UC San Diego

# Lecture 3:

# Some Advanced Topics in Deep Learning

Instructor: Hao Su

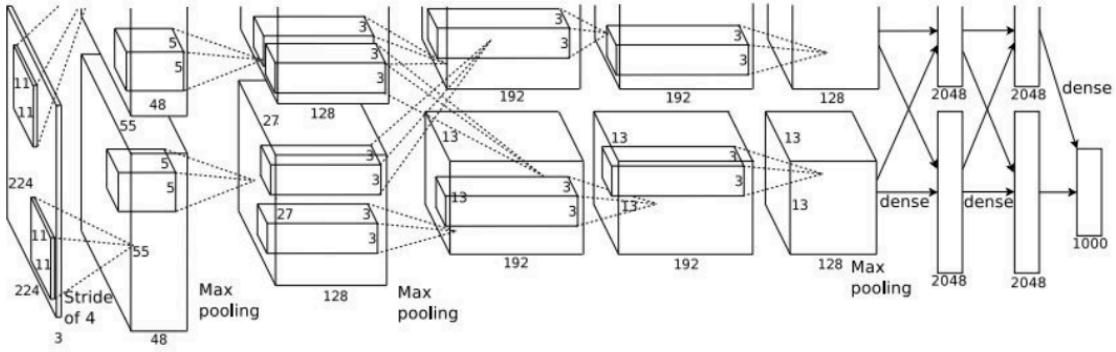
Jan 16, 2018

# Agenda

- **Network Visualization**
- Matrix Calculus and Batch Normalization
- Optimization for Networks
- Theories behind Network Generalizability

# What's going on inside ConvNets?






Class Scores:
1000 numbers

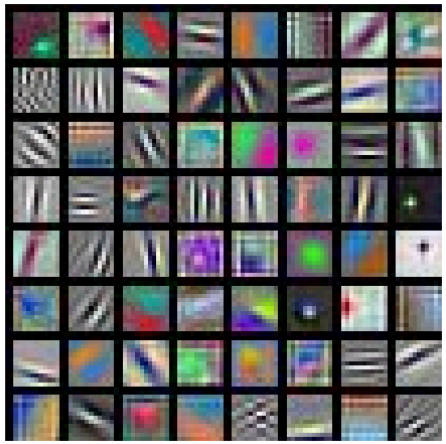
Input Image:
3 x 224 x 224

What are the intermediate features looking for?

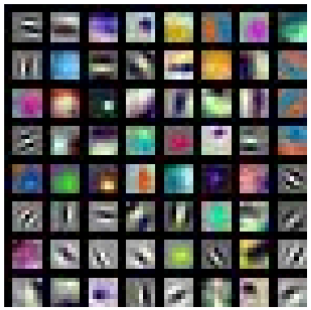
Krizhevsky et al, "ImageNet Classification with Deep Convolutional Neural Networks", NIPS 2012.
Figure reproduced with permission.

[Stanford CS231n]

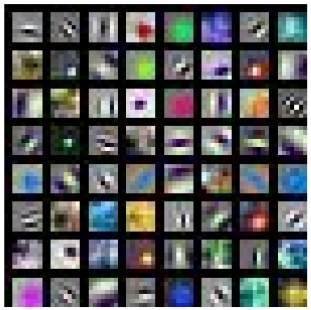
# First Layer: Visualize Filters



AlexNet:
64 x 3 x 11 x 11

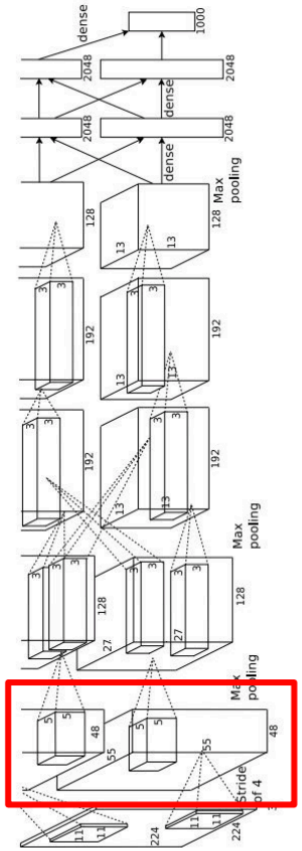


ResNet-18:
64 x 3 x 7 x 7



ResNet-101:
64 x 3 x 7 x 7

DenseNet-121:
64 x 3 x 7 x 7

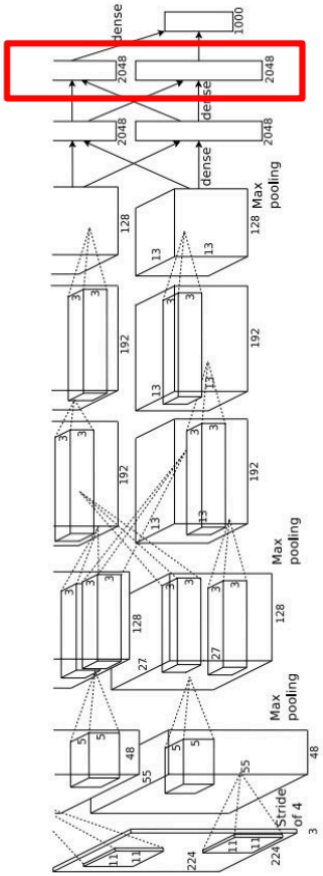



Krizhevsky, "One weird trick for parallelizing convolutional neural networks", arXiv 2014
He et al, "Deep Residual Learning for Image Recognition", CVPR 2016
Huang et al, "Densely Connected Convolutional Networks", CVPR 2017

[Stanford CS231n]

# Last Layer

FC7 layer



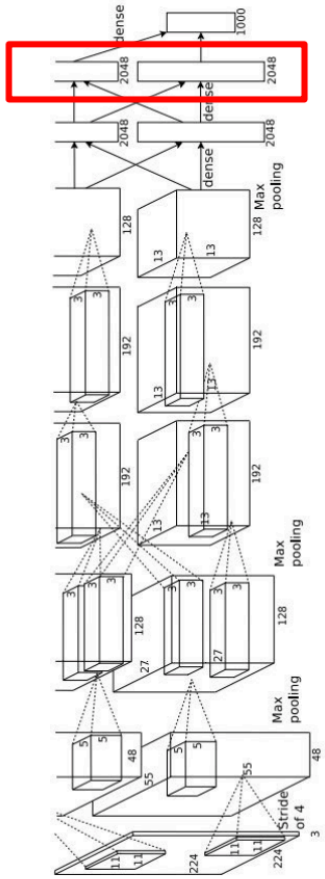

4096-dimensional feature vector for an image (layer immediately before the classifier)

Run the network on many images, collect the feature vectors

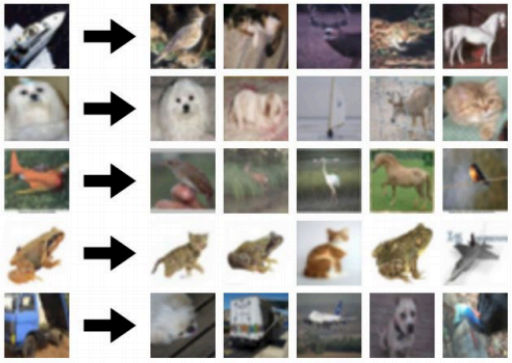
[Stanford CS231n]

# Last Layer: Nearest Neighbors

4096-dim vector

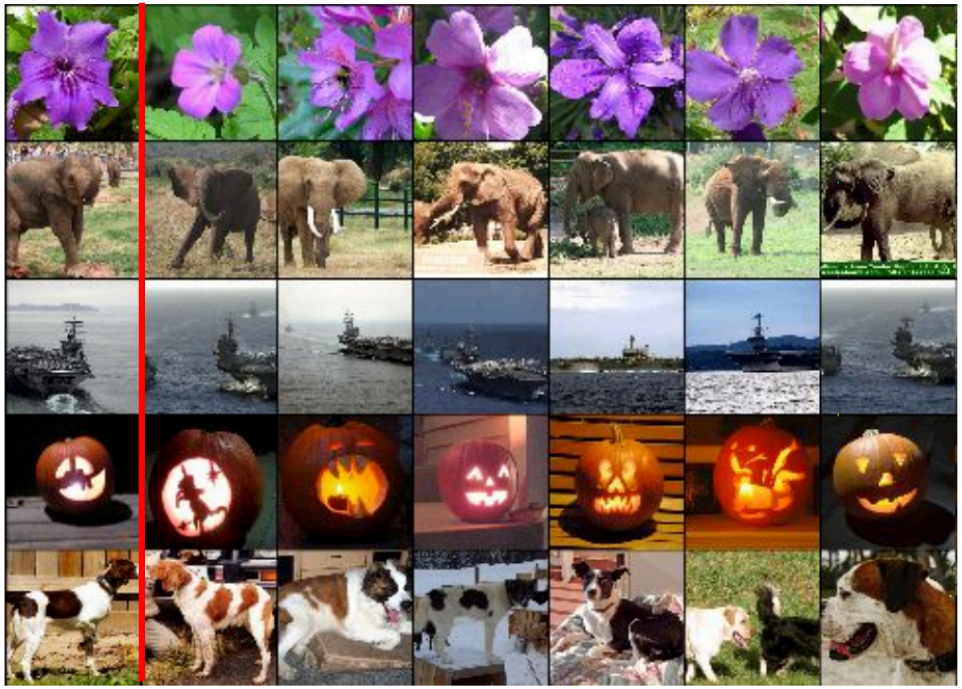



**Recall**: Nearest neighbors in pixel space



Test image

L2 Nearest neighbors in feature space



Krizhevsky et al, "ImageNet Classification with Deep Convolutional Neural Networks", NIPS 2012.
Figures reproduced with permission.

