Deep Learning Fundamentals

Haozhe Xie

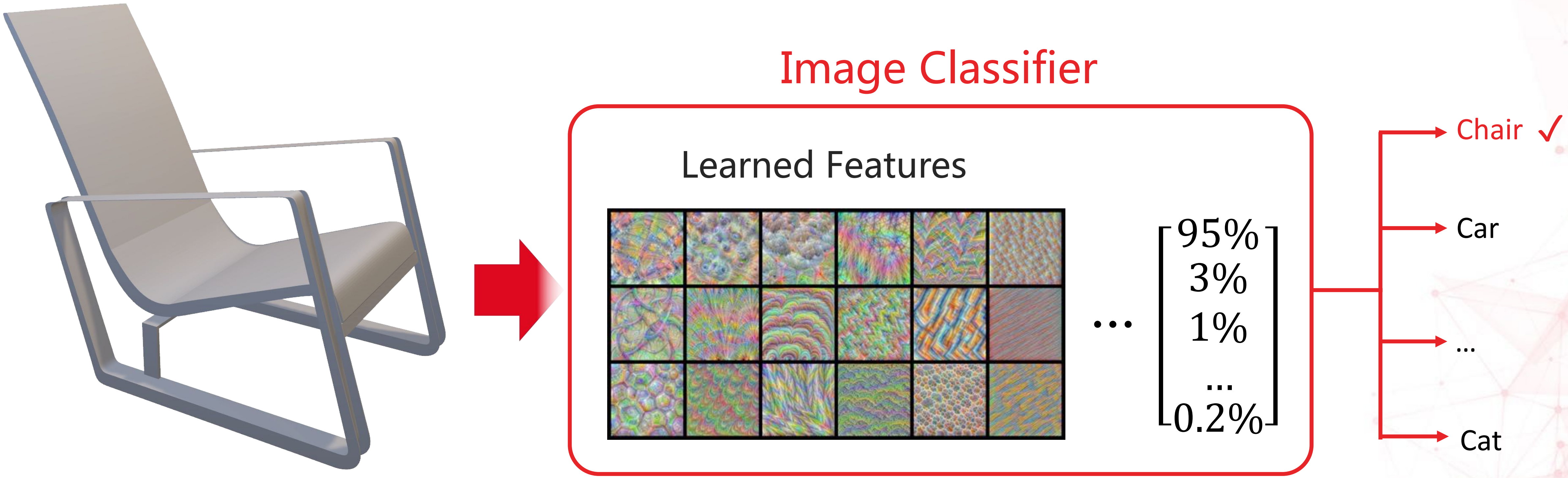
cshzxie@gmail.com

Outline

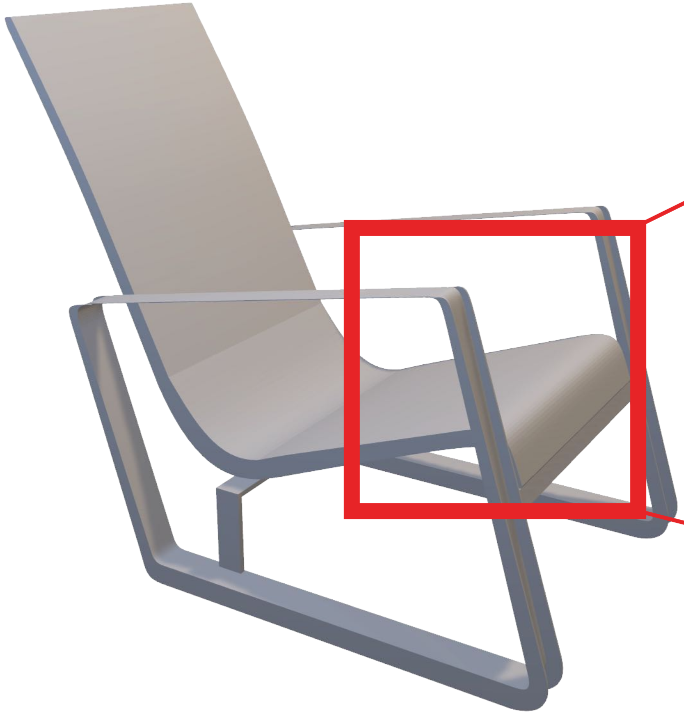
Part I

K-Nearest Neighbor

What is Image Classification?



The Problem: Semantic Gap



```
[[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  93 120 131 127 100  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  80  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  80 101 124 126 119 101 107 114 131 119]
 [ 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  96 117 150 144 120 115 104 107 102  93  87  81  72  79]
 [123 107  96  86  83 112 153 149 122 109 104  75  80 107 112  99]
 [122 121 102  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

Challenges: Viewpoint variation

Other Challenges

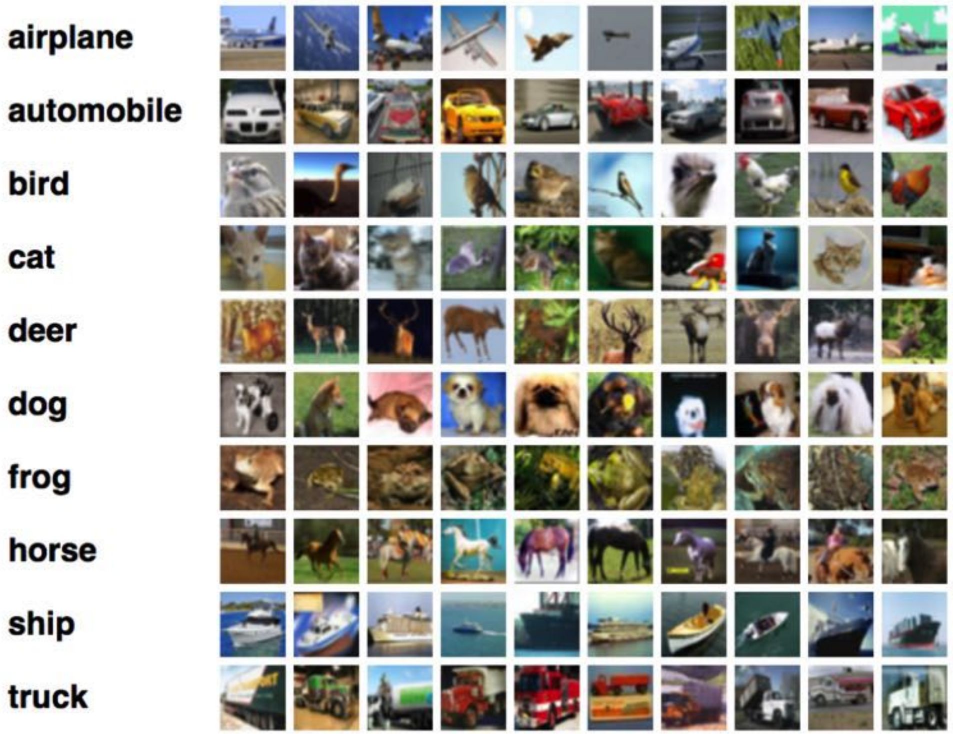
Illumination

Deformation

Occlusion

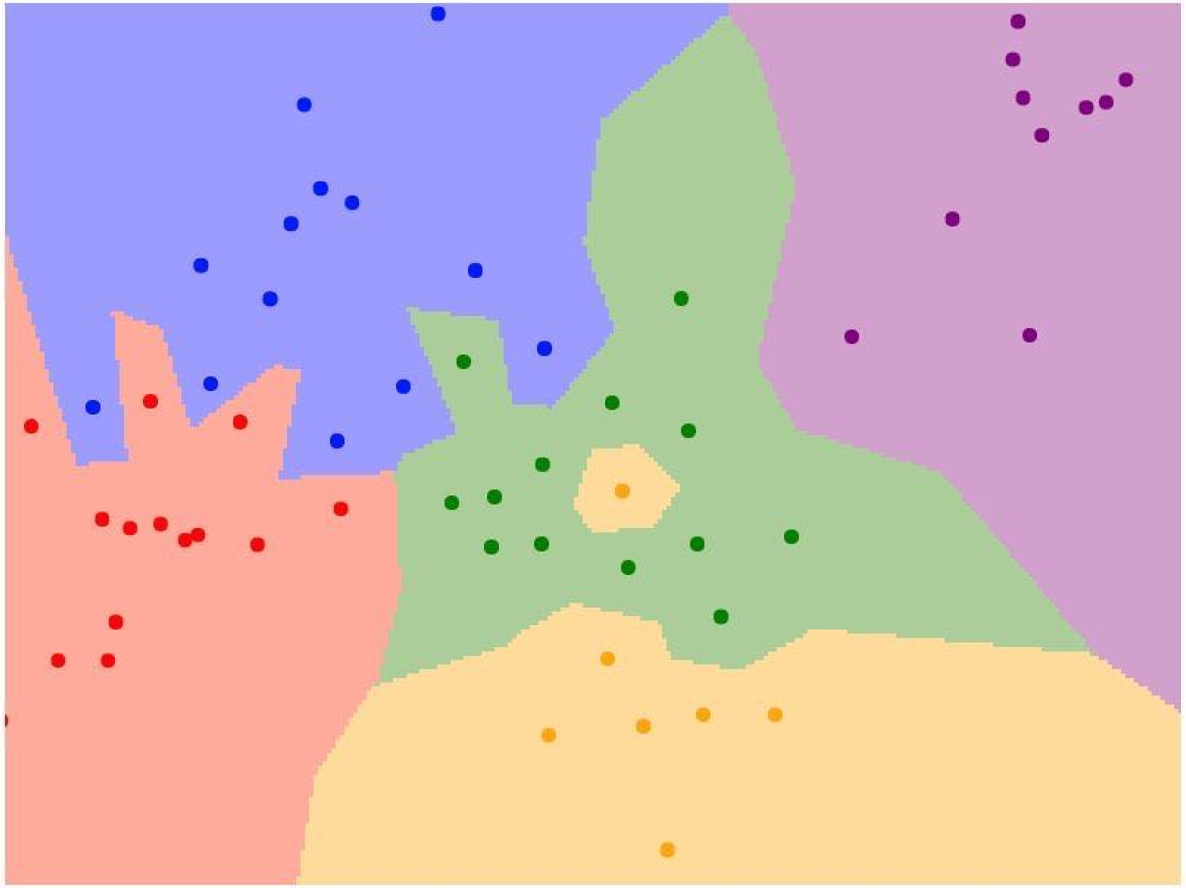
How to Classify Images?

