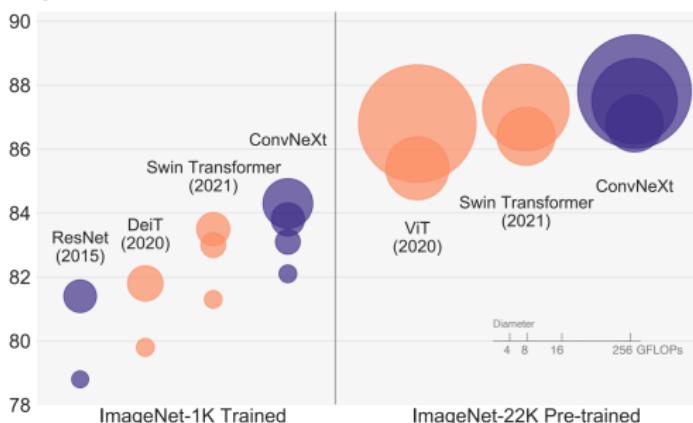
Improvements	Top-1	Δ
ResNet-200	79.0	—
+ Cosine LR Decay	79.3	+0.3
+ Increase training epochs	78.8 †	-0.5
+ EMA of weights	79.1	+0.3
+ Label Smoothing	80.4	+1.3
+ Stochastic Depth	80.6	+0.2
+ RandAugment	81.0	+0.4
+ Dropout on FC	80.7 ‡	-0.3
+ Decrease weight decay	82.2	+1.5
+ Squeeze-and-Excitation	82.9	+0.7
+ ResNet-D	83.4	+0.5

Idea: adopt innovations from Vision transformers

"these designs are not novel even in the ConvNet literature — they have all been researched separately, but not collectively, over the last decade"

- Patchification
- LayerNorm instead of BatchNorm
- More parameters (larger kernels)

ImageNet-1K Acc.



