Advanced Deep Learning for Computer Vision

The Team

Lecturers

Prof. Dr. Matthias
Niessner

Website

https://niessnerlab.org/

Tutors

Lei
Li

Ziya
Erkoç

Lukas
Höllein

David
Rozenberszki

Visual Computing & AI Group at TUM

Photorealistic AI Avatars

NeRFs / 3D Gaussians

3D Semantics / Reconstruction

https://niessnerlab.org/publications.html
https://twitter.com/MattNiessner

Visual Computing Group
Prof. Matthias Nießner

Visual Computing & AI Group at TUM

MeshGPT: 3D Mesh Generation

SceneTex: 3D Scene Texturing

https://niessnerlab.org/publications.html
https://twitter.com/MattNiessner

History of the Lecture

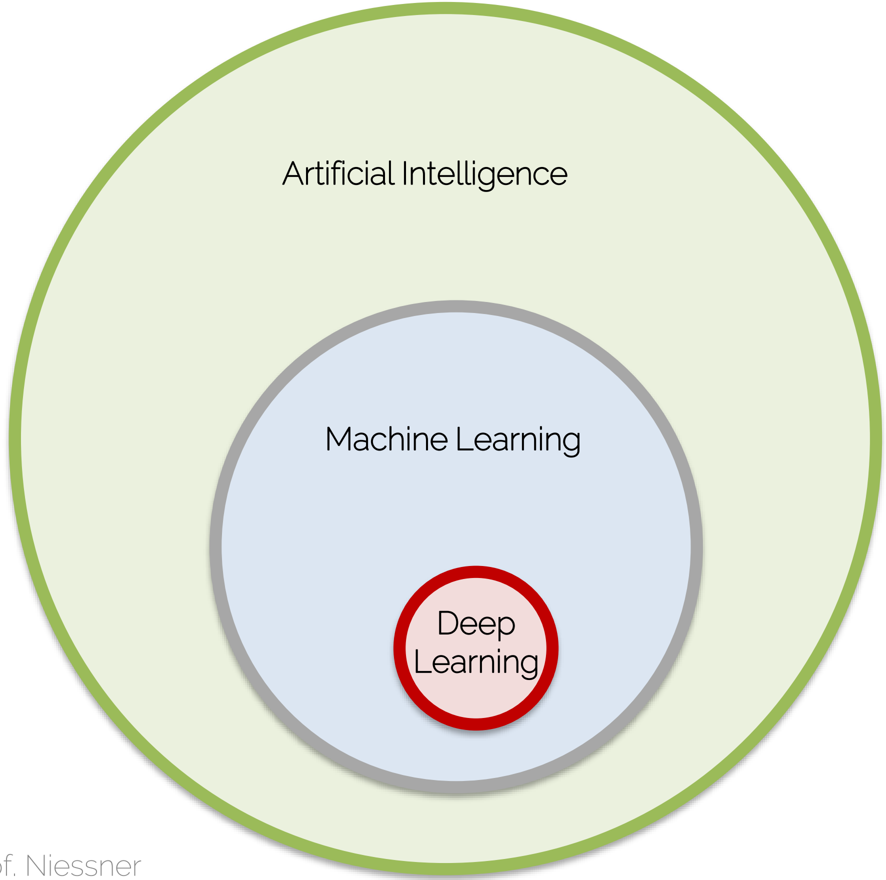
- Follow up on Introduction to Deep Learning (I2DL)
  - https://niessner.github.io/I2DL/
  - Many ADL4CV iterations

- Together with Dynamic Vision and Learning Group
  - https://dvl.in.tum.de/

Prof. Dr. Laura
Leal-Taixé (now at Nvidia)

Basics of DL

AI vs ML vs DL



- Deep Learning
  - ML-methods leveraging neural networks
    - Fit non-linear function to training set through optimization
    - "Hope" that we generalize to unseen training samples

What we assume you know

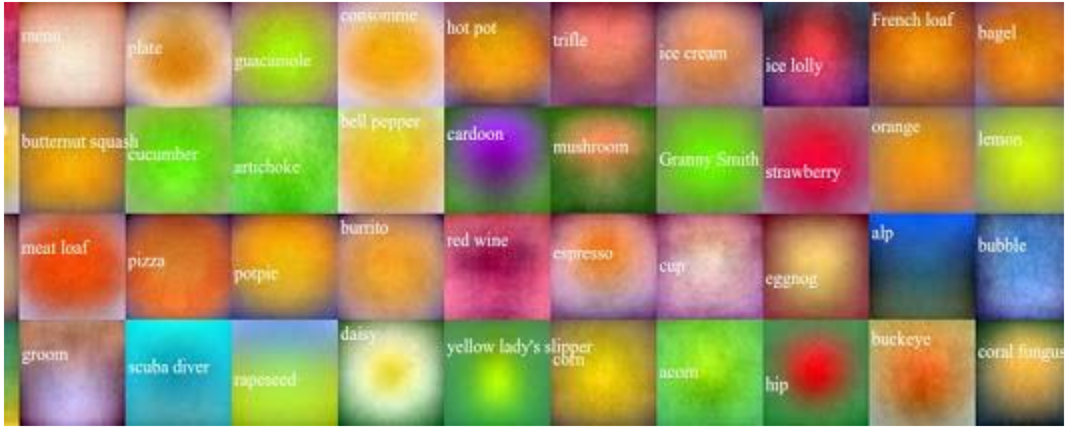
- Linear Algebra & Programming!
- Basics from the Introduction to Deep Learning lecture
  - https://niessner.github.io/I2DL/
- PyTorch (can use TensorFlow…)
- You already trained several models + you know how to debug problems, observe training curves, prepare training/validation/test data

What is a neural network?

Neural network

- Linear score function $f = Wx$

On CIFAR-10



On ImageNet

Credit: Li/Karpathy/Johnson

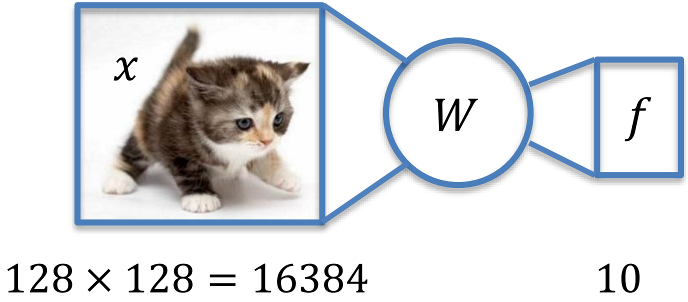
Neural network

- Linear score function $f = Wx$

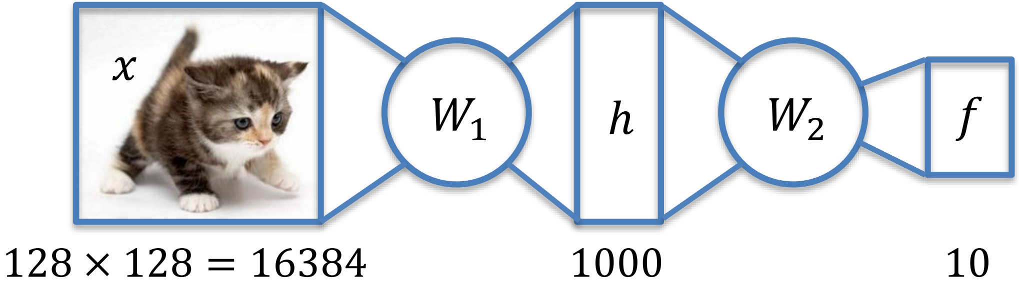
- Neural network is a nesting of 'functions'
  - 2-layers: $f = W_2 \max(0, W_1 x)$
  - 3-layers: $f = W_3 \max(0, W_2 \max(0, W_1 x))$
  - 4-layers: $f = W_4 \tanh\,(\mathrm{W}_3, \max(0, W_2 \max(0, W_1 x)))$
  - 5-layers: $f = W_5 \sigma(W_4 \tanh(\mathrm{W}_3, \max(0, W_2 \max(0, W_1 x))))$
  - … up to hundreds of layers

Neural network

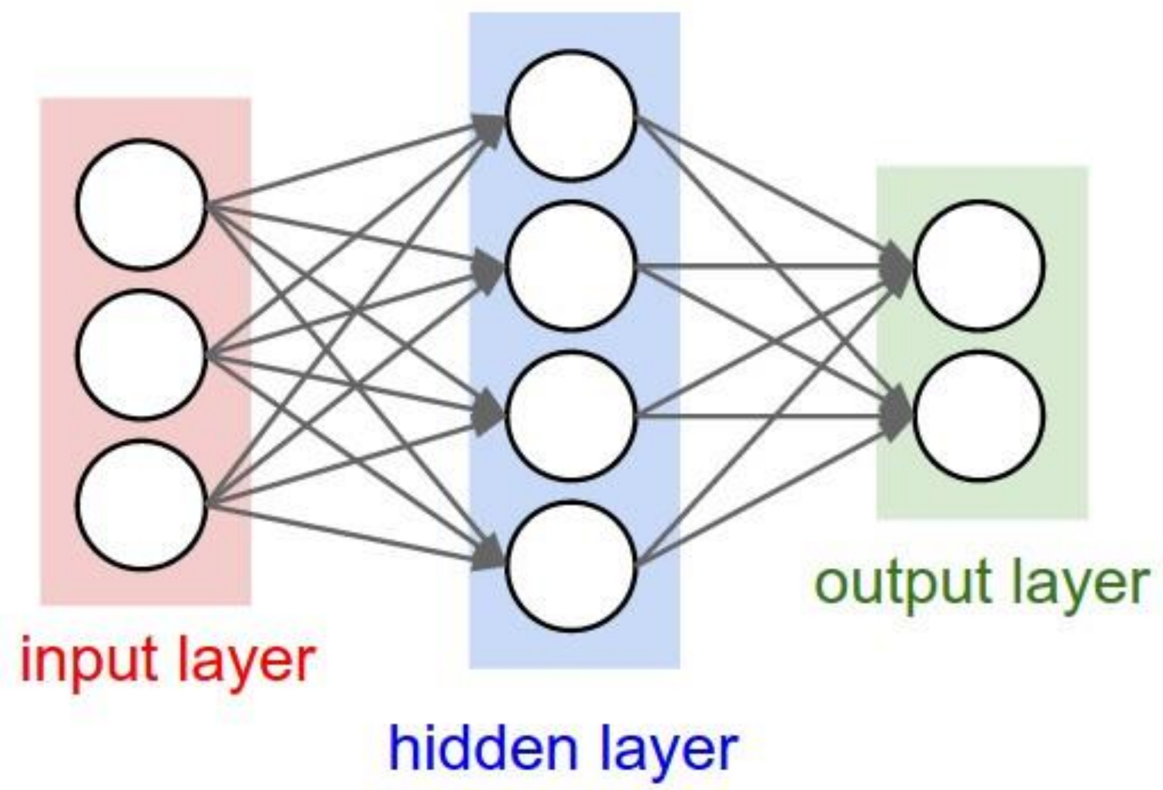
1-layer network: $f = Wx$



2-layer network: $f = W_2 \max(0, W_1 x)$



Neural network

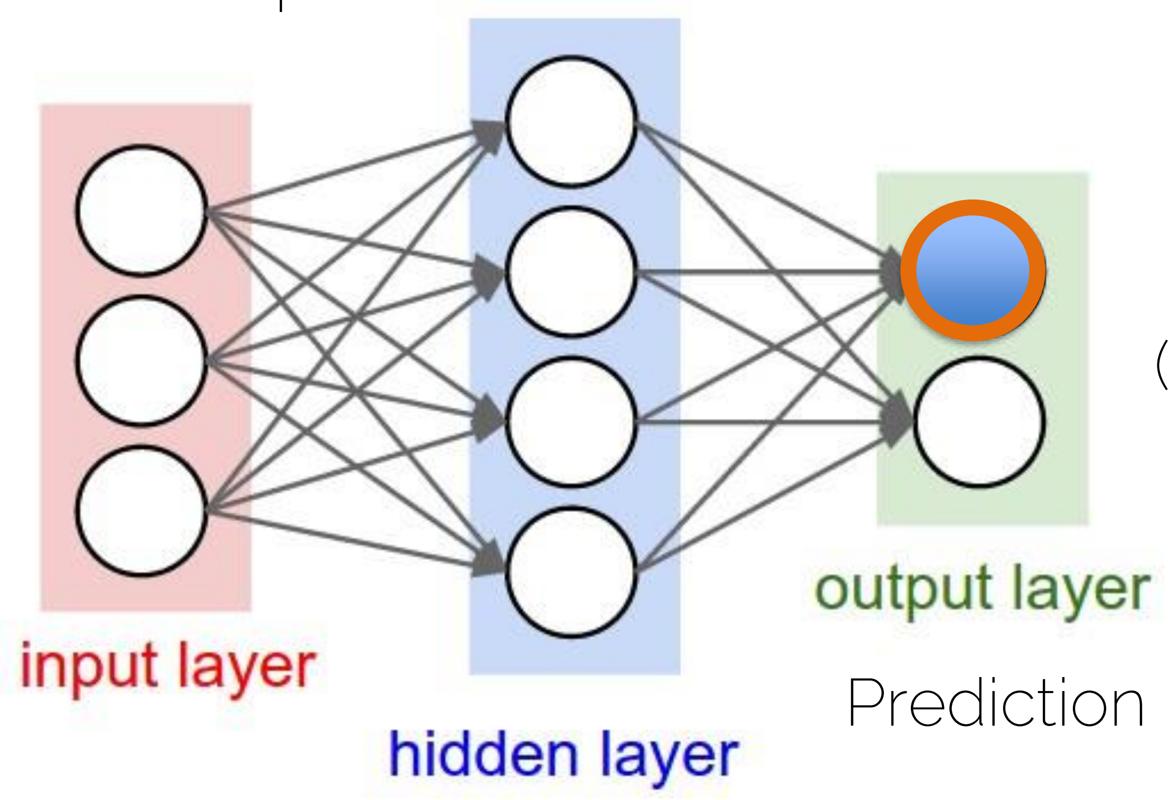


Credit: Li/Karpathy/Johnson

Loss functions

Neural networks

- What is the shape of this function?



Loss (Softmax, Hinge)

Prediction

Loss functions

- Softmax loss function

$$L_i = -\log\left(\frac{e^{s_{y_i}}}{\sum_k e^{s_k}}\right)$$

Evaluate the ground truth score for the image

- Hinge Loss (derived from the Multiclass SVM loss)

$$L_i = \sum_{k \neq y_i} \max(0, s_k - s_{y_i} + 1)$$

Loss functions

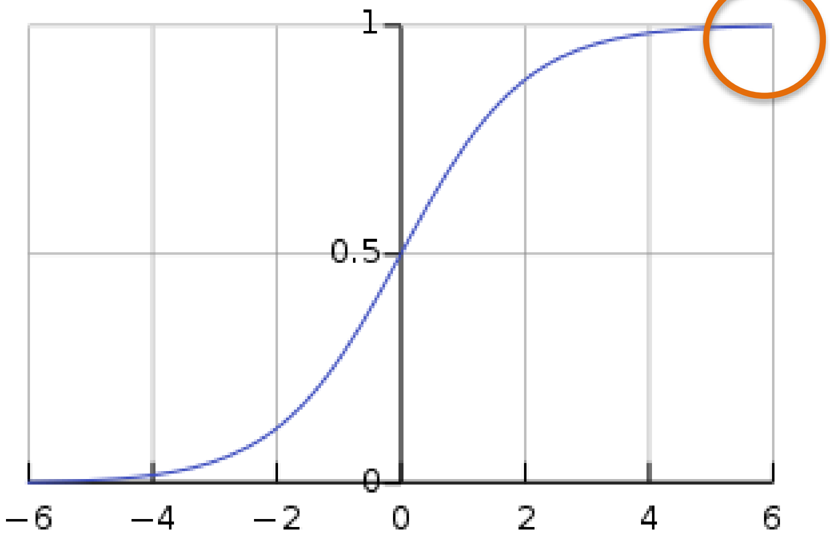
- Softmax loss function
  - Optimizes until the loss is zero
- Hinge Loss (derived from the Multiclass SVM loss)
  - Saturates whenever it has learned a class "well enough"

Activation functions

Sigmoid

$$\sigma(x) = \frac{1}{1+e^{-x}}$$

Forward



$$x = 6$$

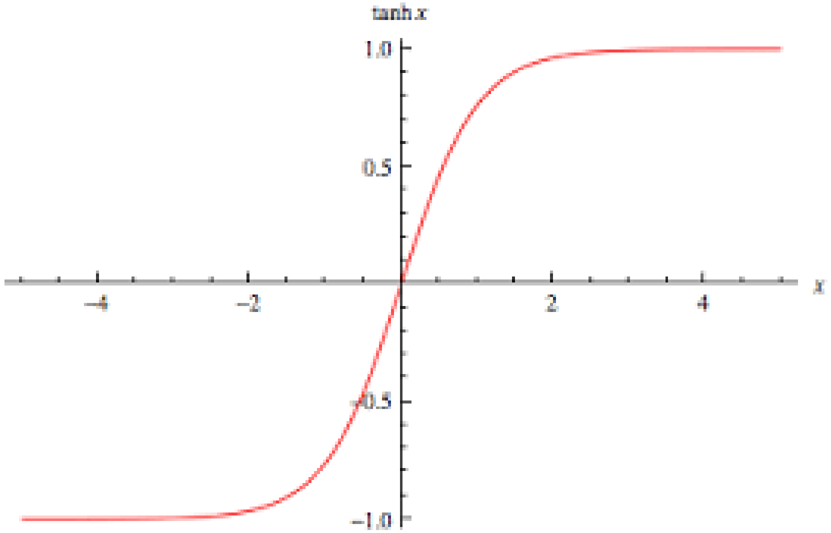
✘ Saturated neurons kill the gradient flow

$$\frac{\partial L}{\partial x} = \frac{\partial \sigma}{\partial x}\frac{\partial L}{\partial \sigma}$$

$$\frac{\partial \sigma}{\partial x}$$

$$\frac{\partial L}{\partial \sigma}$$

tanh



✗ Still saturates

✓ Zero-centered

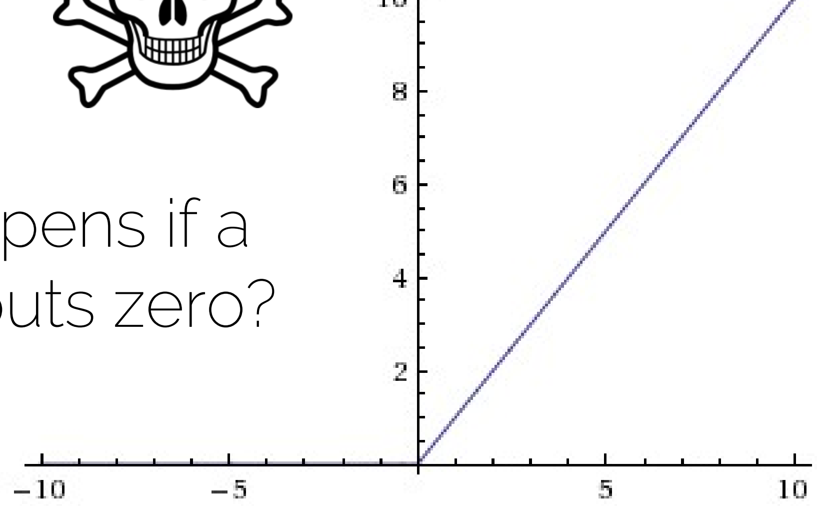
✗ Still saturates

LeCun 1991

Rectified Linear Units (ReLU)

✗ Dead ReLU

What happens if a ReLU outputs zero?