- Data-Driven Approach
  - Collect a dataset of images and labels
  - Use Machine Learning to train a classifier
  - Evaluate the classifier on new images



Naïve Imager Classifier: Nearest Neighbor

```python
import numpy as np

class NearestNeighbor:
  def __init__(self):
    pass

  def train(self, X, y):
    """ X is N x D where each row is an example. Y is 1-dimension of size N """
    # the nearest neighbor classifier simply remembers all the training data
    self.Xtr = X
    self.ytr = y

  def predict(self, X):
    """ X is N x D where each row is an example we wish to predict label for """
    num_test = X.shape[0]
    # lets make sure that the output type matches the input type
    Ypred = np.zeros(num_test, dtype = self.ytr.dtype)

    # loop over all test rows
    for i in xrange(num_test):
      # find the nearest training image to the i'th test image
      # using the L1 distance (sum of absolute value differences)
      distances = np.sum(np.abs(self.Xtr - X[i,:]), axis = 1)
      min_index = np.argmin(distances) # get the index with smallest distance
      Ypred[i] = self.ytr[min_index] # predict the label of the nearest example

    return Ypred
```

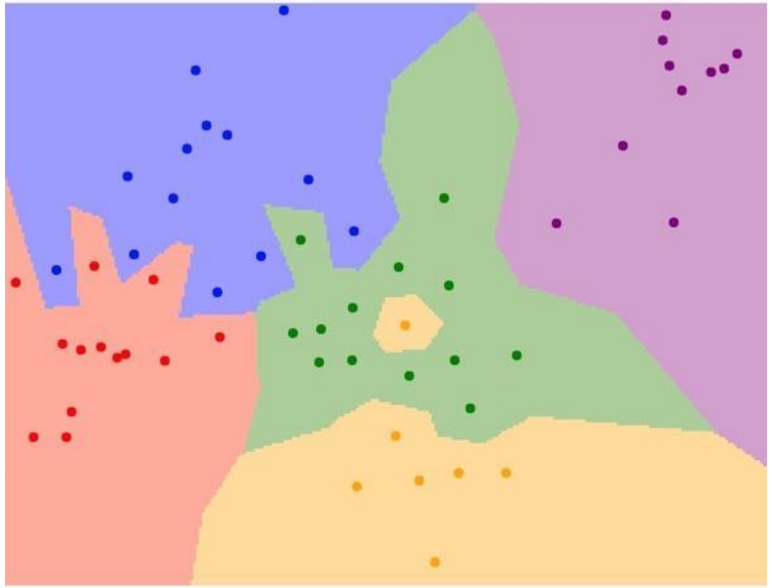


Distance Metric:

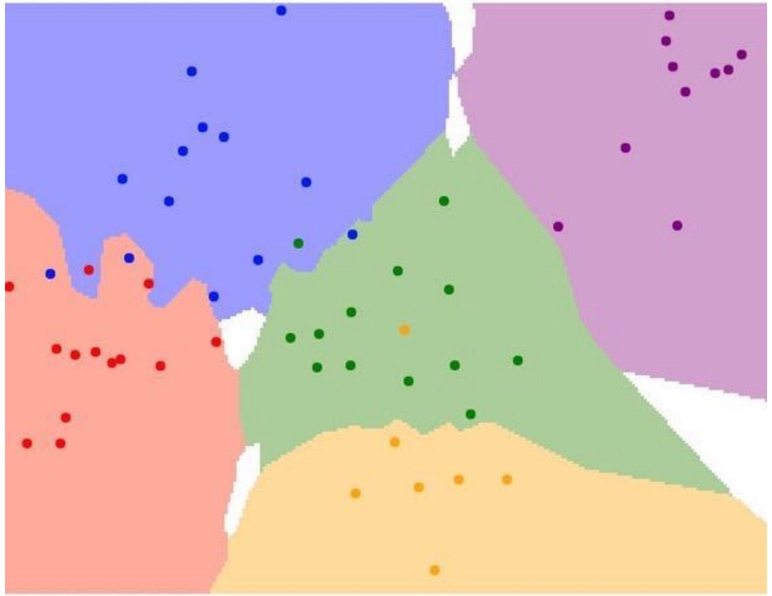
$$d_1(I_1, I_2) = \sum_p \left| I_1^p - I_2^p \right|$$

Naïve Imager Classifier: K-Nearest Neighbor

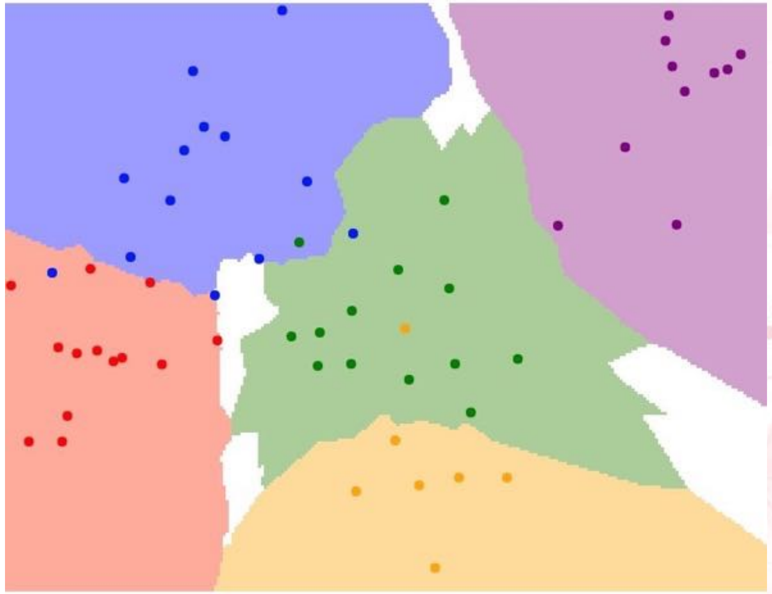
Instead of copying label from nearest neighbor, take majority vote from K closest points.



k=1



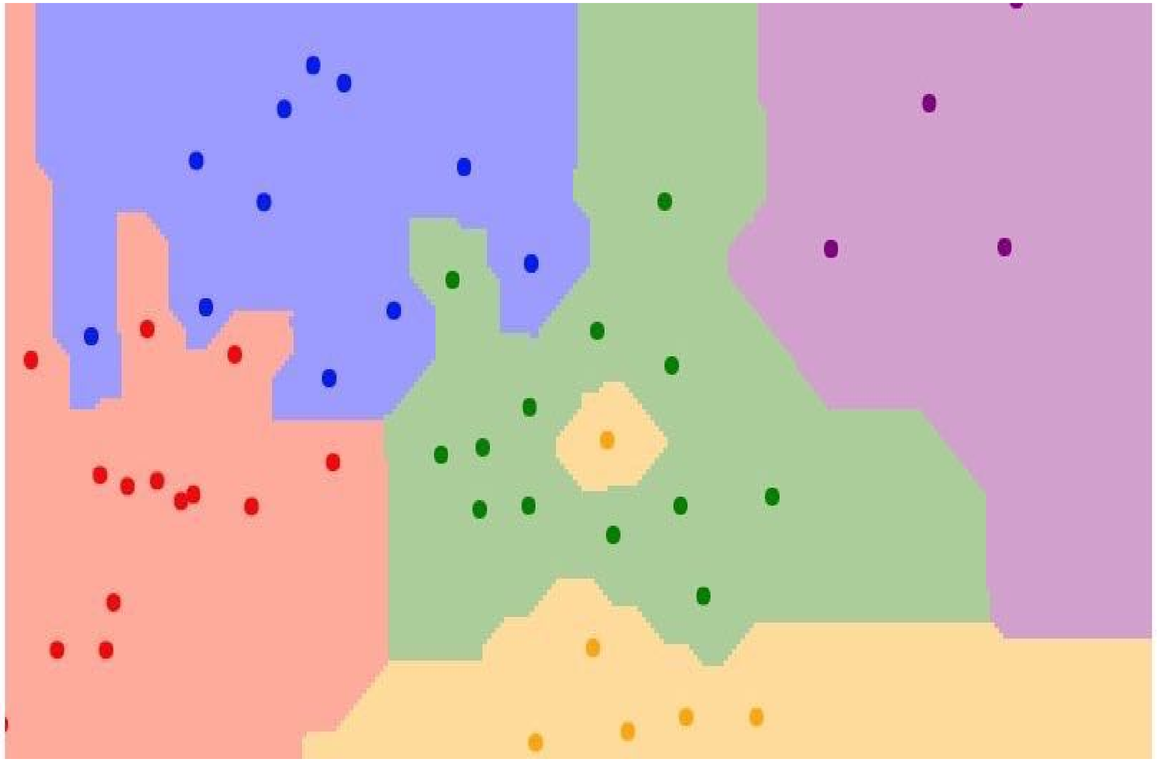
k=3

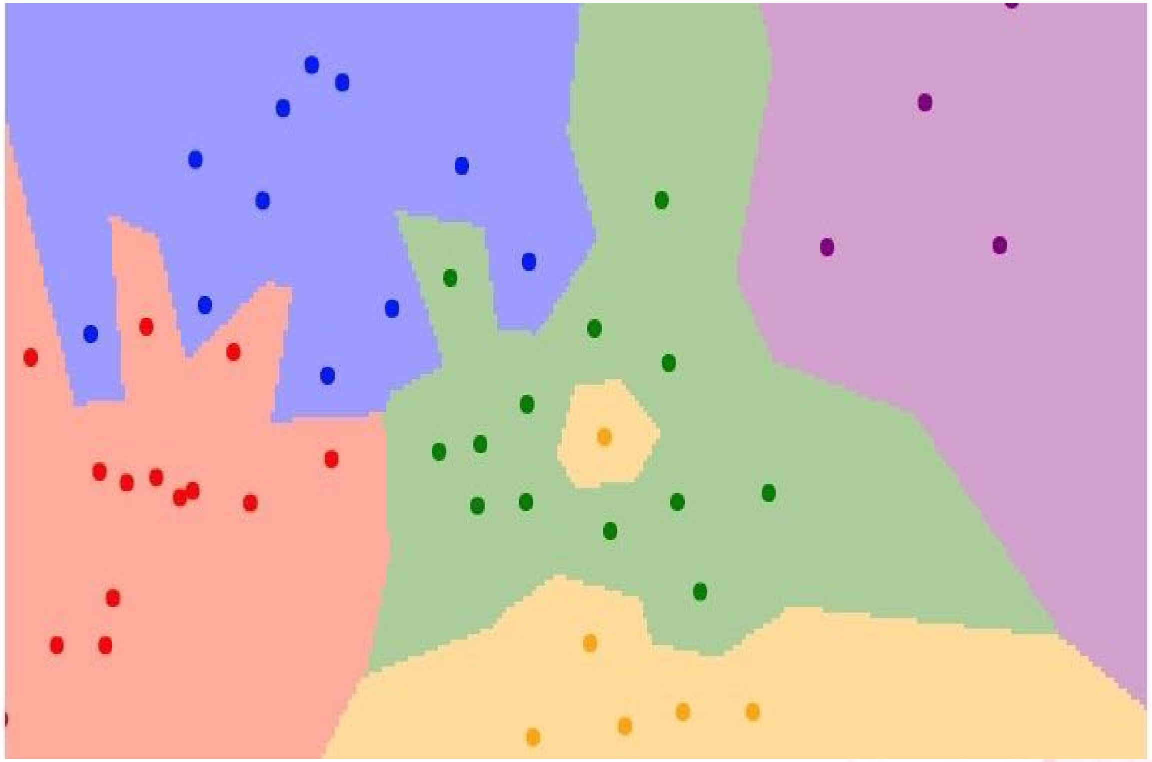


k=5

Naïve Imager Classifier: Nearest Neighbor

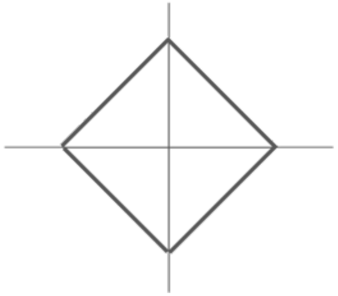
K-Nearest Neighbors: Distance Metrics





L1 (Manhattan) distance

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



L2 (Euclidean) distance

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

K-Nearest Neighbors: Hyperparameters

- What is the best value of k to use?
- What is the best distance to use?

- These are **hyperparameters**: choices about the algorithm that we set rather than learn
- Very problem-dependent.
