



# Introduction to Computer Vision

## Lecture 6 - Deep Learning III

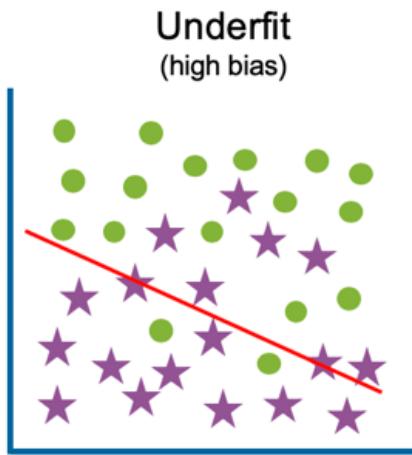
Prof. He Wang

# Logistics

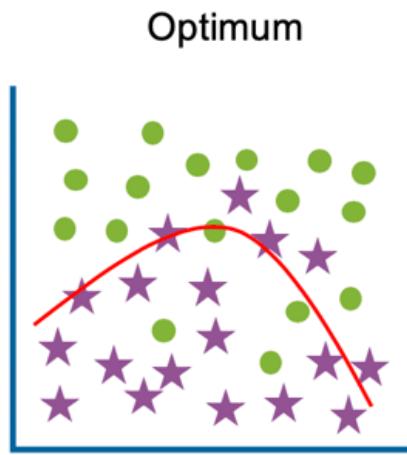
- Assignment 1: due on 3/30 11:59PM (Saturday)
  - If 1 day (0 - 24 hours) past the deadline, 15% off
  - If 2 day (24 - 48 hours) past the deadline, 30% off
  - Zero credit if more than 2 days.

# **Underfitting & Overfitting**

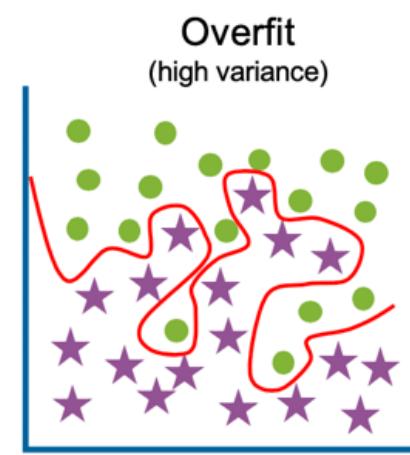
# Underfitting and Overfitting



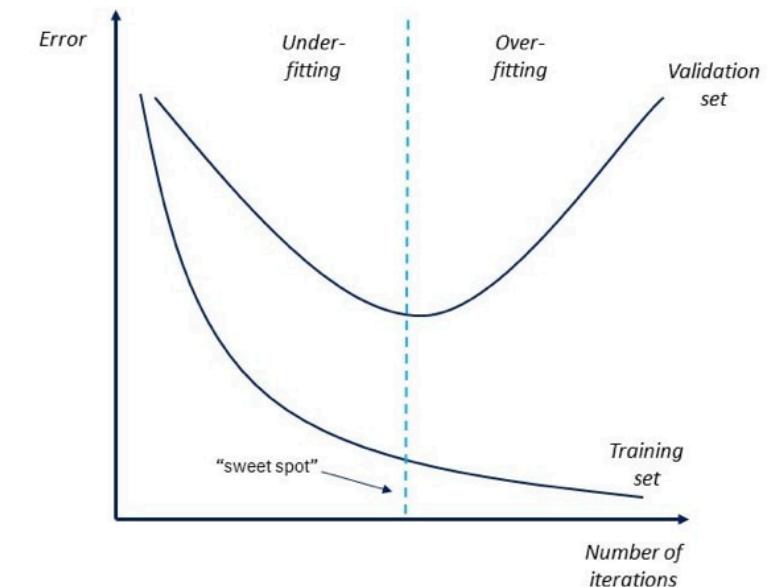
High training error  
High test error



Low training error  
Low test error



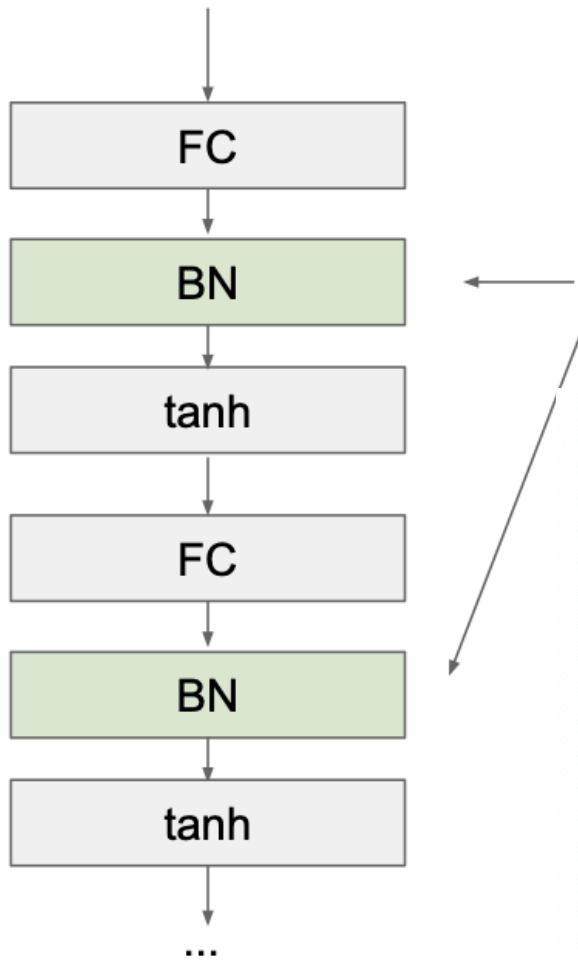
Low training error  
High test error



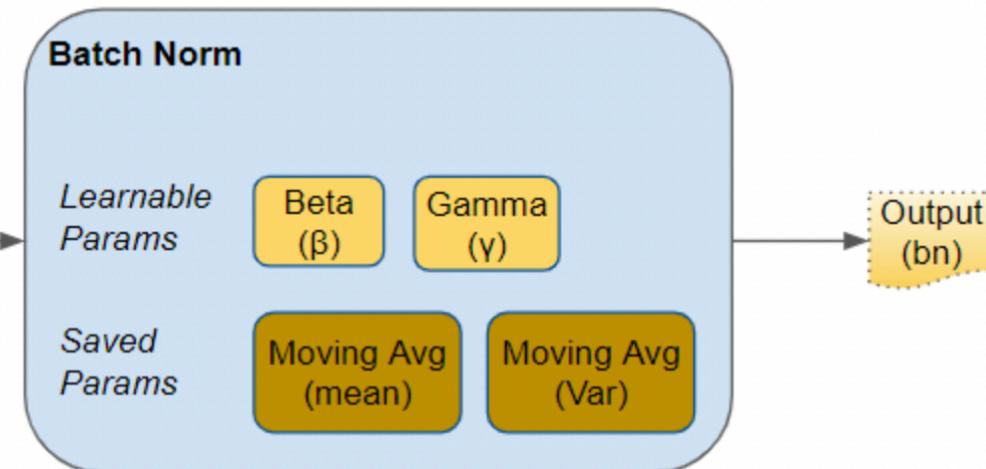
# Underfitting

- Underfitting on the train set: usually caused by limited model capacity or unsatisfactory optimization
  - Batch normalization
  - Skip link
- Overfitting on the test set

# Batch Normalization



Usually inserted after Fully Connected or Convolutional layers, and before nonlinearity.



# Batch Normalization: Train Mode

**Input:**  $x : N \times D$

**Learnable scale and shift parameters:**

$\gamma, \beta : D$

Learning  $\gamma = \sigma$ ,  
 $\beta = \mu$  will recover the identity function!

$$\mu_j = \frac{1}{N} \sum_{i=1}^N x_{i,j}$$

Per-channel mean,  
shape is D

$$\sigma_j^2 = \frac{1}{N} \sum_{i=1}^N (x_{i,j} - \mu_j)^2$$

Per-channel var,  
shape is D

$$\hat{x}_{i,j} = \frac{x_{i,j} - \mu_j}{\sqrt{\sigma_j^2 + \varepsilon}}$$

Normalized x,  
Shape is  $N \times D$

$$y_{i,j} = \gamma_j \hat{x}_{i,j} + \beta_j$$

Output,  
Shape is  $N \times D$

# Batch Normalization: Eval Mode (Test-Time)

**Input:**  $x : N \times D$

**Learnable scale and shift parameters:**

$\gamma, \beta : D$

During testing batchnorm becomes a linear operator!

Can be fused with the previous fully-connected or conv layer

$$\mu_{rms} = \text{(Running average of values seen during training)}$$

$$\mu_{rms} \leftarrow \rho\mu_{rms} + (1 - \rho)\mu_i$$

$$\sigma_{rms}^2 = \text{(Running average of values seen during training)}$$

$$\hat{x}_{i,j} = \frac{x_{i,j} - \mu_{rms,j}}{\sqrt{\sigma_{rms,j}^2 + \epsilon}}$$

$$y_{i,j} = \gamma_j \hat{x}_{i,j} + \beta_j$$

Per-channel mean, shape is D

Per-channel var, shape is D

Normalized x, Shape is N x D

Output, Shape is N x D

# BatchNorm Helps!

- Makes deep networks **much** easier to train!
- Improves gradient flow
- Allows higher learning rates, faster convergence
- Networks become more robust to initialization
- Acts as regularization during training

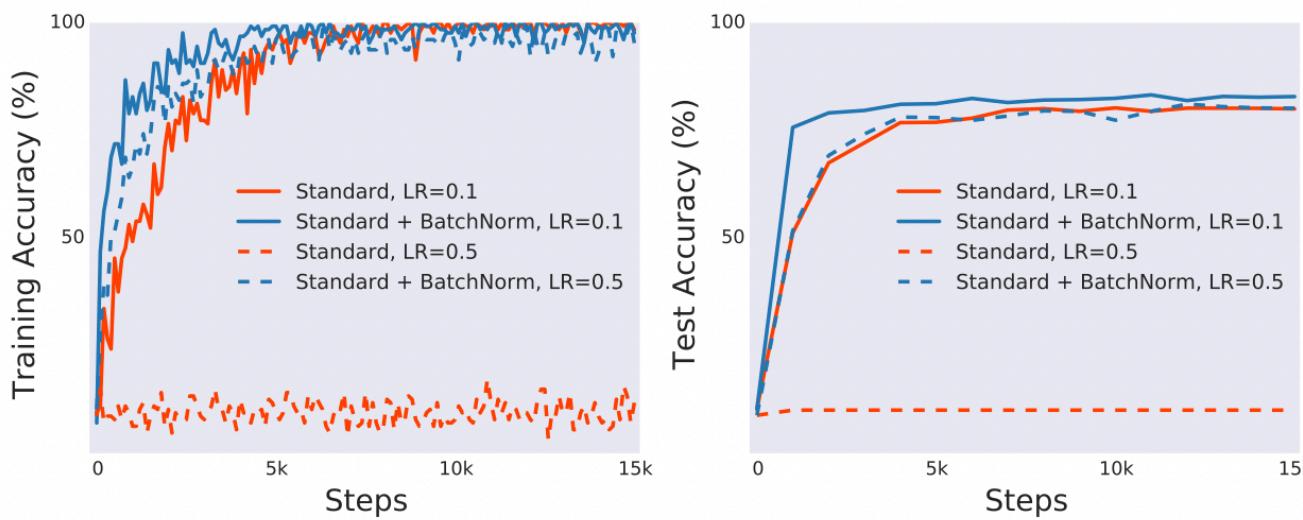


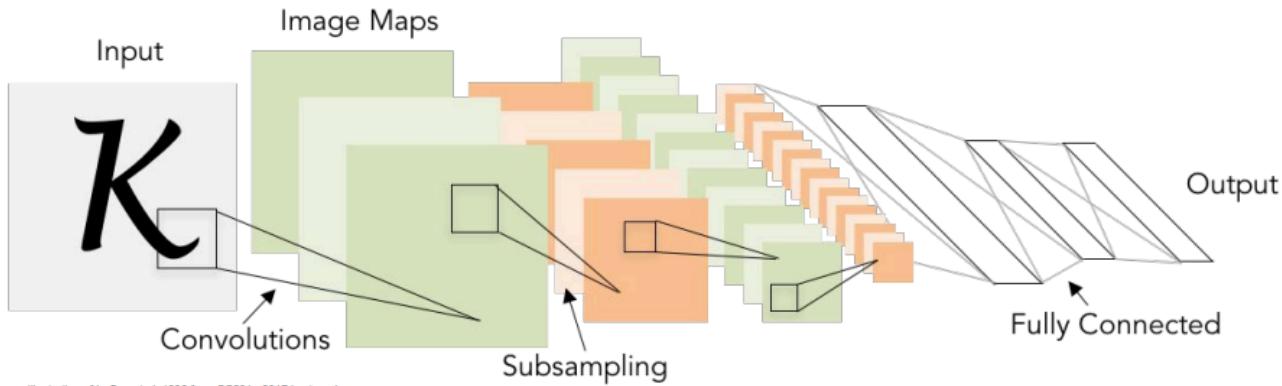
Figure 1: Comparison of (a) training (optimization) and (b) test (generalization) performance of a standard VGG network trained on CIFAR-10 with and without BatchNorm (details in Appendix A).

# Stacking More Layers

- ConvNets stack CONV,POOL,FC layers
- Trend towards smaller filters and deeper architectures
- Trend towards getting rid of POOL/FC layers (just CONV)
- Historically architectures looked like

**$[(\text{Conv-BN-ReLU})^*N - \text{POOL?}]^*m - (\text{FC-BN-RELU})^*K - \text{FC-SoftMax}$**

No BN at the last layer.