1 Introduction

2 First generation of Neural Networks

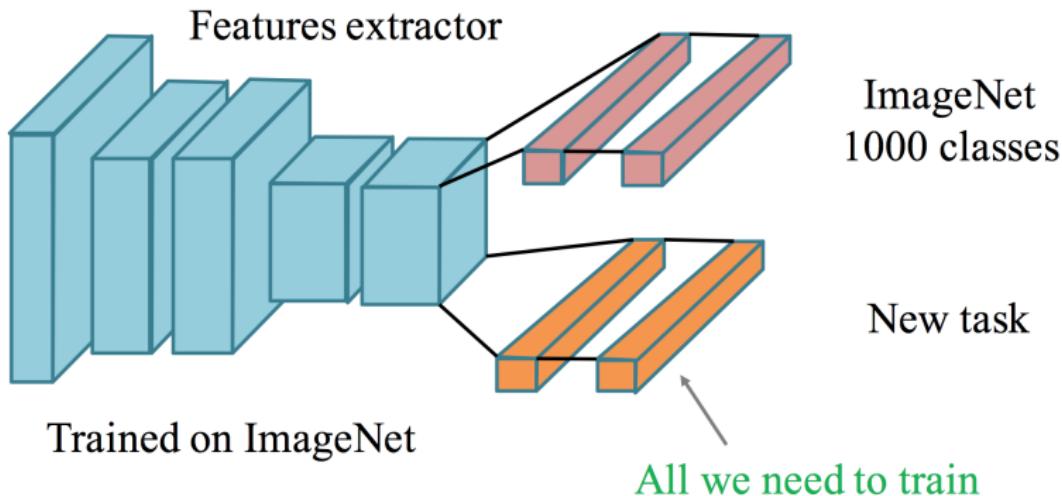
3 Modern Neural Networks

4 Convolutional Neural Networks

5 Transfer learning

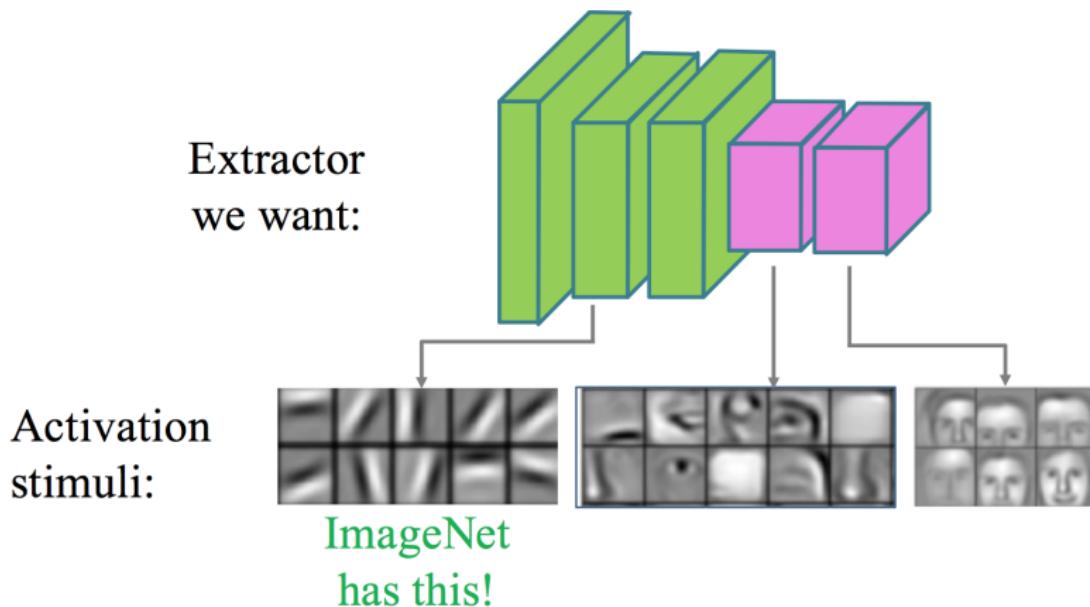
6 Regularizing Neural Networks

- Deep networks learn complex features extractor, but we need lots of data to train it from scratch!
- What if we can reuse an existing features extractor for a new task?



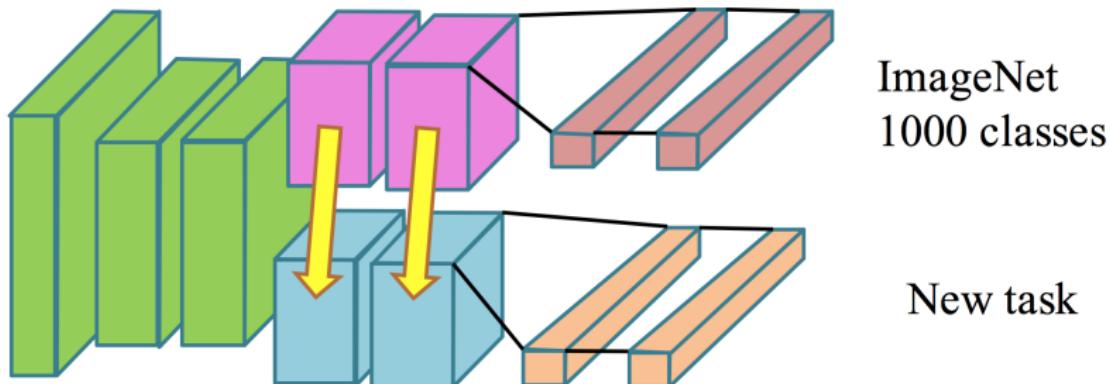
- You need less data to train (for training only final MLP)
- It works if a domain of a new task is similar to ImageNet's
- E.g. won't work for human emotions classification, ImageNet doesn't have people faces in the dataset!

- But what if we need to classify human emotions?
- Maybe we can partially reuse ImageNet features extractor?



Honglak Lee, <http://web.eecs.umich.edu/~honglak/icml09-ConvolutionalDeepBeliefNetworks.pdf>

- You can initialize deeper layers with values from ImageNet
- This is called **fine-tuning**, because you do not start with a random initialization
- Propagate all gradients with smaller learning rate



- Simple differentiable computational blocks can be combined into graphs and still efficiently trained
- Convolutional architectures achieve superhuman performance in many vision problems
- Graphs trained on large datasets can be reused for other tasks