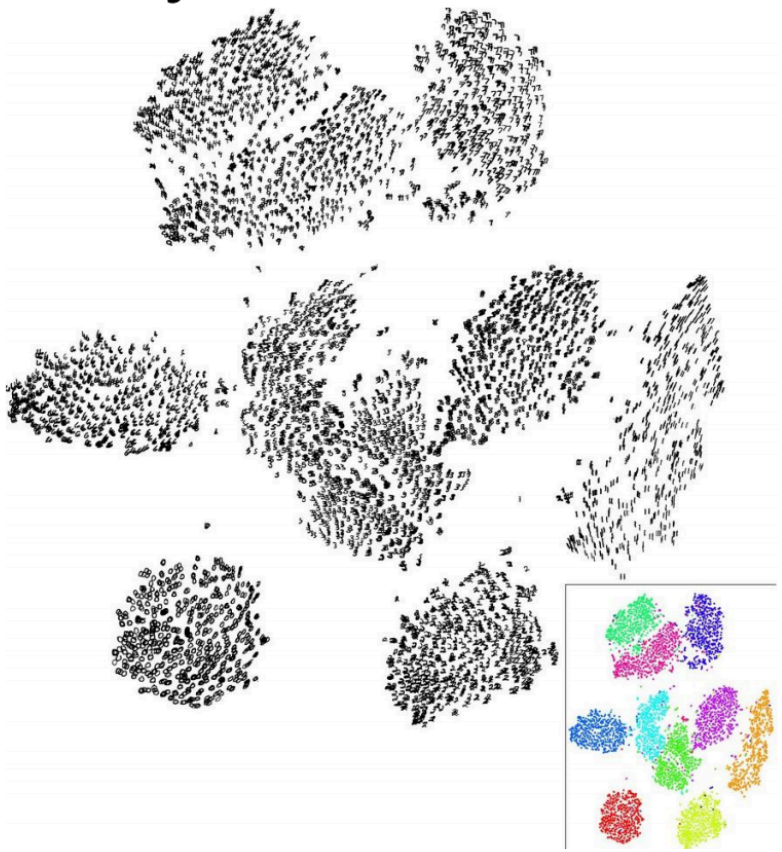
[Stanford CS231n]

# Last Layer: Dimensionality Reduction

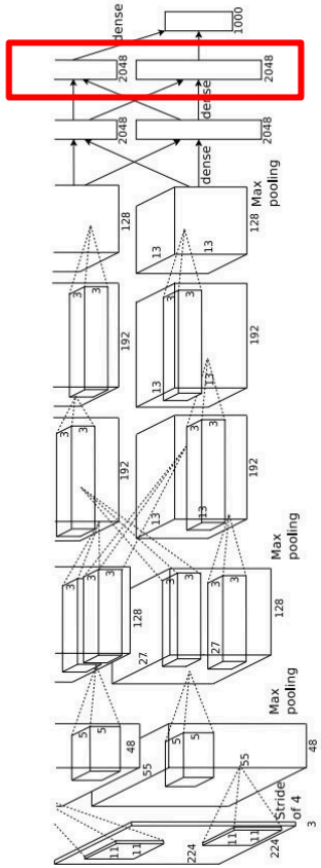
Visualize the “space” of FC7 feature vectors by reducing dimensionality of vectors from 4096 to 2 dimensions

Simple algorithm: Principle Component Analysis (PCA)

More complex: **t-SNE**



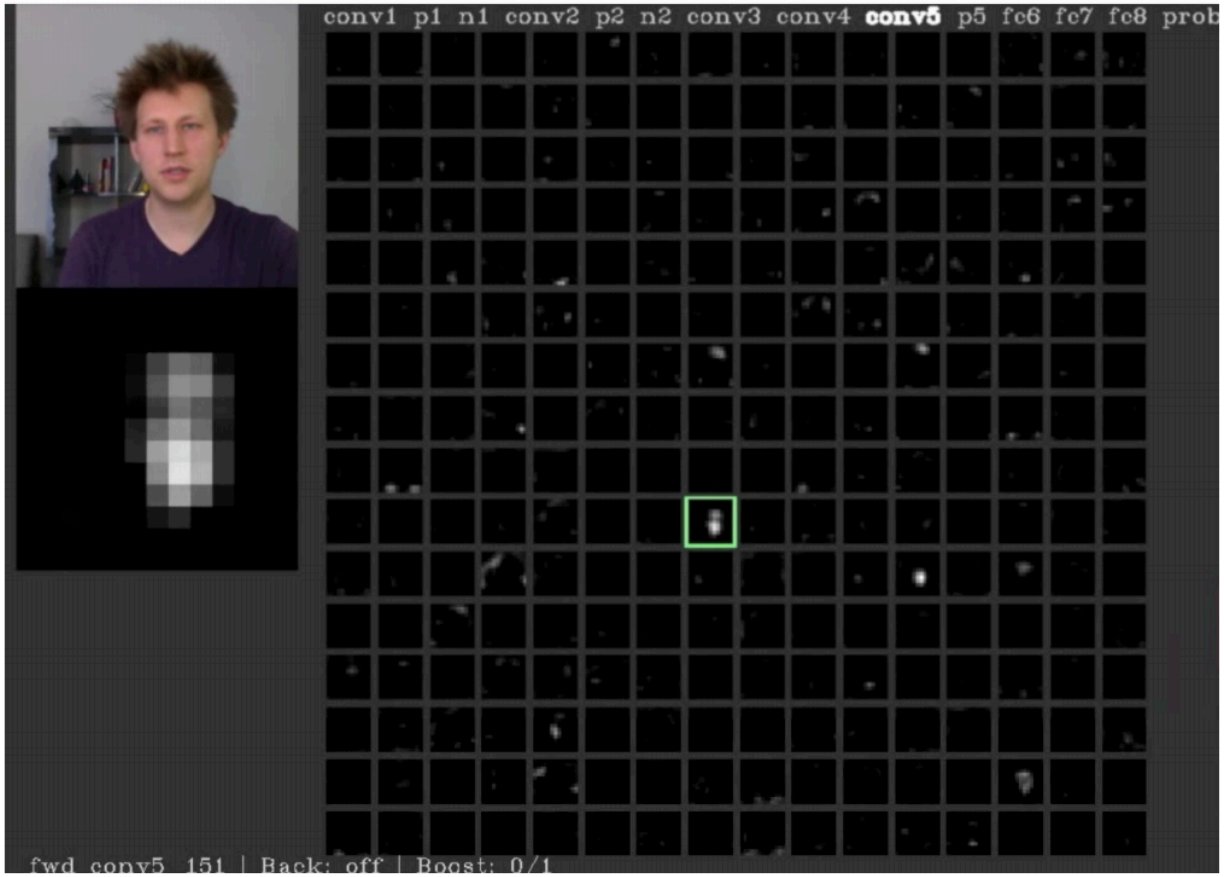
Van der Maaten and Hinton, “Visualizing Data using t-SNE”, JMLR 2008
Figure copyright Laurens van der Maaten and Geoff Hinton, 2008. Reproduced with permission.




[Stanford CS231n]

# Visualizing Activations

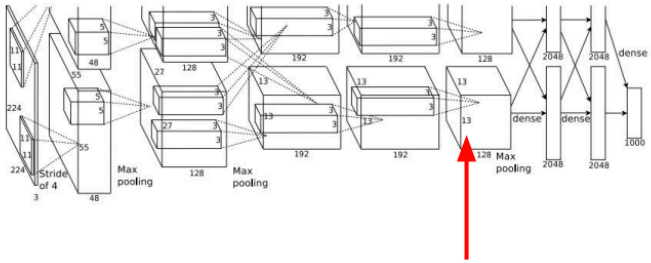
conv5 feature map is 128x13x13; visualize as 128 13x13 grayscale images




Yosinski et al, "Understanding Neural Networks Through Deep Visualization", ICML DL Workshop 2014.
Figure copyright Jason Yosinski, 2014. Reproduced with permission.

[Stanford CS231n]

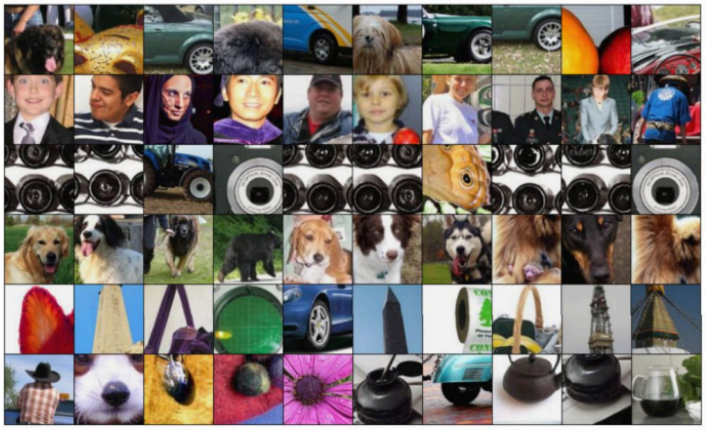
# Maximally Activating Patches




Pick a layer and a channel; e.g. conv5 is 128 x 13 x 13, pick channel 17/128

Run many images through the network, record values of chosen channel

Visualize image patches that correspond to maximal activations

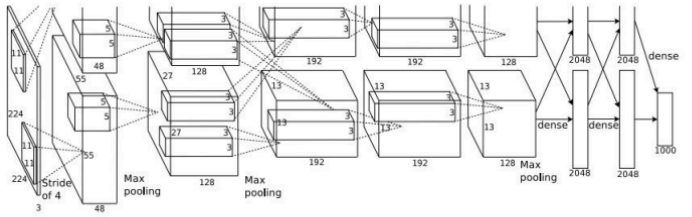


Springenberg et al, "Striving for Simplicity: The All Convolutional Net", ICLR Workshop 2015
Figure copyright Jost Tobias Springenberg, Alexey Dosovitskiy, Thomas Brox, Martin Riedmiller, 2015; reproduced with permission.

[Stanford CS231n]

# Saliency Maps

How to tell which pixels matter for classification?