# Why BatchNorm Works?

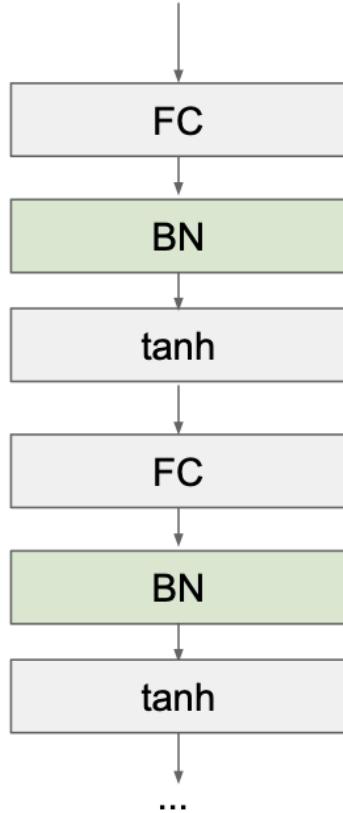
- Original hypothesis: mitigate the “internal covariate shift”
  - *“Training Deep Neural Networks is complicated by the fact that the distribution of each layer’s inputs changes during training, as the parameters of the previous layers change. This slows down the training by requiring lower learning rates and careful parameter initialization, and makes it notoriously hard to train models with saturating nonlinearities.”*
  - *“We refer to the change in the distributions of internal nodes of a deep network, in the course of training, as Internal Covariate Shift.”*

# Why BatchNorm Works?

- New findings: BatchNorm smoothens the loss landscape
  - BatchNorm may not reduce the internal covariate shift.
  - Batch normalization is effective because it smooths and, in turn, simplifies the optimization function that is being solved when training the network.
  - *“This ensures, in particular, that the gradients are more predictive and thus allow for use of larger range of learning rates and faster network convergence.”*

[How Does Batch Normalization Help Optimization? \(No, It Is Not About Internal Covariate Shift\)](#), 2018.

# Pros and Cons of BatchNorm

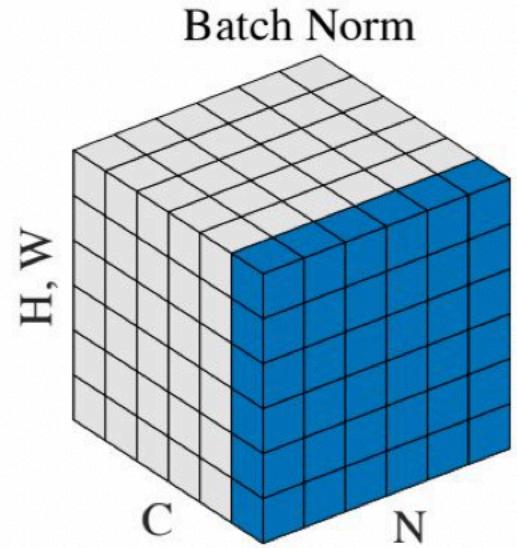


- Makes deep networks **much easier to train!**
- Improves gradient flow
- Allows higher learning rates, faster convergence
- Networks become more robust to initialization
- Acts as regularization during training
- Zero overhead at test-time: can be fused with conv!
- Behaves differently during training and testing: this is a very common source of bugs!

# Problems with Batch Normalization

- If batch size in the training time is too small (like 1), then  $\mu, \sigma$  in a training batch can be very random.
- There will be a big discrepancy between  $\mu, \sigma$  in a training batch and  $\mu_{rms}, \sigma_{rms}$  at test time. —> **Misaligned objective during training and testing**
- May lead to huge performance drop at test time even for training data.

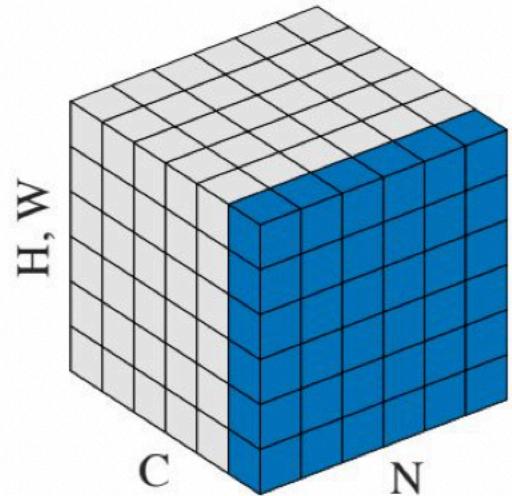
# Get Rid of Batch Dimension?



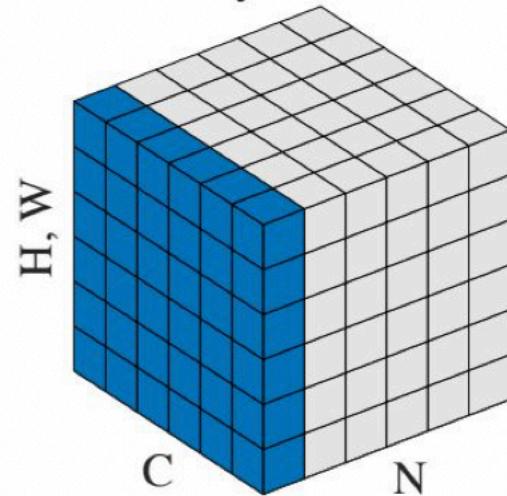
Can we remove the dependence on the batch dimension?  
We then don't have the discrepancy between train and eval modes.

# Normalization Techniques

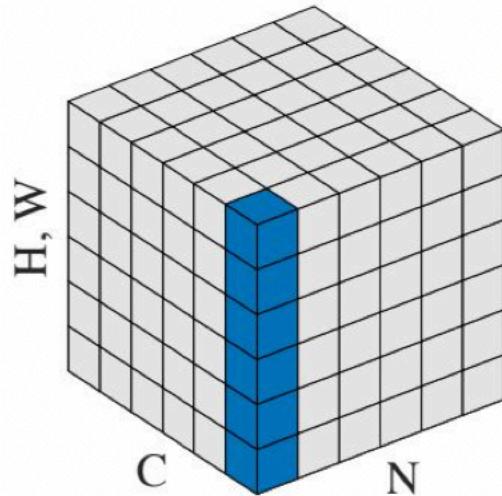
Batch Norm



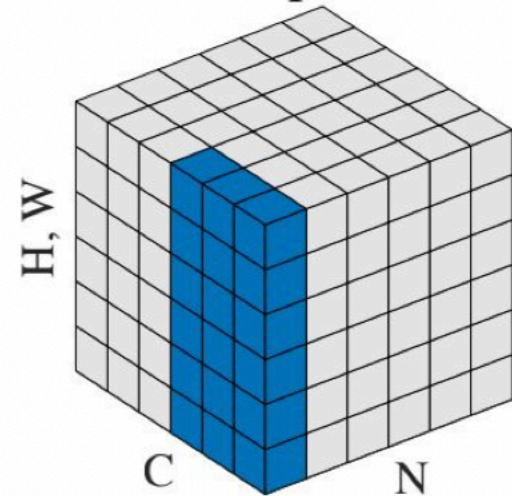
Layer Norm



Instance Norm



**Group Norm**



Widely used in NLP!

Style transfer!

# Group Normalization

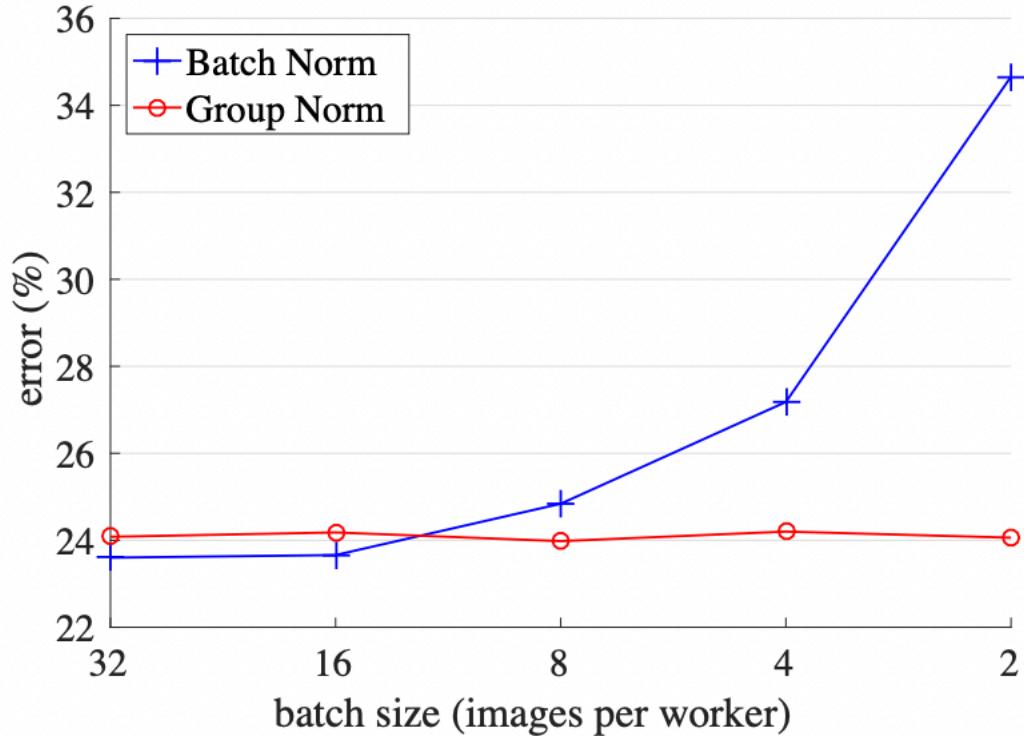


Figure 1. **ImageNet classification error vs. batch sizes.** This is a ResNet-50 model trained in the ImageNet training set using 8 workers (GPUs), evaluated in the validation set.

GroupNorm outperforms  
BatchNorm, when

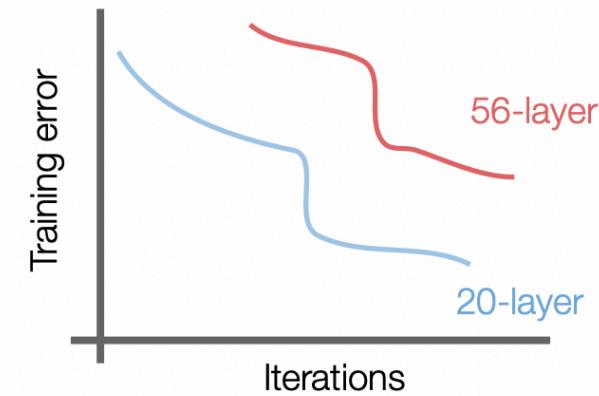
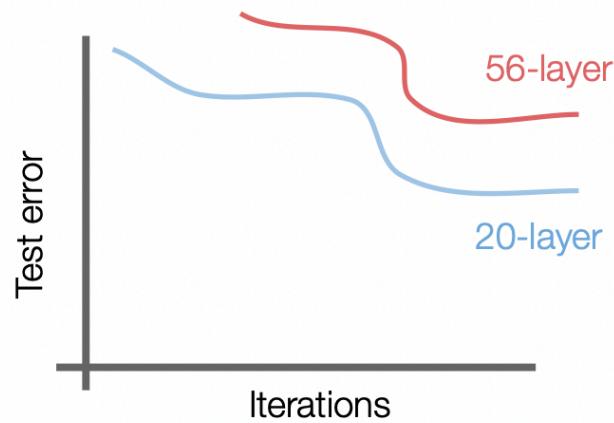
- Batch size is small
- Instances in a batch are highly correlated

# Problems of CNN Training

- Underfitting on the train set: usually caused by limited model capacity or unsatisfactory optimization
  - Batch normalization
  - ResNet or Skip links
- Overfitting on the test set

# Problems When CNN Gets Really Deep

What happens when we continue stacking deeper layers on a “plain” convolutional neural network?



56-layer model performs worse on both test and training error  
-> The deeper model performs worse, but it's not caused by overfitting!

# Problems When CNN Gets Really Deep

Fact: Deep models have more representation power (more parameters) than shallower models.

Hypothesis: the problem is an *optimization* problem,  
**deeper models are harder to optimize**

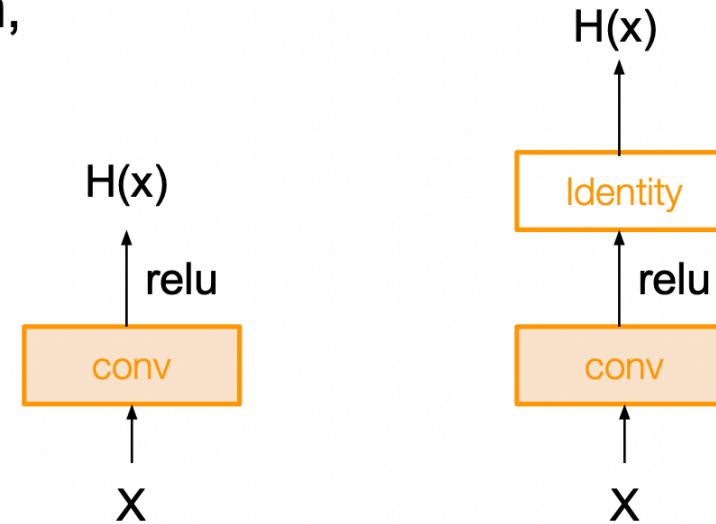
# Problems When CNN Gets Really Deep

Fact: Deep models have more representation power (more parameters) than shallower models.