1 Introduction

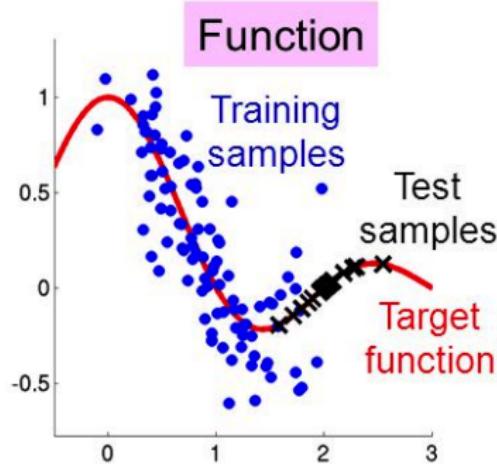
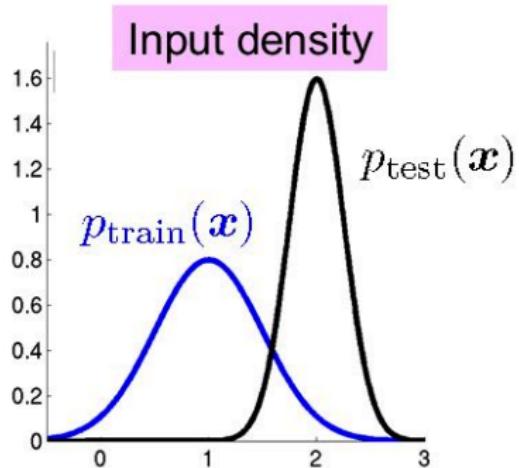
2 First generation of Neural Networks

3 Modern Neural Networks

4 Convolutional Neural Networks

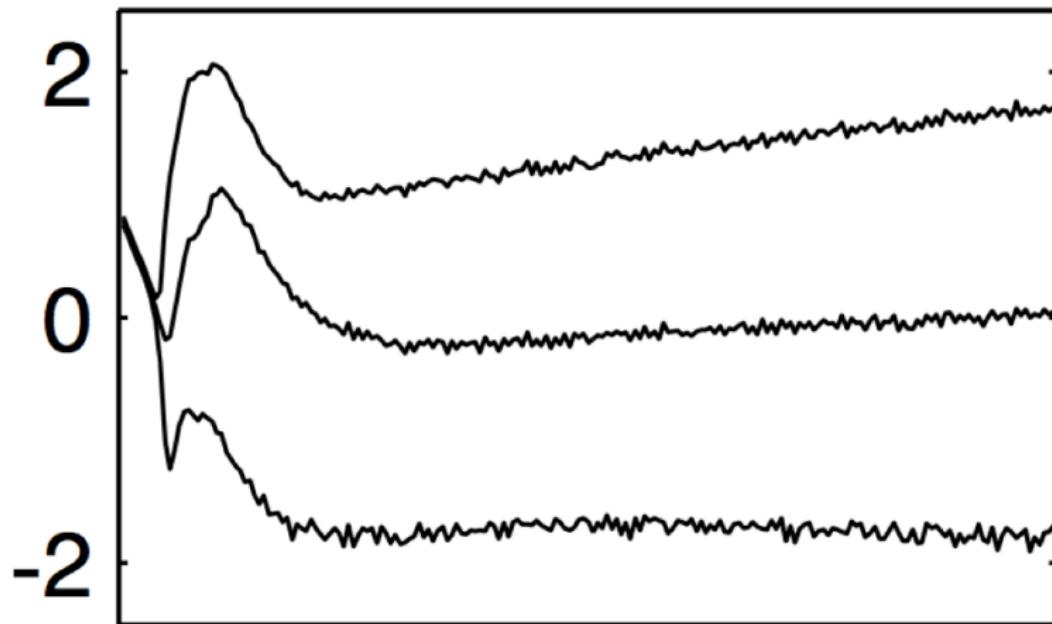
5 Transfer learning

6 Regularizing Neural Networks



Input distribution in neural network

- Input distribution changes over the course of training
- Example: 3 fully connected layers, 100 activations each
- Strong internal covariate shift



- First idea: standardize output of a layer
- $f(\mathbf{x}; b) = \mathbf{x} + b - \mathbb{E}(\mathbf{x} + b)$
- If the backprop ignores dependence of sample average on b , then after step $b + g$:

$$f(\mathbf{x}; b) = \mathbf{x} + b + g - \mathbb{E}(\mathbf{x} + b + g) = \mathbf{x} + b - \mathbb{E}(\mathbf{x} + b)$$

- Parameter b can grow indefinitely
- Gradients should take normalization into account!