- Must try them all out and see what works best.

K-Nearest Neighbors: Hyperparameters

- Idea #1: Choose hyperparameters that work best on the data

  BAD: K = 1 always works perfectly on training data

- Idea #2: Split data into train and test, choose hyperparameters that work best on test data

  BAD: No idea how algorithm will perform on new data

- Idea #3: Split data into train, validation, and test; choose hyperparameters on validation and evaluate on test

  Better!

K-Nearest Neighbors: Hyperparameters

- Idea #4: Cross-Validation: Split data into folds, try each fold as validation and average the results

Fold 1	Fold 2	Fold 3	Fold 4	Fold 5	Fold 6	Test
Fold 1	Fold 2	Fold 3	Fold 4	Fold 5	Fold 6	Test
Fold 1	Fold 2	Fold 3	Fold 4	Fold 5	Fold 6	Test

Note: Useful for small datasets, but not used too frequently in deep learning

Drawbacks of K-Nearest Neighbors

- k-Nearest Neighbor on images never used
  - Very slow at test time
  - Distance metrics on pixels are not informative

Original

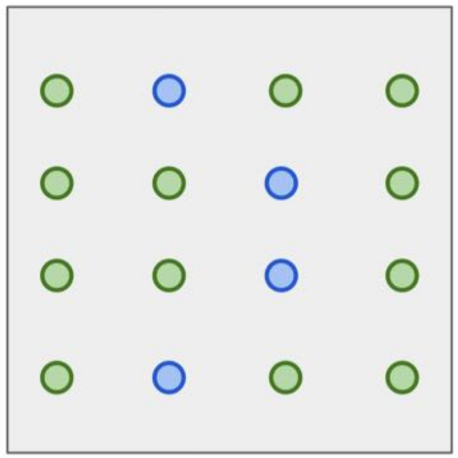
Shifted

Tinted

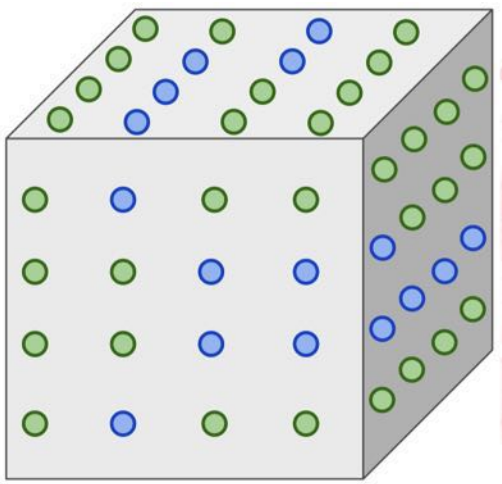
Drawbacks of K-Nearest Neighbors

- k-Nearest Neighbor on images never used
  - Very slow at test time
  - Distance metrics on pixels are not informative
  - Curse of dimensionality

Dimension = 1
Points = 4



Dimension = 2
Points = 4^2



Dimension = 3
Points = 4^3

K-Nearest Neighbors: Summary

- In image classification we start with a **training set** of images and labels, and must predict labels on the **test set**;
- The K-Nearest Neighbors classifier predicts labels based on nearest training examples;
- Distance metric and K are hyperparameters;
- Choose hyperparameters using the validation set; only run on the test set once at the very end.

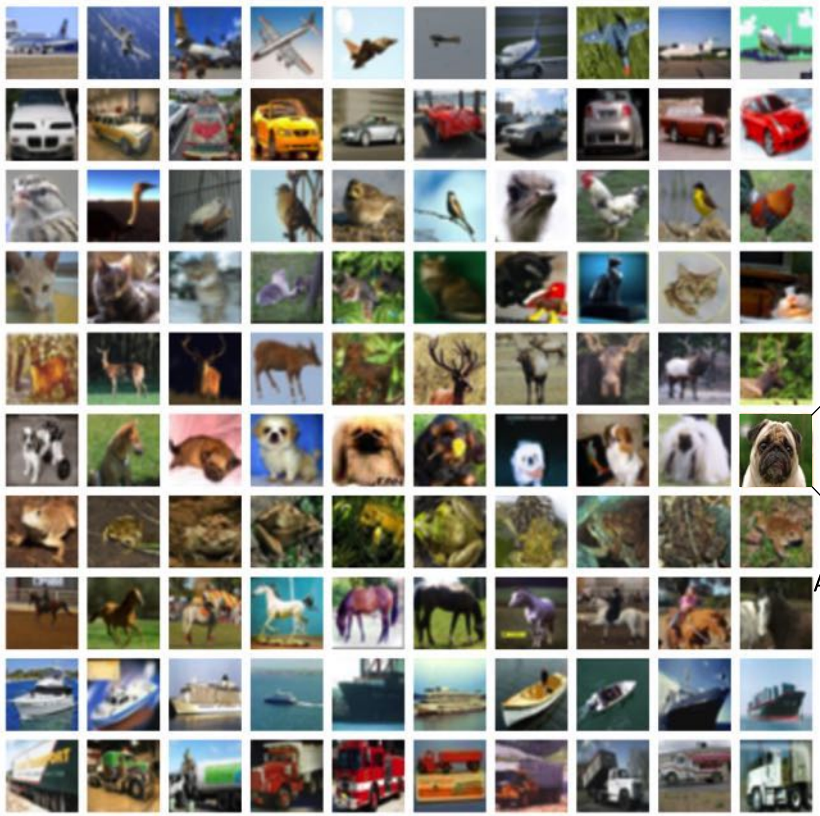
Part II

Linear Classifier

Linear Classifier

Recall CIFAR 10

airplane
automobile
bird
cat
deer
dog
frog
horse
ship
truck



50,000 training images of sizes 32x32x3
10,000 test images.

Image

Array of 32x32x3 numbers
3072 numbers in total

$$f(x, W)$$

10 numbers giving class scores

Parameters or weights

Linear Classifier

$$\overset{10 \times 1}{\boxed{f(x, W)}} = \underset{10 \times 3072}{\boxed{W}}\,\overset{3072 \times 1}{\boxed{x}} + \underset{10 \times 1}{b}$$

Image

Array of 32x32x3 numbers
3072 numbers in total

$$\text{Image} \rightarrow f(x, W) \rightarrow \text{10 numbers giving class scores}$$

Parameters
or weights

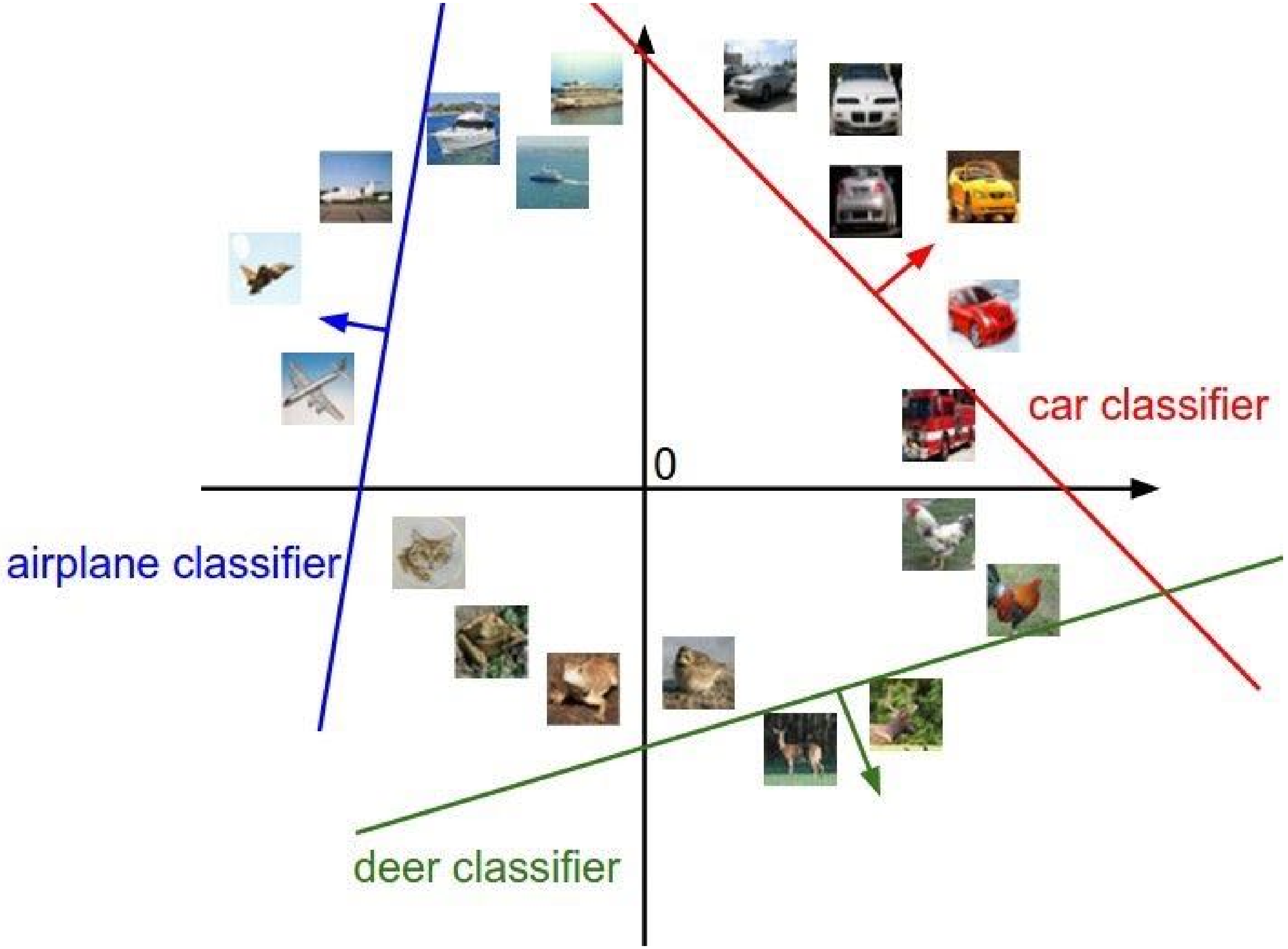
An Example of Linear Classifier

Example with an image with 4 pixels, and 3 classes (cat/dog/ship)

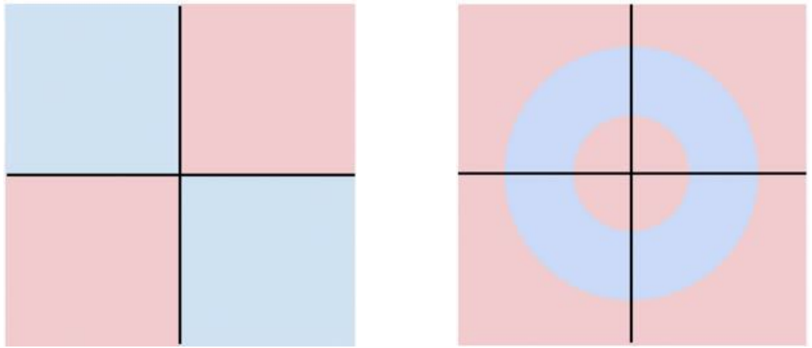
56	231
24	2

$$
\begin{bmatrix} 0.2 & -0.5 & 0.1 & 2.0 \\ 1.5 & 1.3 & 2.1 & 0.0 \\ 0 & 0.25 & 0.2 & -0.3 \end{bmatrix}
\times
\begin{bmatrix} 56 \\ 231 \\ 24 \\ 2 \end{bmatrix}
+
\begin{bmatrix} 1.1 \\ 3.2 \\ -1.2 \end{bmatrix}
=
\begin{array}{c} \begin{bmatrix} -96.8 \\ 437.9 \\ 61.95 \end{bmatrix} \end{array}
\begin{array}{l} \text{Cat Score} \\ \text{Dog Score} \\ \text{Ship Score} \end{array}
$$

Linear Classifiable



- Not Linearly Classifiable Cases



Part III

Loss Functions and Optimization

Things TODO in Linear Classifier

- Quantify the Classification Scores
  - Loss Function
- Find the Parameters Effectively
  - Optimization

Loss Functions

- What is Loss Function?