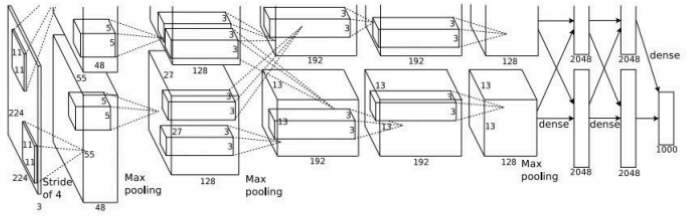

Dog

Simonyan, Vedaldi, and Zisserman, "Deep Inside Convolutional Networks: Visualising Image Classification Models and Saliency Maps", ICLR Workshop 2014.
Figures copyright Karen Simonyan, Andrea Vedaldi, and Andrew Zisserman, 2014; reproduced with permission.

[Stanford CS231n]

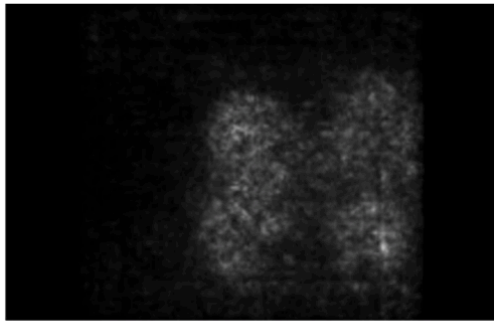
# Saliency Maps

How to tell which pixels matter for classification?




Dog

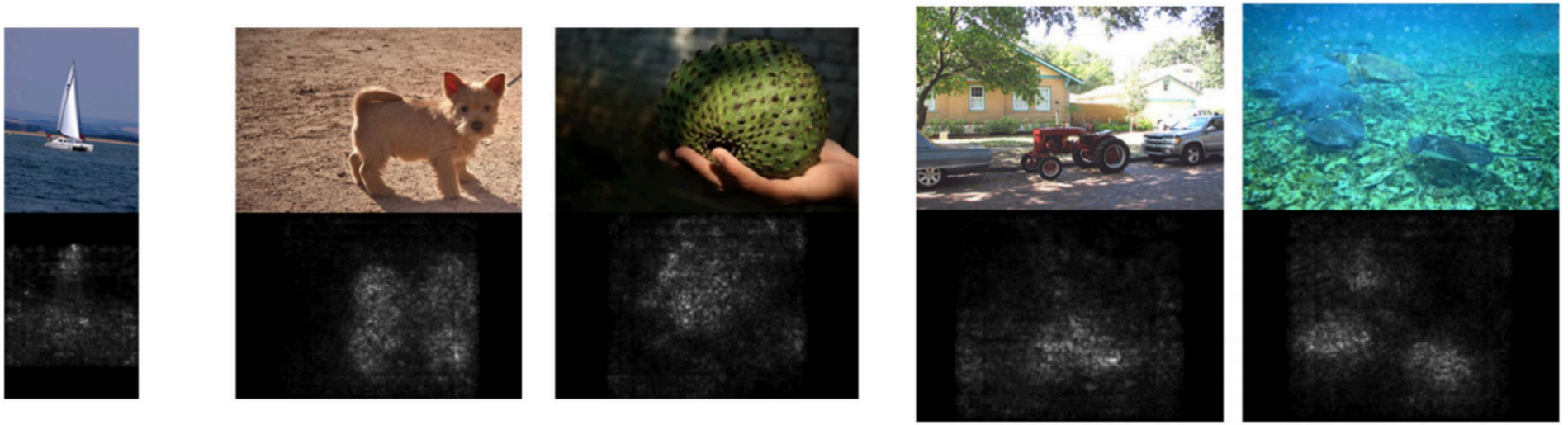
Compute gradient of (unnormalized) class score with respect to image pixels, take absolute value and max over RGB channels



Simonyan, Vedaldi, and Zisserman, "Deep Inside Convolutional Networks: Visualising Image Classification Models and Saliency Maps", ICLR Workshop 2014.
Figures copyright Karen Simonyan, Andrea Vedaldi, and Andrew Zisserman, 2014; reproduced with permission.

[Stanford CS231n]

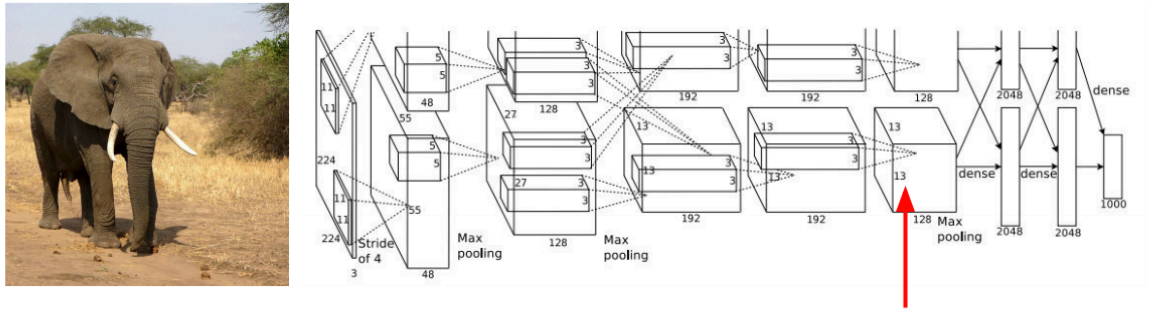
# Saliency Maps



Simonyan, Vedaldi, and Zisserman, "Deep Inside Convolutional Networks: Visualising Image Classification Models and Saliency Maps", ICLR Workshop 2014.
Figures copyright Karen Simonyan, Andrea Vedaldi, and Andrew Zisserman, 2014; reproduced with permission.

[Stanford CS231n]

# Intermediate Features via (guided) backprop




Pick a single intermediate neuron, e.g. one value in 128 x 13 x 13 conv5 feature map

Compute gradient of neuron value with respect to image pixels

Zeiler and Fergus, "Visualizing and Understanding Convolutional Networks", ECCV 2014
Springenberg et al, "Striving for Simplicity: The All Convolutional Net", ICLR Workshop 2015

[Stanford CS231n]

# Visualizing CNN features: Gradient Ascent

**(Guided) backprop**:
Find the part of an image that a neuron responds to

**Gradient ascent**:
Generate a synthetic image that maximally activates a neuron

$$I^* = \arg\max_I f(I) + R(I)$$

Neuron value

Natural image regularizer

[Stanford CS231n]

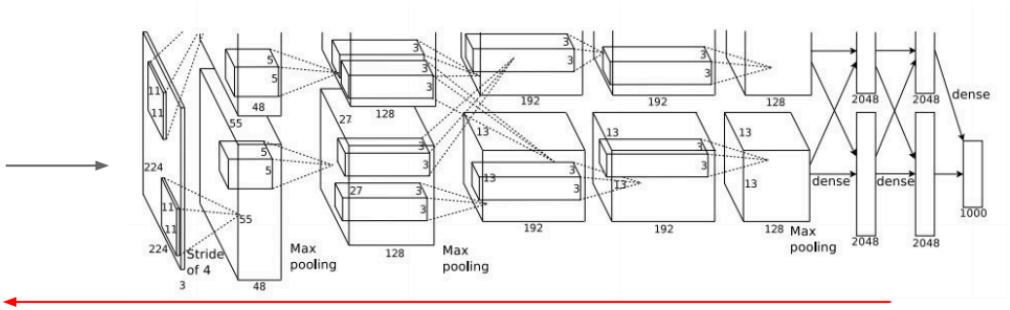
# Visualizing CNN features: Gradient Ascent

$$\arg\max_I S_c(I) - \lambda \|I\|_2^2$$

score for class c (before Softmax)

1. Initialize image to zeros

zero image




Repeat:

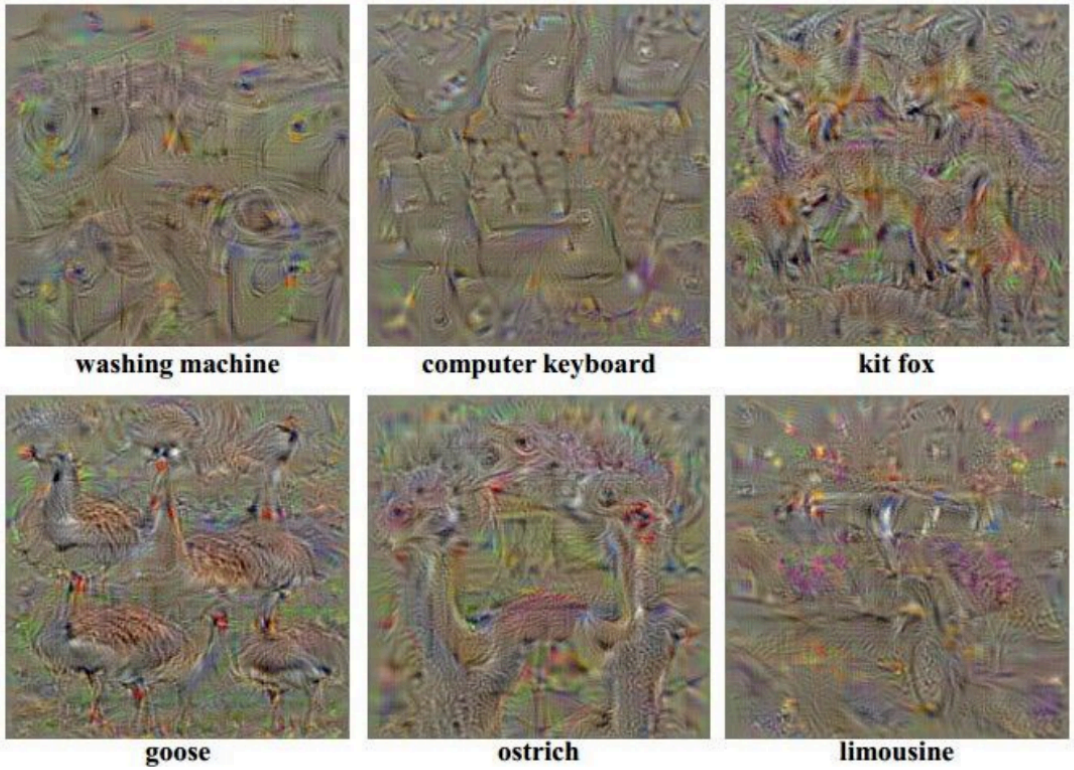
2. Forward image to compute current scores
3. Backprop to get gradient of neuron value with respect to image pixels
4. Make a small update to the image

[Stanford CS231n]

# Visualizing CNN features: Gradient Ascent

$$\arg\max_I S_c(I) - \lambda \|I\|_2^2$$

Simple regularizer: Penalize L2 norm of generated image




Yosinski et al, "Understanding Neural Networks Through Deep Visualization", ICML DL Workshop 2014. Figure copyright Jason Yosinski, Jeff Clune, Anh Nguyen, Thomas Fuchs, and Hod Lipson, 2014. Reproduced with permission.