- Normalizations layer:

$$\hat{\mathbf{x}} = \text{Norm}(\mathbf{x}, \mathcal{X})$$

- We should be able to calculate gradients:

$$\frac{\partial \text{Norm}(\mathbf{x}, \mathcal{X})}{\partial \mathbf{x}} \quad \text{and} \quad \frac{\partial \text{Norm}(\mathbf{x}, \mathcal{X})}{\partial \mathcal{X}}$$

- Standardization requires inverse square root of covariance matrix:

$$\{\text{Cov}(\mathbf{x})\}^{-1/2} (\mathbf{x} - \mathbb{E}\mathbf{x})$$

- Normalize each layer activation independently:

$$\hat{x}^{(k)} = \frac{x^{(k)} - \mathbb{E}[x^{(k)}]}{\sqrt{\text{Var}[x^{(k)}]}}$$

- Estimate mean and variance of input based on current mini-batch

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_1 \dots m\}$;

Parameters to be learned: γ, β

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

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

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

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

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

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

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

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

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

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

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

- Average mean and variance estimates over all training batches:

for $k = 1 \dots K$ **do**

// For clarity, $x \equiv x^{(k)}$, $\gamma \equiv \gamma^{(k)}$, $\mu_{\mathcal{B}} \equiv \mu_{\mathcal{B}}^{(k)}$, etc.

Process multiple training mini-batches \mathcal{B} , each of size m , and average over them:

$$\text{E}[x] \leftarrow \text{E}_{\mathcal{B}}[\mu_{\mathcal{B}}]$$

$$\text{Var}[x] \leftarrow \frac{m}{m-1} \text{E}_{\mathcal{B}}[\sigma_{\mathcal{B}}^2]$$

- Use these new estimates for inference:

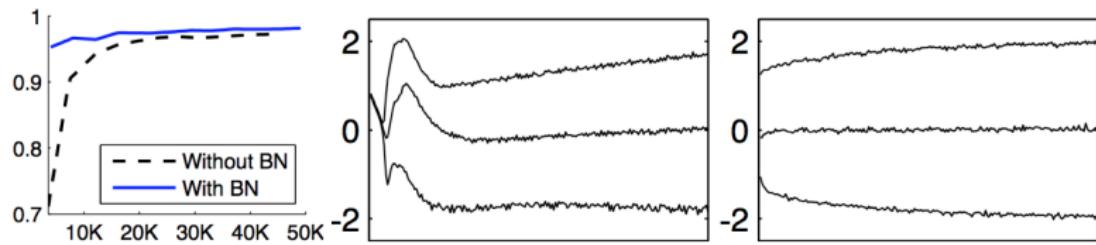
$$y = \frac{\gamma}{\sqrt{\text{Var}[x]+\epsilon}} \cdot x + \left(\beta - \frac{\gamma \text{E}[x]}{\sqrt{\text{Var}[x]+\epsilon}} \right)$$

- Usually BatchNorm is inserted before nonlinearity
- Learning rate can be increased
- Optimization becomes robust to parameter scale:

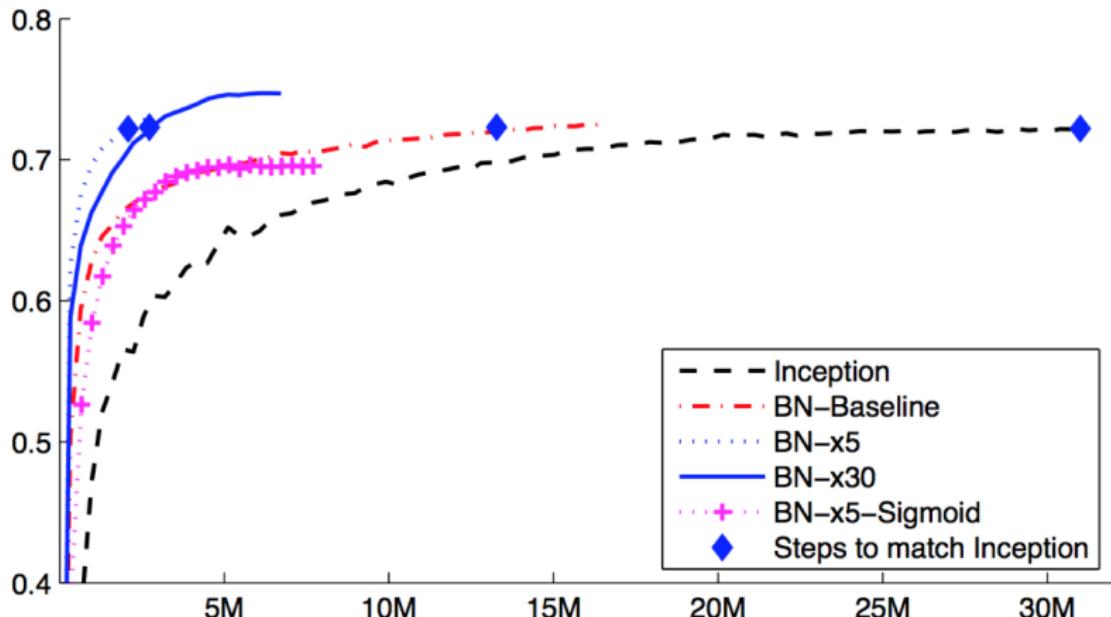
$$\frac{\partial \text{BN}((aW)u)}{\partial u} = \frac{\partial \text{BN}(Wu)}{\partial u}$$
$$\frac{\partial \text{BN}((aW)u)}{\partial (aW)} = \frac{1}{a} \cdot \frac{\partial \text{BN}(Wu)}{\partial W}$$