Hypothesis: the problem is an *optimization* problem, deeper models are harder to optimize

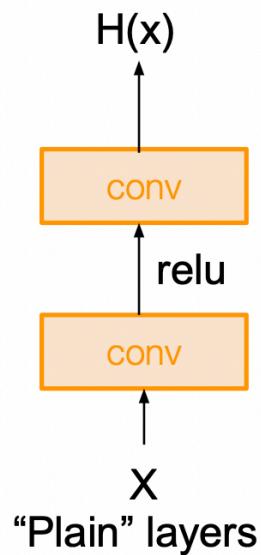
What should the deeper model learn to be at least as good as the shallower model?

A solution by construction is copying the learned layers from the shallower model and setting additional layers to identity mapping.



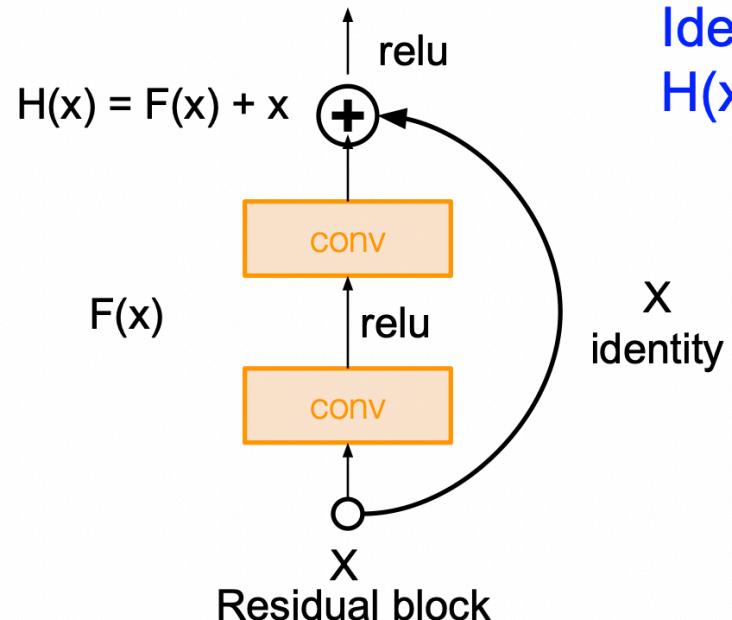
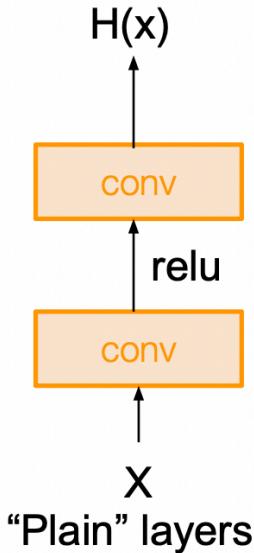
# Residual Link

Solution: Use network layers to fit a residual mapping instead of directly trying to fit a desired underlying mapping



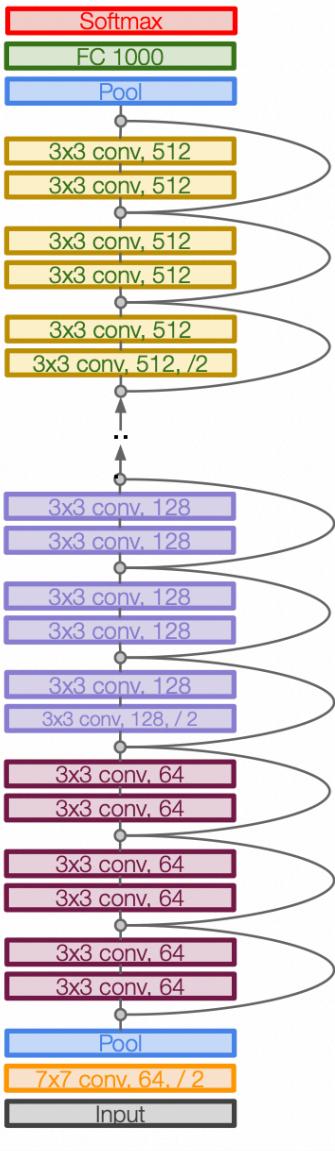
# Residual Link

Solution: Use network layers to fit a residual mapping instead of directly trying to fit a desired underlying mapping



Identity mapping:  
 $H(x) = x$  if  $F(x) = 0$

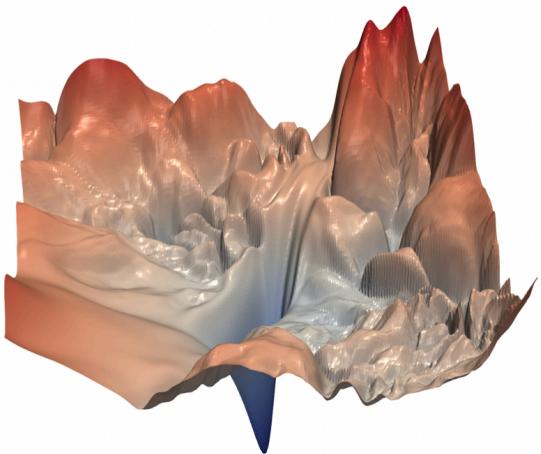
# From the Perspective of Gradient BP



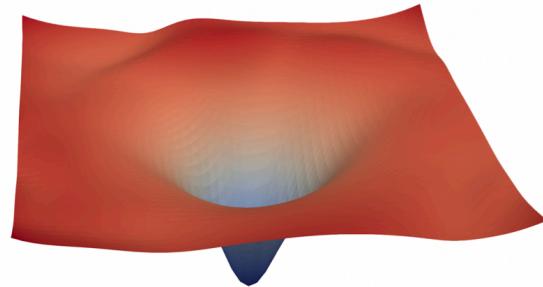
Skip links provide bypasses for gradients to backpropagate.

# Loss Landscape

- “When networks become sufficiently deep, neural loss landscapes quickly transition from being nearly convex to being highly chaotic. This transition from convex to chaotic behavior coincides with a dramatic drop in generalization error, and ultimately to a lack of trainability.”



(a) without skip connections



(b) with skip connections

Figure 1: The loss surfaces of ResNet-56 with/without skip connections. The proposed filter normalization scheme is used to enable comparisons of sharpness/flatness between the two figures.

# Loss Landscape

- “*skip connections promote flat minimizers and prevent the transition to chaotic behavior, which helps explain why skip connections are necessary for training extremely deep networks.*”

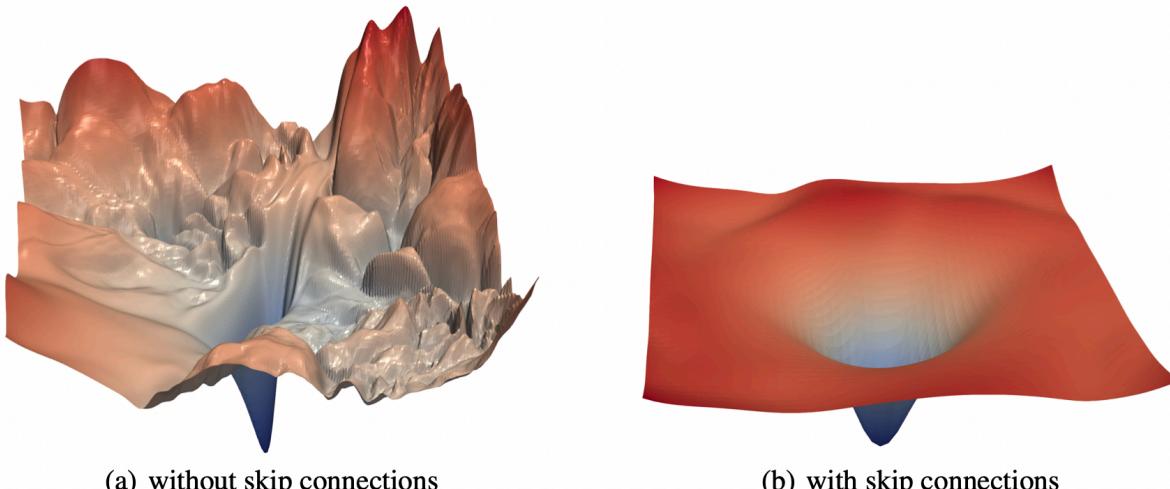
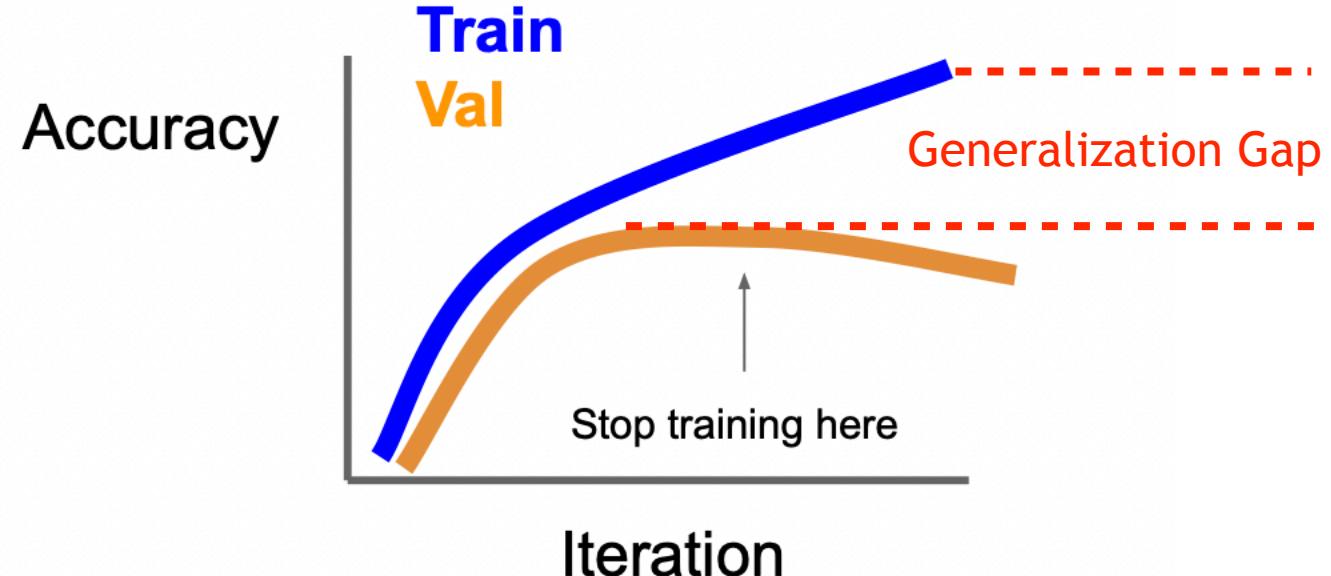
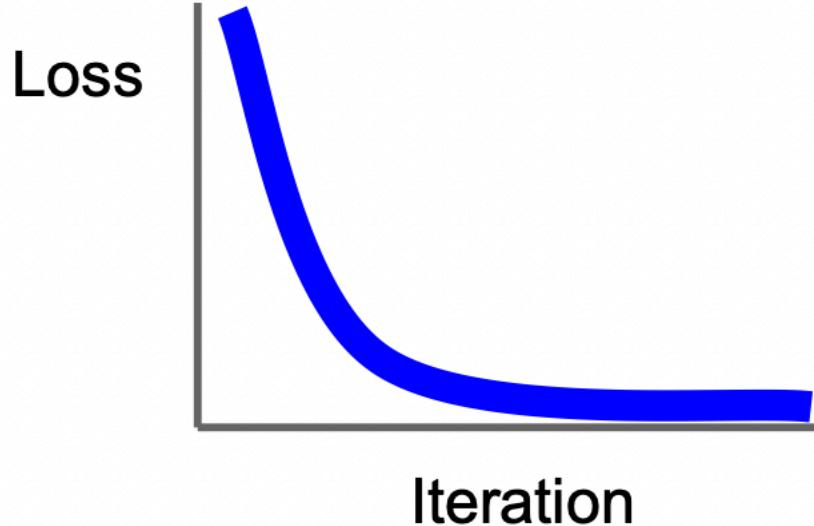


Figure 1: The loss surfaces of ResNet-56 with/without skip connections. The proposed filter normalization scheme is used to enable comparisons of sharpness/flatness between the two figures.