[Stanford CS231n]

# Network Visualization

- Still an open area!

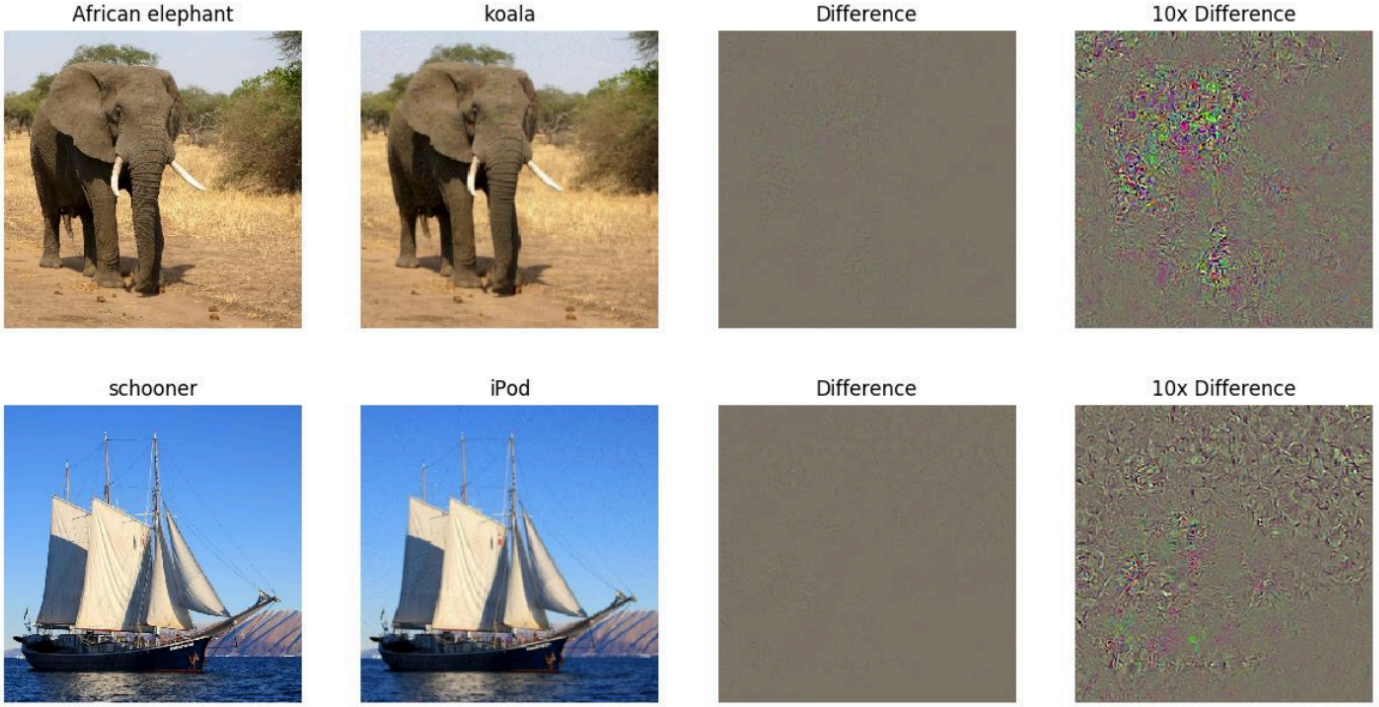
- For only for eyes, but to induce useful characteristics for network diagnosis

# Fooling Images / Adversarial Examples

(1) Start from an arbitrary image
(2) Pick an arbitrary class
(3) Modify the image to maximize the class
(4) Repeat until network is fooled

[Stanford CS231n]

# Fooling Images / Adversarial Examples




[Stanford CS231n]

# Agenda

- Network Visualization
- **Batch Normalization and Matrix Calculus**
- Optimization for Networks
- Theories behind Network Generalizability

# Batch Normalization

**Algorithm 2** Batch normalization [Ioffe and Szegedy, 2015]

**Input:** Values of $x$ over minibatch $x_1 \dots x_B$, where $x$ is a certain channel in a certain feature vector

**Output:** Normalized, scaled and shifted values $y_1 \dots y_B$

1: $\mu = \frac{1}{B}\sum_{b=1}^{B} x_b$

2: $\sigma^2 = \frac{1}{B}\sum_{b=1}^{B}(x_b - \mu)^2$

3: $\hat{x}_b = \frac{x_b - \mu}{\sqrt{\sigma^2 + \epsilon}}$

4: $y_b = \gamma \hat{x}_b + \beta$

- Accelerates training and makes initialization less sensitive
- Zero mean and unit variance feature vectors

[Stanford STATS385]

# Calculus Example

- A simple example:

$$y = W_3 W_2 W_1 x$$

$$W_3 \in \mathbb{R}^{1\times 100}, \quad W_2 \in \mathbb{R}^{100\times 100}, \quad W_1 \in \mathbb{R}^{100\times 10}, \quad x \in \mathbb{R}^{10}, \| x \| \approx 1$$

All elements sampled i.i.d from $N(0,1)$

---

$$\frac{\partial y}{\partial W_1} = ? \qquad \frac{\partial y}{\partial W_2} = ? \qquad \frac{\partial y}{\partial W_3} = ?$$

Hint: $tr(ABC) = tr(CAB) = tr(BCA)$

$$\nabla_A \mathrm{tr} AB = B^T$$

# Calculus Example

- A simple example:

$$y = W_3 W_2 W_1 x$$

$$W_3 \in \mathbb{R}^{1\times 100},\quad W_2 \in \mathbb{R}^{100\times 100},\quad W_1 \in \mathbb{R}^{100\times 10},\quad x \in \mathbb{R}^{10}, \| x \| \approx 1$$

All elements sampled i.i.d from $N(0,1)$

---

$$\frac{\partial y}{\partial W_1} = ? \qquad \frac{\partial y}{\partial W_2} = ? \qquad \frac{\partial y}{\partial W_3} = ?$$

$$\mathbb{E} \| W_1 \|_F^2 = ? \qquad \mathbb{E} \| W_2 \|_F^2 = ? \qquad \mathbb{E} \| W_3 \|_F^2 = ?$$

Hint: $\mathrm{Var}(X+Y) = \mathrm{Var}(X) + \mathrm{Var}(Y)$

# Calculus Example

- A simple example:

$$y = W_3 W_2 W_1 x$$

$$W_3 \in \mathbb{R}^{1\times 100}, \quad W_2 \in \mathbb{R}^{100\times 100}, \quad W_1 \in \mathbb{R}^{100\times 10}, \quad x \in \mathbb{R}^{10}, \| x \| \approx 1$$

All elements sampled i.i.d from $N(0,1)$

---

$$\frac{\partial y}{\partial W_1} = ? \qquad \frac{\partial y}{\partial W_2} = ? \qquad \frac{\partial y}{\partial W_3} = ?$$

$$\mathbb{E} \| W_1 \|_F^2 = ? \qquad \mathbb{E} \| W_2 \|_F^2 = ? \qquad \mathbb{E} \| W_3 \|_F^2 = ?$$

Let $y_1 = W_1 x, \quad y_2 = W_2 W_1 x$

$$\mathbb{E} \| y_1 \|_2^2 \leq ? \qquad \mathbb{E} \| y_2 \|_2^2 \leq ? \qquad \mathbb{E} \| y \|_2^2 \leq ?$$

Hint: $\| A \|_2 \leq \| A \|_F$

# Why Batch Normalization is Effective?

- A simple example:

$$y = W_3 s_3 W_2 s_2 W_1 s_1 x$$

$$W_3 \in \mathbb{R}^{1\times 100}, \quad W_2 \in \mathbb{R}^{100\times 100}, \quad W_1 \in \mathbb{R}^{100\times 10}, \quad x \in \mathbb{R}^{10}, \| x \| \approx 1$$

All elements sampled i.i.d from $N(0,1)$

---

$$\frac{\partial y}{\partial W_1} = ? \qquad \frac{\partial y}{\partial W_2} = ? \qquad \frac{\partial y}{\partial W_3} = ?$$

# Why Batch Normalization is Effective?

- A simple example:

$$y = W_3 s_3 W_2 s_2 W_1 s_1 x$$

$$W_3 \in \mathbb{R}^{1\times 100}, \quad W_2 \in \mathbb{R}^{100\times 100}, \quad W_1 \in \mathbb{R}^{100\times 10}, \quad x \in \mathbb{R}^{10}, \| x \| \approx 1$$

All elements sampled i.i.d from $N(0,1)$

---

$$\frac{\partial y}{\partial W_1} = ? \qquad \frac{\partial y}{\partial W_2} = ? \qquad \frac{\partial y}{\partial W_3} = ?$$