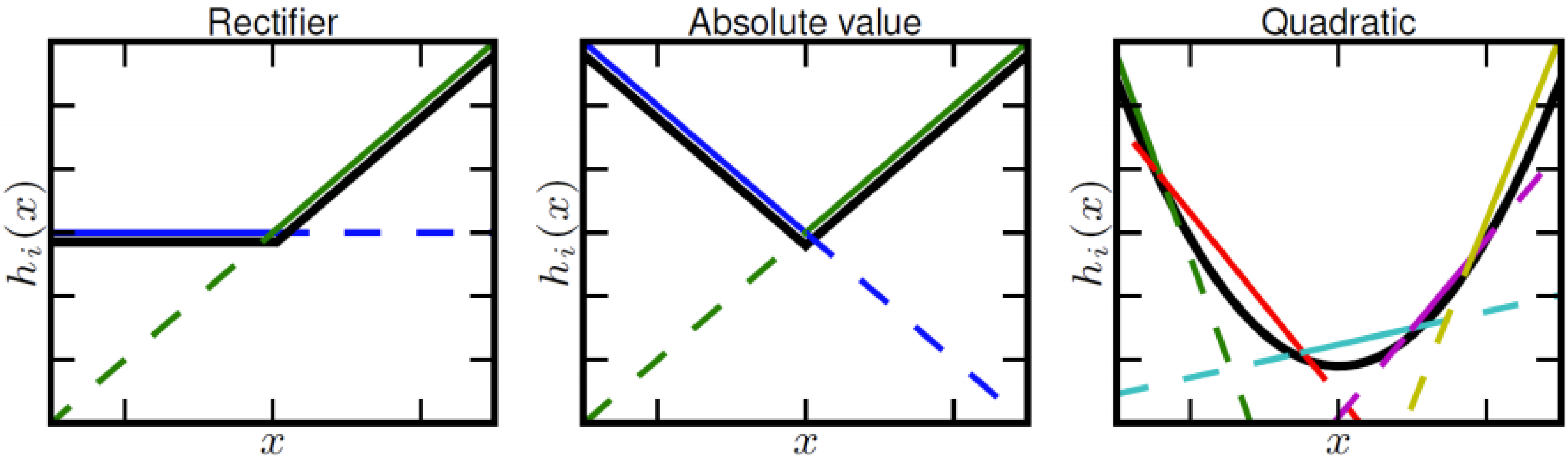


Large and consistent gradients ✓

✓ Fast convergence

✓ Does not saturate

Maxout units



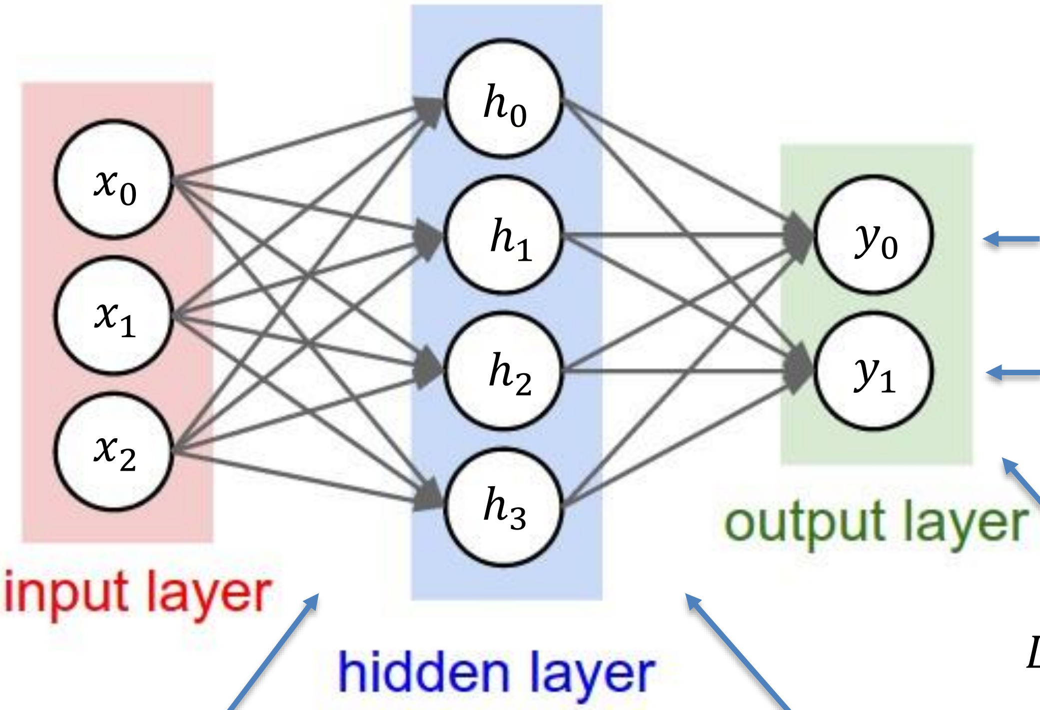
k=2 k=2 k=5

✓ Generalization of ReLUs
✓ Linear regimes
✓ Does not die
✓ Does not saturate

✗ Increase of the number of parameters

Optimization

Gradient Descent for Neural Networks



t_0

t_1

$$\nabla_{w,b} f_{\{x,t\}}(w) = \begin{bmatrix} \frac{\partial f}{\partial w_{0,0,0}} \\ \cdots \\ \cdots \\ \frac{\partial f}{\partial w_{l,m,n}} \\ \cdots \\ \cdots \\ \frac{\partial f}{\partial b_{l,m}} \end{bmatrix}$$

$$L_i = (y_i - t_i)^2$$

$$h_j = A(b_{0,j} + \sum_k x_k w_{0,j,k})$$

$$y_i = A(b_{1,i} + \sum_j h_j w_{1,i,j})$$

Just simple: $A(x) = \max(0, x)$

Stochastic Gradient Descent (SGD)

$$\theta^{k+1} = \theta^k - \alpha \nabla_\theta L(\theta^k, x_{\{1..m\}}, y_{\{1..m\}})$$

$$\nabla_\theta L = \frac{1}{m} \sum_{i=1}^{m} \nabla_\theta L_i$$

$\boldsymbol{k}$ now refers to $\boldsymbol{k}$-th iteration

$\boldsymbol{m}$ training samples in the current batch

Gradient for the $\boldsymbol{k}$-th batch

Note the terminology: iteration vs epoch

Gradient Descent with Momentum

$$v^{k+1} = \beta \cdot v^k + \nabla_\theta L(\theta^k)$$

accumulation rate
('friction', momentum)

velocity

Gradient of current minibatch

$$\theta^{k+1} = \theta^k - \alpha \cdot v^{k+1}$$

model

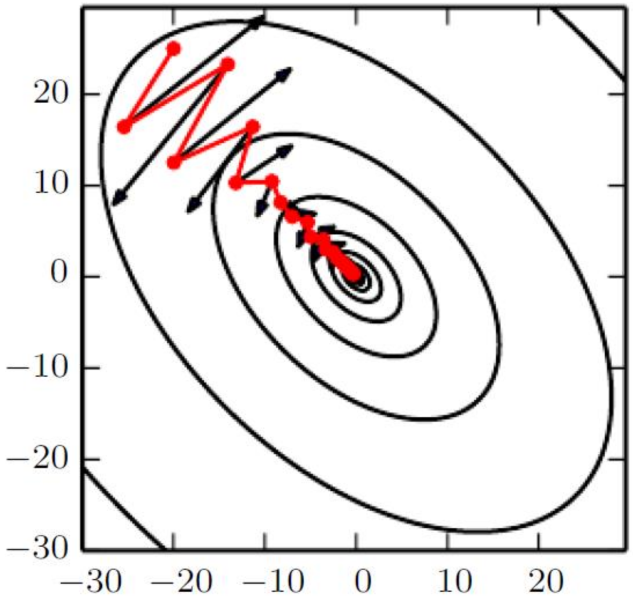
learning rate

velocity

Exponentially-weighted average of gradient

Important: velocity $\boldsymbol{v}^k$ is vector-valued!

Gradient Descent with Momentum



Step will be largest when a sequence of gradients all point to the same direction

Hyperparameters are α, β
β is often set to 0.9

$$\theta^{k+1} = \theta^{k} - \alpha \cdot v^{k+1}$$

Fig. credit: I. Goodfellow

RMSProp

$$s^{k+1} = \beta \cdot s^k + (1-\beta)\boxed{[\nabla_\theta L \circ \nabla_\theta L]}$$

Element-wise multiplication

$$\theta^{k+1} = \theta^k - \alpha \cdot \frac{\nabla_\theta L}{\sqrt{s^{k+1}} + \epsilon}$$

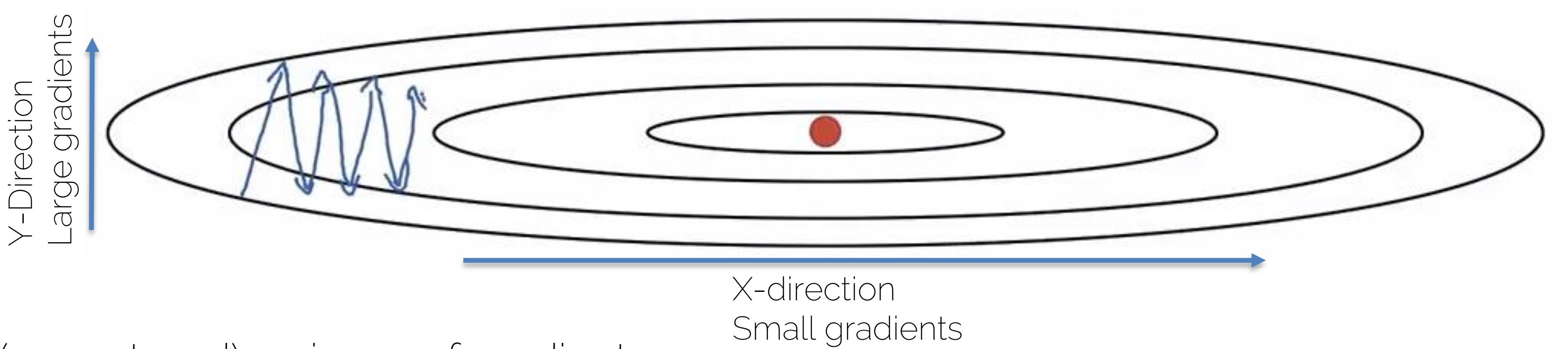
Hyperparameters: α, β, ϵ

Needs tuning!

Often 0.9

Typically 10^{-8}

RMSProp



(uncentered) variance of gradients
-> second momentum

$$\boxed{s^{k+1} = \beta \cdot s^k + (1 - \beta)[\nabla_\theta L \circ \nabla_\theta L]}$$

$$\theta^{k+1} = \theta^k - \alpha \cdot \frac{\nabla_\theta L}{\sqrt{s^{k+1}} + \epsilon}$$

We're dividing by square gradients:
- Division in Y-Direction will be large
- Division in X-Direction will be small

Can increase learning rate!

Fig. credit: A. Ng

Adaptive Moment Estimation (Adam)

Combines Momentum and RMSProp

$$\boxed{m^{k+1} = \beta_1 \cdot m^k + (1-\beta_1)\nabla_\theta L(\theta^k)}$$

First momentum: mean of gradients

$$\boxed{v^{k+1} = \beta_2 \cdot v^k + (1-\beta_2)[\nabla_\theta L(\theta^k) \circ \nabla_\theta L(\theta^k)]}$$

Second momentum: variance of gradients

$$\theta^{k+1} = \theta^k - \alpha \cdot \frac{m^{k+1}}{\sqrt{v^{k+1}}+\epsilon}$$

Adam

Combines Momentum and RMSProp

$$m^{k+1} = \beta_1 \cdot m^k + (1-\beta_1)\nabla_\theta L(\theta^k)$$

$$v^{k+1} = \beta_2 \cdot v^k + (1-\beta_2)[\nabla_\theta L(\theta^k) \circ \nabla_\theta L(\theta^k)]$$

$$\theta^{k+1} = \theta^k - \alpha \cdot \frac{\hat{m}^{k+1}}{\sqrt{\hat{v}^{k+1}}+\epsilon}$$

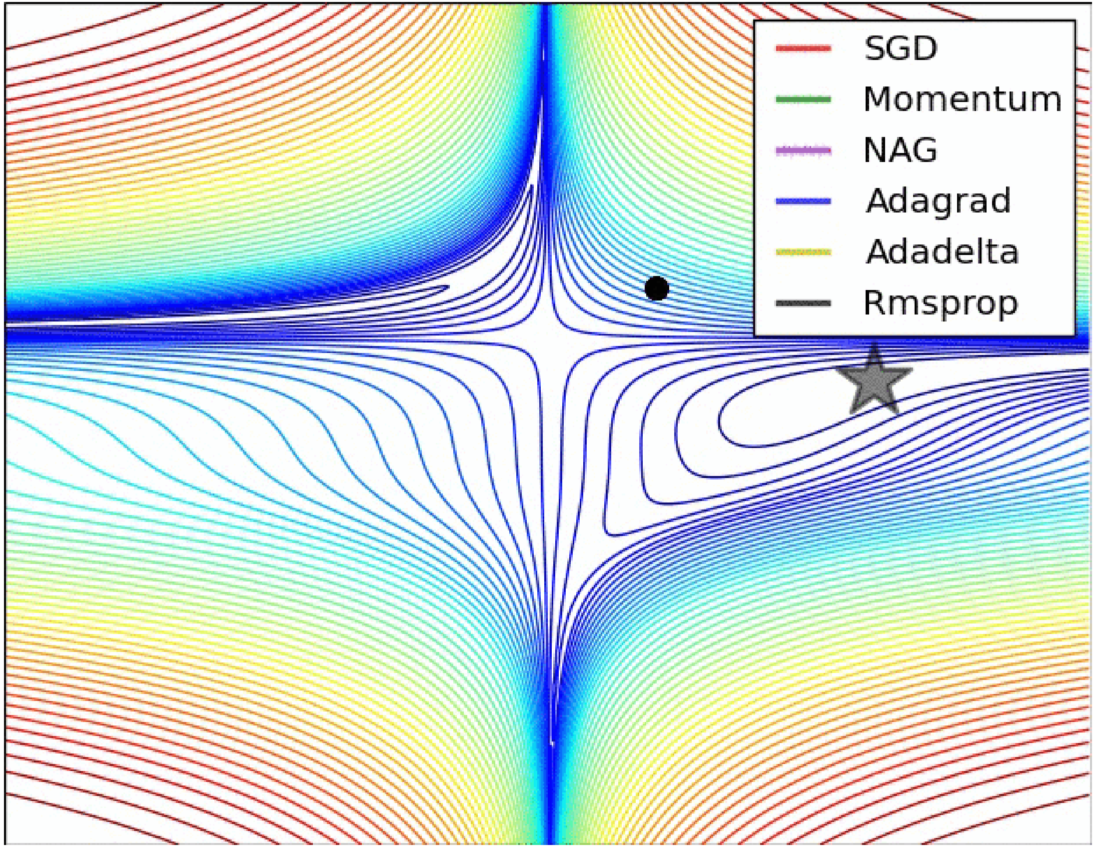
m^{k+1} and v^{k+1} are initialized with zero
-> bias towards zero