# Overfitting

- Underfitting on the train set: usually caused by limited model capacity or unsatisfactory optimization
  - Batch normalization
  - Skip link
- Overfitting on the test set:
  - Data augmentation
  - Regularization
  - Dropout

# The Generalization Gap

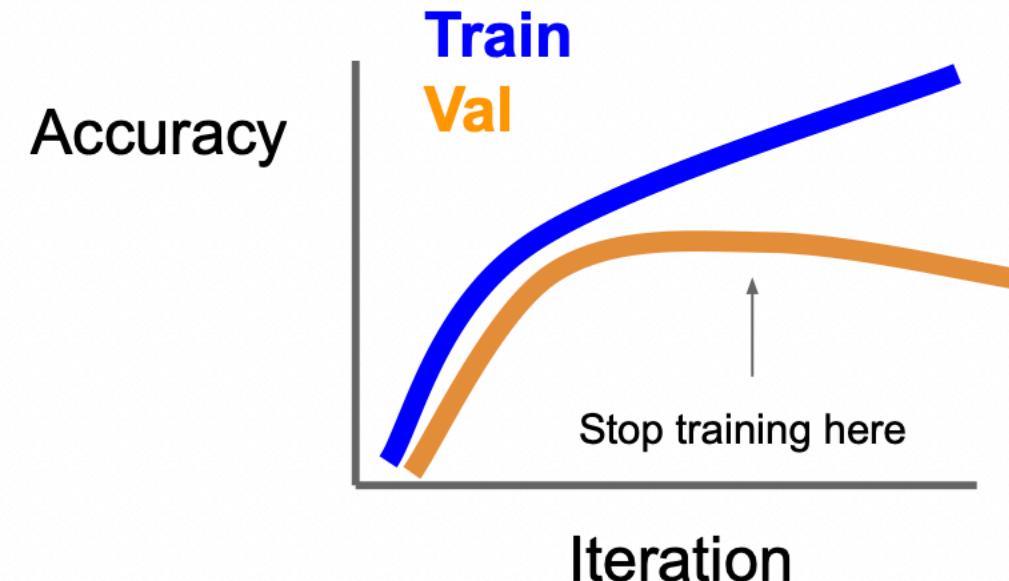
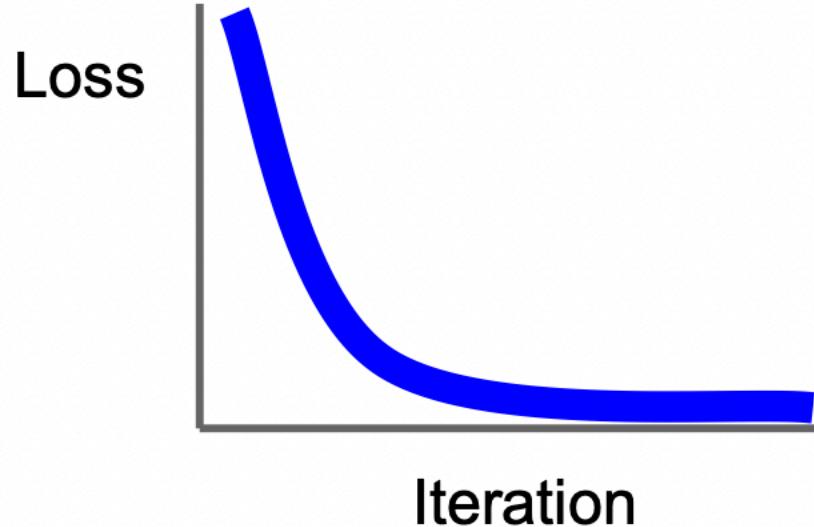


**Generalization gap:** the difference between a model's performance on training data and its performance on unseen data drawn from the same distribution.

# Overfitting

- Underfitting on the train set: usually caused by limited model capacity or unsatisfactory optimization
  - Batch normalization
  - Skip link
- Overfitting on the test set: **usually caused by imbalance between data and model**
  - Data augmentation
  - Regularization
  - Dropout

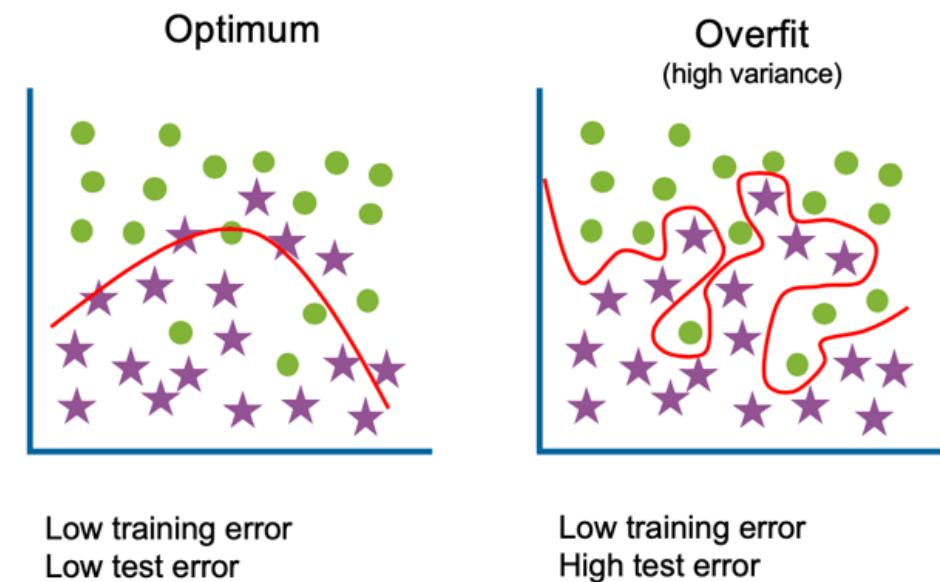
# Early Stopping



Stop training the model when accuracy on the validation set decreases  
Or train for a long time, but always keep track of the model snapshot  
that worked best on val

# The Generalization Gap and Overfitting

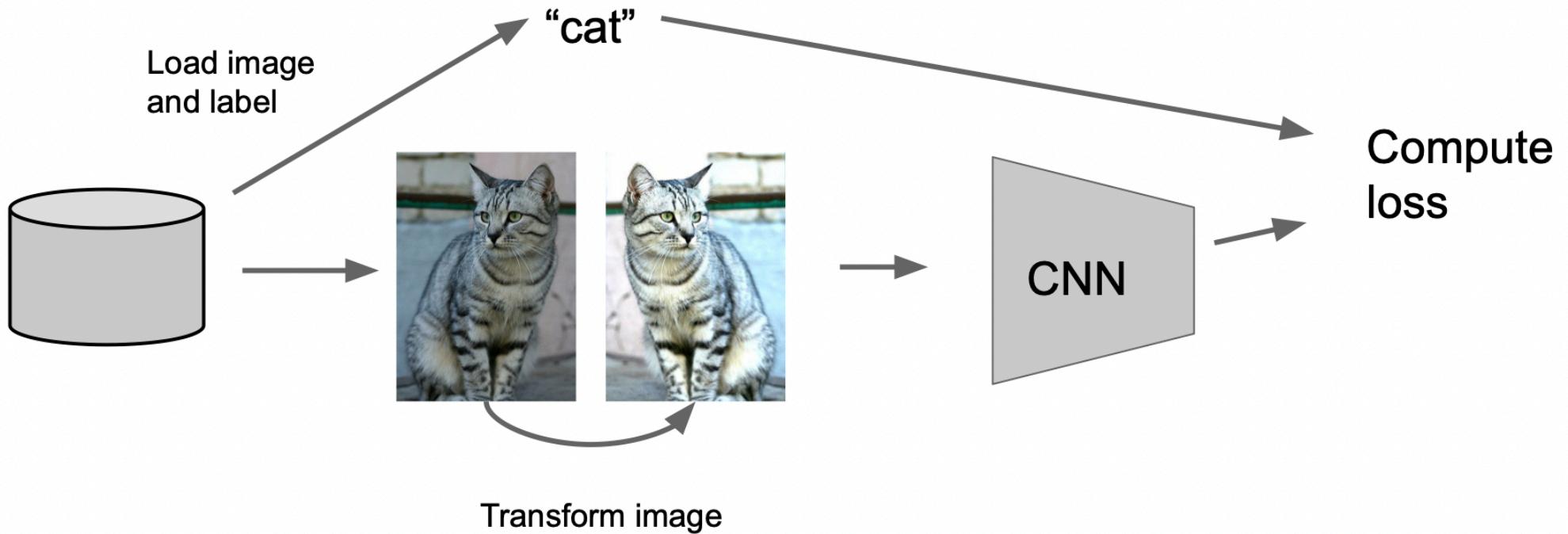
- For an overfitted model, it contains more parameters than can be justified by the data.
- The essence of overfitting is to have unknowingly extracted some of the residual variation (i.e., the **noise**) as if that variation represented underlying model structure
- To minimize generalization gap, we consider to minimize the mismatch between your model and your data.



# From the Perspective of Data

- If your data exhibit sufficient variations, then an appropriate model can't easily overfit.
- To increase the diversity of your data
  - simply collect more data (expensive and time consuming)
  - Data augmentation (free and fast)

# Data Augmentation



**Data augmentation** is a set of techniques to artificially increase the amount of data by generating new data points from existing data. This includes making small changes to data or using deep learning models to generate new data points.

# Simplest Data Augmentation: Horizontal Flip

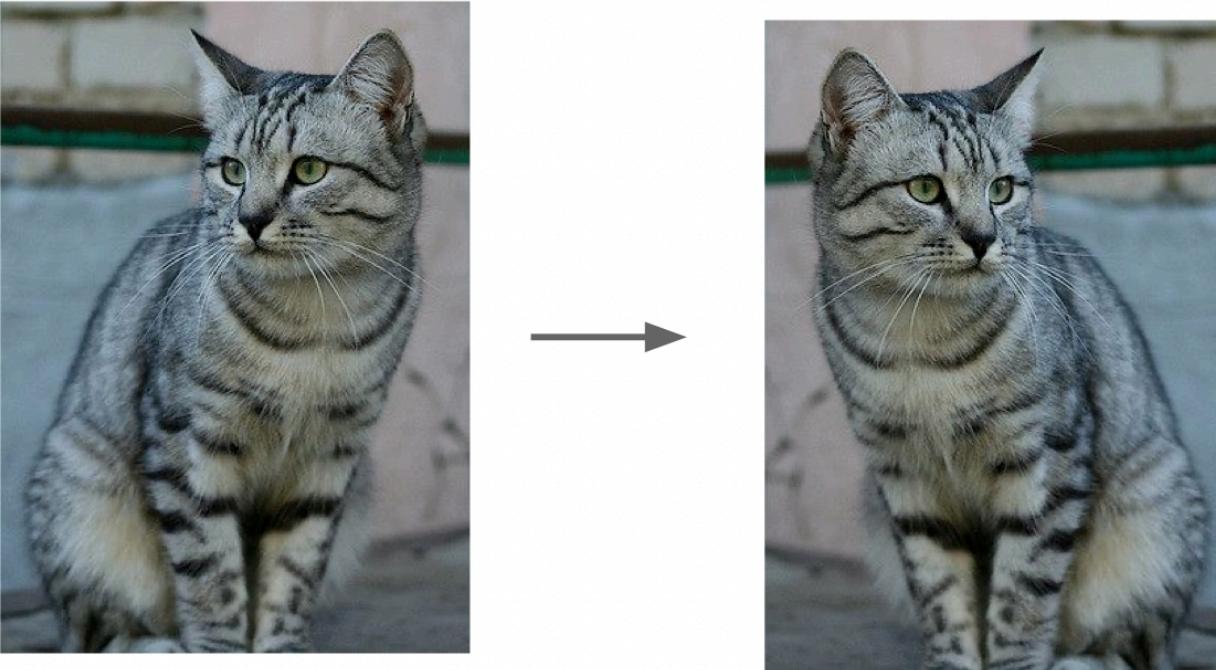
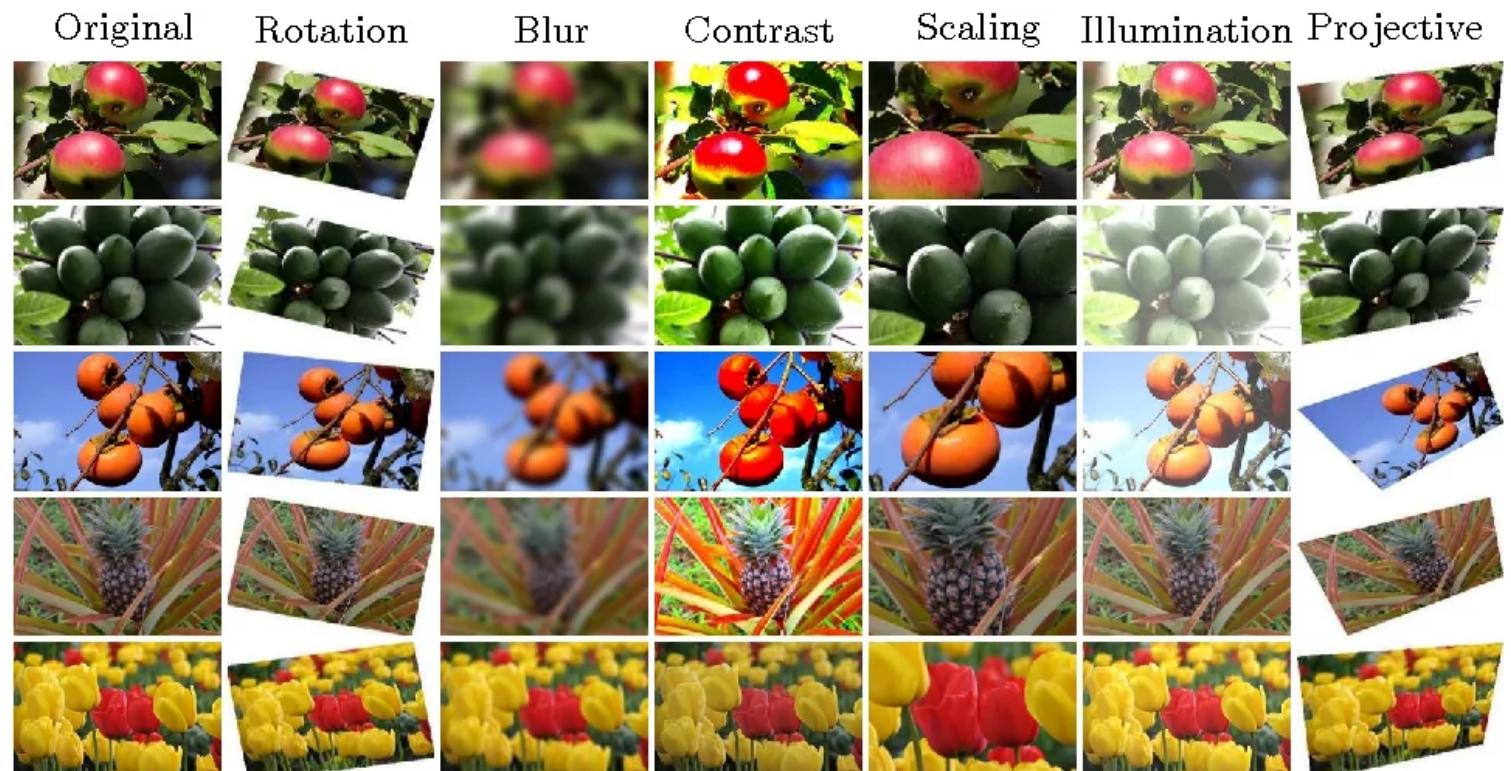


Figure credit: Stanford CS231N

- Data augmentation applies changes to the image while maintaining the label unchanged.
- The thing you care about must be invariant under the transformation of data augmentation.