Update rule: $\Delta w^{t+1} = w^t + \eta \nabla E(w^t)$

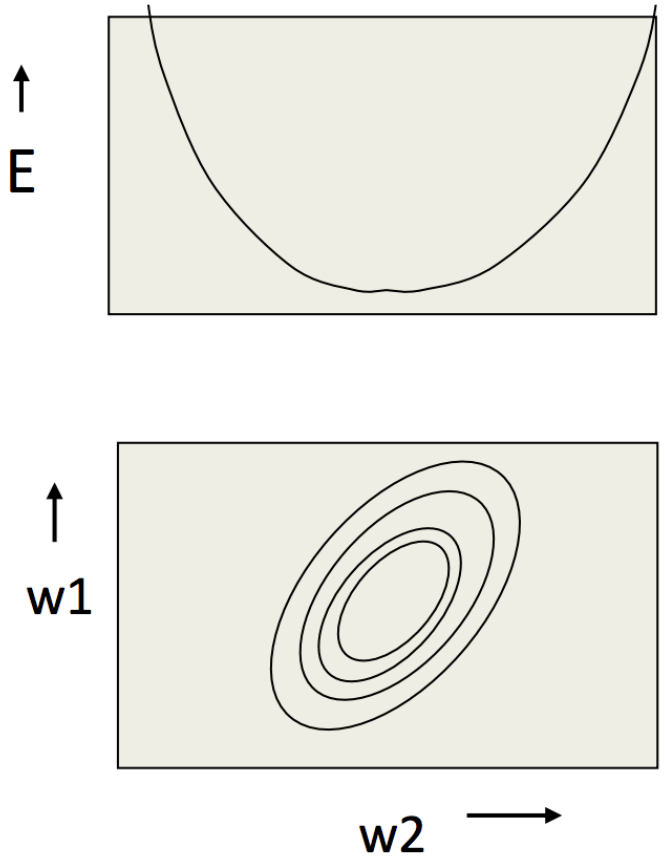
$\eta$ : step size, e.g., stage-wise constant

# Agenda

- Network Visualization
- Matrix Calculus and Batch Normalization
- **Optimization for Networks**
- Theories behind Network Generalizability

# Reminder: The error surface for a linear neuron

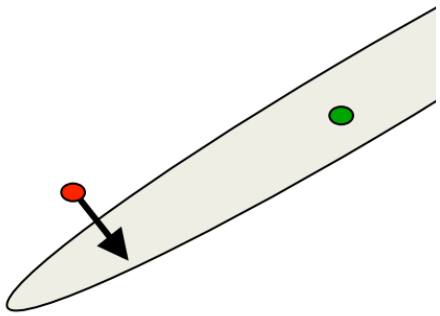
- The error surface lies in a space with a horizontal axis for each weight and one vertical axis for the error.
  - For a linear neuron with a squared error, it is a quadratic bowl.
  - Vertical cross-sections are parabolas.
  - Horizontal cross-sections are ellipses.
- For multi-layer, non-linear nets the error surface is much more complicated.
  - But locally, a piece of a quadratic bowl is usually a very good approximation.




[UToronto CSC321]

# Convergence speed of full batch learning when the error surface is a quadratic bowl

- Going downhill reduces the error, but the direction of steepest descent does not point at the minimum unless the ellipse is a circle.
  - The gradient is big in the direction in which we only want to travel a small distance.
  - The gradient is small in the direction in which we want to travel a large distance.

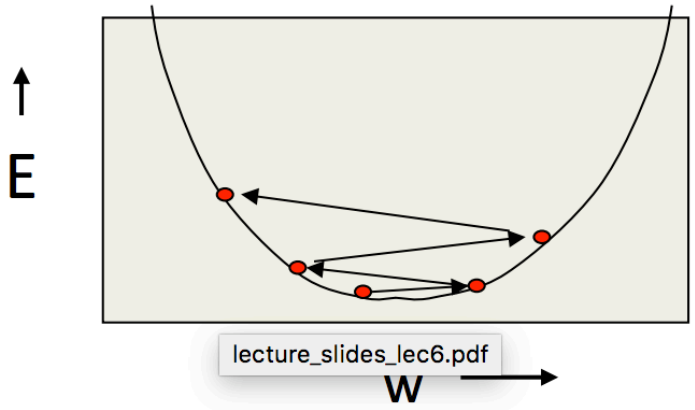


Even for non-linear multi-layer nets, the error surface is locally quadratic, so the same speed issues apply.

[UToronto CSC321]

# How the learning goes wrong

- If the learning rate is big, the weights slosh to and fro across the ravine.
  - If the learning rate is too big, this oscillation diverges.
- What we would like to achieve:
  - Move quickly in directions with small but consistent gradients.
  - Move slowly in directions with big but inconsistent gradients.




[UToronto CSC321]

# Stochastic gradient descent

- If the dataset is highly redundant, the gradient on the first half is almost identical to the gradient on the second half.
  - So instead of computing the full gradient, update the weights using the gradient on the first half and then get a gradient for the new weights on the second half.
  - The extreme version of this approach updates weights after each case. Its called "online".
- Mini-batches are usually better than online.
  - Less computation is used updating the weights.
  - Computing the gradient for many cases simultaneously uses matrix-matrix multiplies which are very efficient, especially on GPUs
- Mini-batches need to be balanced for classes

[UToronto CSC321]

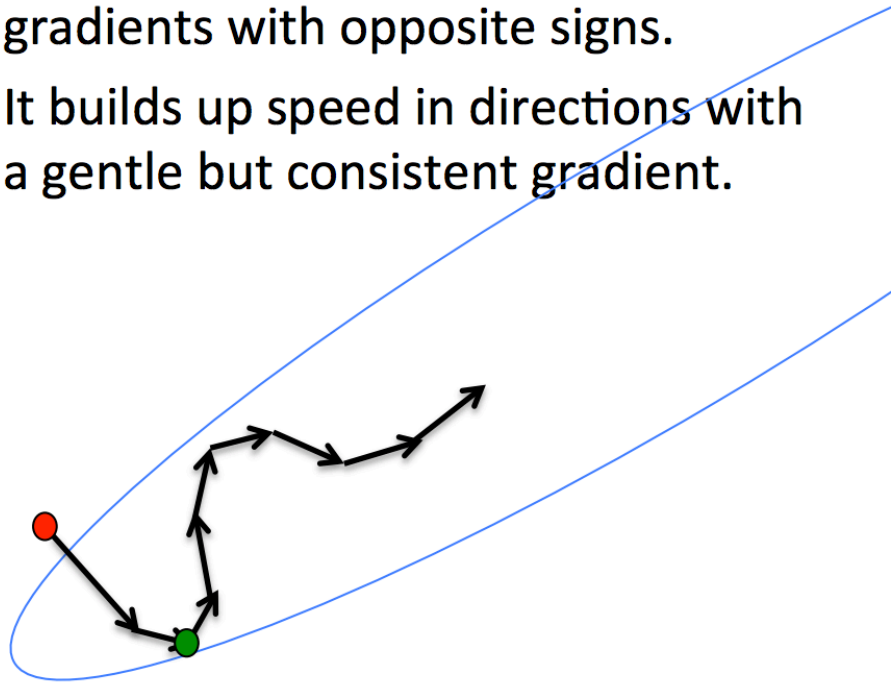
# Momentum Method

## The intuition behind the momentum method

Imagine a ball on the error surface. The location of the ball in the horizontal plane represents the weight vector.

- The ball starts off by following the gradient, but once it has velocity, it no longer does steepest descent.
- Its momentum makes it keep going in the previous direction.

- It damps oscillations in directions of high curvature by combining gradients with opposite signs.
- It builds up speed in directions with a gentle but consistent gradient.



[UToronto CSC321]

# Momentum Method

$$\mathbf{v}(t) = \alpha\, \mathbf{v}(t-1) - \varepsilon \frac{\partial E}{\partial \mathbf{w}}(t)$$

The effect of the gradient is to increment the previous velocity. The velocity also decays by $\alpha$ which is slightly less then 1.

$$\Delta \mathbf{w}(t) = \mathbf{v}(t)$$

The weight change is equal to the current velocity.

$$= \alpha\, \mathbf{v}(t-1) - \varepsilon \frac{\partial E}{\partial \mathbf{w}}(t)$$

$$= \alpha\, \Delta \mathbf{w}(t-1) - \varepsilon \frac{\partial E}{\partial \mathbf{w}}(t)$$

The weight change can be expressed in terms of the previous weight change and the current gradient.

[UToronto CSC321]

# Momentum Method

Image 2: SGD without momentum

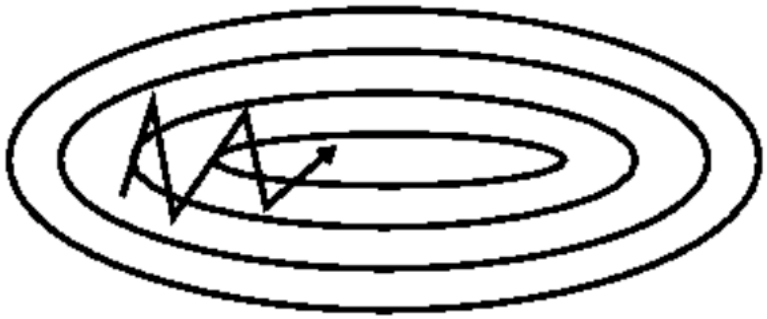


Image 3: SGD with momentum

- It leads to faster and stable convergence.
- Reduced Oscillations

[http://ruder.io/optimizing-gradient-descent/index.html#fn:15]