  - A loss function tells how good our current classifier is.

- Given a dataset of examples: $\{(x_i, y_i)\}_{i=1}^{N}$
- Loss over the dataset is a sum of loss over examples:

$$L = \frac{1}{N}\sum_{i} L_i(f(x_i, W), y_i)$$

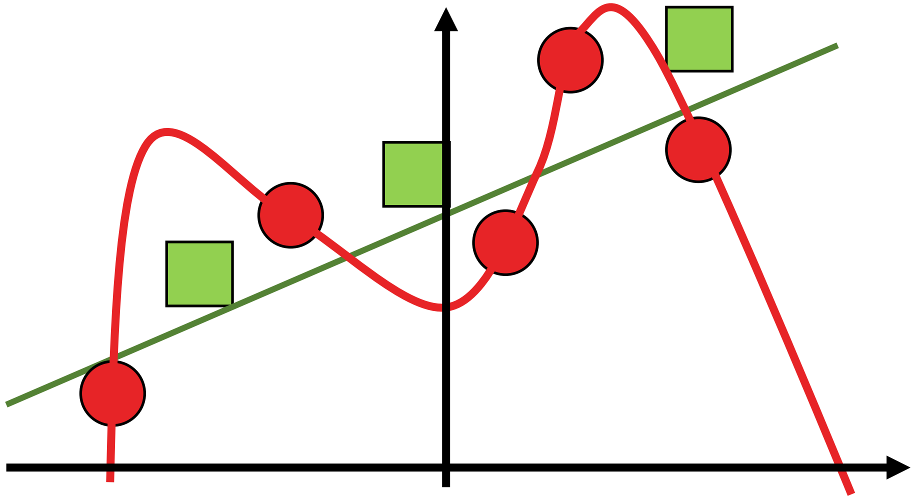
Regularization

$$\underbrace{L = \frac{1}{N}\sum_i L_i(f(x_i, W), y_i)}_{} + \lambda R(W)$$

Regularization: Model should be "simple", so it works on test data

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

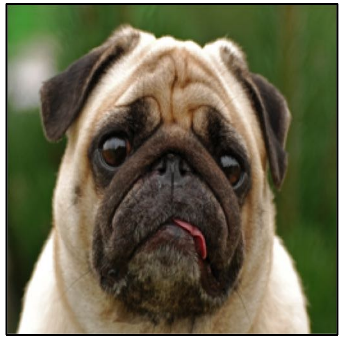
Data Loss: Model predictions should match training data



Regularization

- Common Regularization
  - L2 Regularization: $R(W) = \sum_{j=0}^{p} \beta_j^2$
  - L1 Regularization: $R(W) = \sum_{j=0}^{p} |\beta_j|$
  - Elastic net (L1 + L2): $R(W) = \alpha \sum_{j=0}^{p} \beta_j^2 + \sum_{j=0}^{p} |\beta_j|$
  - Max norm regularization
  - Dropout
  - Fancier: Batch normalization, stochastic depth

Softmax Classifier



3.2	Cat Score
5.1	Dog Score
-1.7	Ship Score

- Scores = unnormalized log probabilities of the classes

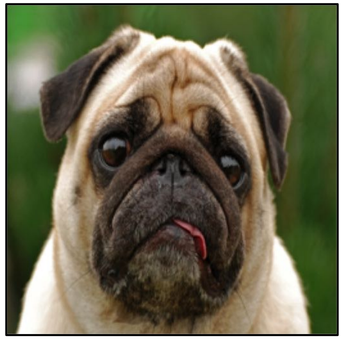
$$P(Y=k|X=x_i)=\frac{e^{s_k}}{\sum_j e^{s_j}} \qquad \text{where } s=f(x_i,\mathbf{W})$$

Want to maximize the log likelihood (loss function) to minimize the negative log likelihood of the correct class:

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

- In Summary: $L_i=-\log(\frac{e^{s_{y_i}}}{\sum_j e^{s_j}})$

Softmax Classifier



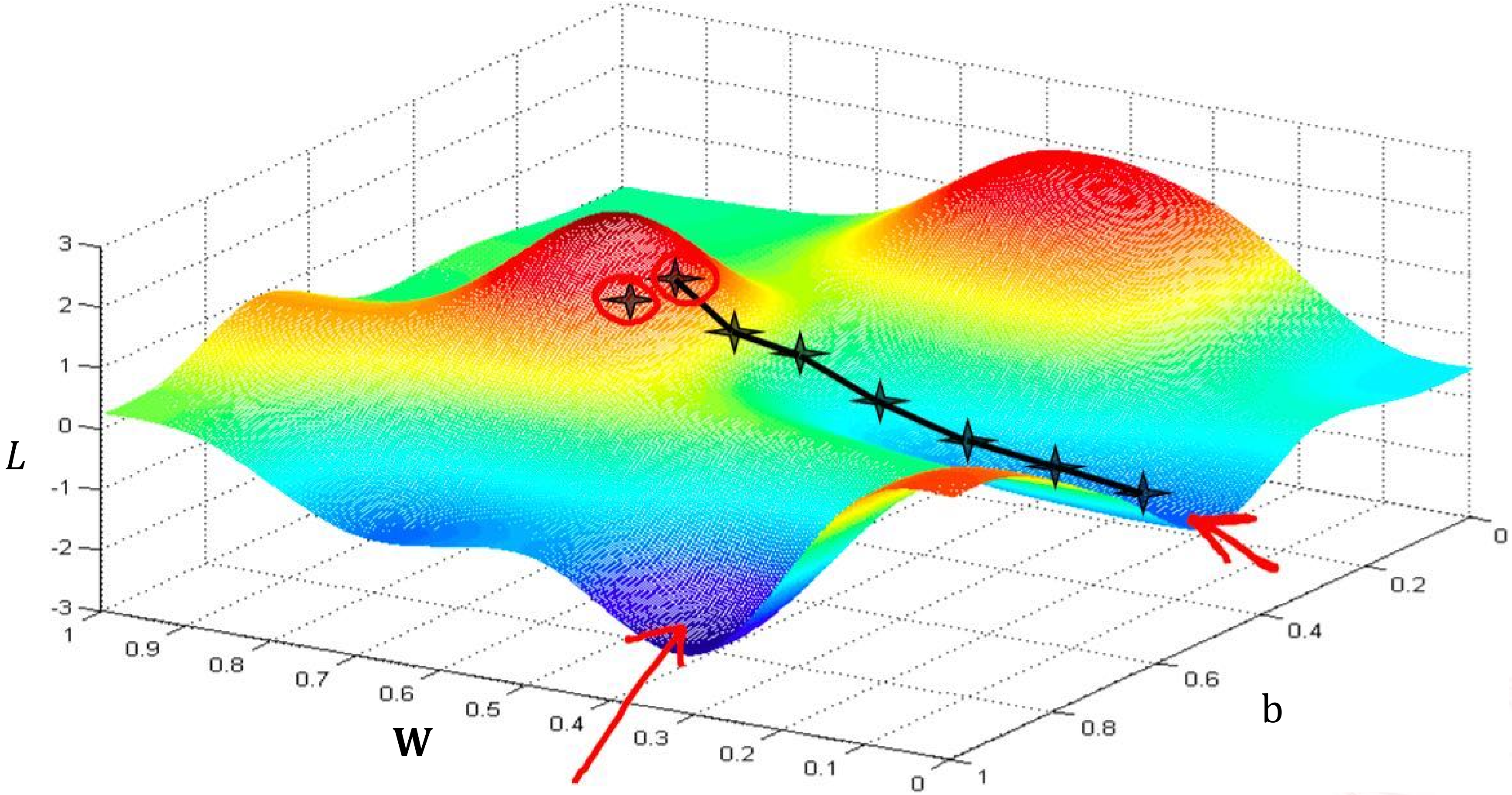
$$L_i = -\log\left(\frac{e^{s_{y_i}}}{\sum_j e^{s_j}}\right)$$

	unnormalized log probabilities	exp →	unnormalized probabilities	normalize →	normalized probabilities
Cat	3.2		24.5		0.13
Dog	5.1		164.0		0.87
Ship	-1.7		0.18		0.00

$$L_i = -\log(0.87) = 0.06$$

Optimization

- How to find the best **W** and b?