Typically, bias-corrected moment updates

$$\hat{m}^{k+1} = \frac{m^k}{1-\beta_1}$$

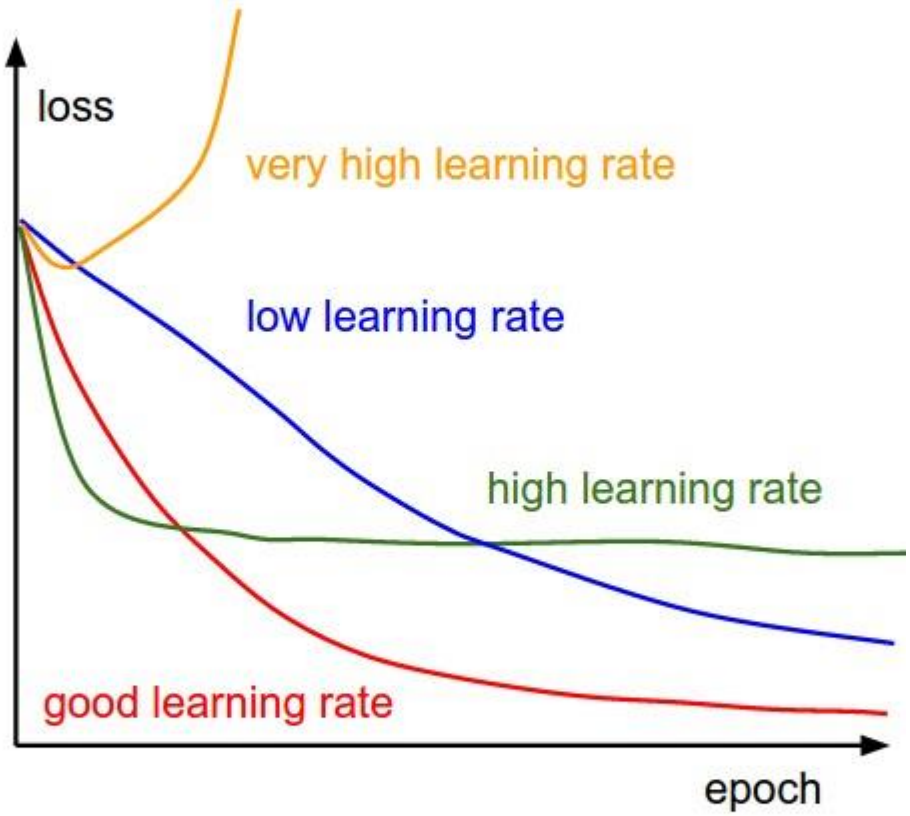
$$\hat{v}^{k+1} = \frac{v^k}{1-\beta_2}$$

Convergence



Training NNs

Importance of Learning Rate



Over- and Underfitting

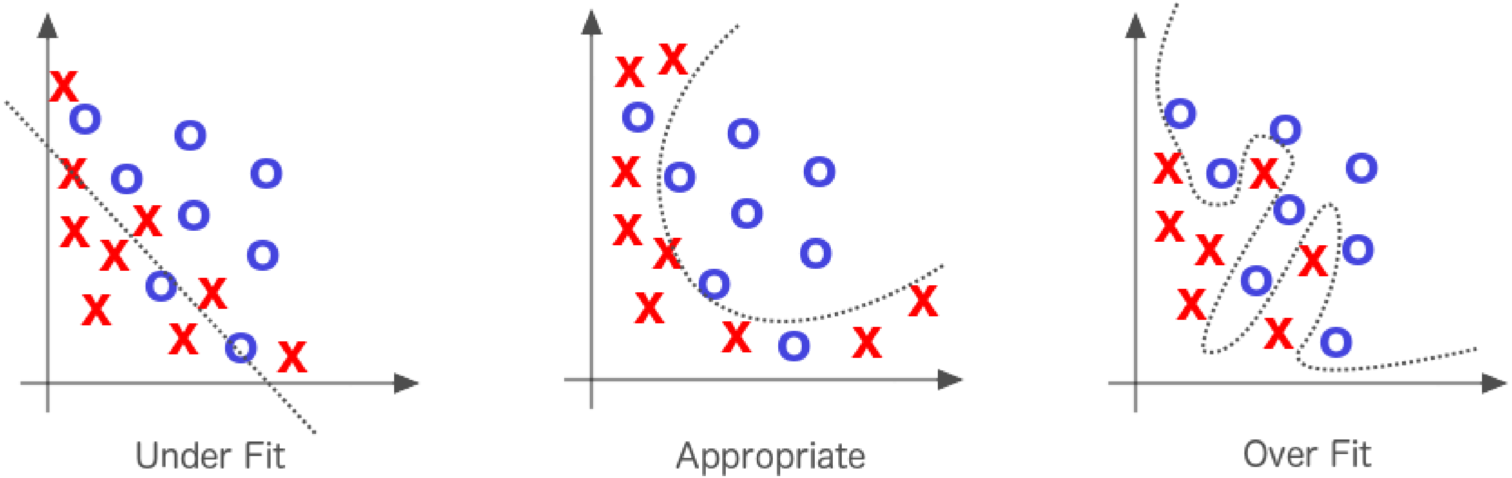
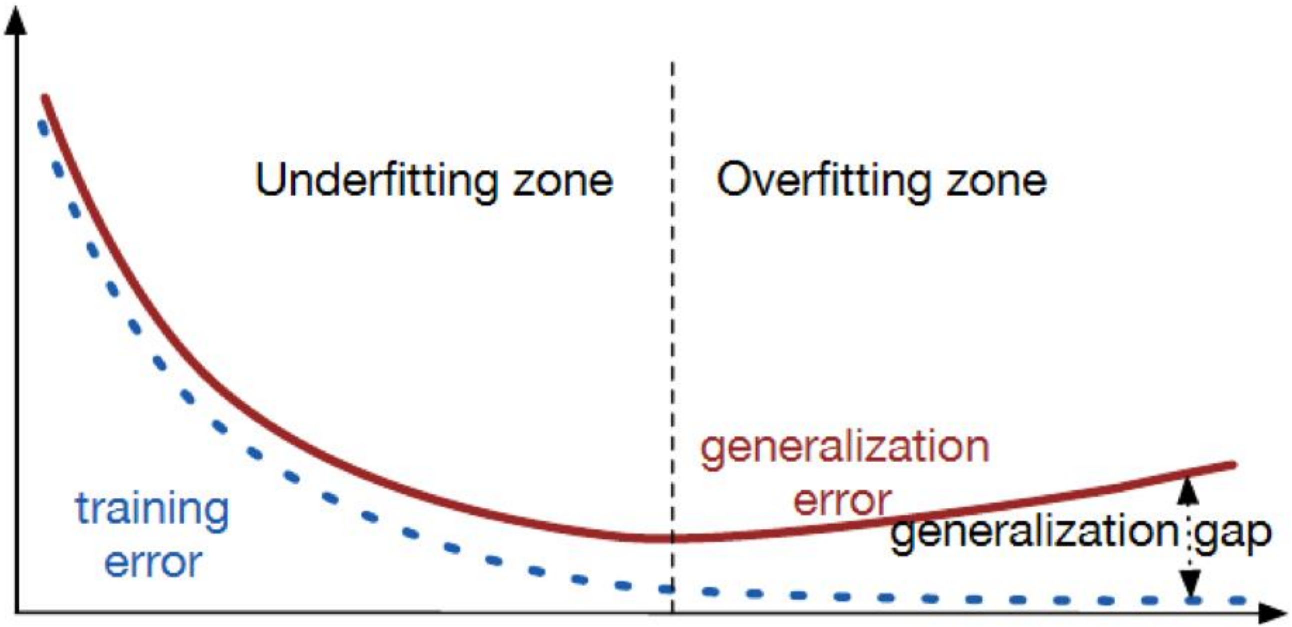


Figure extracted from Deep Learning by Adam Gibson, Josh Patterson, O'Reily Media Inc., 2017

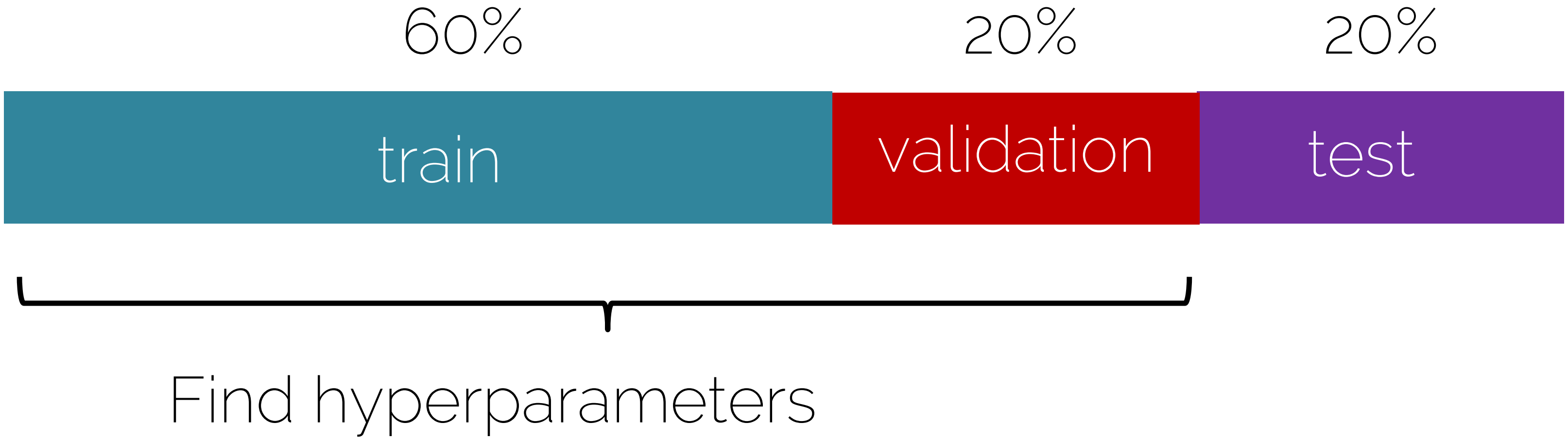
Over- and Underfitting



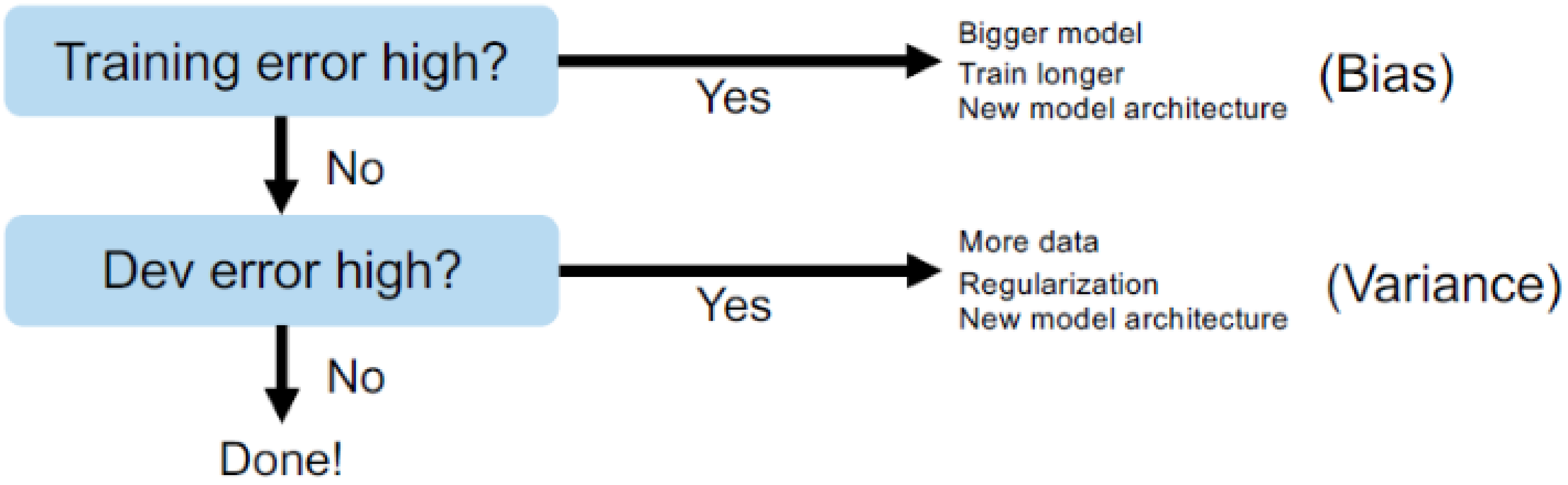
Source: http://srdas.github.io/DLBook/ImprovingModelGeneralization.html

Basic recipe for machine learning

- Split your data



Basic recipe for machine learning



Regularization

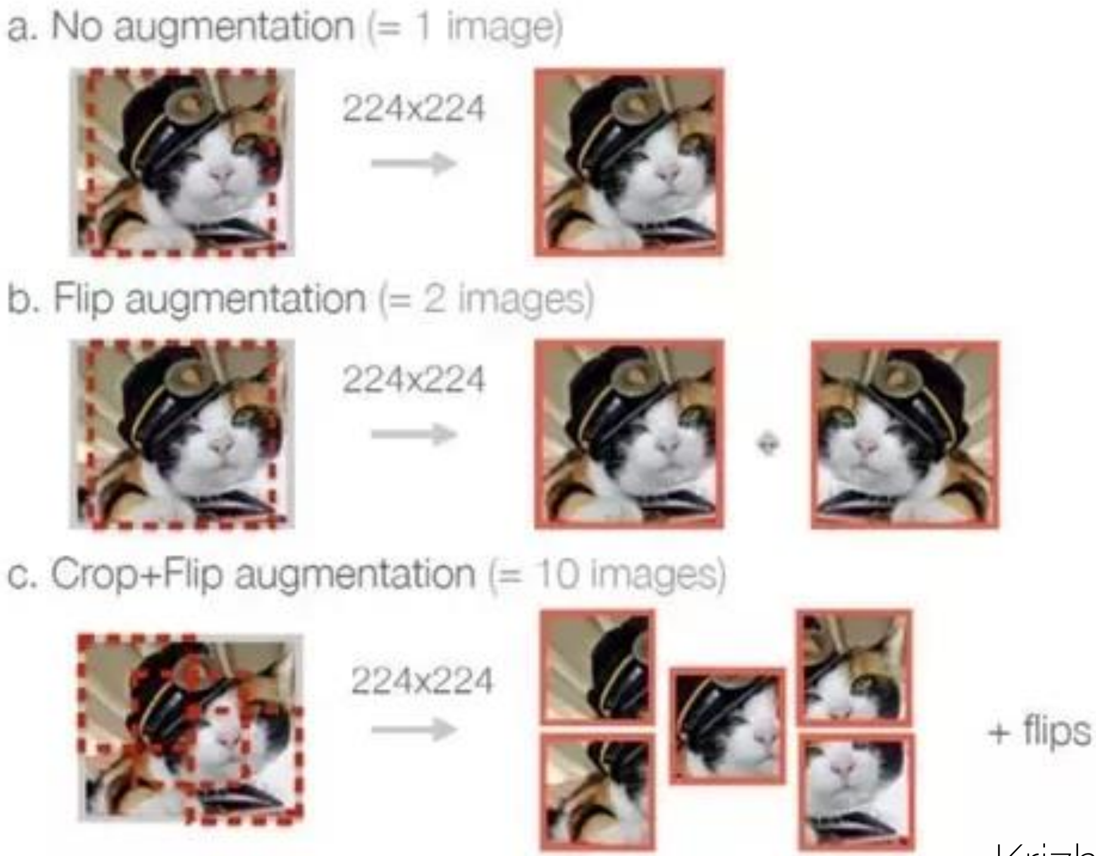
Regularization

- Any strategy that aims to

Lower validation error

Increasing training error

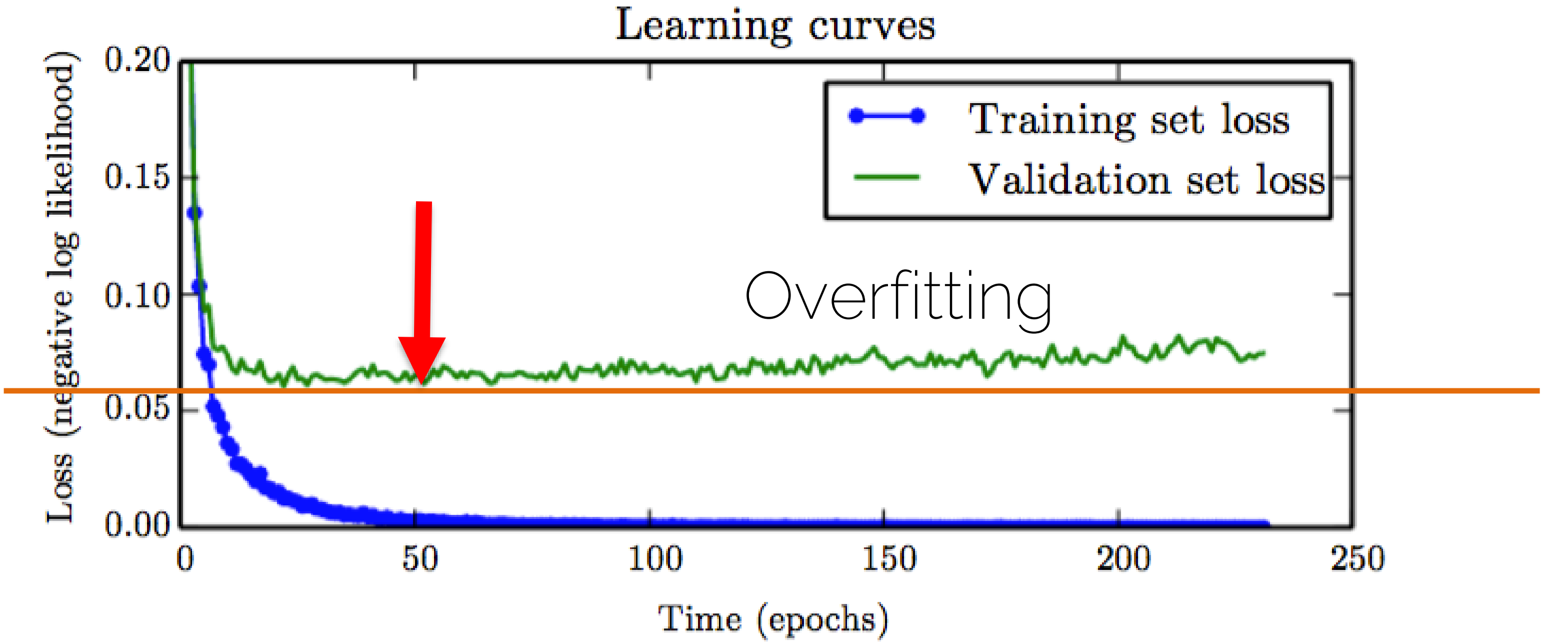
Data augmentation



Krizhevsky 2012

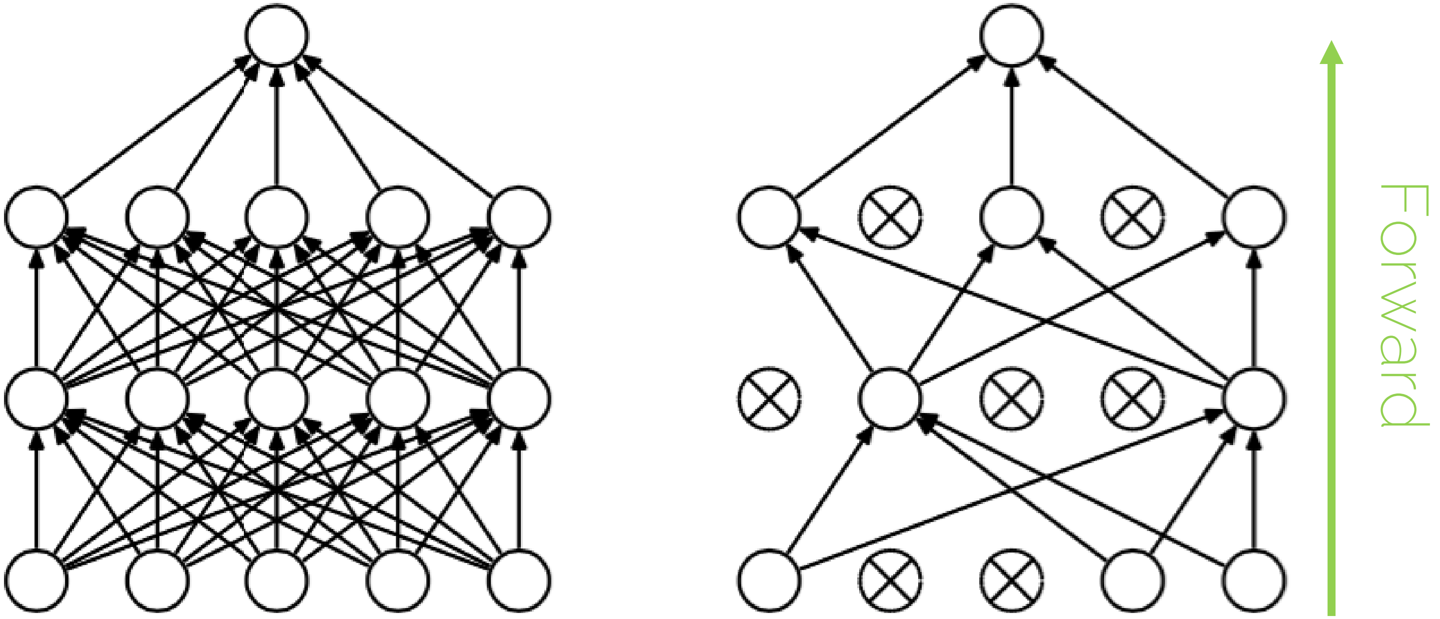
Early stopping

- Training time is also a hyperparameter



Dropout

- Disable a random set of neurons (typically 50%)