# Data Augmentation Gallery

- Position augmentation
  - Scaling
  - Cropping
  - Flipping
  - Padding
  - Rotation
  - Translation
  - Affine transformation
- Color augmentation
  - Brightness
  - Contrast
  - Saturation
  - Hue
- Applying GAN/RL for data augmentation

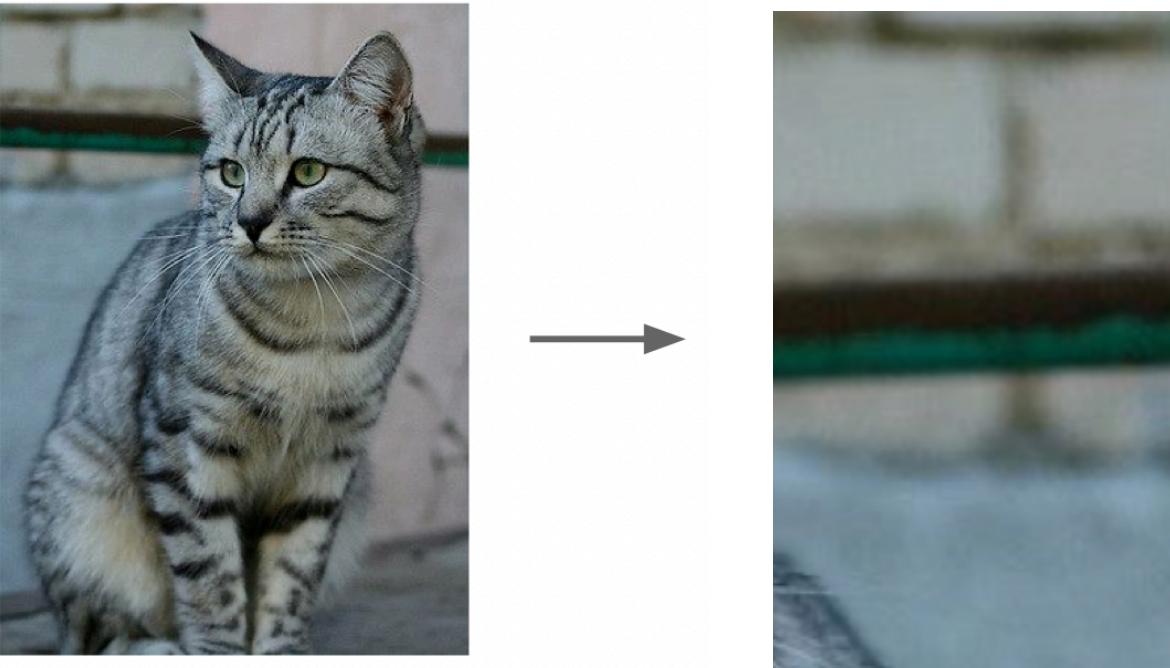


<https://research.aimultiple.com/data-augmentation/>

# Benefit of Using Data Augmentations

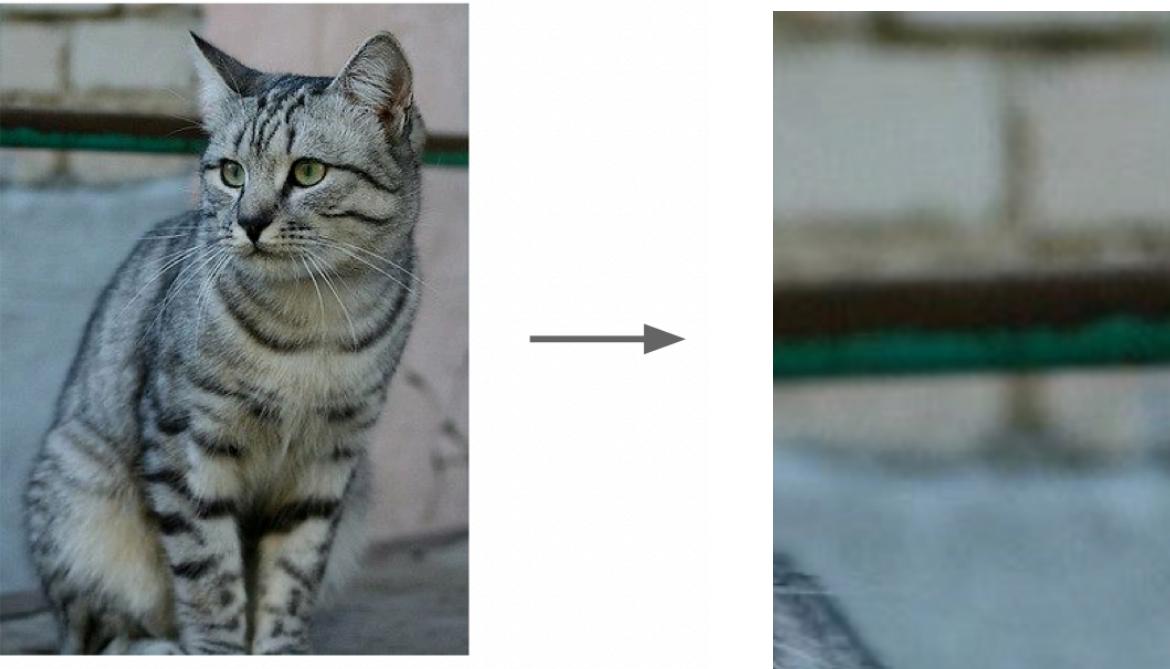
- Improving model prediction accuracy
  - reducing data overfitting and creating variability in data
  - increasing generalization ability of the models
  - helping resolve class imbalance issues in classification

# Doing Right Data Augmentations



- The magnitude of DA can't be too strong. If core information is lost, then model can't learn.

# Doing Right Data Augmentations



- The magnitude of DA can't be too strong. If core information is lost, then model can't learn.
- The magnitude of DA shouldn't be too weak, otherwise no use.
- Decide the magnitude by human or by tuning parameters.

# Overfitting

- Underfitting on the train set: usually caused by limited model capacity or unsatisfactory optimization
  - Batch normalization
  - Skip link
- Overfitting on the test set: usually caused by imbalance between data and model
  - Data augmentation
  - **Regularization**
  - Dropout

# Regularization

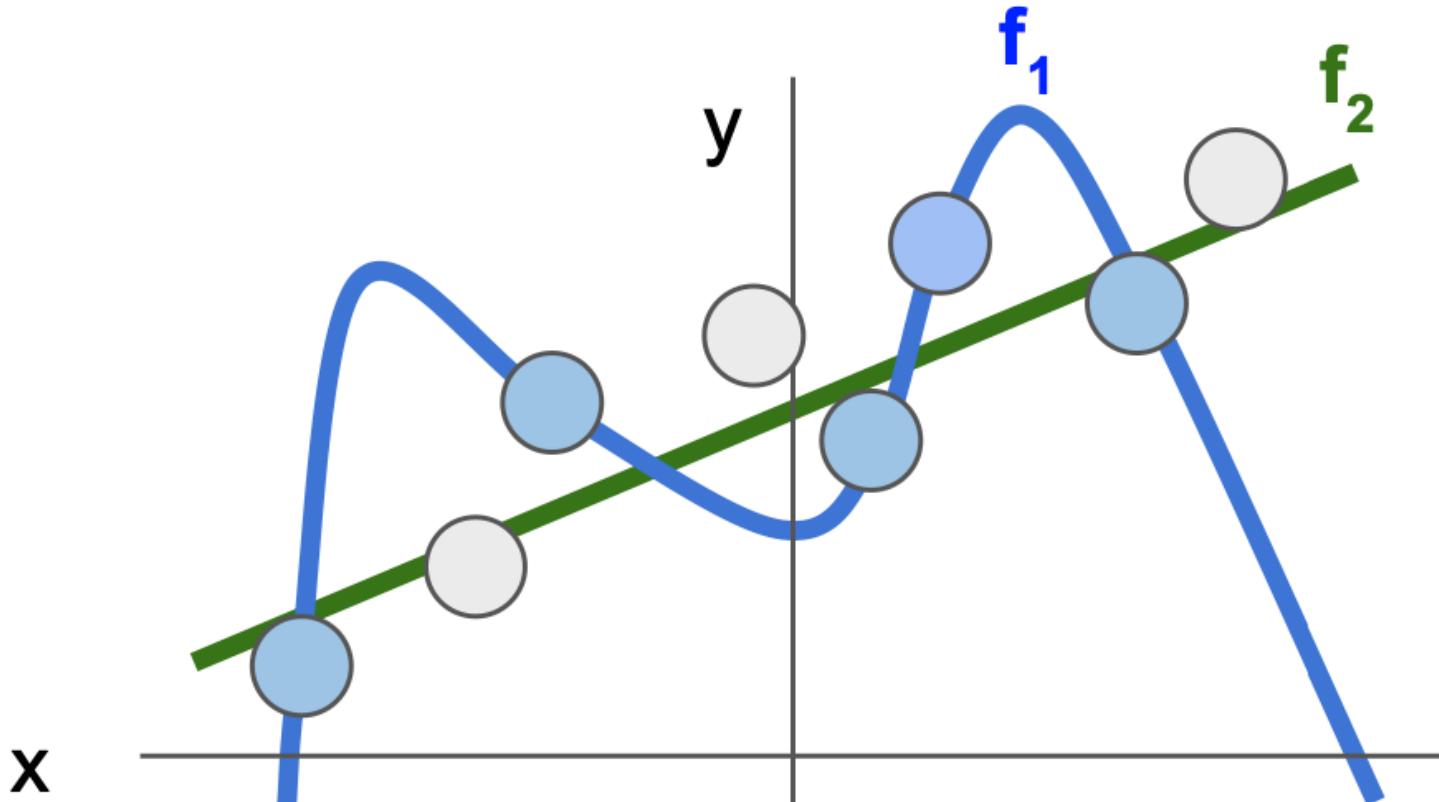
- Avoid the model to be arbitrarily complex

$$L(W) = \underbrace{\frac{1}{N} \sum_{i=1}^N L_i(f(x_i, W), y_i)}_{\text{Data loss: Model predictions should match training data}} + \lambda \underbrace{R(W)}_{\text{Regularization: Prevent the model from doing } \textit{too well} \text{ on training data}}$$

**Data loss:** Model predictions should match training data

**Regularization:** Prevent the model from doing *too well* on training data

# Regularization: Prefer Simpler Models



Regularization pushes against fitting the data  
too well so we don't fit noise in the data

# Regularization

- Avoid the model to be arbitrarily complex

$$L(W) = \underbrace{\frac{1}{N} \sum_{i=1}^N L_i(f(x_i, W), y_i)}_{\text{Data loss: Model predictions should match training data}} + \lambda \underbrace{R(W)}_{\text{Regularization: Prevent the model from doing } \textit{too well} \text{ on training data}}$$

**Data loss:** Model predictions should match training data

**Regularization:** Prevent the model from doing *too well* on training data

**Occam's Razor:** Among multiple competing hypotheses, the simplest is the best,  
William of Ockham 1285-1347

# Regularization from the Model Perspective

- Avoid the model to be arbitrarily complex

$$\mathcal{L} = \mathcal{L}_{main} + \lambda R(W)$$

In common use:

L2 regularization

$$R(W) = \sum_k \sum_l W_{k,l}^2 \quad (\text{Weight decay})$$

L1 regularization

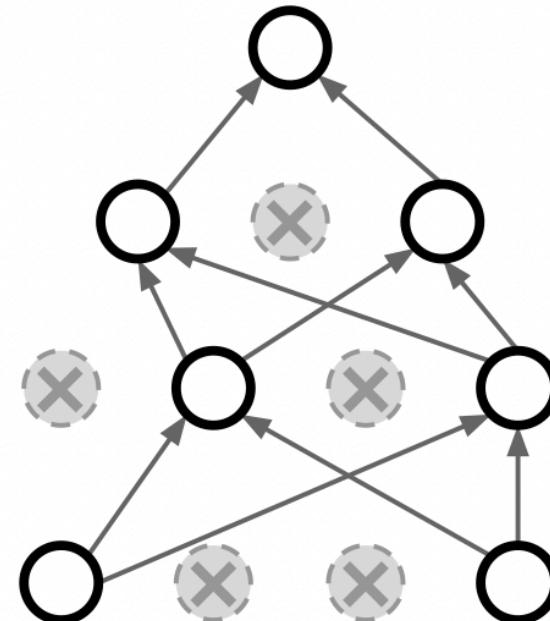
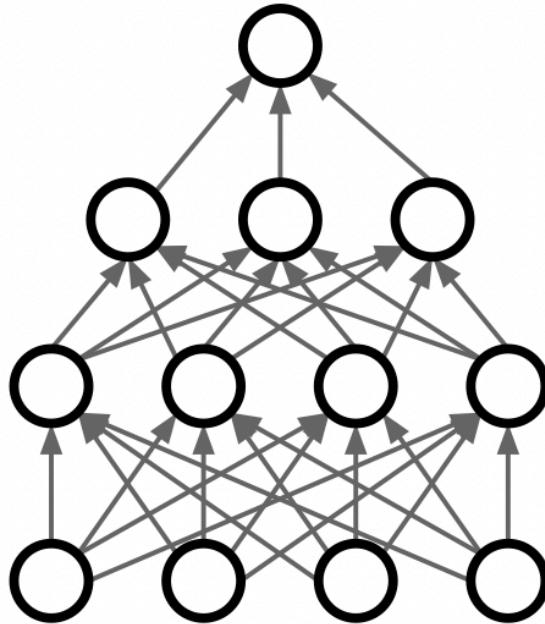
$$R(W) = \sum_k \sum_l |W_{k,l}|$$