Optimization: Gradient Decent

- Stochastic Gradient Descent (SGD)
  - $L(W) = \frac{1}{N}\sum_{i=1}^{N} L_i(x_i, y_i, W) + \lambda R(W)$
  - $\nabla_W L(W) = \frac{1}{N}\sum_{i=1}^{N} \nabla_W L_i(x_i, y_i, W) + \lambda \nabla_W R(W)$

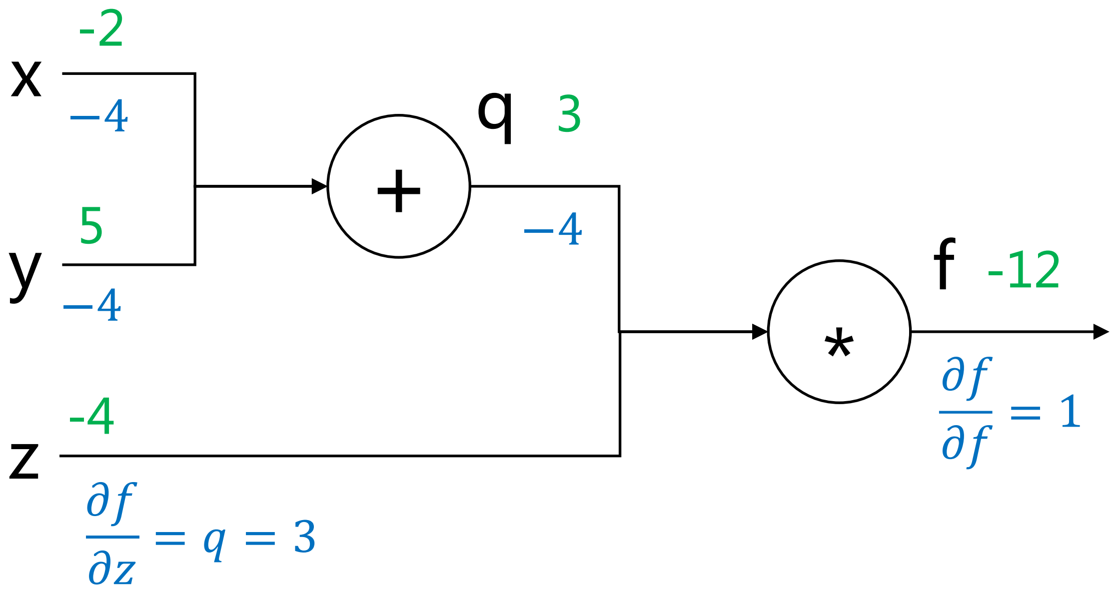
```
# Vanilla Minibatch Gradient Descent
while True:
    data_batch = sample_training_data(data, 64)
    weights_grad = eval_gradient(loss_func, data_batch, weights)
    weights -= weights_grad * learning_rate
```

Part IV

Backpropagation and Neural Networks

Backpropagation

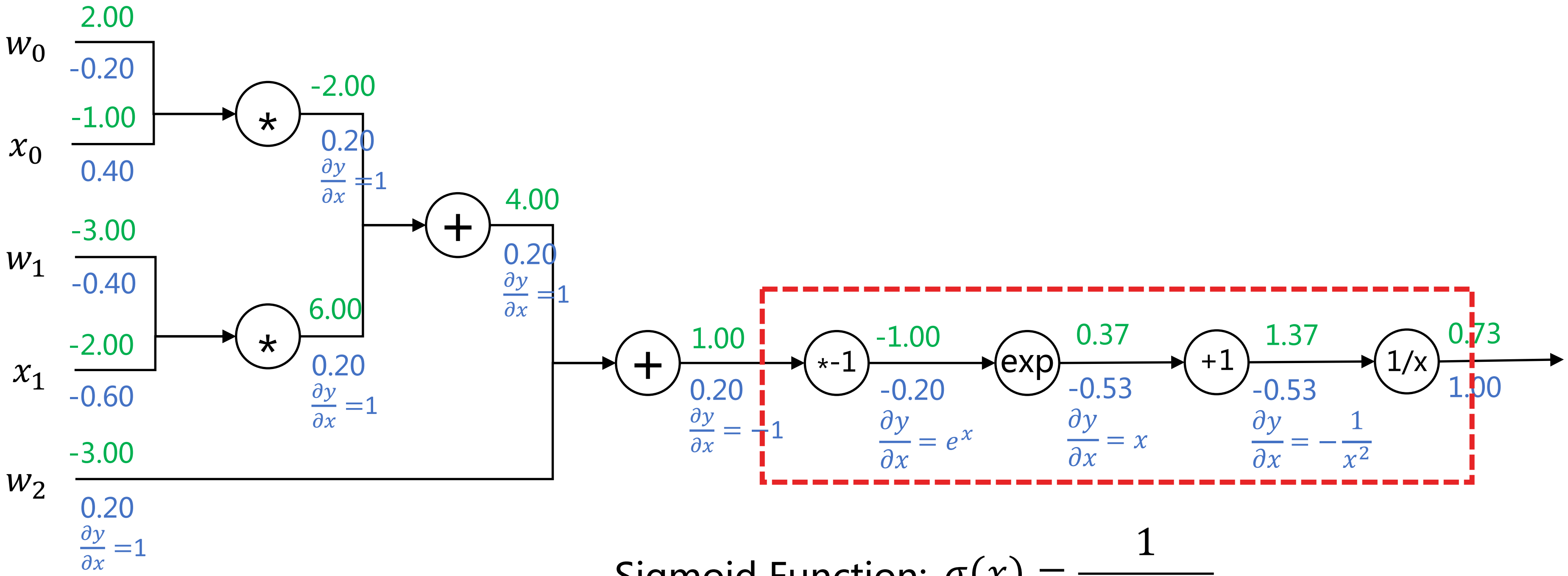
- How to get the gradient?
- $f(x, y, z) = (x + y)z$



- Given x = -2, y = 5, z = -4
- $q = x + y \qquad \frac{\partial q}{\partial x} = 1 \quad \frac{\partial q}{\partial y} = 1$
- $f = qz \qquad \frac{\partial f}{\partial q} = z \quad \frac{\partial f}{\partial z} = q$
- $\frac{\partial f}{\partial x} = \frac{\partial f}{\partial q}\frac{\partial q}{\partial x} \qquad \frac{\partial f}{\partial y} = \frac{\partial f}{\partial q}\frac{\partial q}{\partial y}$

Backpropagation

- $f(\mathrm{W},\mathrm{x}) = \frac{1}{1+e^{-(w_0x_0+w_1x_1+w_2)}}$



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

Backpropagation

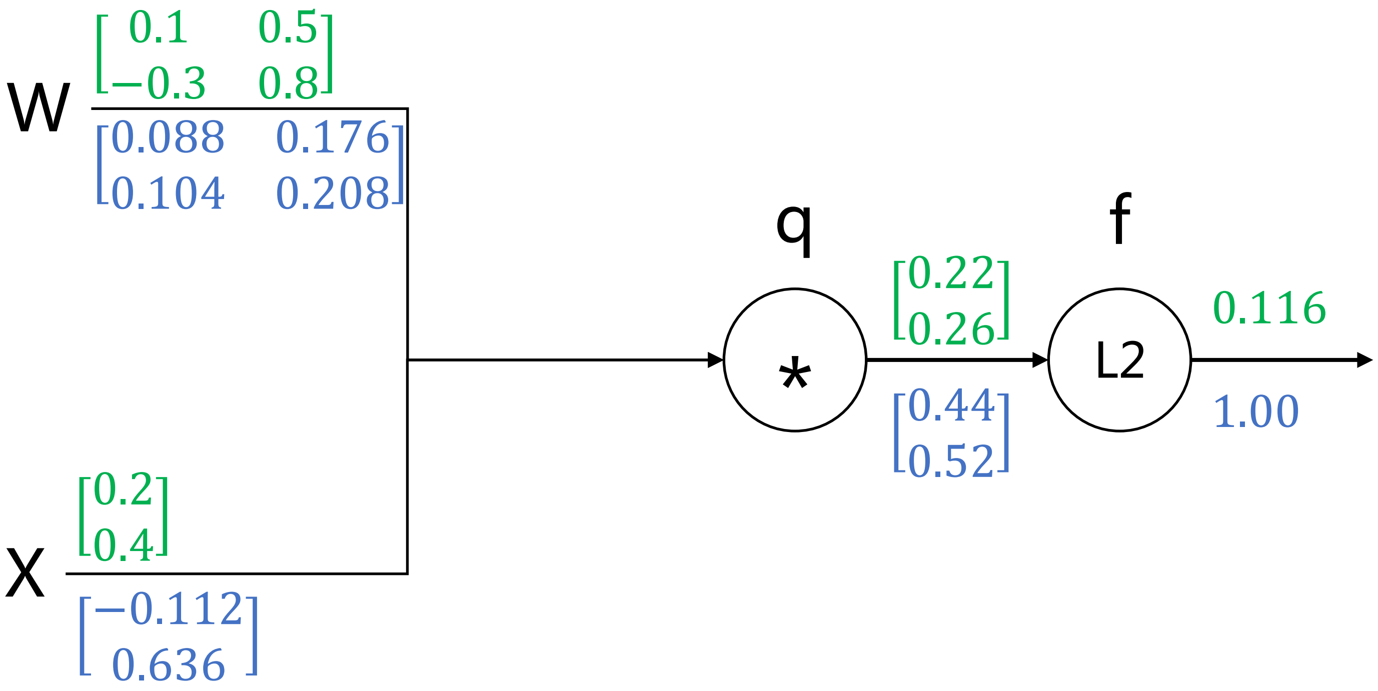
- Let $z = x + y$. What if x and y are vectors?
  - $x = (x_1, x_2)$, $y = (y_1, y_2)$
  - $z = (x_1 + y_1, x_2 + y_2)$
  - $\frac{\partial z_1}{\partial x_1} = 1$, $\frac{\partial z_1}{\partial x_2} = 0$, $\frac{\partial z_2}{\partial x_1} = 0$, $\frac{\partial z_1}{\partial x_2} = 1$
  - $\boxed{\frac{\partial z}{\partial x} = \begin{bmatrix} \frac{\partial z_1}{\partial x_1} & \frac{\partial z_1}{\partial x_2} \\ \frac{\partial z_2}{\partial x_1} & \frac{\partial z_1}{\partial x_2} \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}}$ Jacobian matrix

Backpropagation

- Let $y = \max(0, x)$ and x is a vector of size 4096. What is the size of the Jacobian matrix?
  - 4096×4096
- What the size of the Jacobian matrix if we use a minibatch of size 100?
  - 409600×409600
- What does the Jacobian matrix look like?