(a) Standard Neural Net

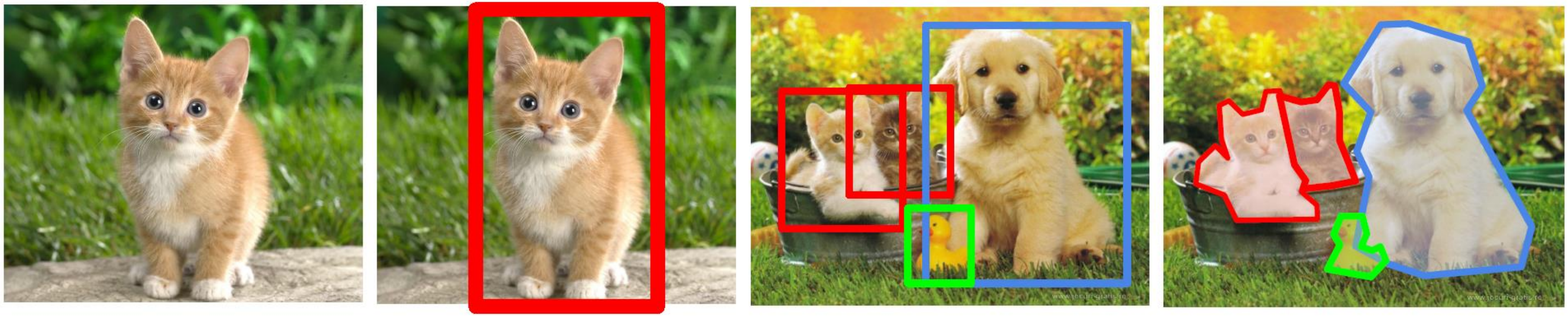
(b) After applying dropout.

Srivastava 2014

How to deal with images?

Using CNNs in Computer Vision

Classification	Classification + Localization	Object Detection	Instance Segmentation



CAT	CAT	CAT, DOG, DUCK	CAT, DOG, DUCK

Single object

Multiple objects

Image filters

- Each kernel gives us a different image filter

Input

Edge detection

$$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$$

Box mean

$$\frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

Sharpen

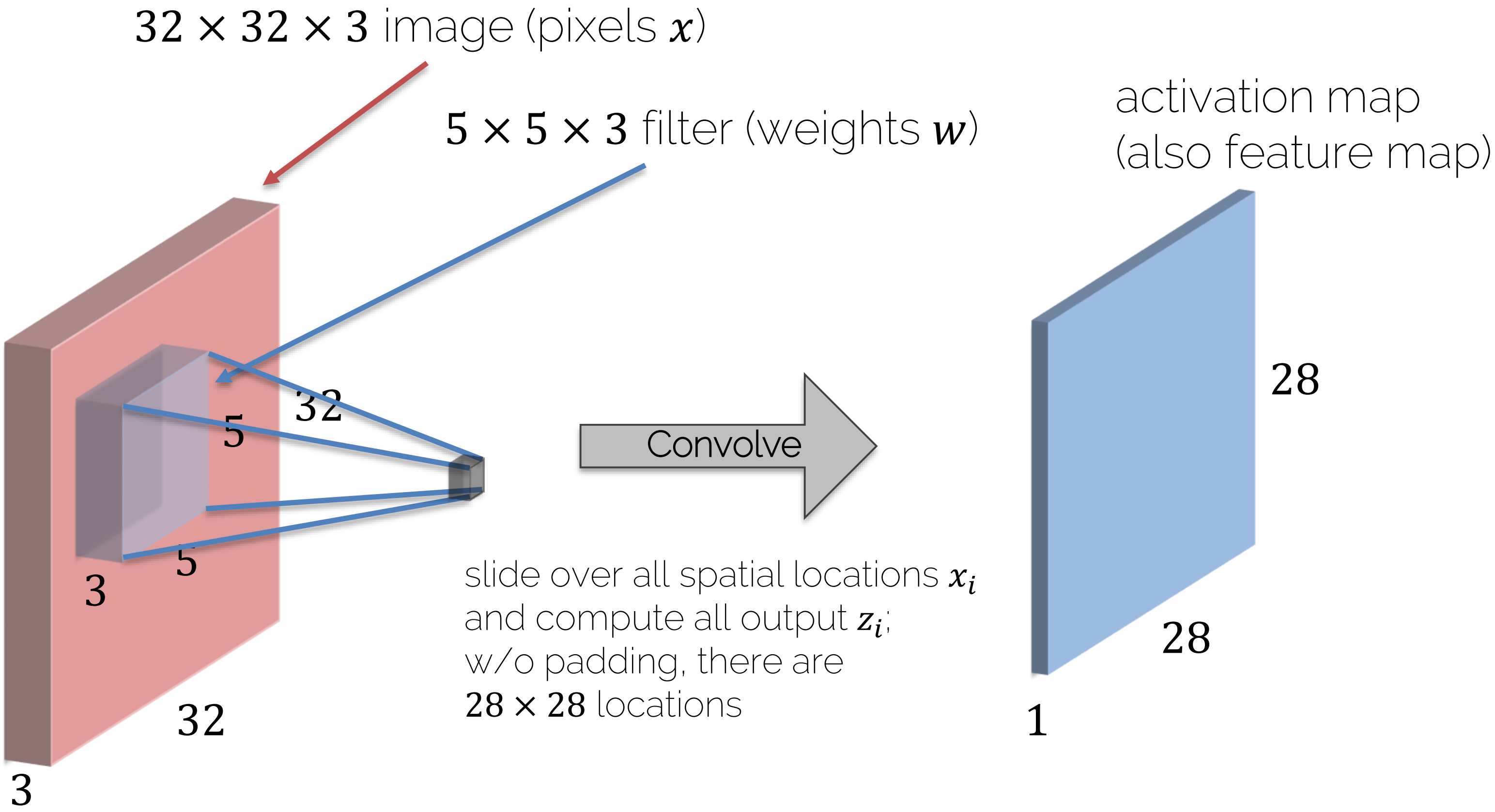
$$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$$

Gaussian blur

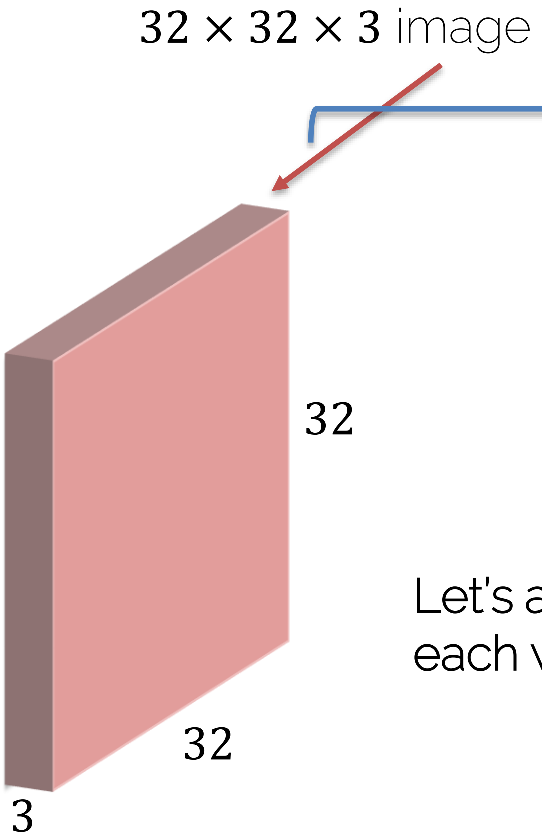
$$\frac{1}{16}\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$$

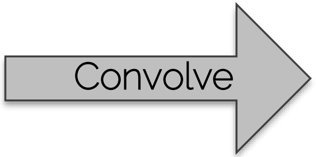
LET'S LEARN THESE FILTERS!

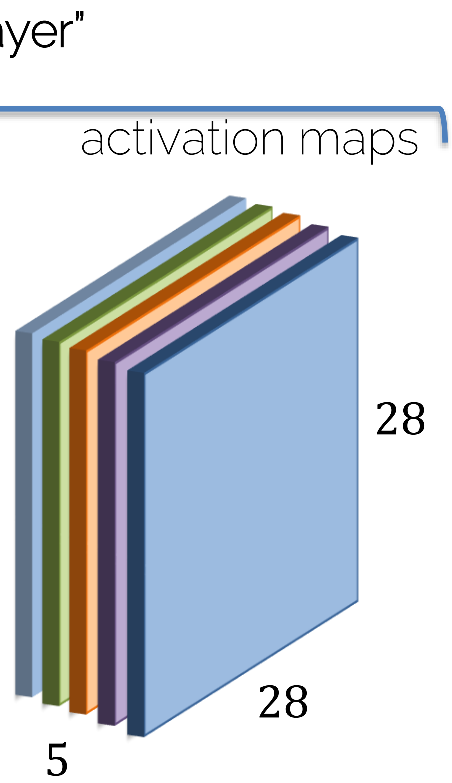
Convolutions on RGB Images



Convolution Layer



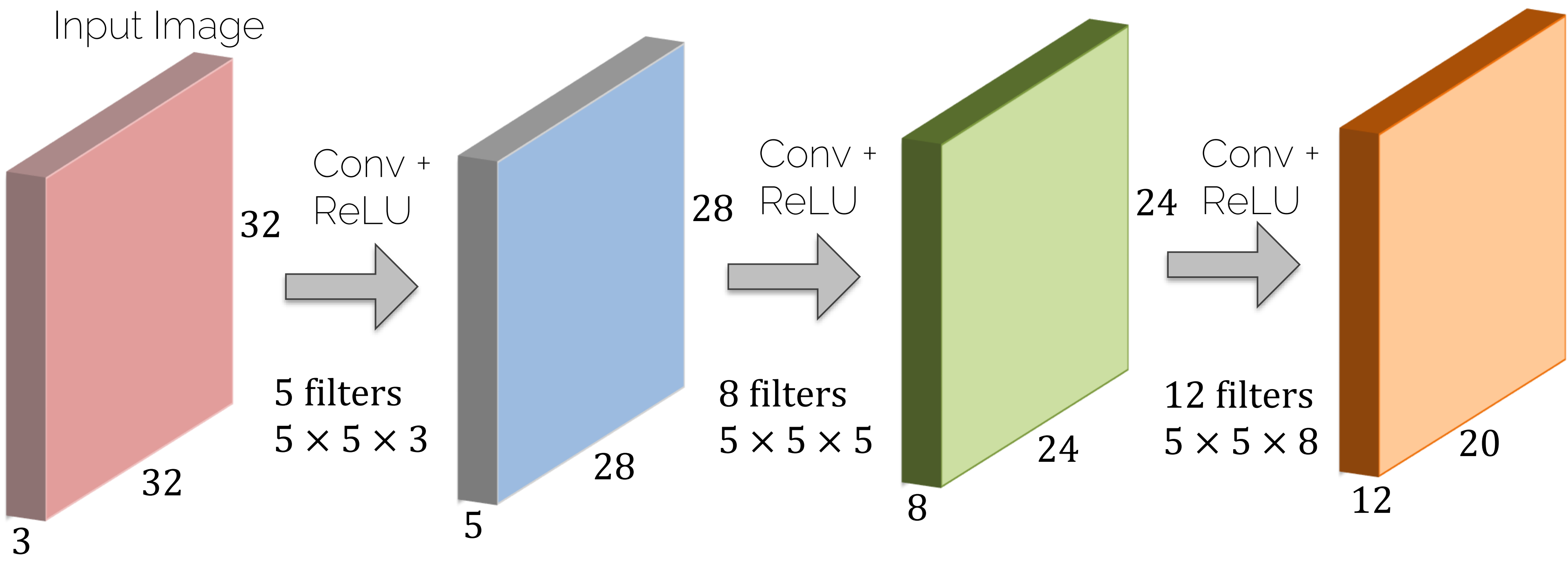




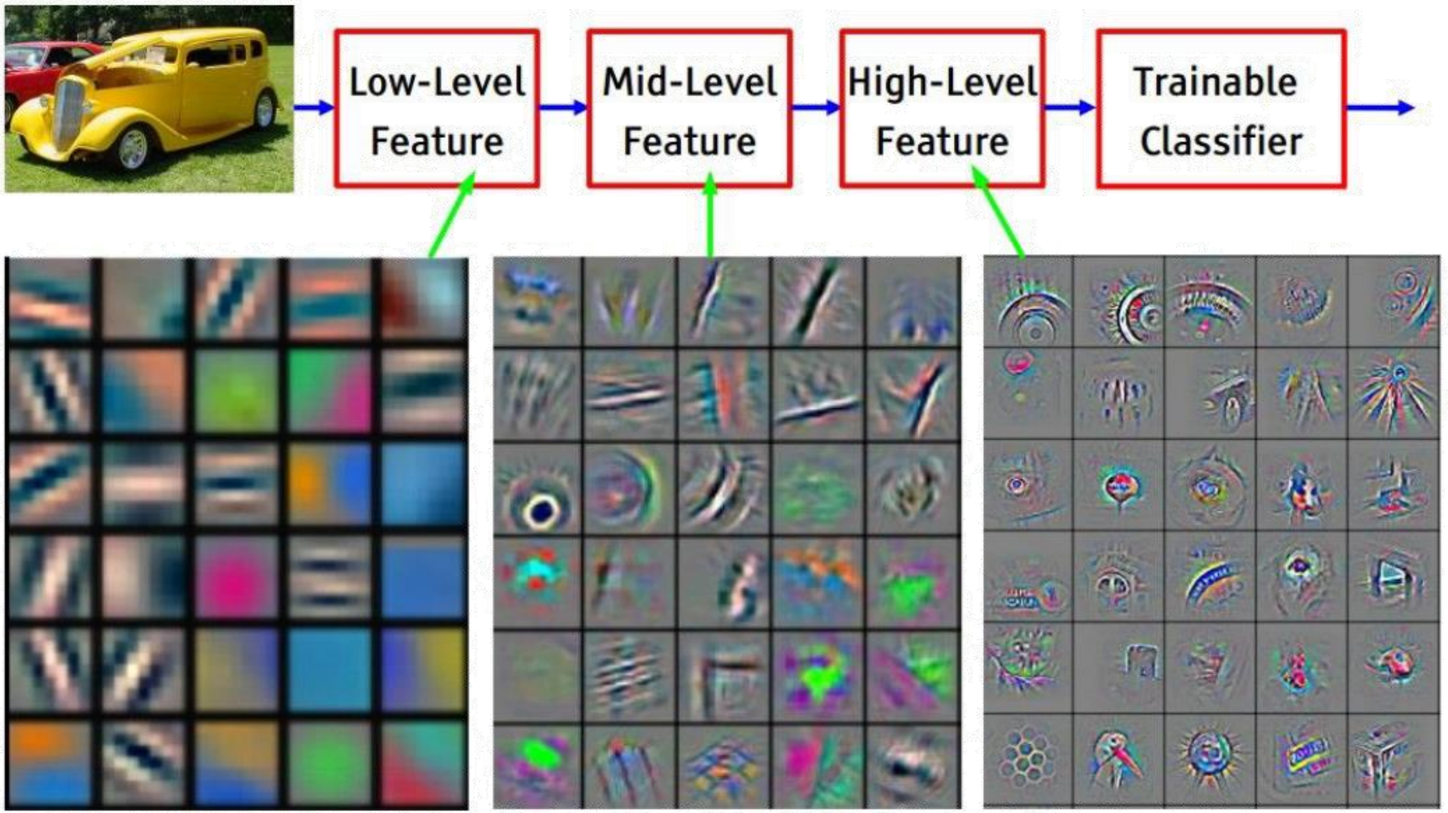
Let's apply **five** filters,
each with different weights!