# ADAM: An Improved Moment Method

First order and second order moment estimation

$$m_t = \beta_1 m_{t-1} + (1-\beta_1) g_t$$

$$v_t = \beta_2 v_{t-1} + (1-\beta_2) g_t^2$$

Bias correction

$$\hat{m}_t = \frac{m_t}{1-\beta_1^t}$$

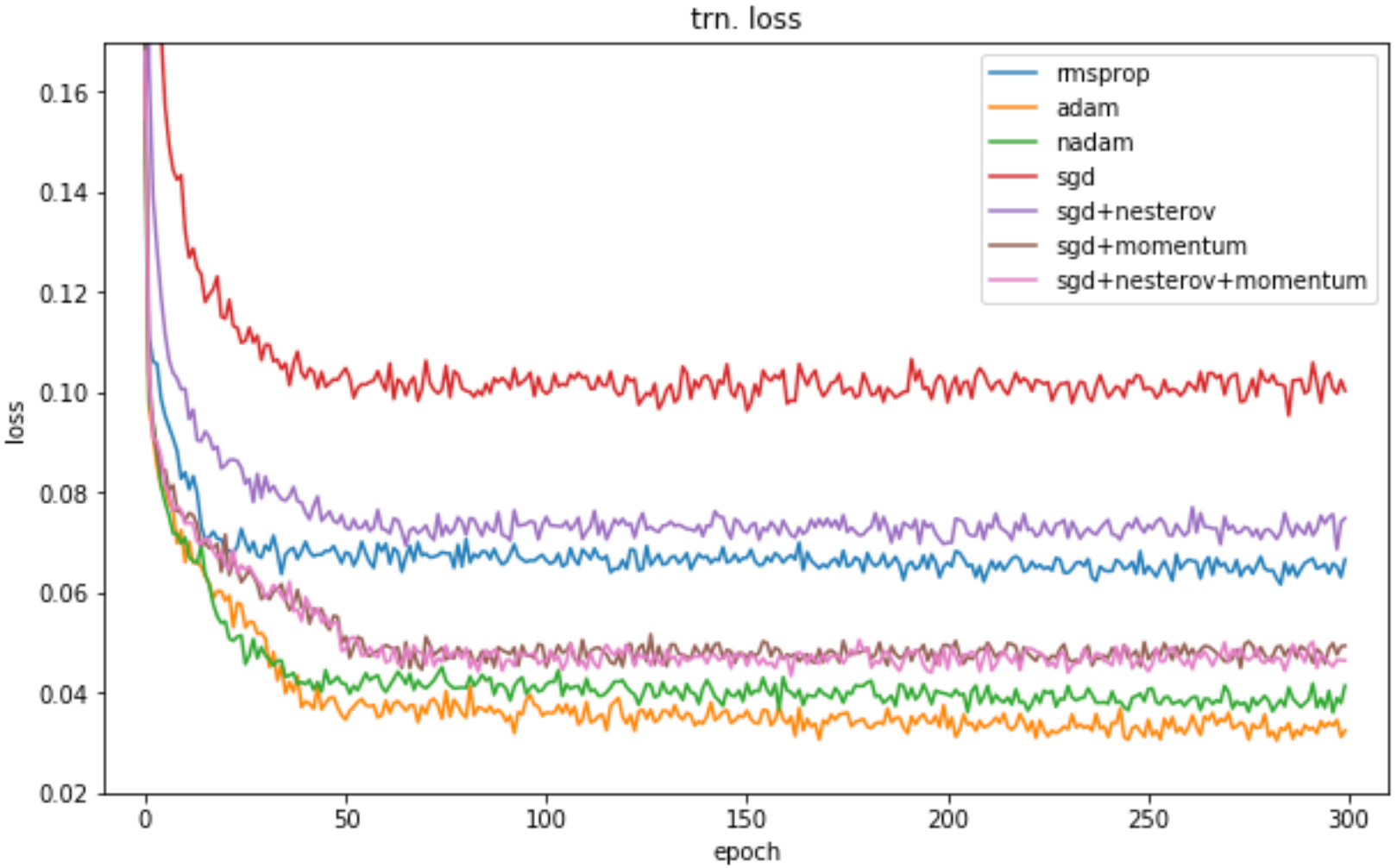
$$\hat{v}_t = \frac{v_t}{1-\beta_2^t}$$

Adam update rule:

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{\hat{v}_t} + \epsilon} \hat{m}_t$$

[UToronto CSC321]

# Learning Curve



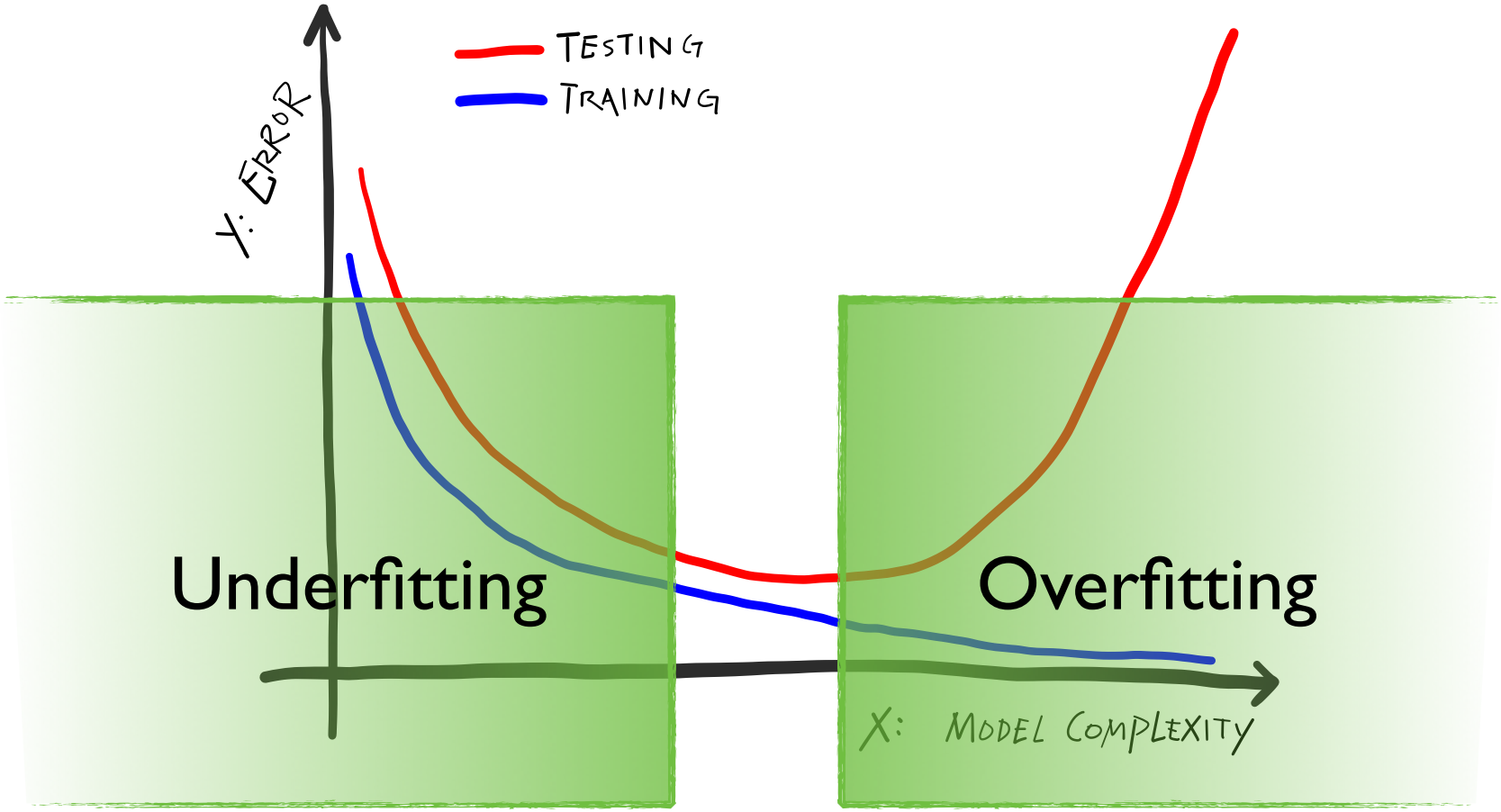

[UToronto CSC321]

# Agenda

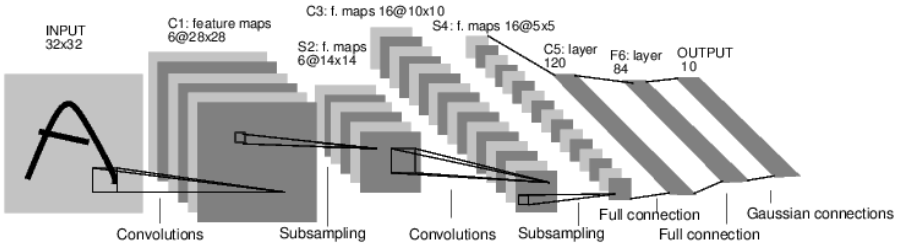
- Network Visualization
- Matrix Calculus and Batch Normalization
- Optimization for Networks
- **Analysis behind Network Generalizability**

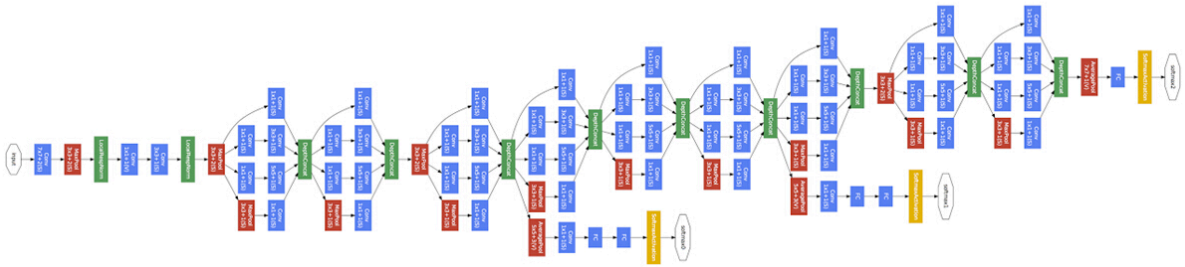
# Model Selection

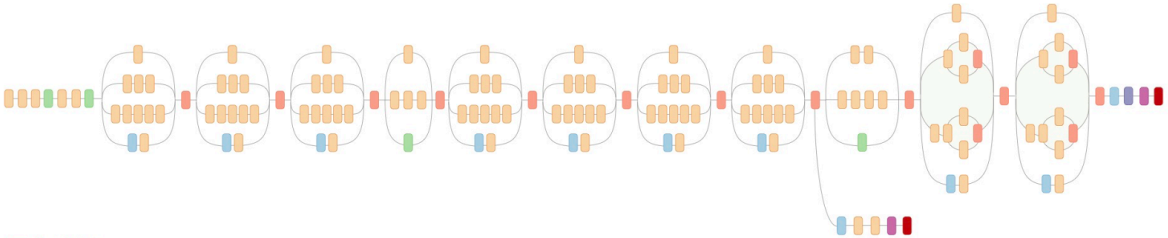
[Chiyuan Zhang, ICLR'17]