- Dropout layers can be removed
- Weight regularization can be reduced

BatchNorm example



BatchNorm example



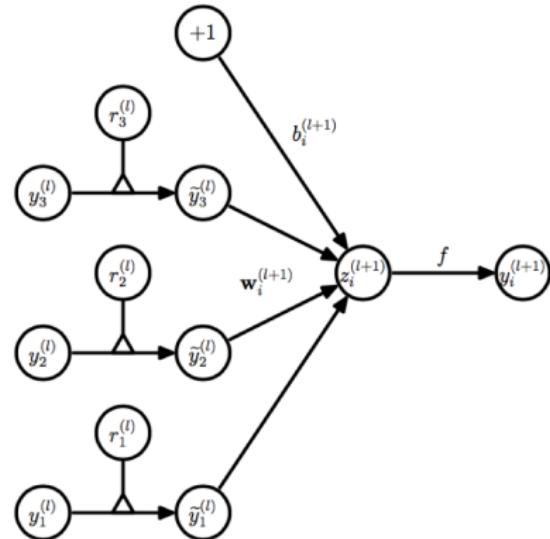
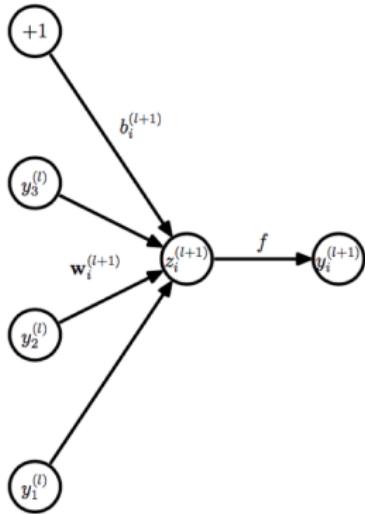
- Weights reparametrization:

$$\mathbf{w} = \frac{g}{\|\mathbf{v}\|} \mathbf{v}$$

- Fixes the Euclidean norm of \mathbf{w} to g
- Gradients:

$$\nabla_g L = \frac{\nabla_{\mathbf{w}} L \cdot \mathbf{v}}{\|\mathbf{v}\|}, \quad \nabla_{\mathbf{v}} L = \frac{g}{\|\mathbf{v}\|} \nabla_{\mathbf{w}} L - \frac{g \nabla_g L}{\|\mathbf{v}\|^2} \mathbf{v}$$

Model	Test Error
Maxout [6]	11.68%
Network in Network [17]	10.41%
Deeply Supervised [16]	9.6%
ConvPool-CNN-C [26]	9.31%
ALL-CNN-C [26]	9.08%
our CNN, mean-only B.N.	8.52%
our CNN, weight norm.	8.46%
our CNN, normal param.	8.43%
our CNN, batch norm.	8.05%
ours, W.N. + mean-only B.N.	7.31%



- Randomly disable outputs of a layer:

$$r_{jk} \sim \text{Bernoulli}(p)$$

$$v_{j+1}(\mathbf{x}) = f_{j+1}(v_j(\mathbf{x}) \otimes r_j; w_j)$$

- Prevents co-adaptation of neurons and makes them more robust to random perturbations
- Similar to training 2^n models with shared weights
- During inference: multiply weights of dropout layer by p
- Works well with max-norm regularization
- Variational dropout can learn separate dropout weight for each node

- Prevents co-adaptation of neurons and makes them more robust to random perturbations
- Similar to training 2^n models with shared weights
 - During inference: multiply weights of dropout layer by p
 - Works well with max-norm regularization
 - Variational dropout can learn separate dropout weight for each node

- Prevents co-adaptation of neurons and makes them more robust to random perturbations
- Similar to training 2^n models with shared weights
- During inference: multiply weights of dropout layer by p
- Works well with max-norm regularization
- Variational dropout can learn separate dropout weight for each node