CNN Prototype

ConvNet is concatenation of Conv Layers and activations



CNN learned filters



Feature visualization of convolutional net trained on ImageNet from [Zeiler & Fergus 2013]

Pooling Layer: Max Pooling

Single depth slice of input

3	1	3	5
6	0	7	9
3	2	1	4
0	2	4	3

Max pool with
2×2 filters and stride 2

'Pooled' output

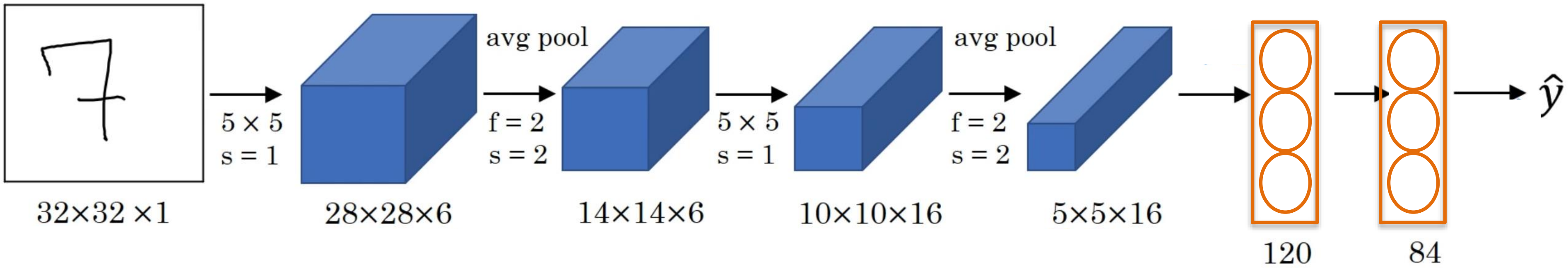
6	9
3	4

Classic CNN architectures

LeNet

- Digit recognition: 10 classes

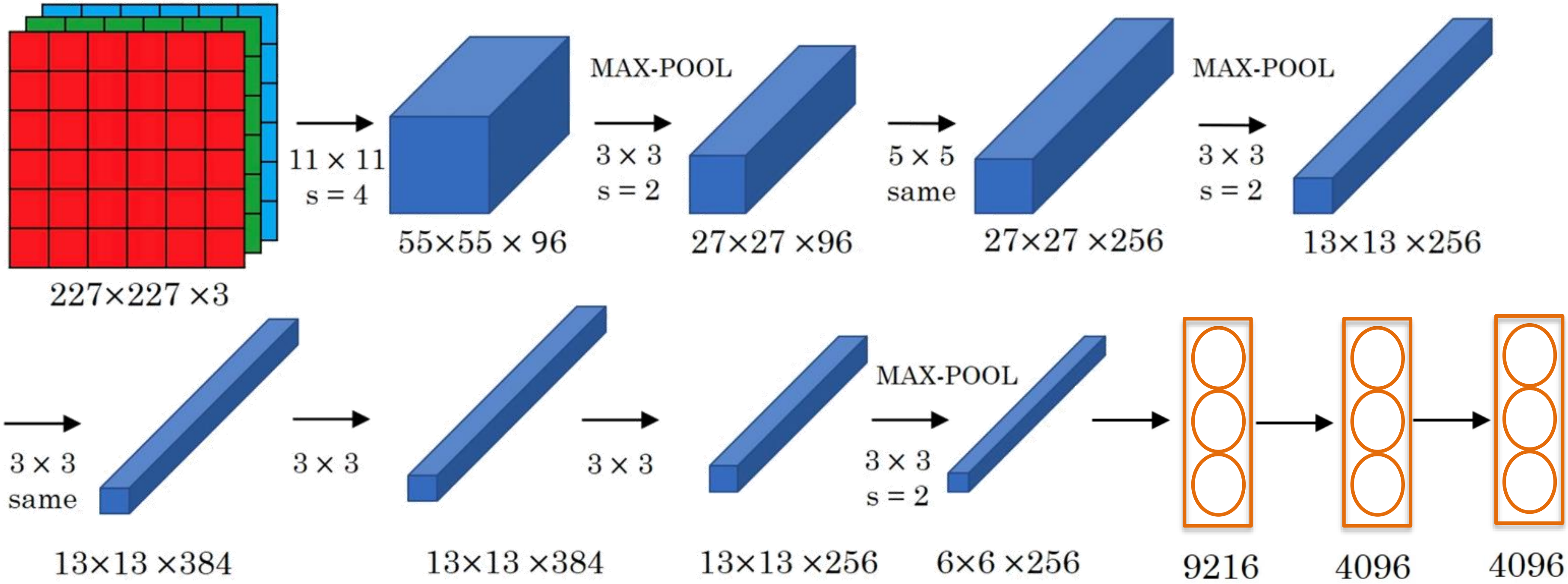
60k parameters



- Conv -> Pool -> Conv -> Pool -> Conv -> FC
- As we go deeper: Width, height ↓ Number of filters ↑

AlexNet

[Krizhevsky et al. 2012]



- Softmax for 1000 classes

VGGNet

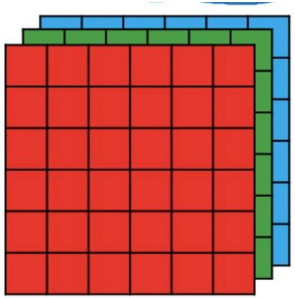
[Simonyan and Zisserman 2014]

- Striving for simplicity
- CONV = 3x3 filters with stride 1, same convolutions
- MAXPOOL = 2x2 filters with stride 2

VGGNet

Conv=3x3,s=1,same
Maxpool=2x2,s=2

Still very common: VGG-16



224×224 ×3

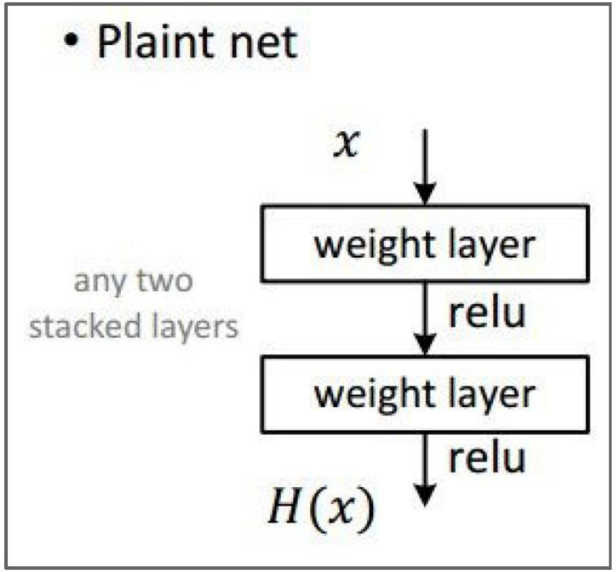
→ [CONV 64] ×2 → 224×224×64 → POOL → 112×112 ×64 → [CONV 128] ×2 → 112×112 ×128 → POOL → 56×56 ×128

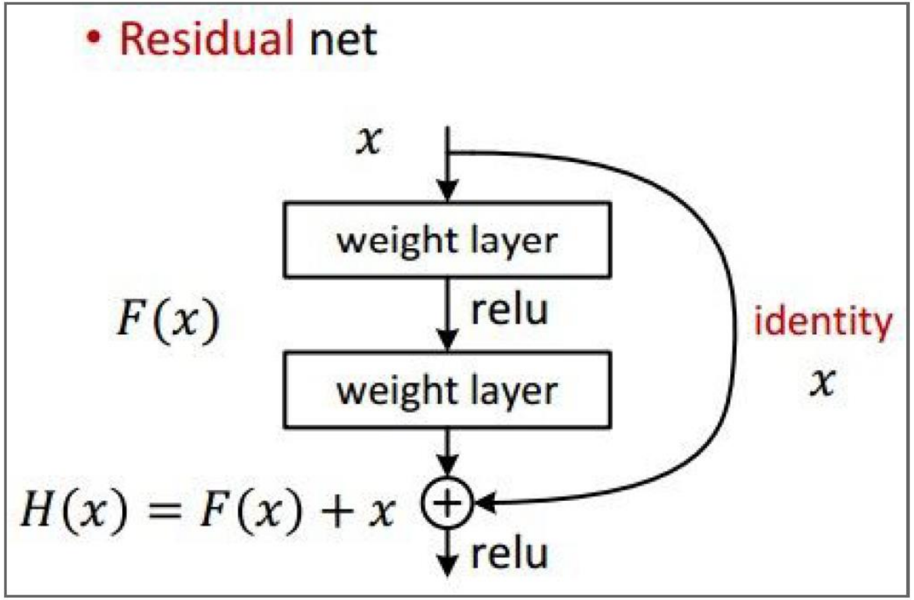
→ [CONV 256] ×3 → 56×56 ×256 → POOL → 28×28 ×256 → [CONV 512] × 3 → 28×28 ×512 → POOL → 14×14×512

→ [CONV 512] ×3 → 14×14 ×512 → POOL → 7×7×512 → FC 4096 → FC 4096 → Softmax 1000

ResNet

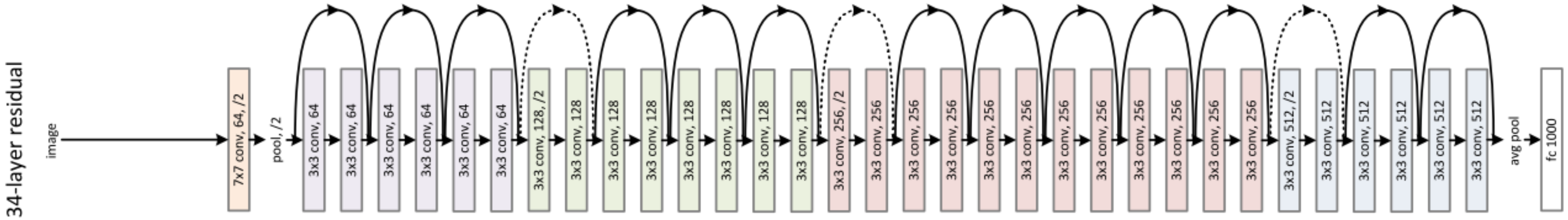
[He et al. 2015]





ResNet

[He et al. 2015]

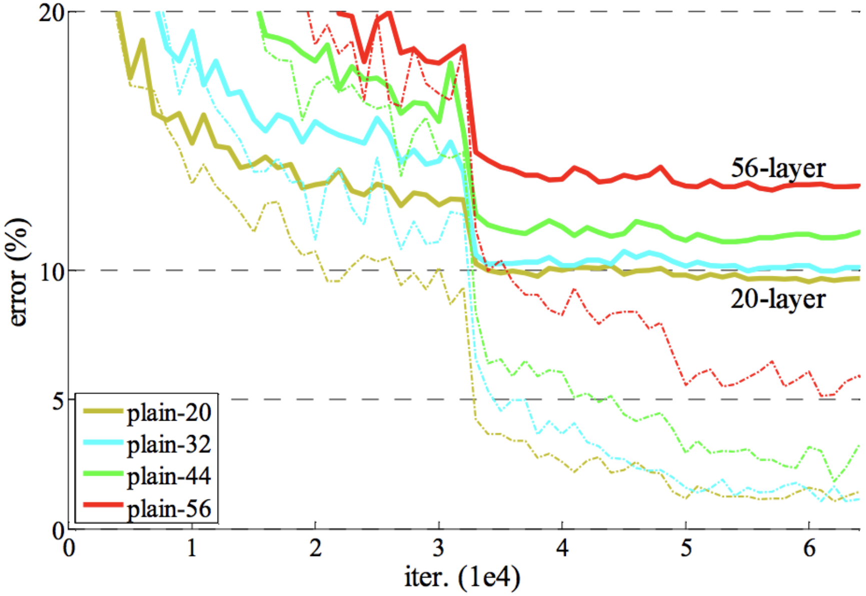


- Xavier/2 initialization
- SGD + Momentum (0.9)
- Learning rate 0.1, divided by 10 when plateau
- Mini-batch size 256
- Weight decay of 1e-5
- No dropout

ResNet

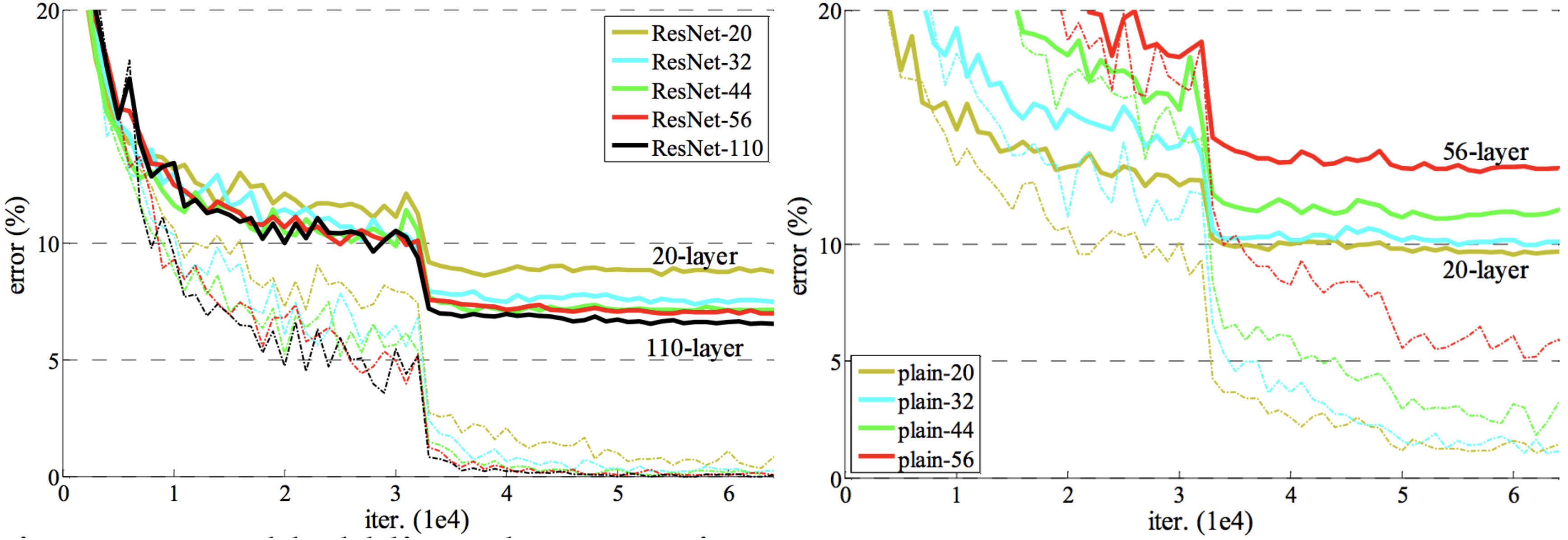
- If we make the network deeper, at some point performance starts to degrade

- Too many parameters, the optimizer cannot properly train the network

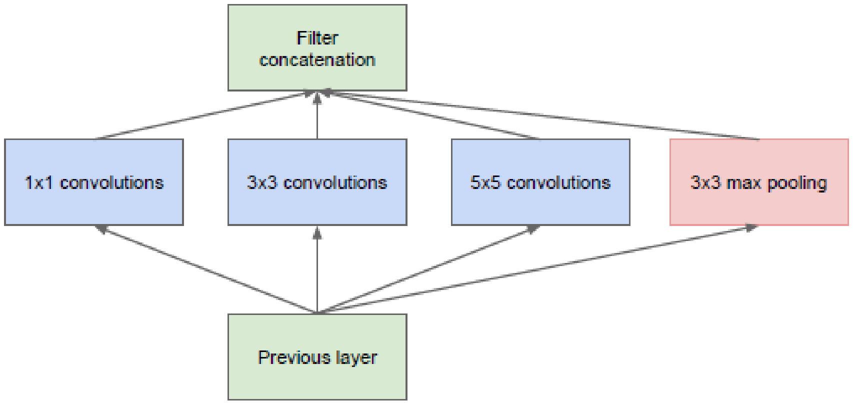


ResNet

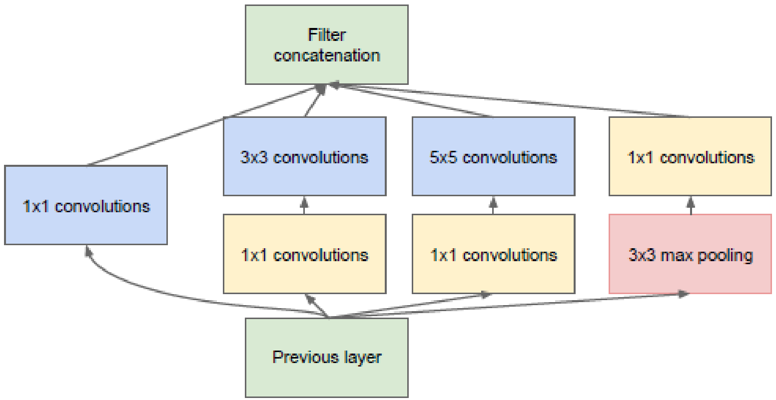
- If we make the network deeper, at some point performance starts to degrade