Elastic net (L1 + L2)

$$R(W) = \sum_k \sum_l \beta W_{k,l}^2 + |W_{k,l}|$$

# Dropout

In each forward pass, randomly set some neurons to zero  
Probability of dropping is a hyperparameter; 0.5 is common



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

Slide credit: Stanford CS231N

# Dropout

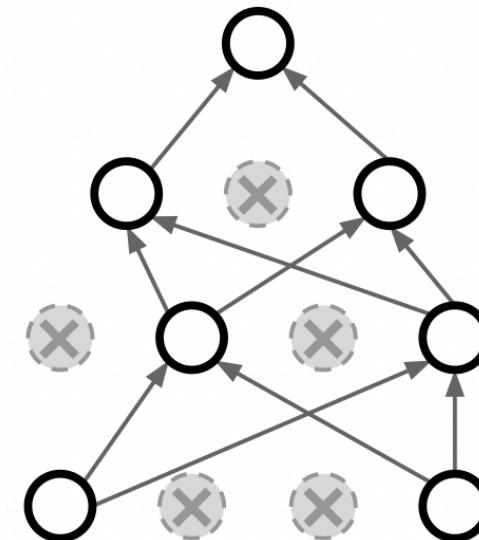
```
p = 0.5 # probability of keeping a unit active. higher = less dropout

def train_step(X):
    """ X contains the data """

    # forward pass for example 3-layer neural network
    H1 = np.maximum(0, np.dot(W1, X) + b1)
    U1 = np.random.rand(*H1.shape) < p # first dropout mask
    H1 *= U1 # drop!
    H2 = np.maximum(0, np.dot(W2, H1) + b2)
    U2 = np.random.rand(*H2.shape) < p # second dropout mask
    H2 *= U2 # drop!
    out = np.dot(W3, H2) + b3

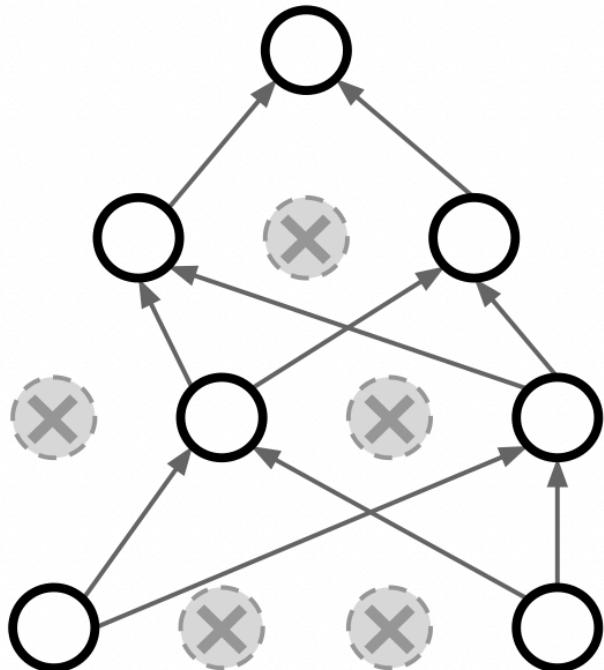
    # backward pass: compute gradients... (not shown)
    # perform parameter update... (not shown)
```

Example forward pass with a 3-layer network using dropout



# Dropout

How can this possibly be a good idea?



Forces the network to have a redundant representation;  
Prevents co-adaptation of features



# Dropout: Test Time

```
""" Vanilla Dropout: Not recommended implementation (see notes below) """  
  
p = 0.5 # probability of keeping a unit active. higher = less dropout  
  
def train_step(X):  
    """ X contains the data """  
  
    # forward pass for example 3-layer neural network  
    H1 = np.maximum(0, np.dot(W1, X) + b1)  
    U1 = np.random.rand(*H1.shape) < p # first dropout mask  
    H1 *= U1 # drop!  
    H2 = np.maximum(0, np.dot(W2, H1) + b2)  
    U2 = np.random.rand(*H2.shape) < p # second dropout mask  
    H2 *= U2 # drop!  
    out = np.dot(W3, H2) + b3  
  
    # backward pass: compute gradients... (not shown)  
    # perform parameter update... (not shown)  
  
def predict(X):  
    # ensembled forward pass  
    H1 = np.maximum(0, np.dot(W1, X) + b1) * p # NOTE: scale the activations  
    H2 = np.maximum(0, np.dot(W2, H1) + b2) * p # NOTE: scale the activations  
    out = np.dot(W3, H2) + b3
```

## Dropout Summary

drop in train time

scale at test time

At test time all neurons are active always  
=> We must scale the activations so that for each neuron:  
output at test time = expected output at training time

# BatchNorm as a Regularization

- BatchNorm forces the output before activations to follow a certain Gaussian distribution, which limits the capacity of a model ==> Regularization
- BatchNorm thus helps alleviate overfitting.
- With BatchNorm, people may not need dropout.

# Summary of Mitigating Overfitting

- Principle:
  - to balance the **data** variability and the **model** capacity
- Techniques:
  - **Data augmentation** (from the data perspective)
  - **BatchNorm** (from the data perspective)
  - **Regularization** (from the model perspective)
  - **Dropout** (from the model perspective)
  - ...

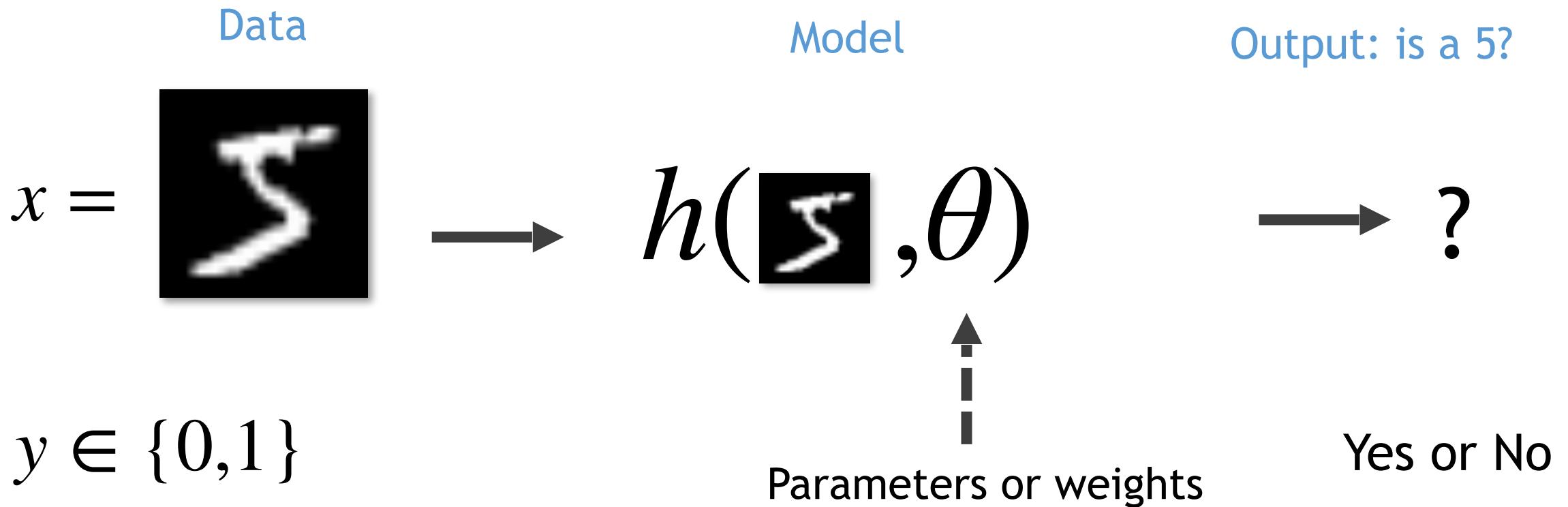
(Always good to use)

(used only for large FC layers)

# Classification

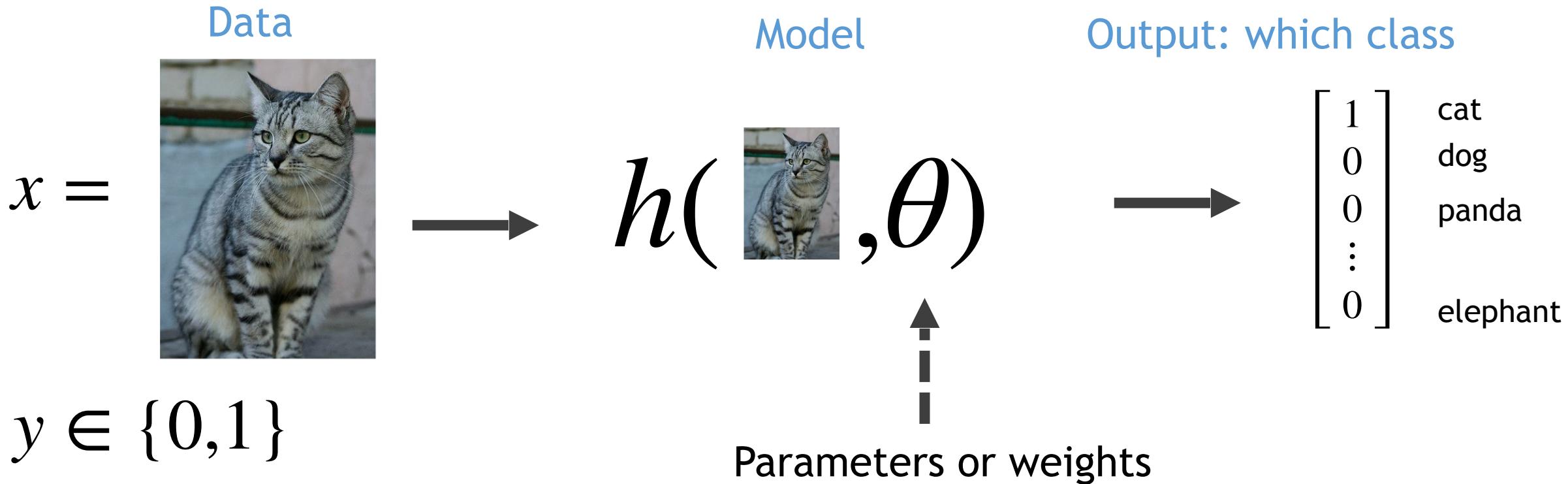
# Image Classification

- Image classification is a core task in computer vision.
- An example of binary classification:



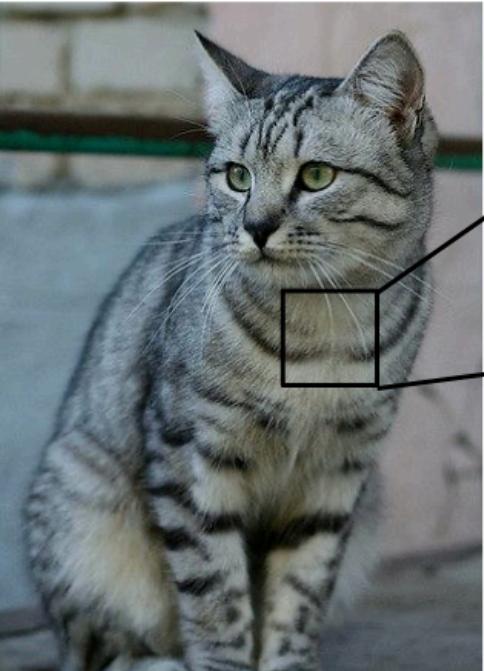
# Image Classification

- Classic definition: image classification is to categorize an image into several known classes ( $N$ ).
- Image classification is very important for **semantic understanding**.