- Prevents co-adaptation of neurons and makes them more robust to random perturbations
- Similar to training 2^n models with shared weights
- During inference: multiply weights of dropout layer by p
- Works well with max-norm regularization
- Variational dropout can learn separate dropout weight for each node

- Prevents co-adaptation of neurons and makes them more robust to random perturbations
- Similar to training 2^n models with shared weights
- During inference: multiply weights of dropout layer by p
- Works well with max-norm regularization
- Variational dropout can learn separate dropout weight for each node

- L_2 regularization: adds $\lambda \|\mathbf{w}\|^2$ to the loss function
- Penalizes peaky weight vectors
- Prevents neurons from focusing on one strong input
- Something similar to dropout
- Wager, Wang, Liang. Dropout Training as Adaptive Regularization:
“We show that the dropout regularizer is first-order equivalent to an L2 regularizer applied after scaling the features by an estimate of the inverse diagonal Fisher information matrix”

- L_2 regularization: adds $\lambda \|\mathbf{w}\|^2$ to the loss function
- Penalizes peaky weight vectors
 - Prevents neurons from focusing on one strong input
 - Something similar to dropout
- Wager, Wang, Liang. Dropout Training as Adaptive Regularization:
“We show that the dropout regularizer is first-order equivalent to an L2 regularizer applied after scaling the features by an estimate of the inverse diagonal Fisher information matrix”

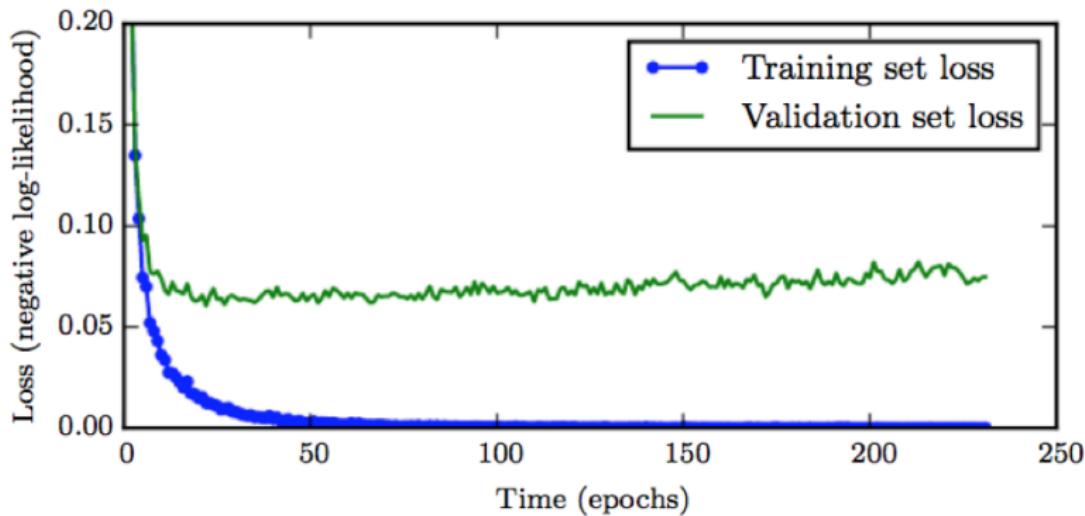
- L_2 regularization: adds $\lambda \|\mathbf{w}\|^2$ to the loss function
- Penalizes peaky weight vectors
- Prevents neurons from focusing on one strong input
- Something similar to dropout
- Wager, Wang, Liang. Dropout Training as Adaptive Regularization:
“We show that the dropout regularizer is first-order equivalent to an L2 regularizer applied after scaling the features by an estimate of the inverse diagonal Fisher information matrix”

- L_2 regularization: adds $\lambda\|\mathbf{w}\|^2$ to the loss function
- Penalizes peaky weight vectors
- Prevents neurons from focusing on one strong input
- Something similar to dropout
- Wager, Wang, Liang. Dropout Training as Adaptive Regularization:
“We show that the dropout regularizer is first-order equivalent to an L2 regularizer applied after scaling the features by an estimate of the inverse diagonal Fisher information matrix”

- L_2 regularization: adds $\lambda\|\mathbf{w}\|^2$ to the loss function
- Penalizes peaky weight vectors
- Prevents neurons from focusing on one strong input
- Something similar to dropout
- Wager, Wang, Liang. Dropout Training as Adaptive Regularization:
“We show that the dropout regularizer is first-order equivalent to an L2 regularizer applied after scaling the features by an estimate of the inverse diagonal Fisher information matrix”