Inception layer



(a) Inception module, naïve version

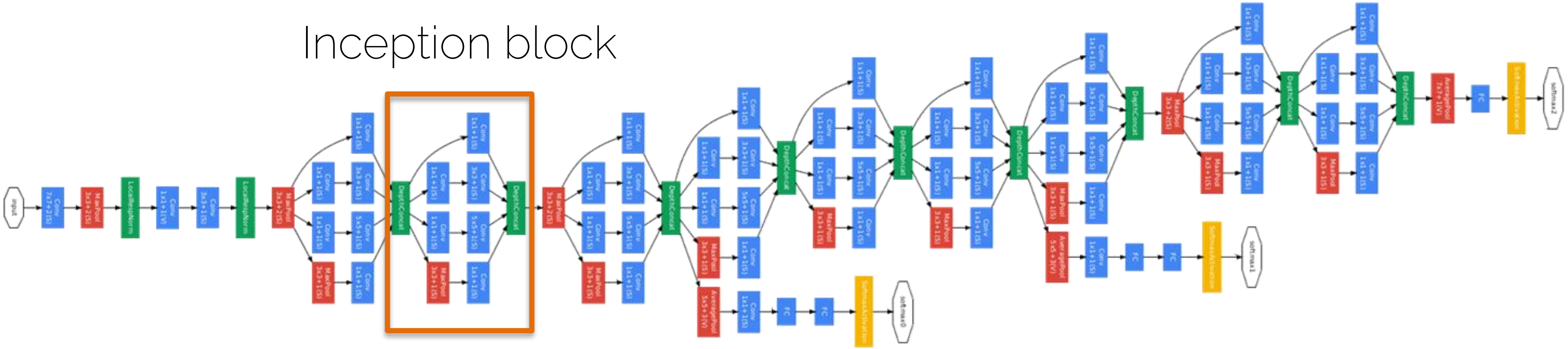


(b) Inception module with dimensionality reduction

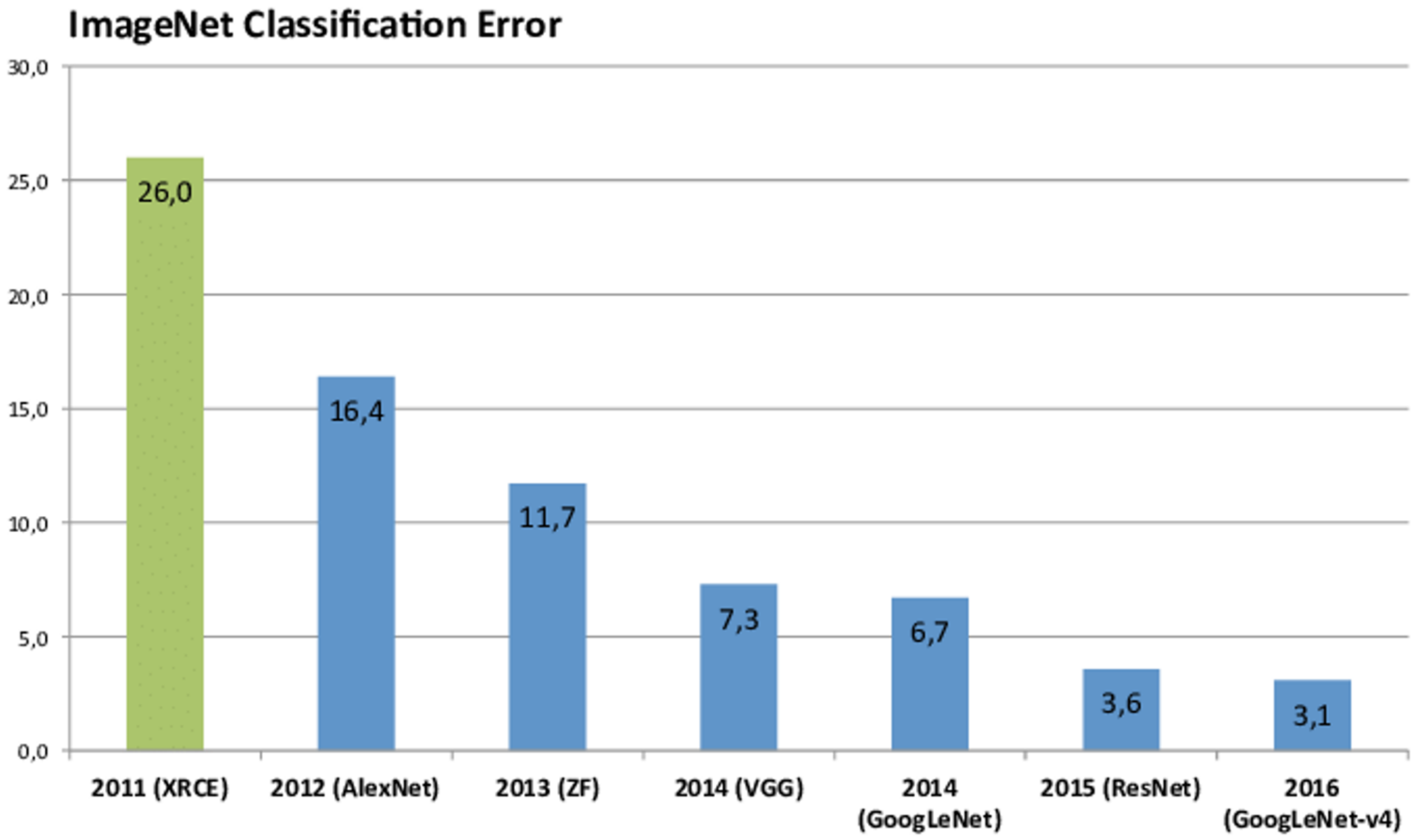
GoogLeNet: using the inception layer

[Szegedy et al. 2014]

Inception block



CNN Architectures

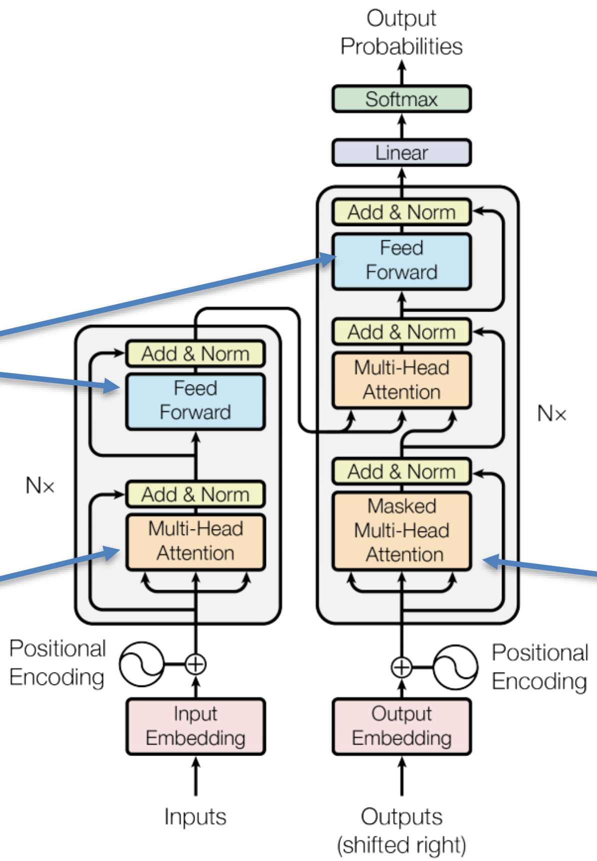


Transformers

Transformers

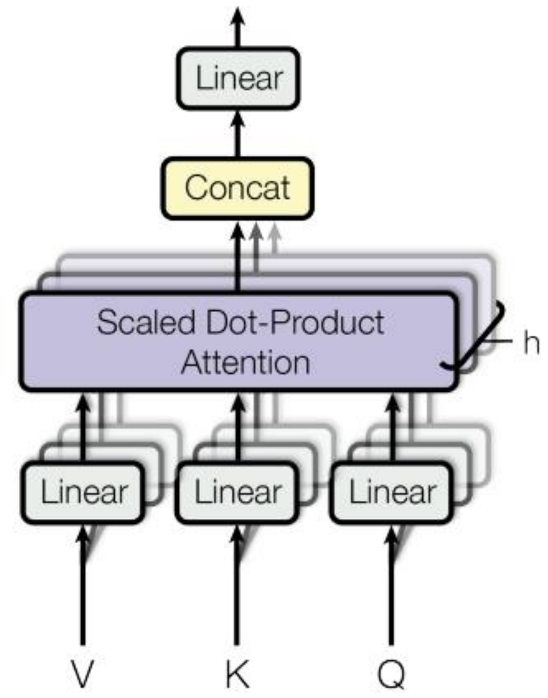
Fully connected layer

Multi-Head Attention on the "encoder"



Masked Multi-Head Attention on the "decoder"

Multi-Head Attention



Intuition: Take the query Q, find the most similar key K, and then find the value V that corresponds to the key.

In other words, learn V, K, Q where:
V – here is a bunch of interesting things.
K – here is how we can index some things.
Q – I would like to know this interesting thing.

Loosely connected to Neural Turing Machines (Graves et al.).

Multi-Head Attention

Index the values via a differentiable operator.

Multiply queries with keys

Get the values

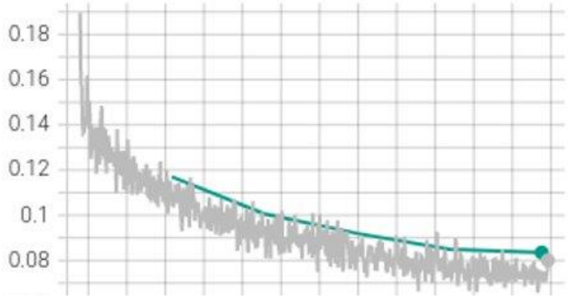
$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

To train them well, divide by $\sqrt{d_k}$, "probably" because for large values of the key's dimension, the dot product grows large in magnitude, pushing the softmax function into regions where it has extremely small gradients.

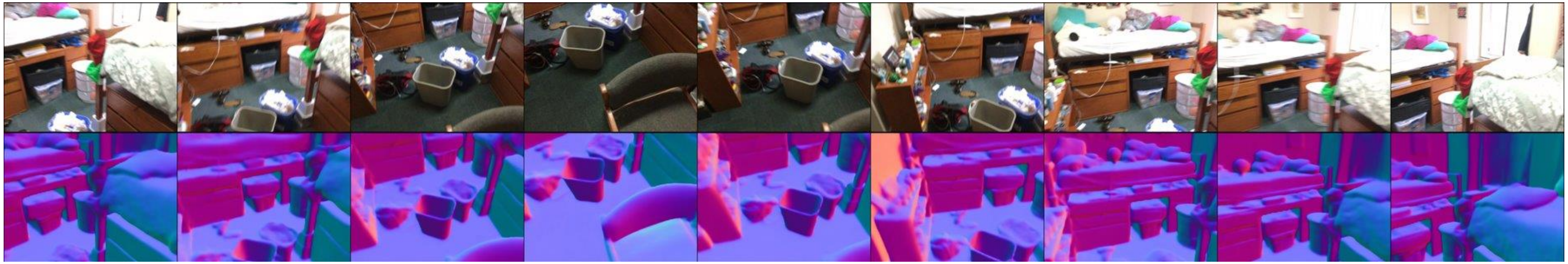
How to train your neural network?

Setup Visualizations

- Always visualize train and validation loss curves.



- Check data loading and augmentation by visualizing samples.



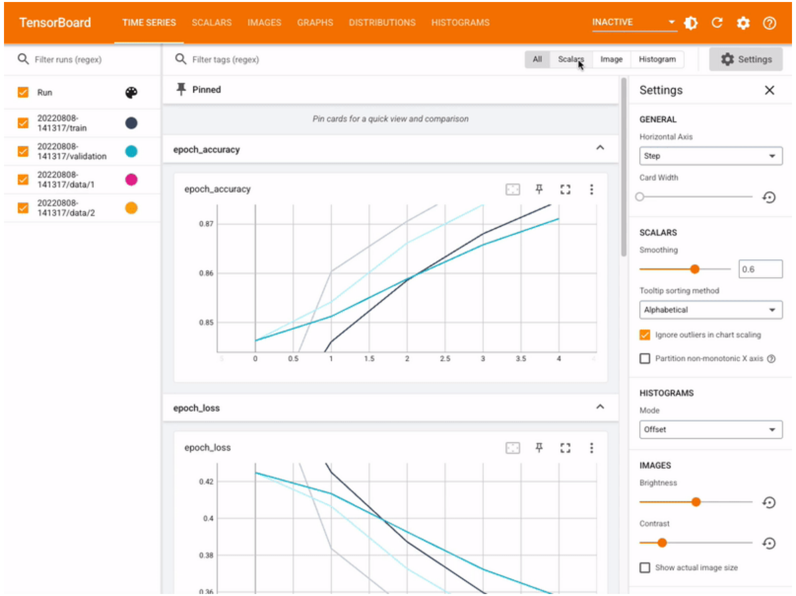
Setup Visualizations

- TensorBoard is easy to setup

  https://pytorch.org/tutorials/recipes/recipes/tensorboard_with_pytorch.html
  https://www.tensorflow.org/tensorboard/

- And provides an easy-to-use interface for visualizing image batches, metrics, histograms, videos …

  https://pytorch.org/docs/stable/tensorboard.html?highlight=summarywriter#

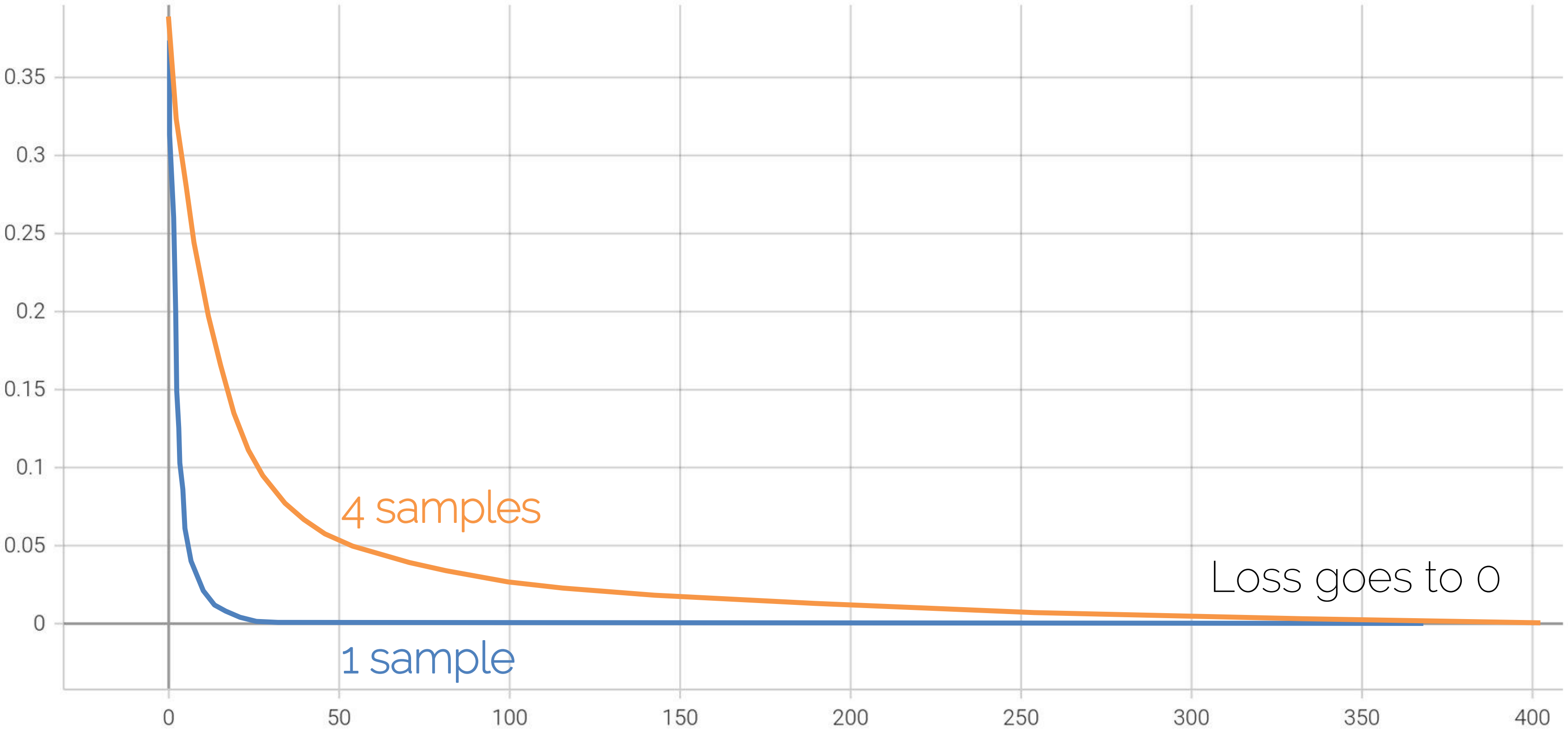


Is data loading correct?

- Data output (target): overfit to single training sample (needs to have 100% because it just memorizes input)
  - It's irrespective of input !!!

- Data input: overfit to a handful (e.g., 4) training samples
  - It's now conditioned on input data

Overfitting curves



Debugging: overfitting -> generalization

- Move from overfitting to a hand-full of samples
  - 5, 10, 100, 1000…
  - At some point, we should see generalization

- Apply common sense: can we overfit to the current number of samples?

- Always be aware of network parameter count!

Check timings

- How long does each iteration take?
  - Get precise timings!!!
  - If an iteration takes > 500ms, things get dicey…

- Where is the bottleneck: data loading vs backprop?
  - Speed up data loading: smaller resolutions, compression, train from SSD – e.g., network training is good idea
  - Speed up backprop

- Estimate total timings: how long until you see some pattern? How long till convergence?

Network architecture

- 100% mistake so far: "let's use super big network and train for two weeks and we see where we stand." [because we desperately need those 2%...]

- Start with simplest network possible: rule of thumb divide #layers you started with by 5.

- Get debug cycles down – ideally, minutes!!!