# Challenges

## The Problem: Semantic Gap



This image by [Nikita](#) is  
licensed under [CC-BY 2.0](#)

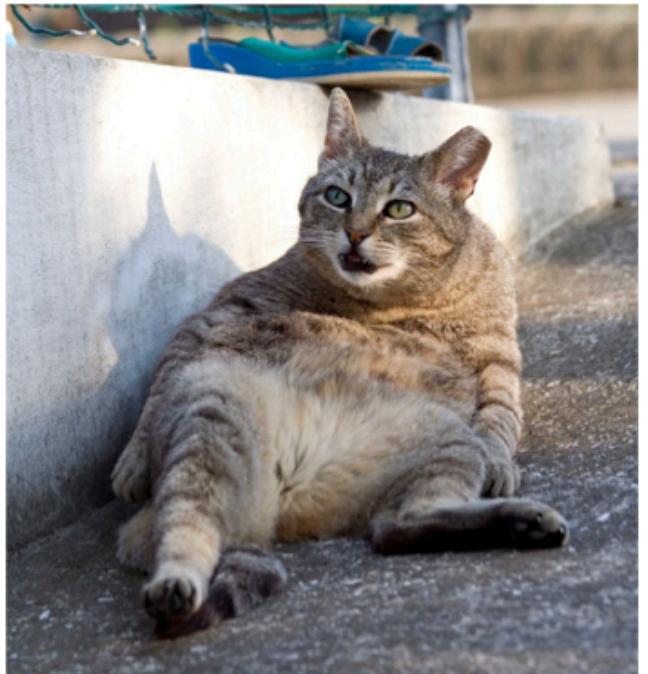
```
[[105 112 108 111 104 99 106 99 96 103 112 119 104 97 93 87]
 [ 91 98 102 106 104 79 98 103 99 105 123 136 110 105 94 85]
 [ 76 85 90 105 128 105 87 96 95 99 115 112 106 103 99 85]
 [ 99 81 81 93 128 131 127 100 95 98 102 99 96 93 101 94]
 [106 91 61 64 69 91 88 85 101 107 109 98 75 84 96 95]
 [114 108 85 55 55 69 64 54 64 87 112 129 98 74 84 91]
 [133 137 147 103 65 81 88 65 52 54 74 84 102 93 85 82]
 [128 137 144 140 109 95 86 70 62 65 63 63 60 73 86 101]
 [125 133 148 137 119 121 117 94 65 79 80 65 54 64 72 98]
 [127 125 131 147 133 127 126 131 111 96 89 75 61 64 72 84]
 [115 114 109 123 150 148 131 118 113 109 100 92 74 65 72 78]
 [ 89 93 90 97 108 147 131 118 113 114 113 109 106 95 77 80]
 [ 63 77 86 81 77 79 102 123 117 115 117 125 125 130 115 87]
 [ 62 65 82 89 78 71 88 101 124 126 119 101 107 114 131 119]
 [ 63 65 75 88 89 71 62 81 120 138 135 105 81 98 110 118]
 [ 87 65 71 87 106 95 69 45 76 130 126 107 92 94 105 112]
 [118 97 82 86 117 123 116 66 41 51 95 93 89 95 102 107]
 [164 146 112 80 82 120 124 104 76 48 45 66 88 101 102 109]
 [157 170 157 120 93 86 114 132 112 97 69 55 70 82 99 94]
 [130 128 134 161 139 100 109 118 121 134 114 87 65 53 69 86]
 [128 112 96 117 158 144 128 115 104 107 102 93 87 81 72 79]
 [123 107 96 86 83 112 153 149 122 109 104 75 80 107 112 99]
 [122 121 102 80 82 86 94 117 145 148 153 102 58 78 92 107]
 [122 164 148 103 71 56 78 83 93 103 119 139 102 61 69 84]]
```

What the computer sees

An image is a tensor of integers  
between [0, 255]:

e.g. 800 x 600 x 3  
(3 channels RGB)

# Challenges: Pose and Deformation



[This image](#) by [Umberto Salvagnin](#)  
is licensed under [CC-BY 2.0](#)



[This image](#) by [Umberto Salvagnin](#)  
is licensed under [CC-BY 2.0](#)

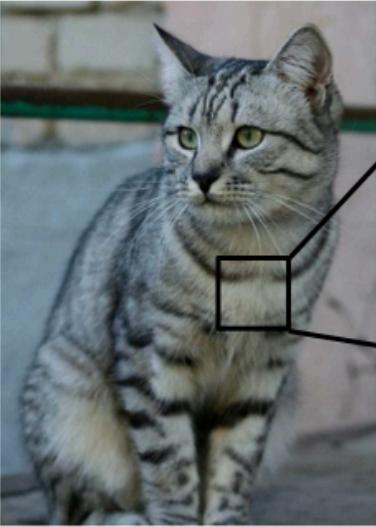


[This image](#) by [sare bear](#) is  
licensed under [CC-BY 2.0](#)

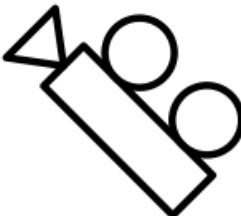
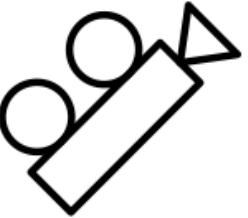


[This image](#) by [Tom Thai](#) is  
licensed under [CC-BY 2.0](#)

# Challenges: Viewpoint Variation



|  |
|--|
| [105 112 108 111 104 99 106 99 96 103 112 119 104 97 93 87]    |
| [ 91 108 106 104 101 98 105 102 100 103 102 109 106 103 94 85] |
| [ 76 85 98 106 120 107 105 87 96 99 100 101 106 103 90 85]     |
| [ 99 81 81 93 129 107 107 109 95 100 102 99 96 93 101 94]      |
| [106 91 61 64 69 91 88 85 101 107 109 98 75 84 86 96]          |
| [114 108 85 55 55 69 64 54 64 87 112 129 98 74 84 91]          |
| [133 137 147 103 65 81 86 65 62 52 54 74 84 102 93 85 82]      |
| [128 137 144 140 109 95 86 78 62 65 63 63 60 73 86 101]        |
| [125 133 148 137 119 121 117 94 65 79 88 65 54 64 72 98]       |
| [127 125 131 147 133 127 121 131 111 96 89 75 61 64 72 94]     |
| [115 114 109 123 150 148 131 118 113 109 108 92 74 65 72 78]   |
| [ 89 93 98 97 108 147 131 118 113 114 113 109 106 95 77 80]    |
| [ 63 77 86 81 77 79 102 123 117 115 117 125 125 130 115 87]    |
| [ 62 65 82 89 78 71 88 101 124 126 119 101 107 114 131 119]    |
| [ 63 65 75 88 89 71 62 81 120 138 135 105 81 98 110 118]       |
| [ 87 65 71 87 106 95 69 45 76 138 126 107 92 94 105 112]       |
| [118 97 82 86 117 123 116 66 41 51 95 93 89 95 102 107]        |
| [164 146 112 88 82 123 121 104 76 48 45 66 88 101 102 109]     |
| [157 170 157 120 93 86 114 132 112 97 69 55 70 82 99 94]       |
| [138 128 134 161 139 108 109 118 121 134 114 87 65 53 69 86]   |
| [128 112 96 117 150 144 120 115 104 107 102 93 87 81 72 79]    |
| [123 107 96 86 83 112 153 149 122 109 104 75 80 107 112 99]    |
| [122 121 102 88 82 86 94 117 145 148 153 102 58 78 92 107]     |
| [122 164 148 103 71 56 78 83 93 103 119 102 61 69 84]          |



All pixels change when  
the camera moves!

# Challenges: Background Variation



[This image](#) is CC0 1.0 public domain



[This image](#) is CC0 1.0 public domain

# Challenges: Illumination



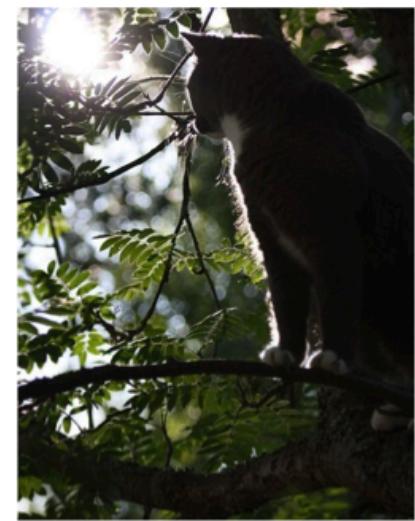
[This image](#) is CC0 1.0 public domain



[This image](#) is CC0 1.0 public domain



[This image](#) is CC0 1.0 public domain



[This image](#) is CC0 1.0 public domain

# Challenges: Occlusion



[This image](#) is [CC0 1.0](#) public domain



[This image](#) is [CC0 1.0](#) public domain



[This image](#) by [jonsson](#) is licensed  
under [CC-BY 2.0](#)

# Challenges: Intraclass Variation



[This image](#) is [CC0 1.0](#) public domain

# The Requirement of A Good Image Classifier

- Invariant with all the aforementioned variations.
- That's why we want to add those data augmentations to mimic these variations.
  - Rotation → Pose/viewpoint
  - Color jittering → illumination
  - Crop and scale → viewpoint
  - ...

# Methods

- Non-parameteric models
  - Nearest Neighbor
- Parametric models
  - CNN
  - ...

# Nearest Neighbor Classifier

deer



bird



plane



cat



car



Training data with labels

?



query data

Distance Metric

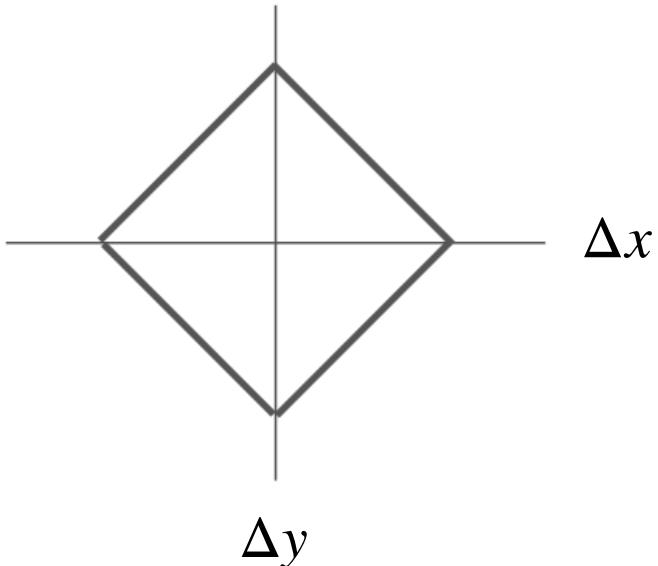


$\rightarrow \mathbb{R}$

# Distance Metric

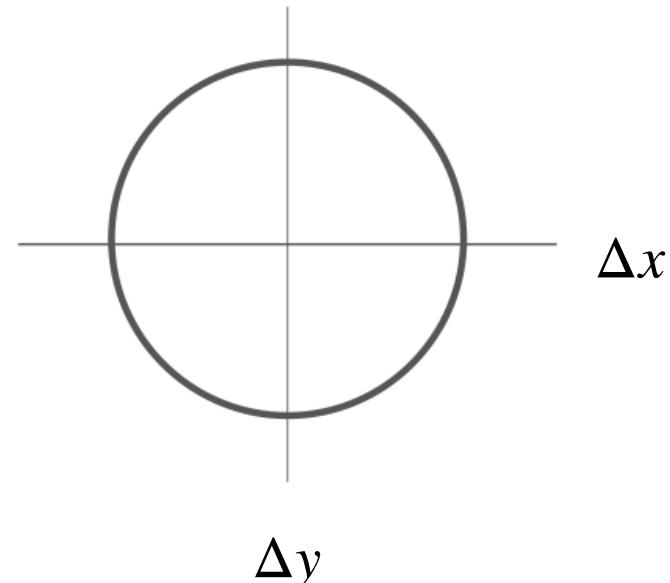
L1 (Manhattan) distance

$$d_1(I_1, I_2) = \sum_p |I_1^p - I_2^p|$$



L2 (Euclidean) distance

$$d_2(I_1, I_2) = \sqrt{\sum_p (I_1^p - I_2^p)^2}$$



# Distance Metric

**L1 distance:**

$$d_1(I_1, I_2) = \sum_p |I_1^p - I_2^p|$$

test image

|    |    |     |     |
|----|----|-----|-----|
| 56 | 32 | 10  | 18  |
| 90 | 23 | 128 | 133 |
| 24 | 26 | 178 | 200 |
| 2  | 0  | 255 | 220 |

training image

|    |    |     |     |
|----|----|-----|-----|
| 10 | 20 | 24  | 17  |
| 8  | 10 | 89  | 100 |
| 12 | 16 | 178 | 170 |
| 4  | 32 | 233 | 112 |

-

pixel-wise absolute value differences

|    |    |    |     |
|----|----|----|-----|
| 46 | 12 | 14 | 1   |
| 82 | 13 | 39 | 33  |
| 12 | 10 | 0  | 30  |
| 2  | 32 | 22 | 108 |

=

add  
→ 456

# Nearest Neighbor Classifier

```
def train(images, labels):  
    # Machine learning!  
    return model
```



Memorize all  
data and labels

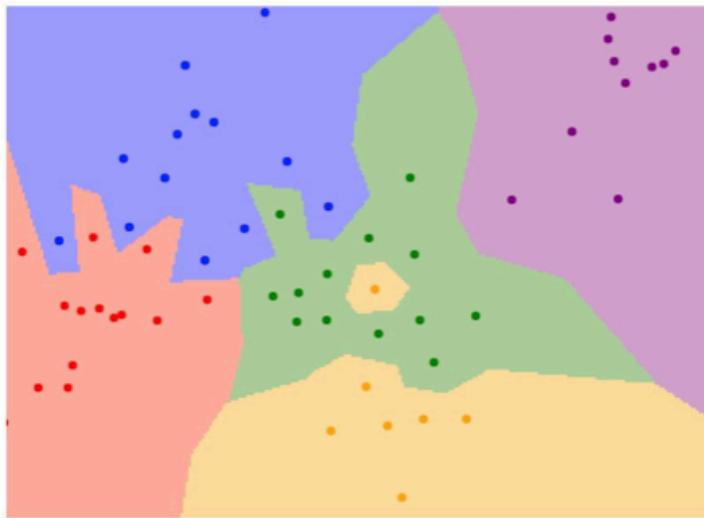
```
def predict(model, test_images):  
    # Use model to predict labels  
    return test_labels
```