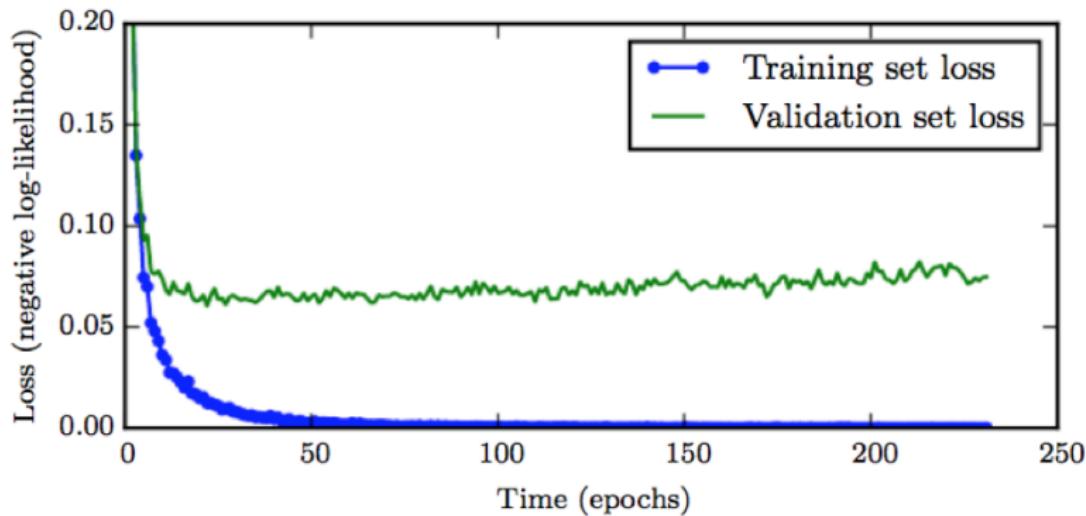
Early stopping

- Select number of optimization steps based on validation quality
- When the optimal number of steps is known, one can retrain the model both on training and validation data
- For a linear model with MSE loss and SGD optimizer, early stopping is equivalent to L_2 regularization



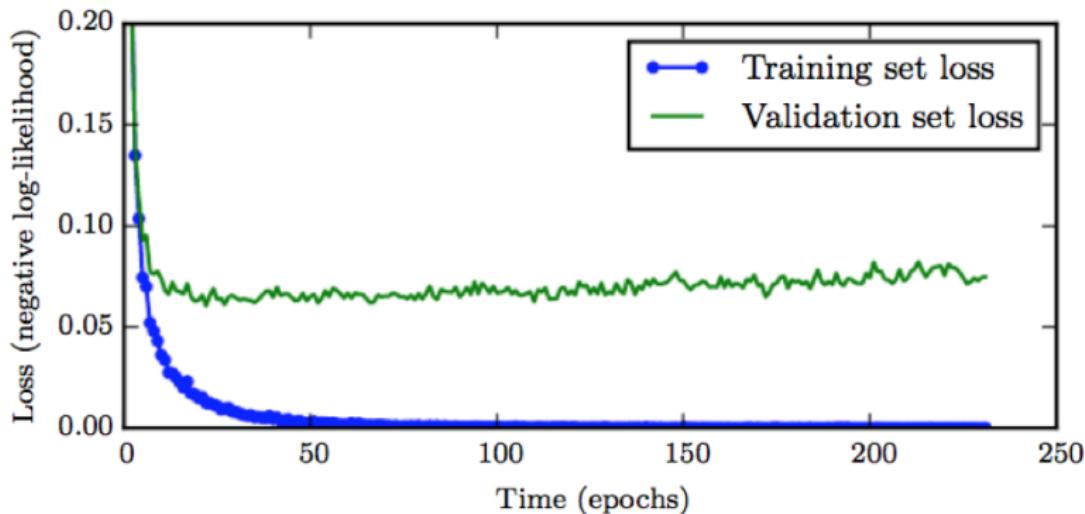
Early stopping

- Select number of optimization steps based on validation quality
- When the optimal number of steps is known, one can retrain the model both on training and validation data
- For a linear model with MSE loss and SGD optimizer, early stopping is equivalent to L_2 regularization