  - $\begin{bmatrix} 1 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & 1 \end{bmatrix}$

Backpropagation

- $f(x, \mathbf{W}) = \|\mathbf{W}x\|^2 = \sum_{i=1}^{n}(\mathrm{W}x)_i^2$



$$f(q) = \|q\|^2 = q_1^2 + q_2^2 + \cdots + q_n^2$$

$$\frac{\partial f}{\partial q_i} = 2q_i$$

$$q = \begin{bmatrix} q_1 \\ \vdots \\ q_n \end{bmatrix} = \mathbf{W}x = \begin{pmatrix} W_{(1,1)}x_1 + \cdots + W_{(1,n)}x_n \\ \vdots \\ W_{(n,1)}x_1 + \cdots + W_{(n,n)}x_n \end{pmatrix}$$

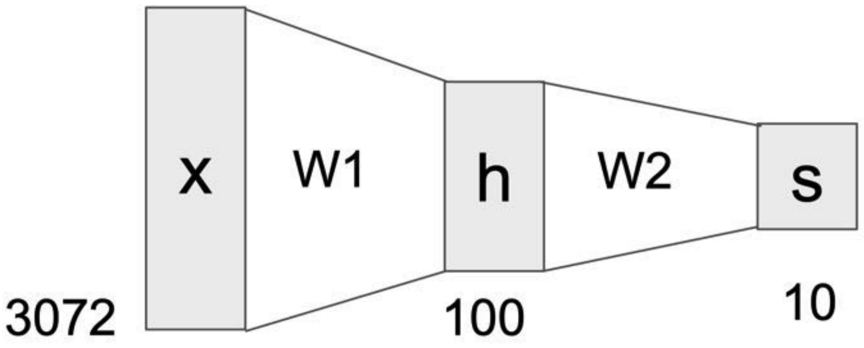
$$\frac{\partial q_k}{\partial W_{(i,j)}} = \mathbf{1}_{k=i}x_j \quad \frac{\partial f}{\partial W_{(i,j)}} = \mathbf{1}_{k=i}x_j \cdot 2\mathrm{q_i}$$

$$\frac{\partial q_k}{\partial x_i} = W_{(k,i)} \qquad \frac{\partial f}{\partial x_i} = \sum_k \frac{\partial f}{\partial q_k}\frac{\partial q_k}{\partial x_i}$$

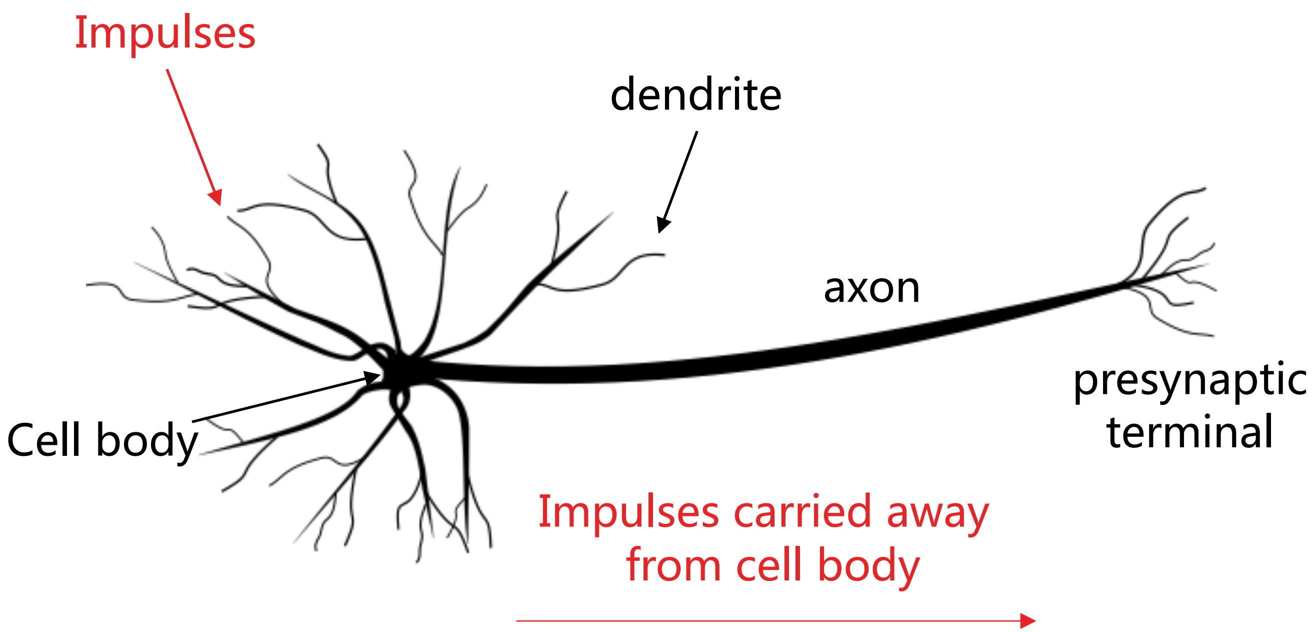
$$= \sum_k 2q_k W_{(k,i)}$$