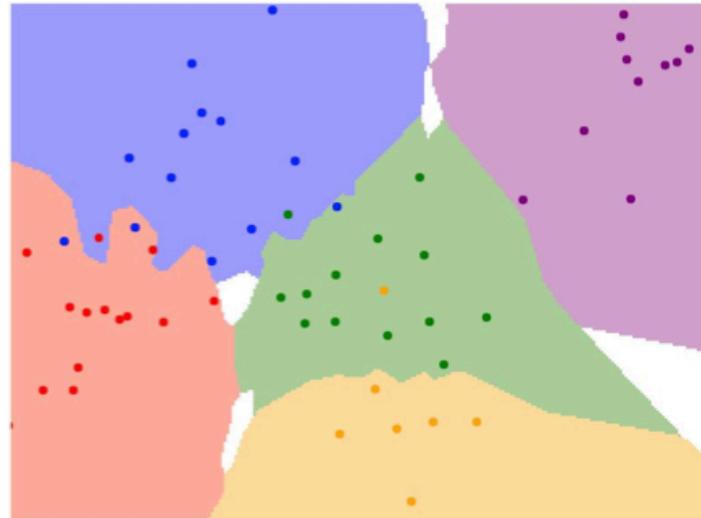
Predict the label  
of the most similar  
training image

# K-Nearest Neighbors

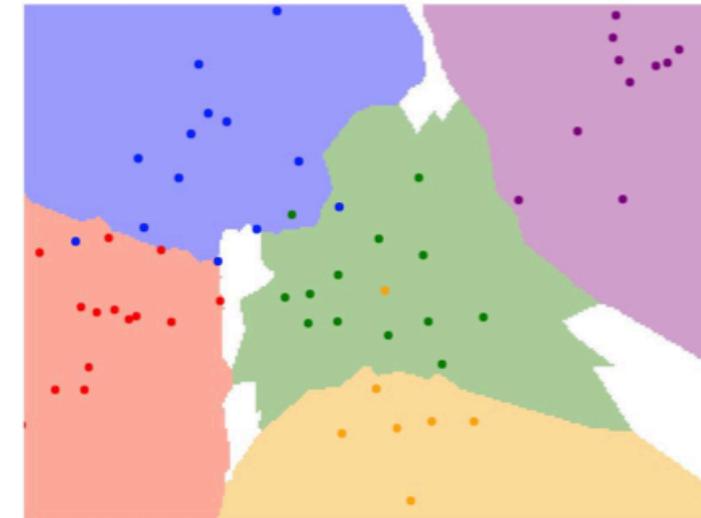
Instead of copying label from nearest neighbor,  
take **majority vote** from K closest points



$K = 1$



$K = 3$



$K = 5$

Similar strategy to RANSAC.

# Problems with Nearest Neighbor Classifier

- KNN with pixel distance never used.
  - Pixel distance as a metric is too sensitive to change in background/illumination/pose/viewpoint/occlusion/... that are not essential to the semantics.
- Very slow at test time.
- However, nearest neighbor-based techniques are still useful when the metrics is learned via a deep neural network and widely used in image retrieval, metric learning, 3D vision, and etc.

# Using CNN for Image Classification

- Things we need to take care:
  - Network architecture
  - Loss functions

# SoftMax vs. SVM

- For image classification, the most widely paradigm is **Softmax classifier** + **cross-entropy loss**.
- For binary classification, people also use SVM loss. However, the extended multiclass SVM loss is rarely used in current trend.

## Multiclass SVM loss:

Given an example  $(x_i, y_i)$  where  $x_i$  is the image and where  $y_i$  is the (integer) label,

and using the shorthand for the scores vector:  $s = f(x_i, W)$

the SVM loss has the form:

$$\begin{aligned} L_i &= \sum_{j \neq y_i} \begin{cases} 0 & \text{if } s_{y_i} \geq s_j + 1 \\ s_j - s_{y_i} + 1 & \text{otherwise} \end{cases} \\ &= \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1) \end{aligned}$$

# Softmax Classifier

- Also called multinomial logistic regression.



Want to interpret raw classifier scores as **probabilities**

$$s = f(x_i; W)$$

$$P(Y = k|X = x_i) = \frac{e^{s_k}}{\sum_j e^{s_j}}$$

Softmax  
Function

|      |            |
|------|------------|
| cat  | <b>3.2</b> |
| car  | 5.1        |
| frog | -1.7       |

# Softmax as a Non-Linear Activation

- SoftMax, also known as Softargmax or normalized exponential function, is a generalization of the **logistic function** to multiple dimensions.
- Definition:  $\sigma(z) : z \in \mathbb{R}^K \rightarrow (0,1)^K$  when  $K > 1$

$$\sigma(z)_i = \frac{\exp(\beta z_i)}{\sum_{j=1}^K \exp(\beta z_j)}$$

- $\beta = 1$  by default.

# Softmax as a Non-Linear Activation

- SoftMax, also known as Softargmax or normalized exponential function, is a generalization of the **sigmoid function** to multiple dimensions.
- Definition:  $\sigma(z) : z \in \mathbb{R}^K \rightarrow (0,1)^K$  when  $K > 1$

$$\sigma(z)_i = \frac{\exp(\beta z_i)}{\sum_{j=1}^K \exp(\beta z_j)}$$

- $\beta = 1$  by default.
- When  $\beta \rightarrow \infty$ , Softmax  $\rightarrow$  argmax.
- When  $K = 2$ ,  $\sigma(\begin{bmatrix} z \\ 0 \end{bmatrix})_1 = \text{Sigmoid}(z)$

# SoftMax Classifier



Want to interpret raw classifier scores as **probabilities**

$$s = f(x_i; W)$$

$$P(Y = k|X = x_i) = \frac{e^{s_k}}{\sum_j e^{s_j}}$$

Softmax Function

cat  
car  
frog

|      |
|------|
| 3.2  |
| 5.1  |
| -1.7 |

Unnormalized  
log-probabilities / logits

Probabilities  
must be  $\geq 0$

|       |
|-------|
| 24.5  |
| 164.0 |
| 0.18  |

unnormalized  
probabilities

exp

normalize

|      |
|------|
| 0.13 |
| 0.87 |
| 0.00 |

probabilities

# SoftMax Classifier



Want to interpret raw classifier scores as **probabilities**

$$s = f(x_i; W)$$

$$P(Y = k|X = x_i) = \frac{e^{s_k}}{\sum_j e^{s_j}}$$

Softmax Function

cat  
car  
frog

3.2  
5.1  
-1.7

Unnormalized  
log-probabilities / logits

Probabilities  
must be  $\geq 0$

24.5  
164.0  
0.18

unnormalized  
probabilities

exp

normalize

Probabilities  
must sum to 1

0.13  
0.87  
0.00

probabilities

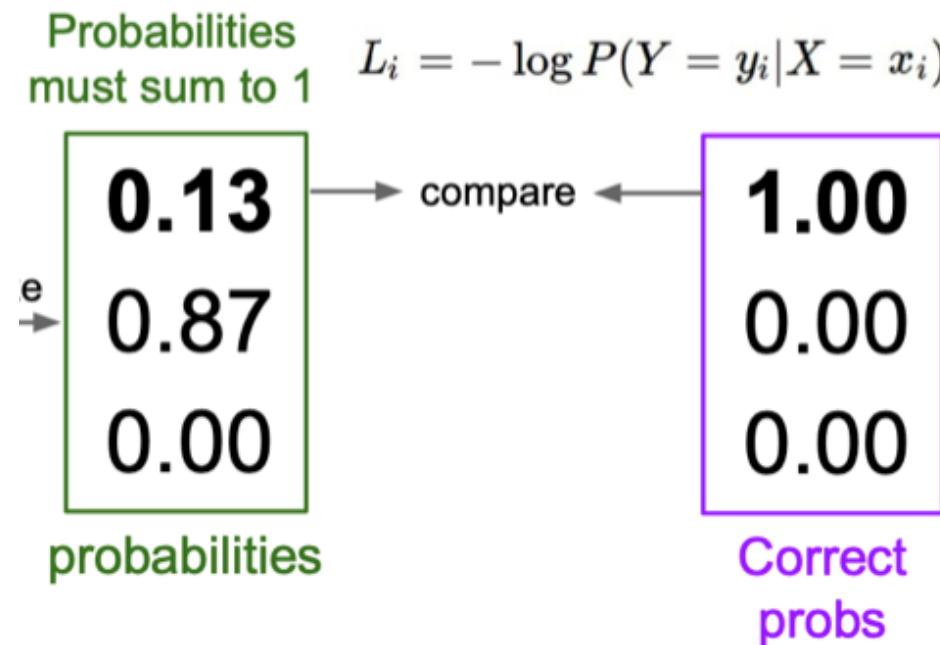
compare

1.00  
0.00  
0.00

Correct  
probs

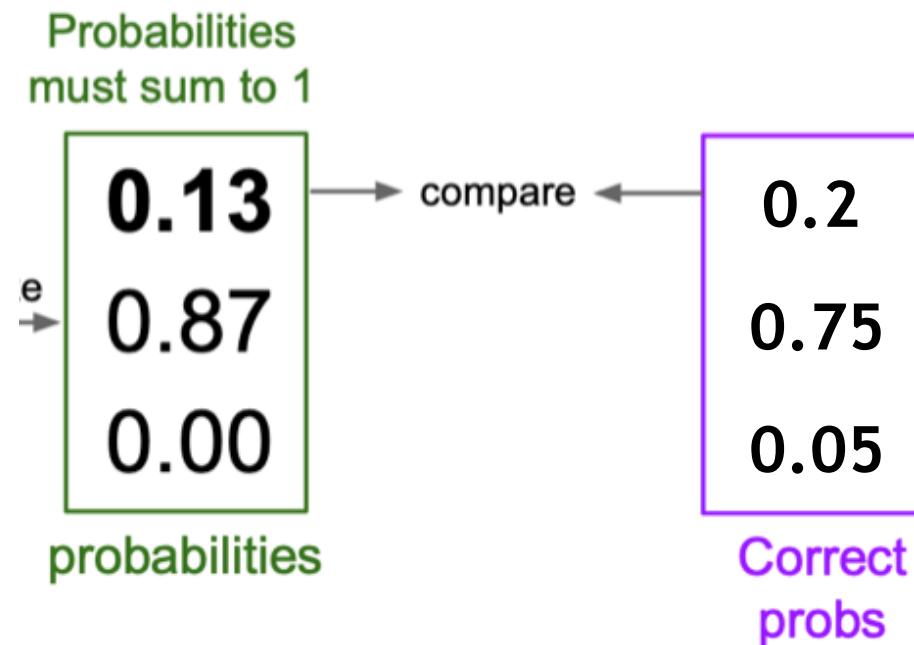
# Negative Log-likelihood Loss

- When correct probability is a one-hot vector, we can simply use negative log likelihood (NLL) loss.



# Loss between Two Discrete Distributions

- When correct probs is not one-hot, e.g. there is uncertainty in the label or the label is smoothed (label smoothing), we need to measure the difference between two distributions



Reference: [label smoothing](#)

# “Distance” between Two Distributions

- One widely used measure is Kullback-Leibler divergence  $D_{KL}(P \parallel Q)$ 
  - a measure of how one **probability distribution**  $Q$  is different from a second, reference probability distribution  $P$ .
  - For discrete probability distributions:

$$D_{\text{KL}}(P \parallel Q) = \sum_{x \in \mathcal{X}} P(x) \log \left( \frac{P(x)}{Q(x)} \right).$$

# Properties of KL Divergence

- Kullback-Leibler divergence  $D_{KL}(P \parallel Q)$

$$D_{\text{KL}}(P \parallel Q) = \sum_{x \in \mathcal{X}} P(x) \log \left( \frac{P(x)}{Q(x)} \right).$$

- $D_{KL}(P \parallel Q) \geq 0$  and  $D_{KL}(P \parallel Q) = 0$  only if  $P = Q$
- It is not a metric, since  $D_{KL}(P \parallel Q) \neq D_{KL}(Q \parallel P)$  and does not satisfy the triangle inequality.

# From KL Divergence to Cross-Entropy

$$\cdot D_{KL}(P || Q) = - \sum_{x \in \mathcal{X}} P(x) \log Q(x) - \left( - \sum_{x \in \mathcal{X}} P(x) \log P(x) \right)$$

$$H(P, Q)$$
$$H(P)$$

- Entropy  $H(P)$
- Cross entropy  $H(P, Q)$

# From KL Divergence to Cross-Entropy

$$\bullet D_{KL}(P || Q) = - \sum_{x \in \mathcal{X}} P(x) \log Q(x) - \left( - \sum_{x \in \mathcal{X}} P(x) \log P(x) \right)$$

$$H(P, Q) \quad H(P)$$

- Entropy  $H(P)$ , cross entropy  $H(P, Q)$
- When  $P$  is the ground truth prob.,  $Q$  is the predicted prob.,  $H(P)$  is a constant.
- **Cross entropy loss:**  $\mathcal{L}_{CE} = H(P, Q) = - \sum_{x \in \mathcal{X}} P(x) \log Q(x)$

# Properties of Cross Entropy Loss

$$\mathcal{L}_{CE} = H(P, Q) = - \sum_{x \in \mathcal{X}} P(x) \log Q(x)$$

- With random initialization,  $\mathcal{L}_{CE} \approx 1/(\# \text{ of classes})$
- No upper bound.
- Minimum = 0.

# Summary of Softmax Classifier



3.2

5.1

-1.7

Want to interpret raw classifier scores as **probabilities**

$$s = f(x_i; W)$$

$$P(Y = k | X = x_i) = \frac{e^{s_k}}{\sum_j e^{s_j}}$$

Softmax  
Function

Maximize probability of correct class

$$L_i = -\log P(Y = y_i | X = x_i)$$

Putting it all together:

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



# Introduction to Computer Vision

Next week: Lecture 7,  
Deep Learning IV