Debugging

- Need train/val/test curves
  - Evaluation needs to be consistent!
  - Numbers need to be comparable

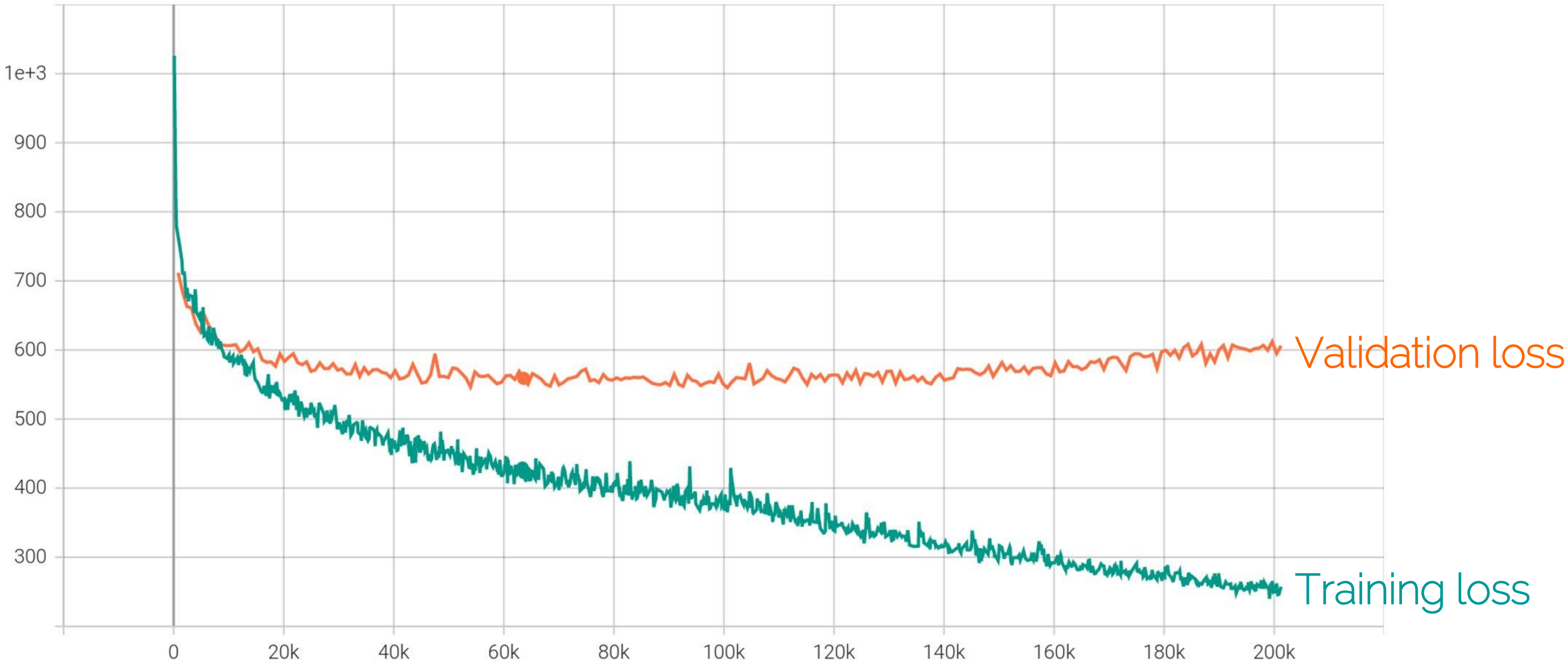
- Only make one change at a time
  - "I've added 5 more layers and double the training size, and now I also trained 5 days longer" – it's better, but WHY?

Overfitting

- ONLY THINK ABOUT THIS ONCE YOUR TRAINING LOSS GOES DOWN AND YOU CAN OVERFIT!

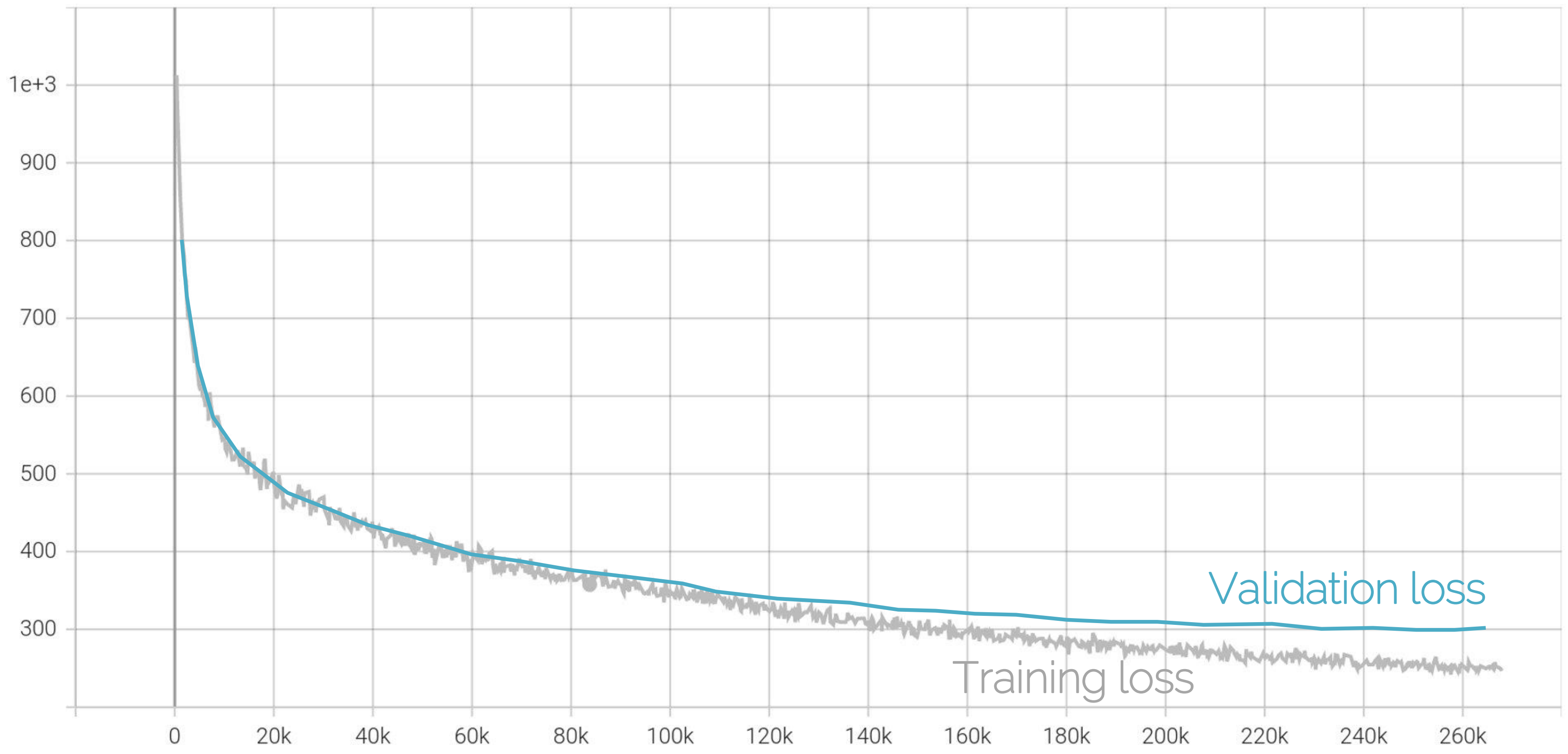
- Typically try this order:
- Network too big – makes things also faster ☺
- More regularization; e.g., weight decay
- Not enough data - makes things slower!
- Dropout - makes things slower!
- Guideline: make training harder -> generalize better

Severe overfitting!

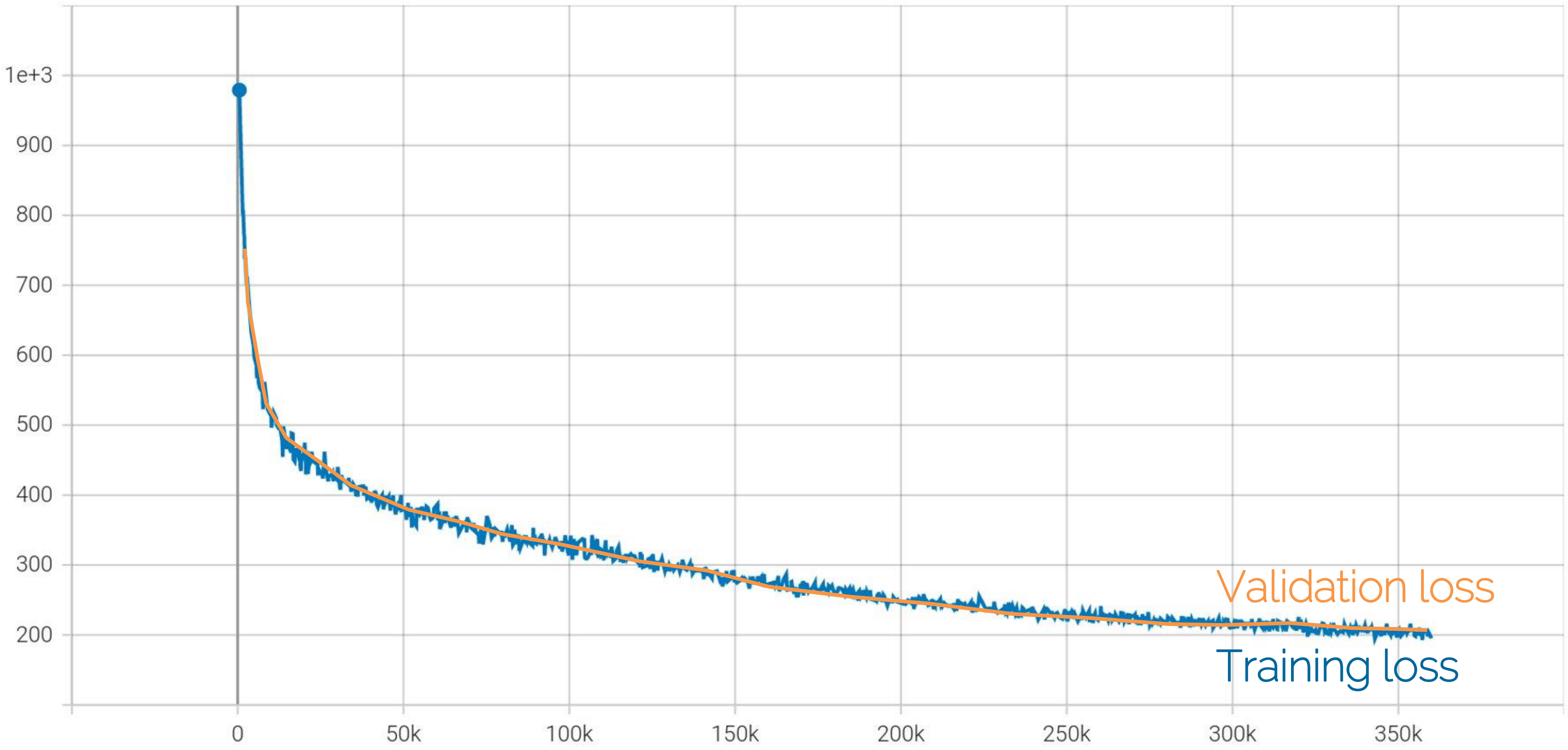


→ Try smaller network size, data augmentations, regularizations.

Moderate overfitting



No overfitting



Pushing the limits!

- PROCEED ONLY IF YOU GENERALIZE AND YOU ADDRESSED OVERFITTING ISSUES!

- Bigger network -> more capacity, more power - needs also more data!
- Better architecture -> ResNet is typically standard, but InceptionNet architectures perform often better (e.g., InceptionNet v4, XceptionNet, etc.)
- Schedules for learning rate decay
- Class-based re-weighting (e.g., give under-represented classes higher weight)
- Hyperparameter tuning: e.g., grid search; apply common sense!

Bad signs…

- Train error doesn't go down…
- Validation error doesn't go down… (ahhh we don't learn)
- Validation performs better than train… (trust me, this scenario is very unlikely – unless you have a bug ☺)
- Test on train set is different error than train… (forgot dropout?)
- Often people mess up the last batch in an epoch…

- You are training set contains test data…
- You debug your algorithm on test data…

“Most common” neural net mistakes

- you didn't try to overfit a single batch first.
- you forgot to toggle train/eval mode for the net.
- you forgot to .zero_grad() (in pytorch) before .backward().
- you passed softmaxed outputs to a loss that expects raw logits.
- you didn't use bias=False for your Linear/Conv2d layer when using BatchNorm, or conversely forget to include it for the output layer

Credit: A. Karpathy

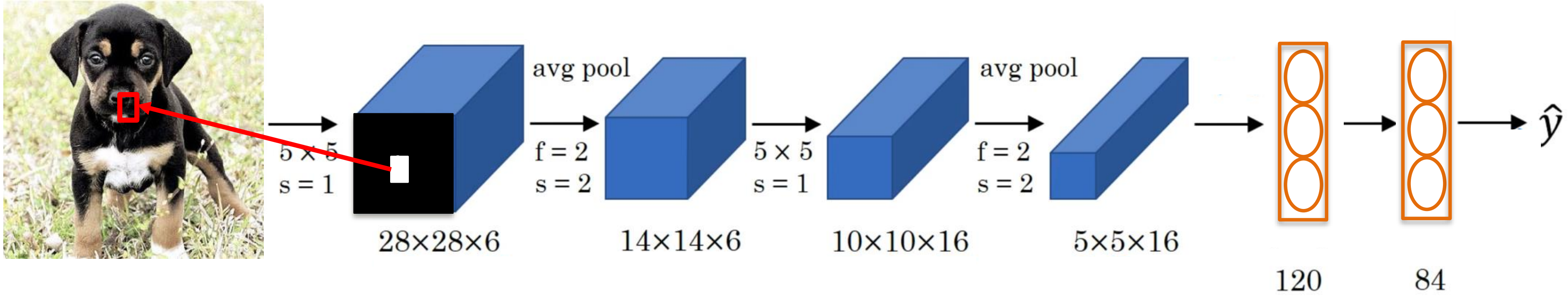
Visualization and Interpretability

Visualization of ConvNets

- Visualization in Image Space
- Visualizing Importance (Occlusion Experiment)
- T-SNE Visualization

Visualization is a great way for debugging!

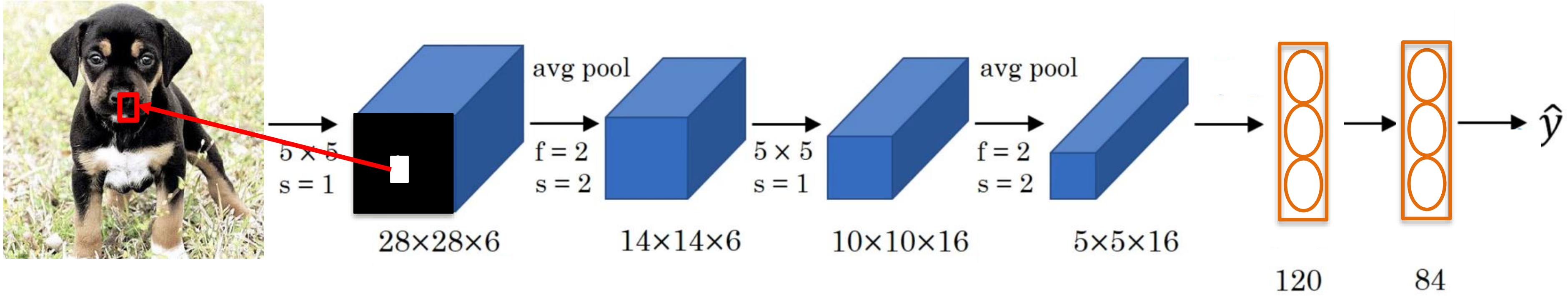
Visualizing in the image space



- Pick a unit in layer 1.
- Find the 9 image patches in your dataset that maximize the unit's activation.

Zeiler and Fergus. „Visualizing and understanding convolutional neural networks". ECCV 2014

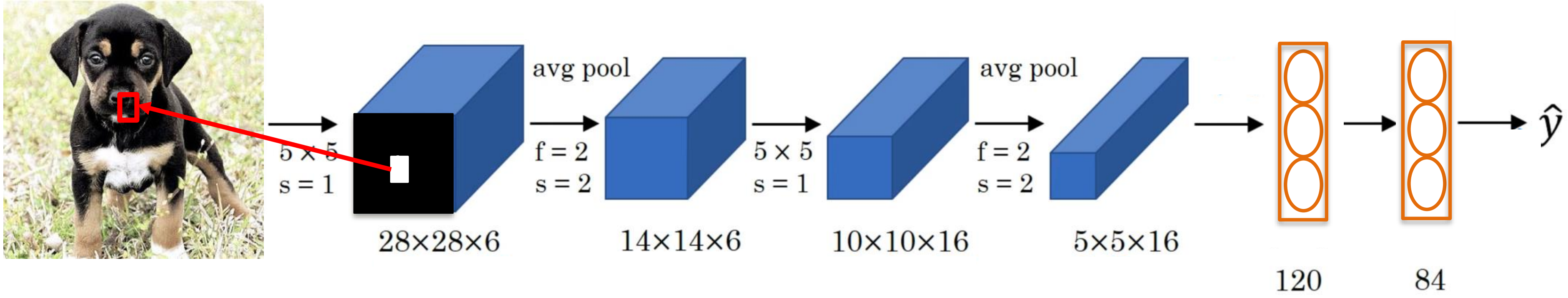
Visualizing in the image space

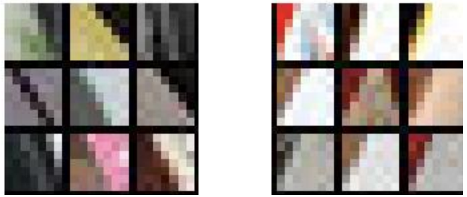


Feature map 1, layer 1, 9 image patches that provided the highest activation

Zeiler and Fergus. „Visualizing and understanding convolutional neural networks". ECCV 2014