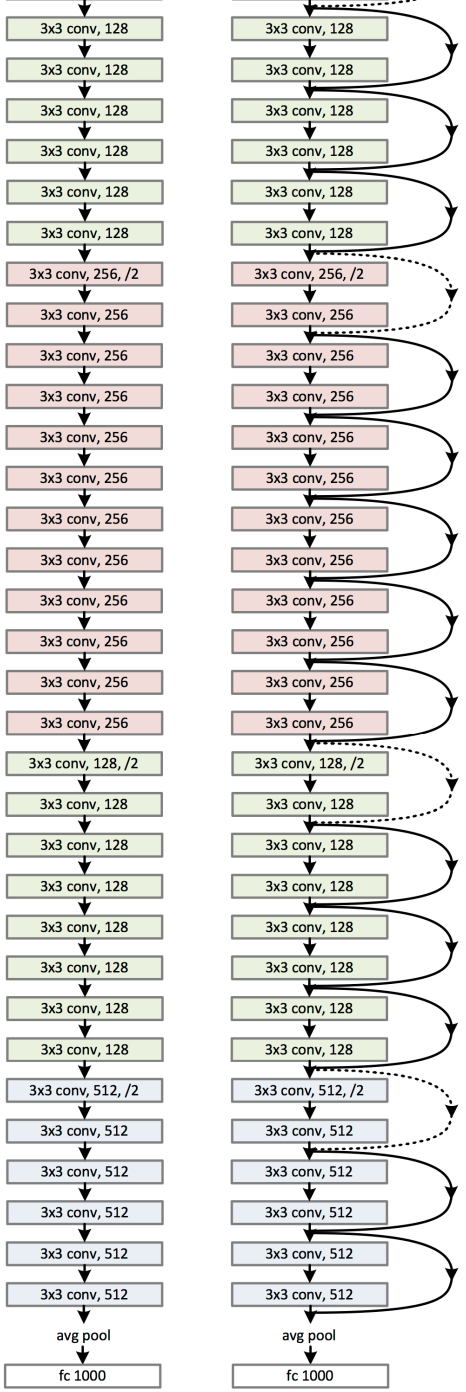


# Deep Learning



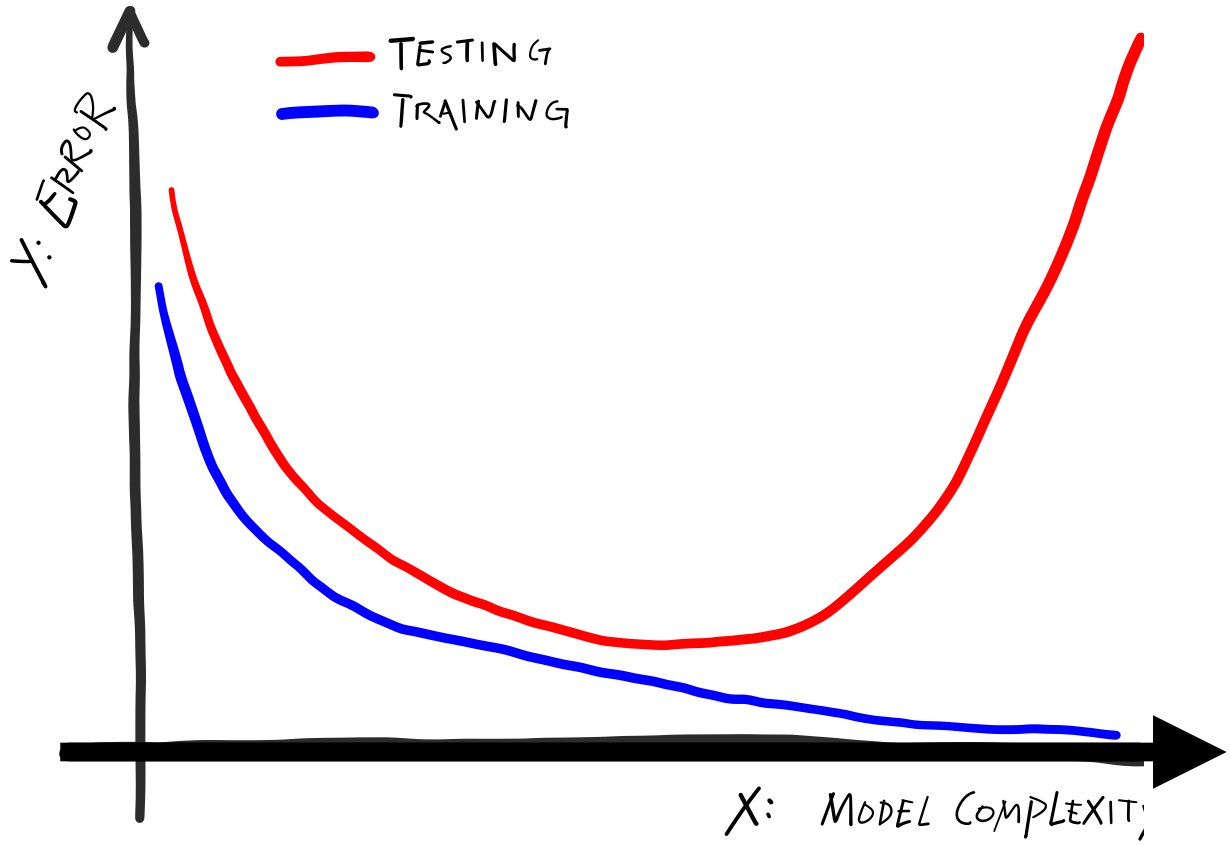





# Bias — Variance

[Chiyuan Zhang, ICLR'17]

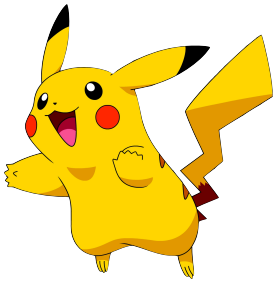



Deep Learning

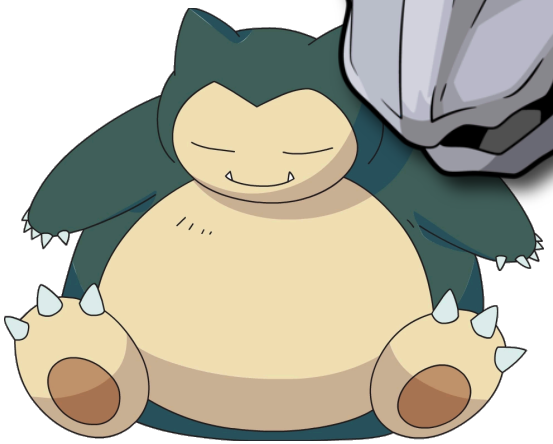
# Parameter Count / Num Training Samples

MLP 1x512
p/n: 24

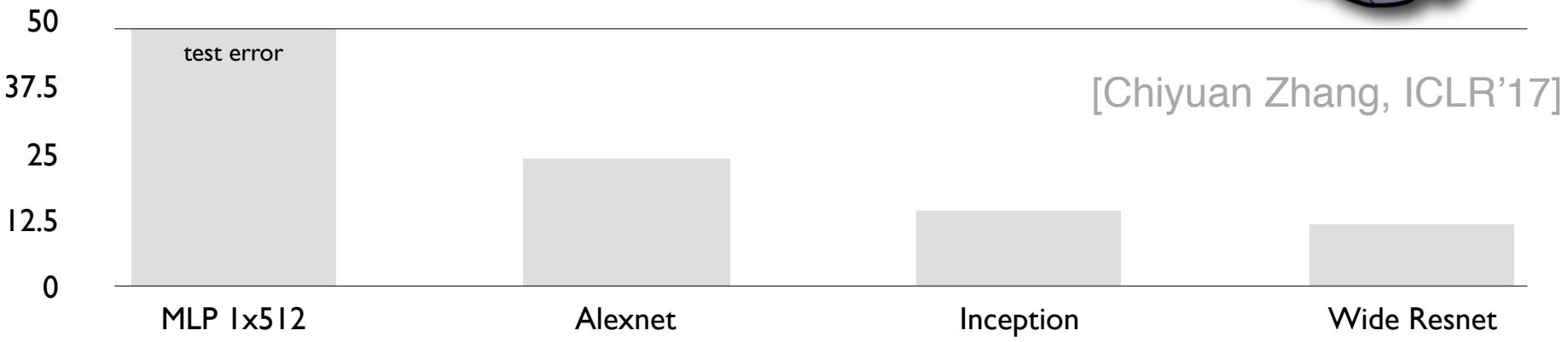
Alexnet
p/n: 28



Inception
p/n: 33



Wide Resnet
p/n: 179

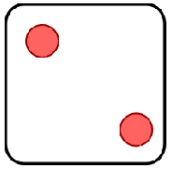



[Chiyuan Zhang, ICLR'17]

# Random Label Dataset

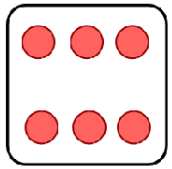
[Chiyuan Zhang, ICLR'17]