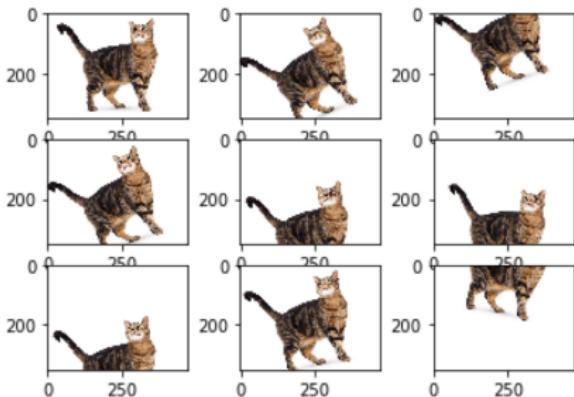


Early stopping

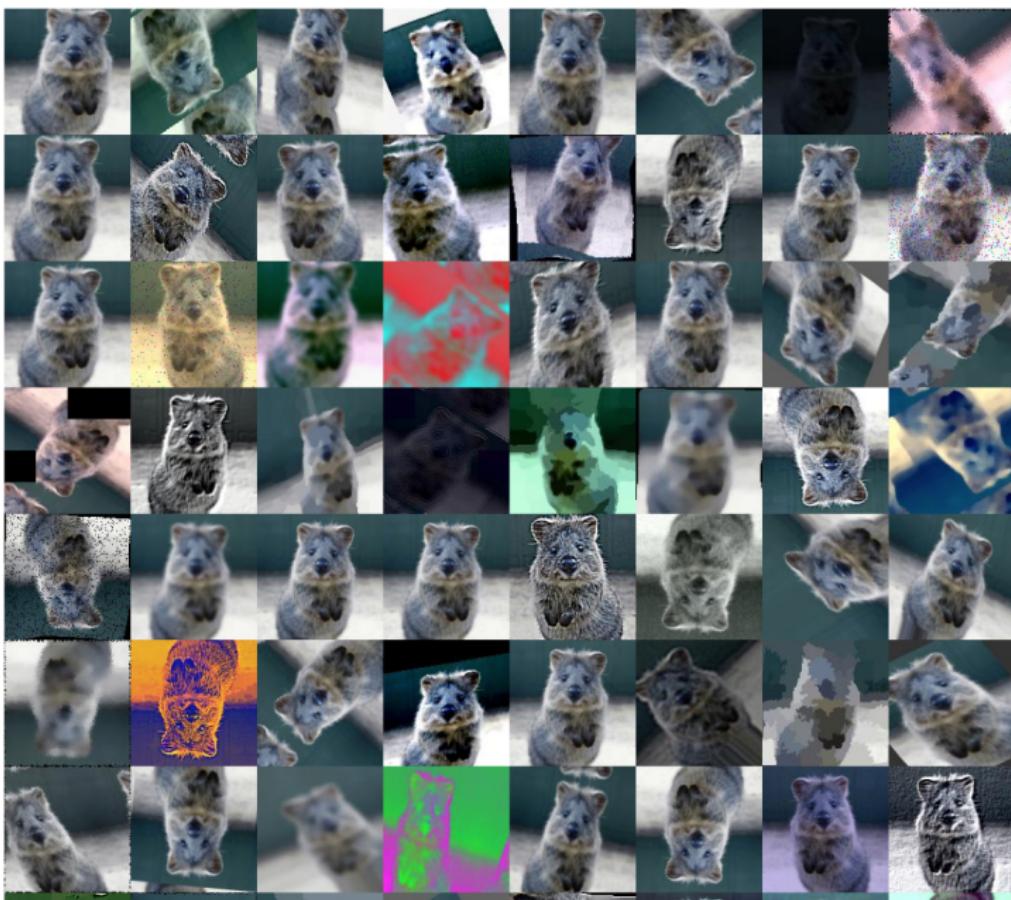
- Select number of optimization steps based on validation quality
- When the optimal number of steps is known, one can retrain the model both on training and validation data
- For a linear model with MSE loss and SGD optimizer, early stopping is equivalent to L_2 regularization



Data augmentation



Data augmentation



- Shift
- Zooming in/out
- Rotation
- Flip
- Distortion
- Shade

- You can even use style transfer!



- Time stretching
- Pitch shifting
- Dynamic Range Compression
- Background noise
- ...

- Improves robustness of the model
- Useful for small samples
- Improves results on large datasets too

- Normalizations improve training both in terms speed and quality
- Regularizations (dropout, max-norm, L_2) prevent co-adaptation
- Data augmentation is useful to improve generalization
- Many other regularization techniques: multi-task learning, adversarial samples, sparsification etc.