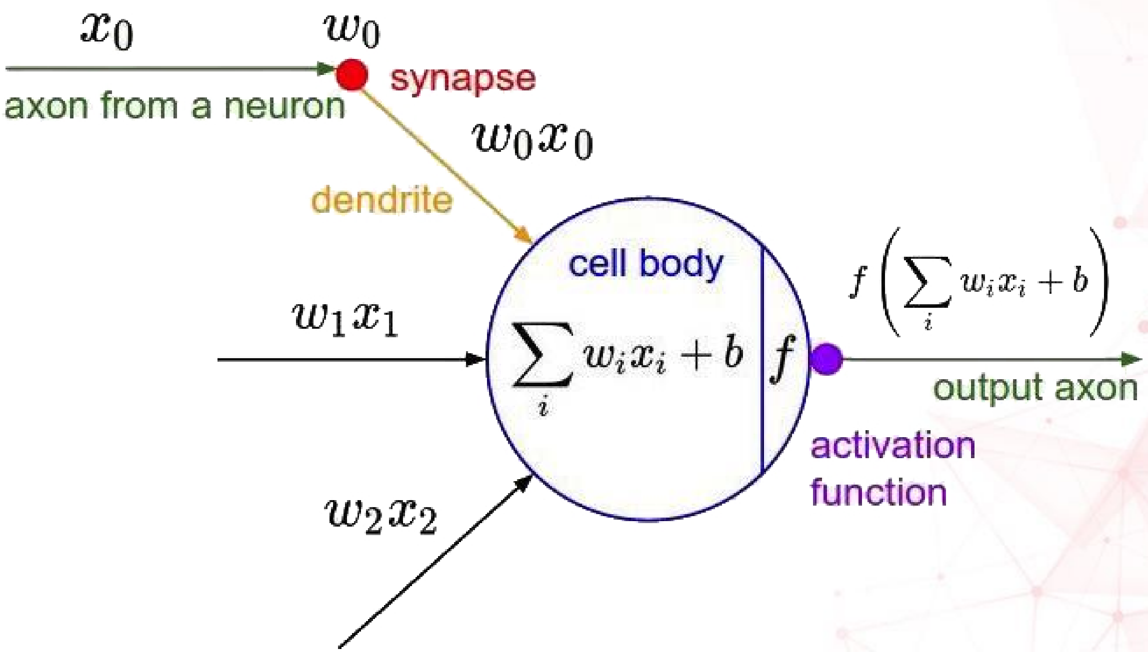
Neural Networks

- Linear Score Function
  - $f = \mathbf{W}x$
- 2-Layer Neural Network
  - $f = \mathbf{W_2} \max(0, \mathbf{W_1}x)$
- 3-Layer Neural Network
  - $f = \mathbf{W_3} \max(\mathbf{W_2} \max(0, \mathbf{W_1}x))$

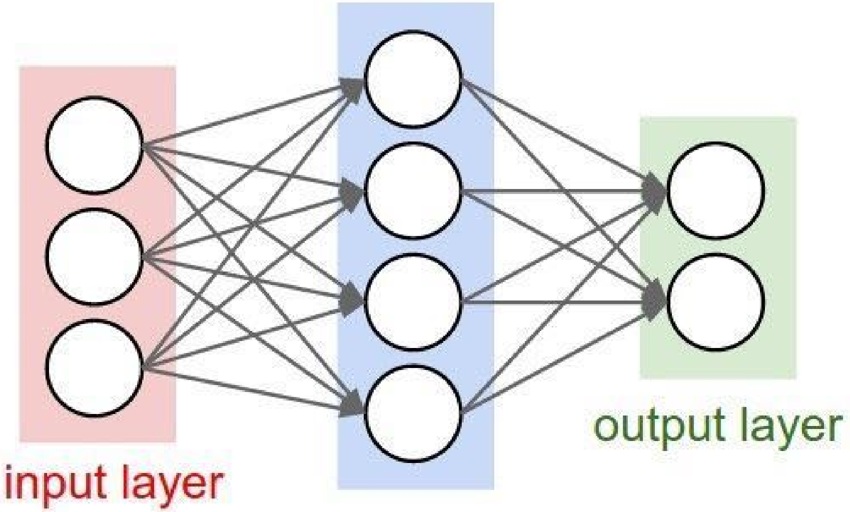


Neural Networks

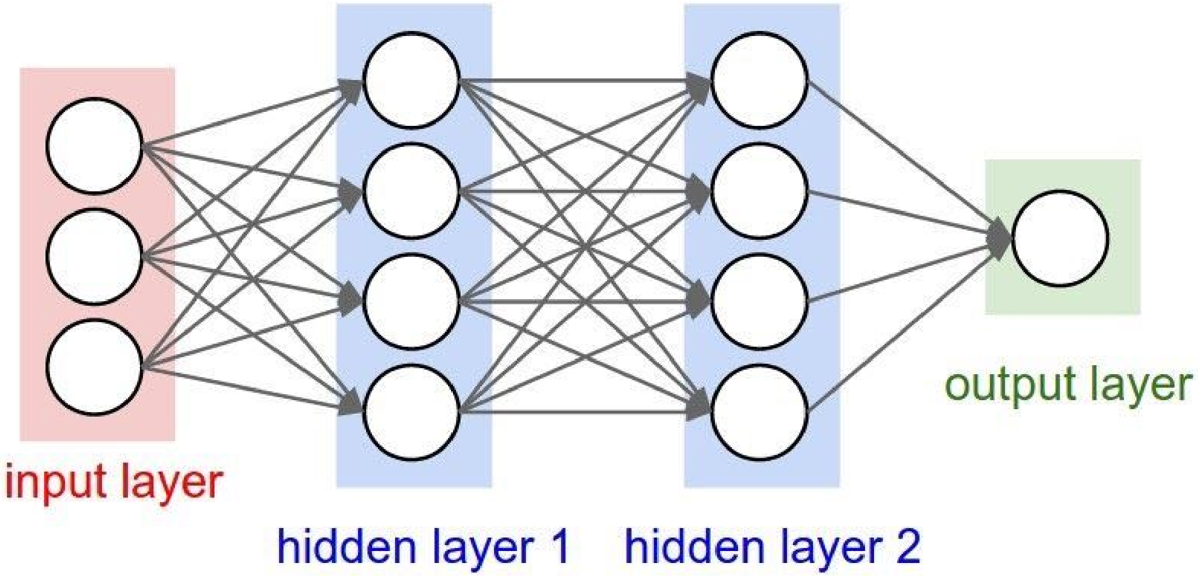




Neural networks: Architectures



Fully Connected Layers



- “2-layer Neural Net”
- “1-hidden-layer Neural Net”

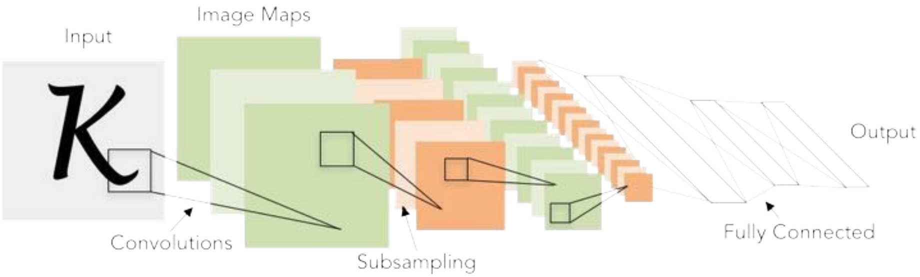
- “3-layer Neural Net”
- “2-hidden-layer Neural Net”

Part V

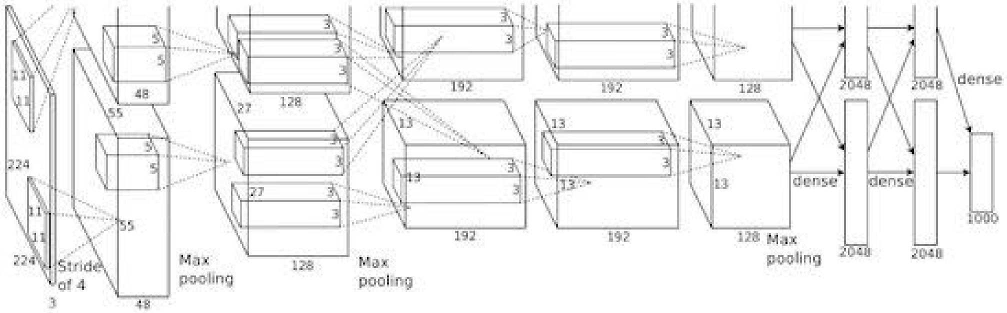
Convolutional Neural Networks

Important Events of CNNs

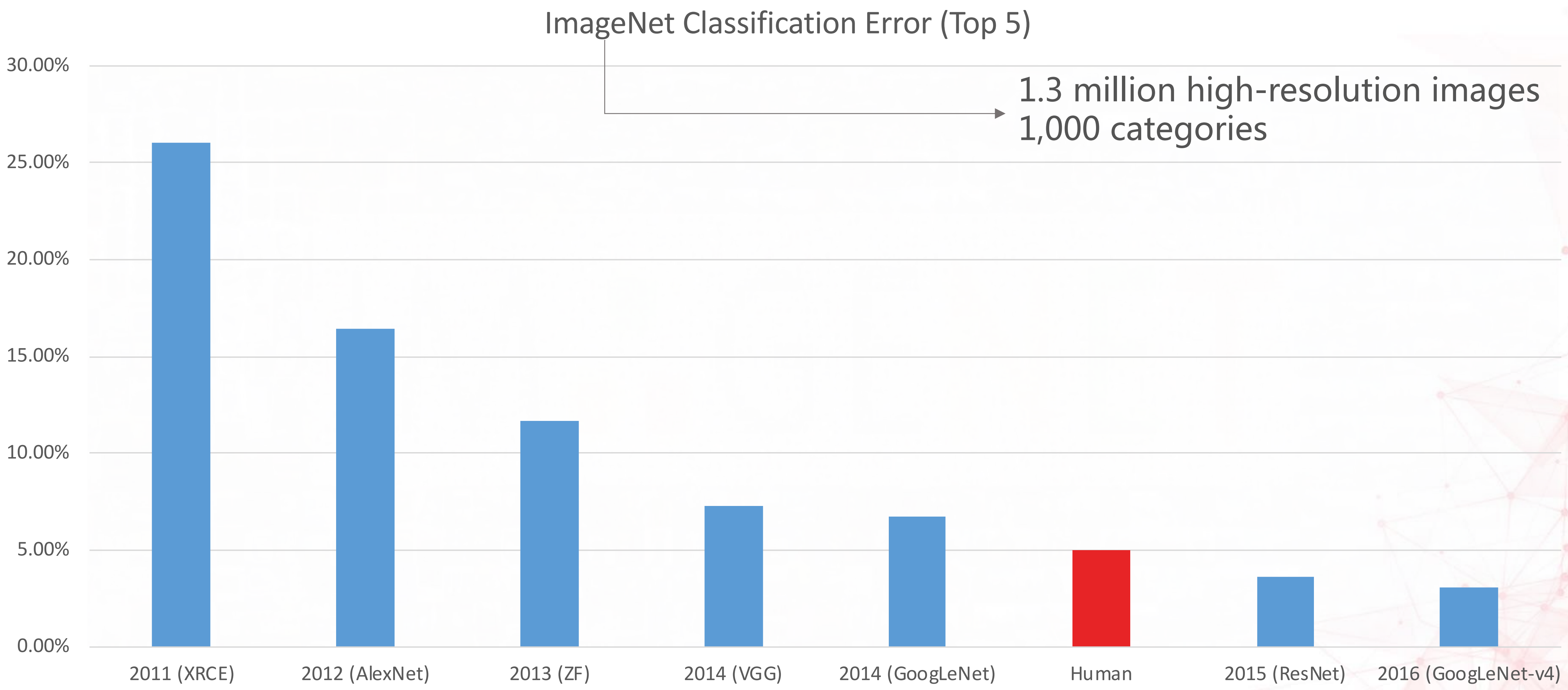
- LeNet-5
  - Used for Document Classification (1998)



- AlexNet
  - Used for Image Classification (2012)

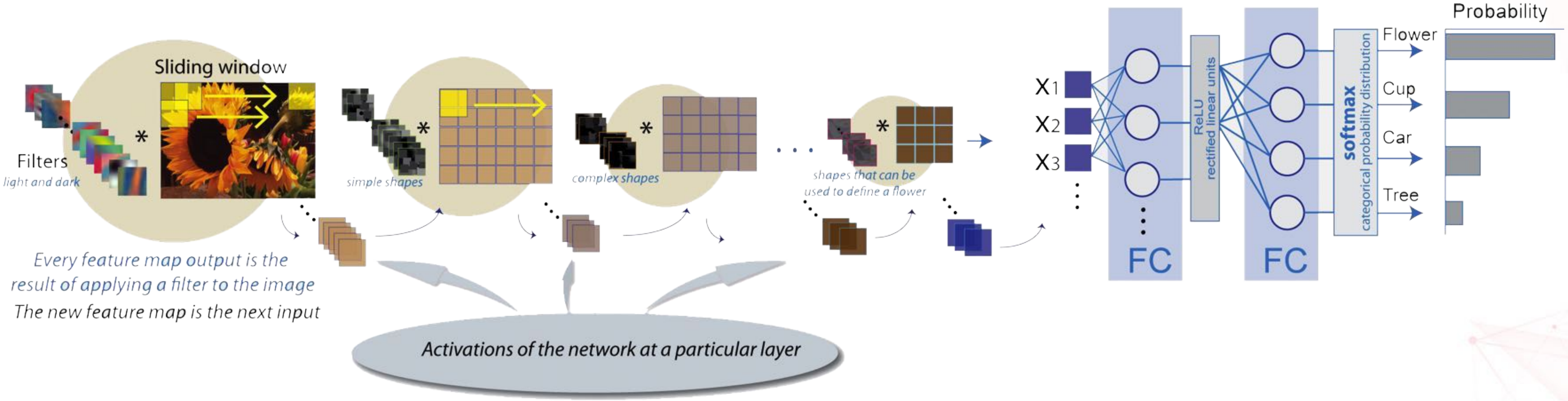