Dog

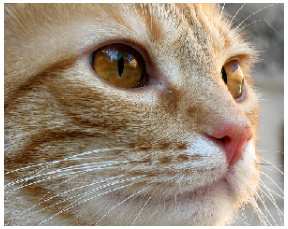


Cat

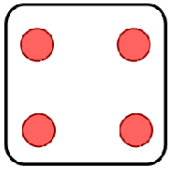
Flower



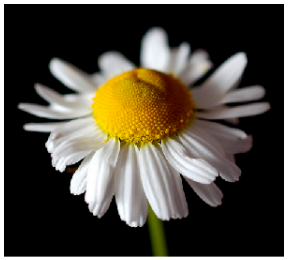
Dog



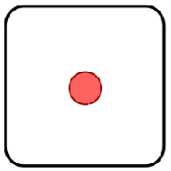
Cat



Bus



Flower

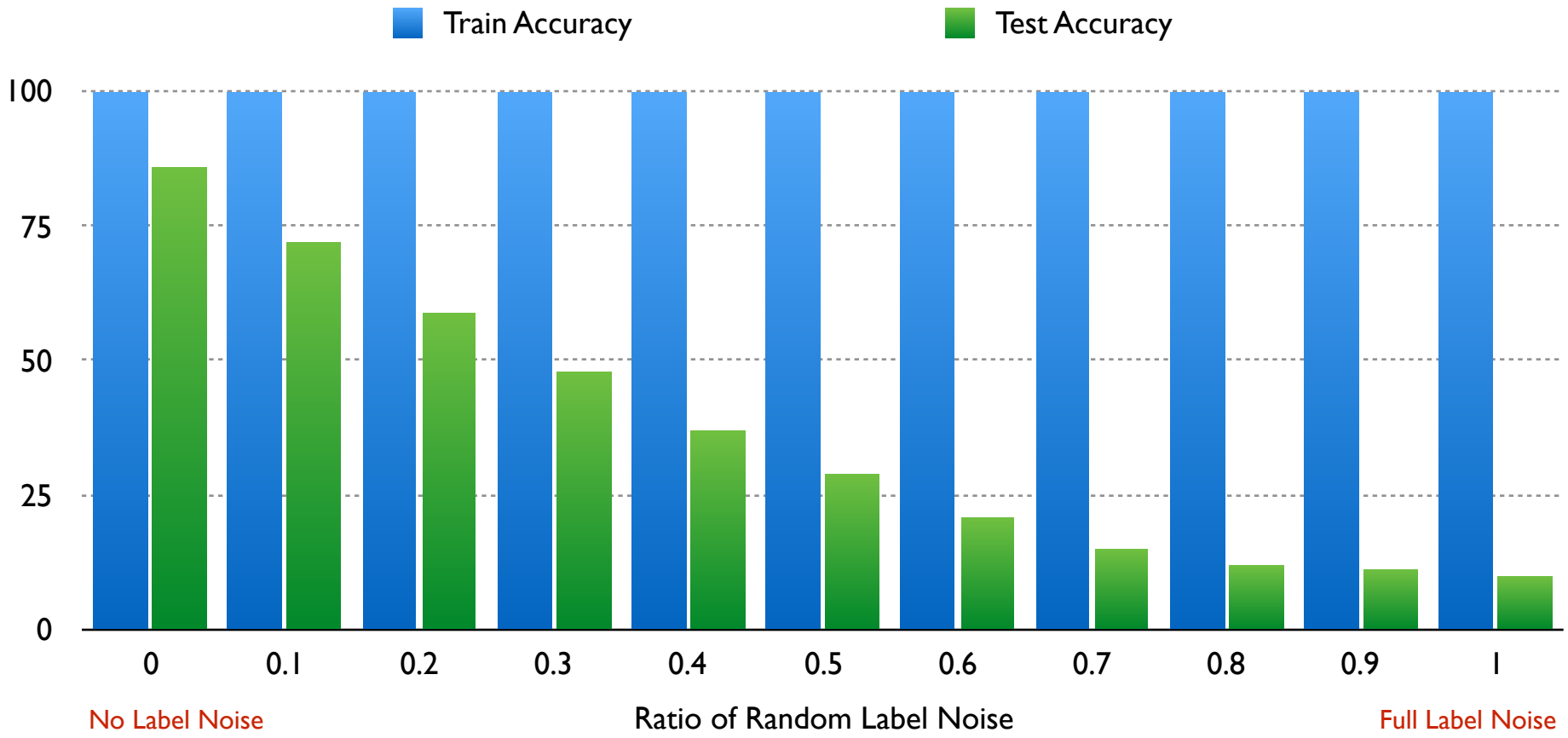


Bird

⋮

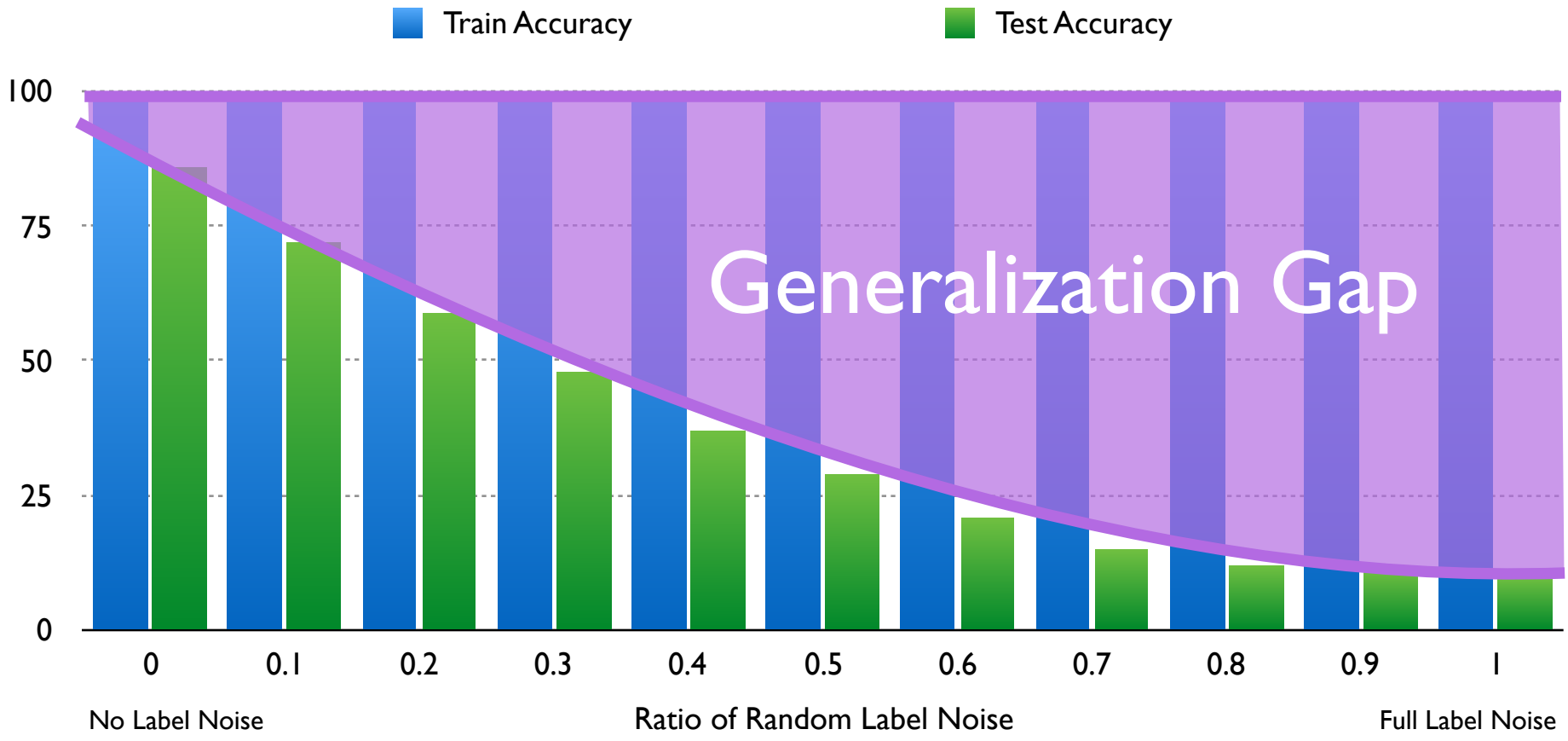
# Randomization Test

[Chiyuan Zhang, ICLR'17]



# Randomization Test

[Chiyuan Zhang, ICLR'17]



# Randomization Test

- Deep Neural Networks Easily fit random labels

[Chiyuan Zhang, ICLR'17]

Deep Neural Networks easily fit random labels.

# Regularizers in Deep Learning

[Chiyuan Zhang, ICLR'17]

- Data augmentation: domain-specific transformations

- Weight decay: l2-regularizer on weights

- Dropout*: randomly mask out responses

* Nitish Srivastava, Geoffrey E Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. JMLR, 15(1):1929–1958, 2014.

# Fitting Random Label with Regularizers

[Chiyuan Zhang, ICLR'17]

| Regularizer | Model | Training Accuracy |
|---|---|---|
| Weight decay | Inception | 100% |
| | Alexnet | Failed to converge |
| | MLP 1x512 | 99.21% |
| Crop Augmentation* | Inception | 99.93% |

IMAGENET

| Regularizer | Model | Training top-5 |
|---|---|---|
| Dropout | Inception V3 | 96.15% |
| Dropout + Weight decay | | 97.95% |

*We need to tune the hyperparams a bit and run for more epochs for this to converge, see paper for details.

# Implicit Regularization

[Chiyuan Zhang, ICLR'17]

A REGULARIZER is ~~a mechanism~~ anything that

~~constrain the model~~ or ~~empower the data.~~ hurt the training process.

# Next Class

- The **optimization algorithm** and **landscape of the loss function** have to be taken into consideration

- Sketch of some latest theoretical investigation into deep learning