Visualizing in the image space





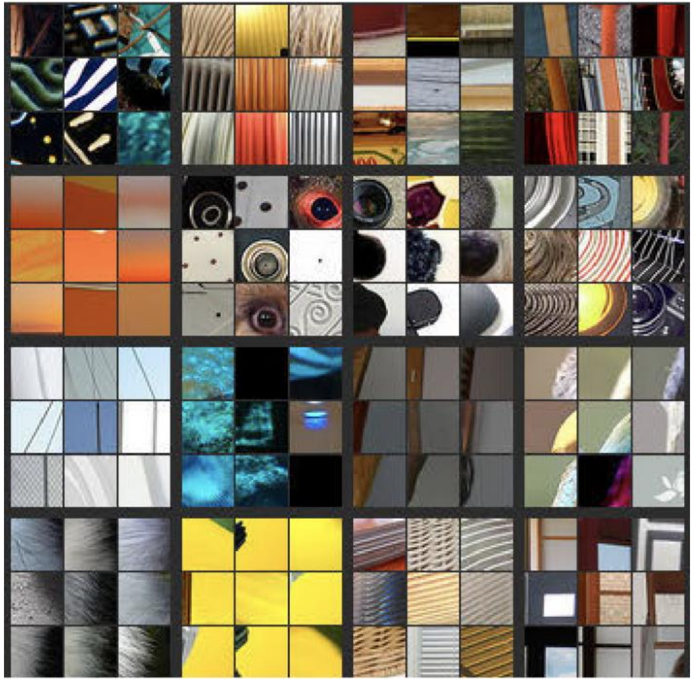
Feature map 2, layer 1, 9 image patches that provided the highest activation

Zeiler and Fergus. „Visualizing and understanding convolutional neural networks". ECCV 2014

Visualizing in the image space

Layer 1

Layer 2



Zeiler and Fergus. „Visualizing and understanding convolutional neural networks". ECCV 2014

Visualizing in the image space

Zoom in, examples of Layer 2

Zeiler and Fergus. „Visualizing and understanding convolutional neural networks". ECCV 2014

Visualizing in the image space

Zoom in, examples of Layer 5

Zeiler and Fergus. „Visualizing and understanding convolutional neural networks“. ECCV 2014

Visualizing in the image space

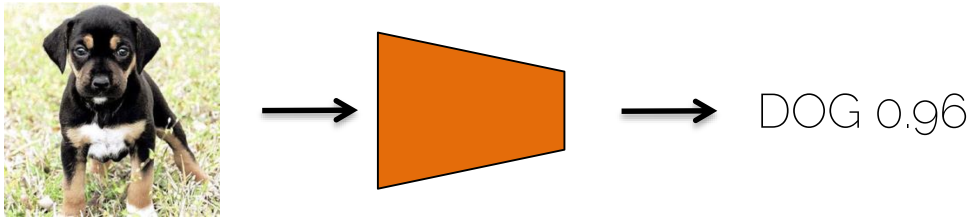
Zoom in, examples of Layer 5

Zeiler and Fergus. „Visualizing and understanding convolutional neural networks". ECCV 2014

Visualizing importance

The occlusion experiment

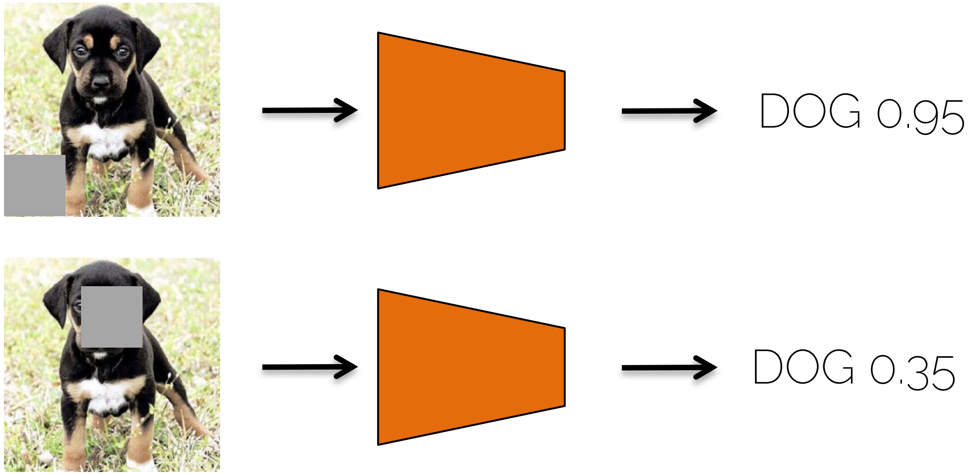
- Block different parts of the image and see how the classification score changes



Zeiler and Fergus. „Visualizing and understanding convolutional neural networks". ECCV 2014

The occlusion experiment

- Block different parts of the image and see how the classification score changes

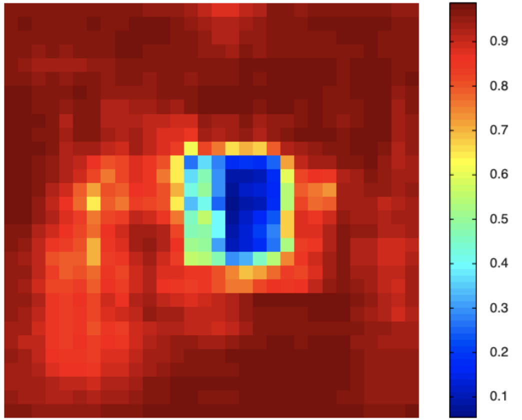


The face of the dog is more important for correct classification

Zeiler and Fergus. „Visualizing and understanding convolutional neural networks". ECCV 2014

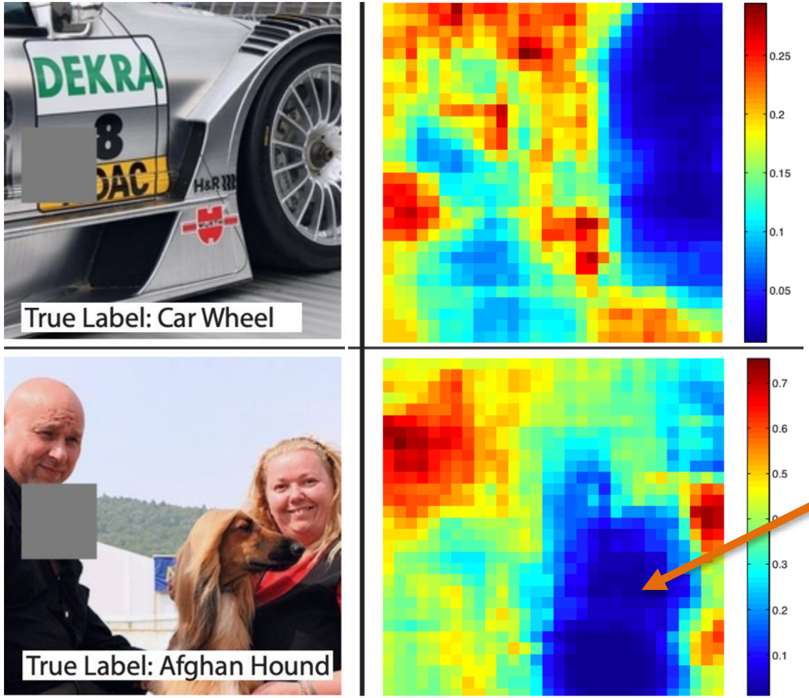
The occlusion experiment

- Create a map, where each pixel represents the classification probability if an occlusion square is placed in that region



Zeiler and Fergus. „Visualizing and understanding convolutional neural networks". ECCV 2014

The occlusion experiment



Most important pixels for classification

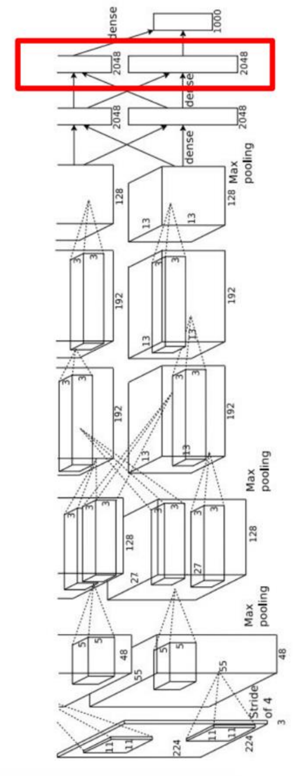
Zeiler and Fergus. „Visualizing and understanding convolutional neural networks“. ECCV 2014

t-SNE

Intuition

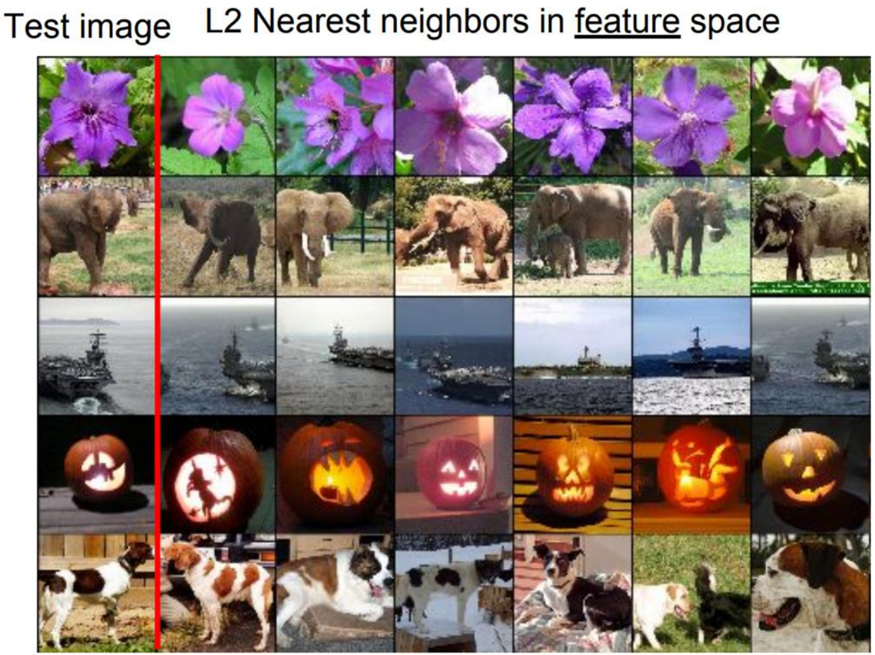
- We want to visualize the last FC layer of AlexNet which dimension 4096

- We do a forward pass of all the images and get their 4096 representations



Intuition

- Nearest neighbor visualization



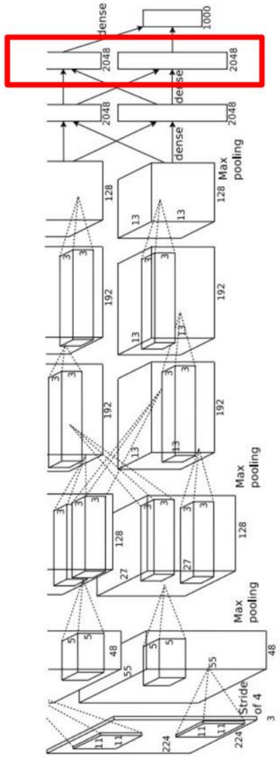


Image credit: Fei-Fei, Yeung, Johnson

Intuition

- How can I visualize these clusters in feature space?
- Map high-dimensional embedding to 2D map which preserves the pairwise distance of the points
- This mapping is done by t-SNE

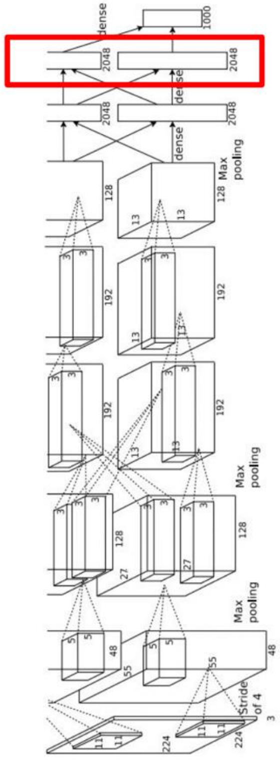
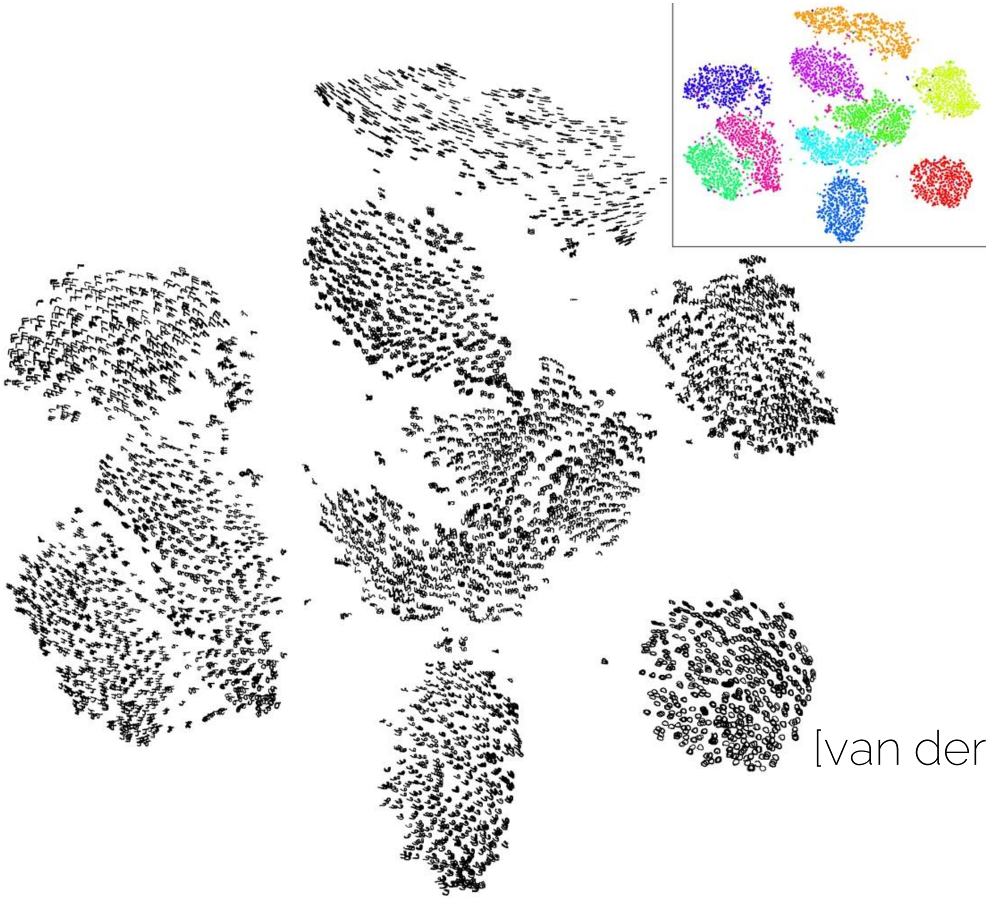


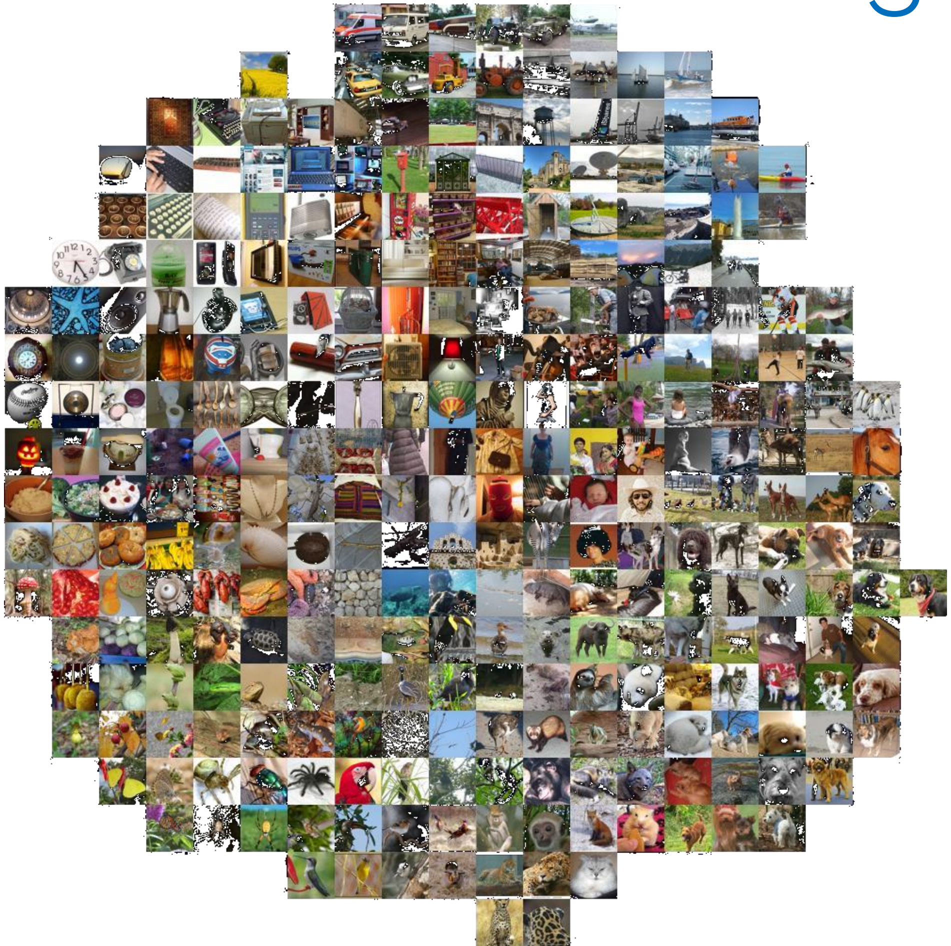
Image credit: Fei-Fei, Yeung, Johnson

t-SNE Visualization: MNIST



[van der Maaten et al.] t-SNE

t-SNE Visualization: ImageNet



t-SNE Visualization: ShapeNet

When is t-SNE worth using?

- You can use it to debug your network
- Good for visualizing the clusters created by a Siamese network

Reading Homework

- [van der Maaten et al. 08] Visualizing Data using t-SNE
  - https://www.jmlr.org/papers/volume9/vandermaaten08a/vandermaaten08a.pdf
- TensorBoard Visualization
  - https://pytorch.org/tutorials/recipes/recipes/tensorboard_with_pytorch.html
  - https://www.tensorflow.org/tensorboard/

Literature

- I2DL Lecture
  - https://niessner.github.io/I2DL/
- Latest Research
  - https://niessnerlab.org/publications.html
- Social Media
  - How to start a research project: https://twitter.com/MattNiessner/status/1441027241870118913
  - Many good feeds for latest research papers

Thanks for watching!