Fast-forward to Today: CNNs are Everywhere



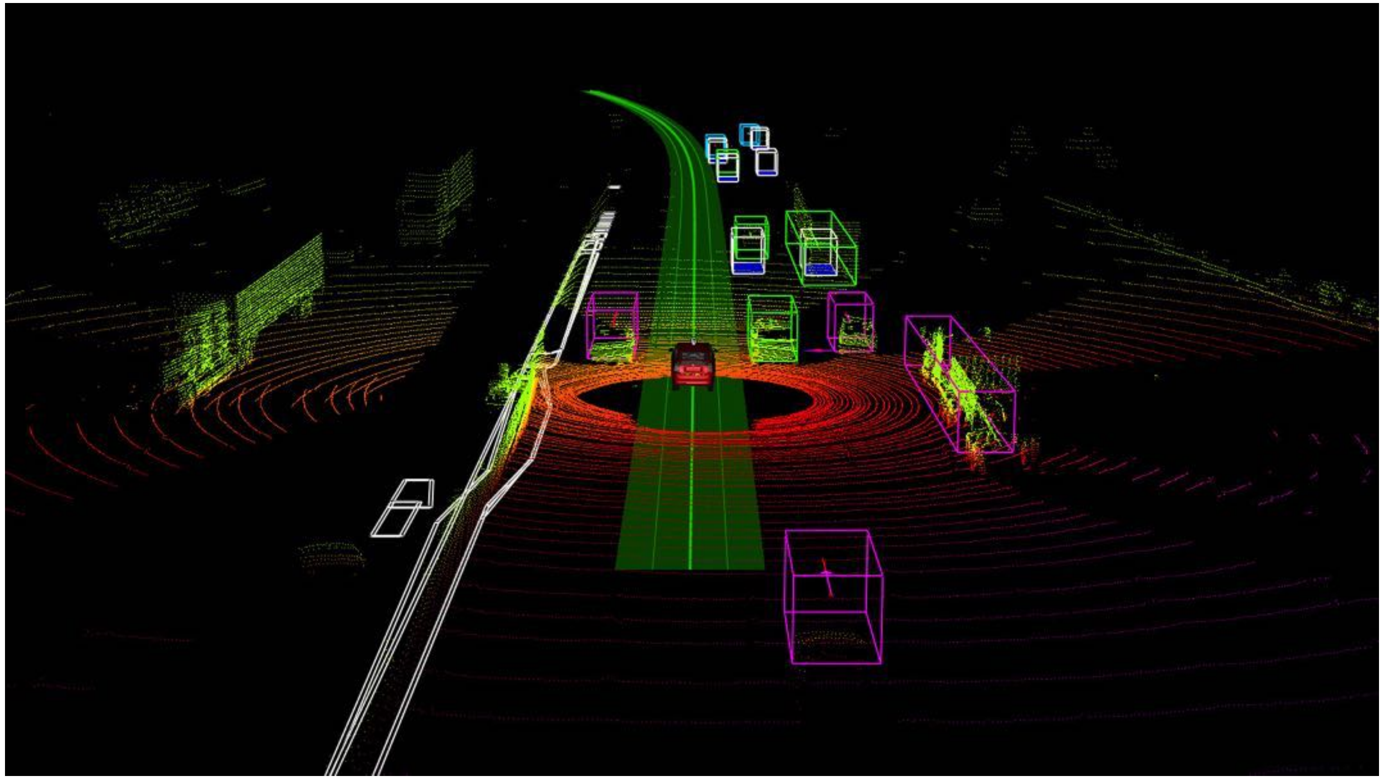
Fast-forward to Today: CNNs are Everywhere

- Image Classification



Fast-forward to Today: CNNs are Everywhere

- Object Detection in 2D/3D



Fast-forward to Today: CNNs are Everywhere

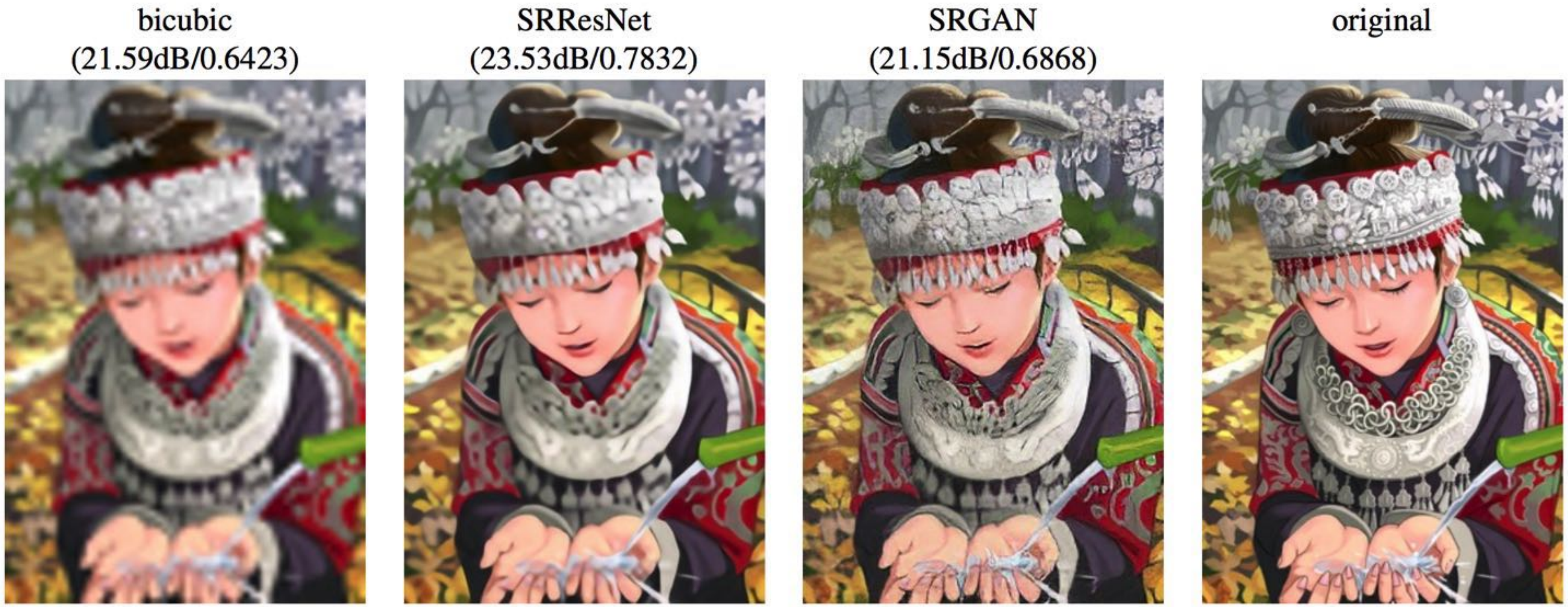
- Semantic Segmentation

Fast-forward to Today: CNNs are Everywhere

- Pose Estimation

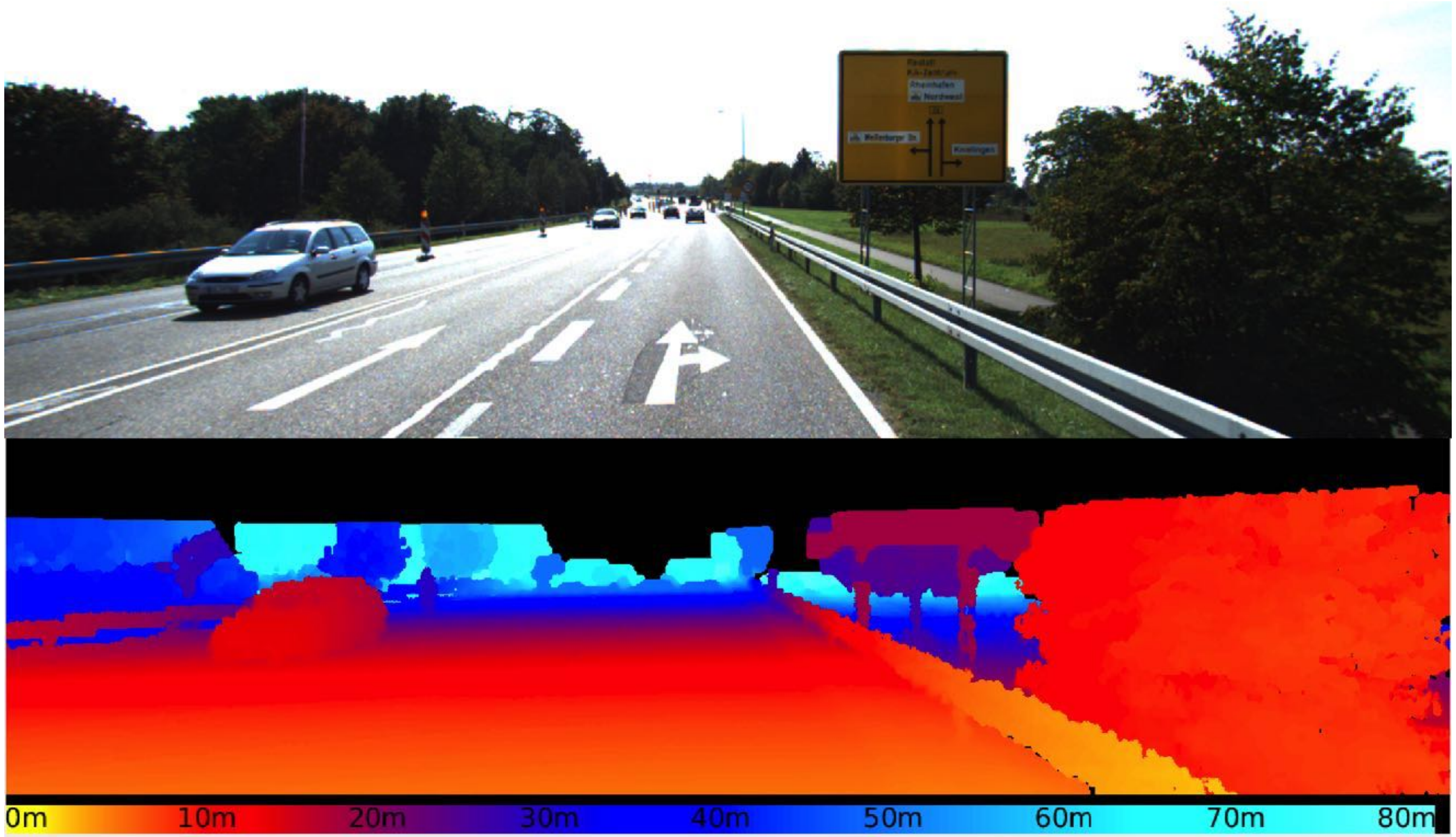
Fast-forward to Today: CNNs are Everywhere

- Image Super Resolution



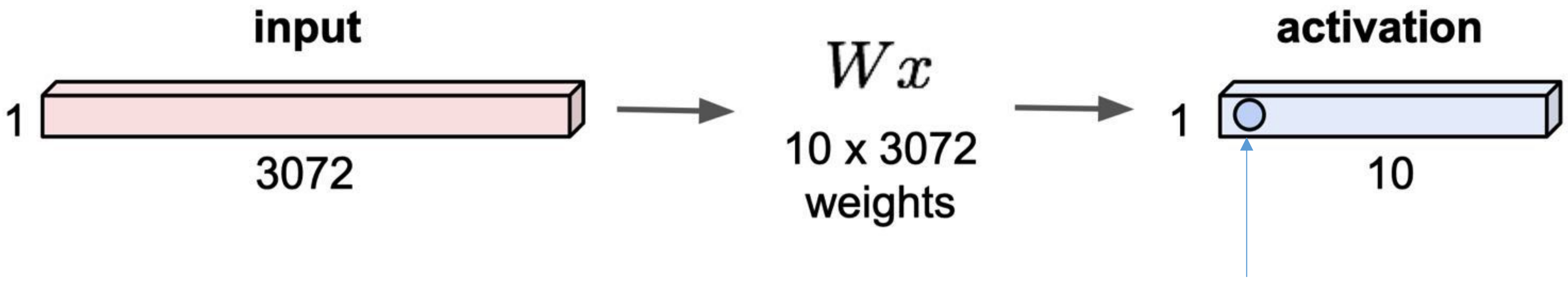
Fast-forward to Today: CNNs are Everywhere

- Depth Estimation



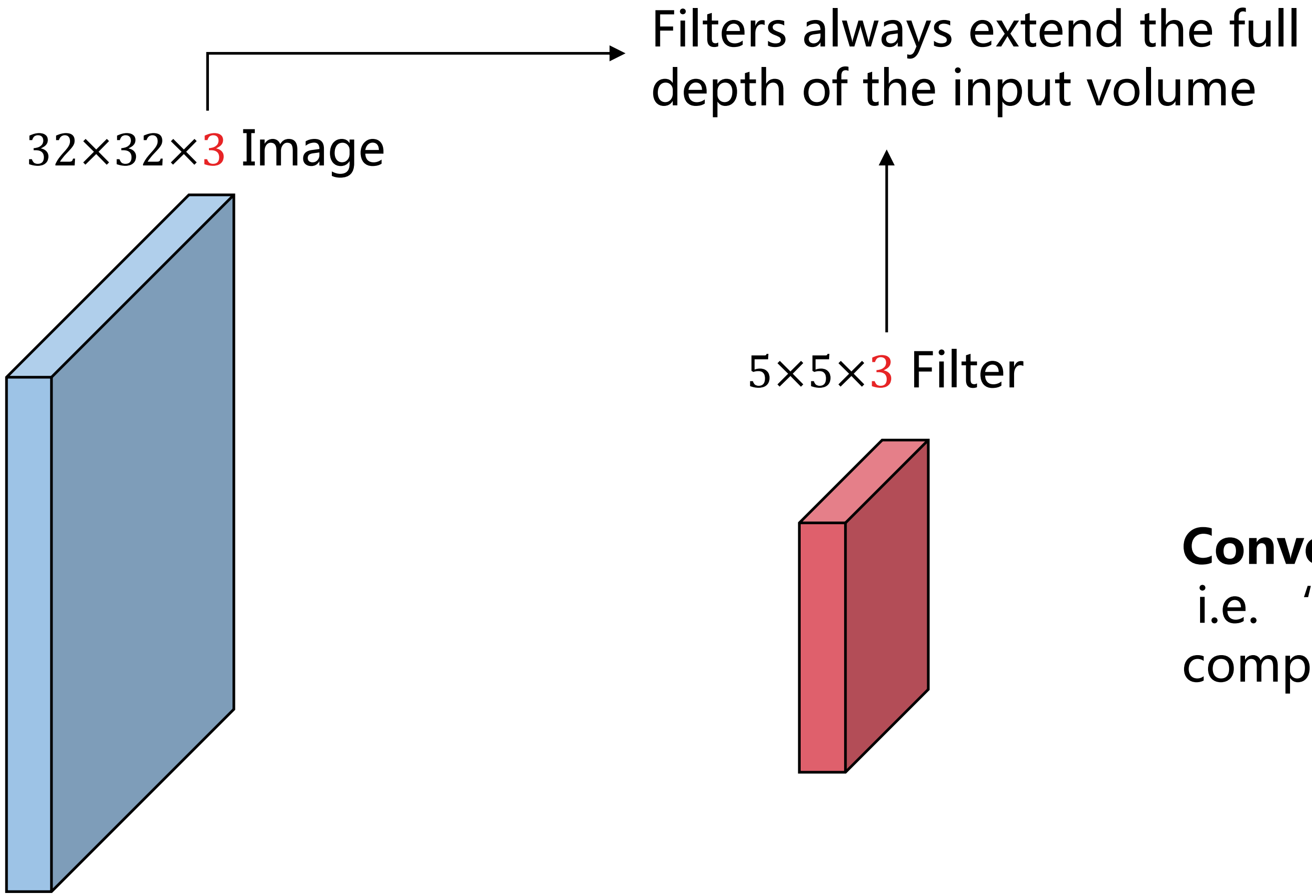
Recap: Fully Connected Layer

- Given an image of size $32 \times 32 \times 3$



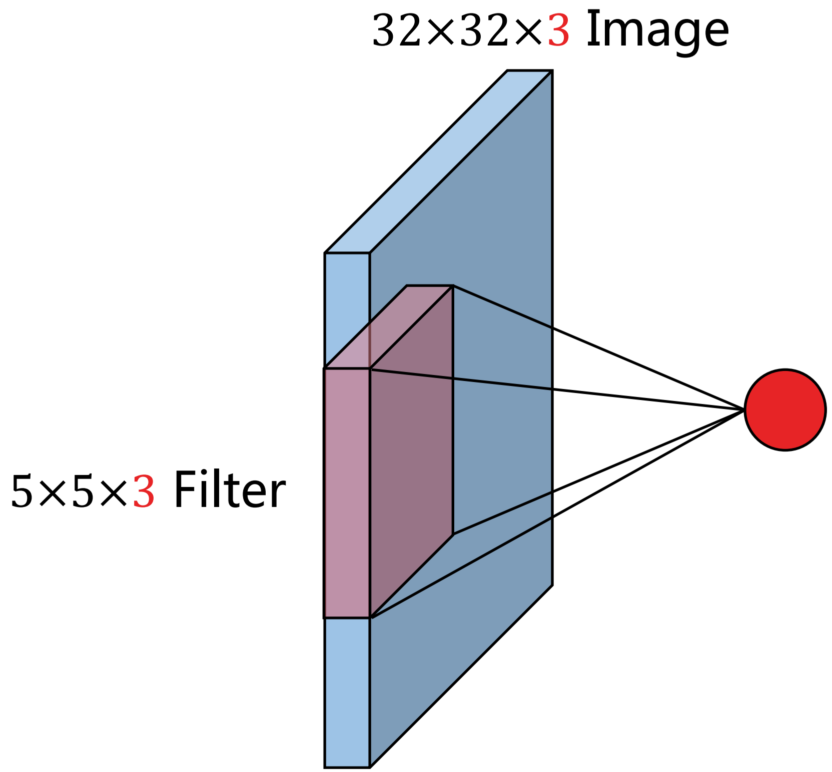
1 number
the result of taking a dot product between a row of W and the input (a 3072-dimensional dot product)

Convolutional Layer



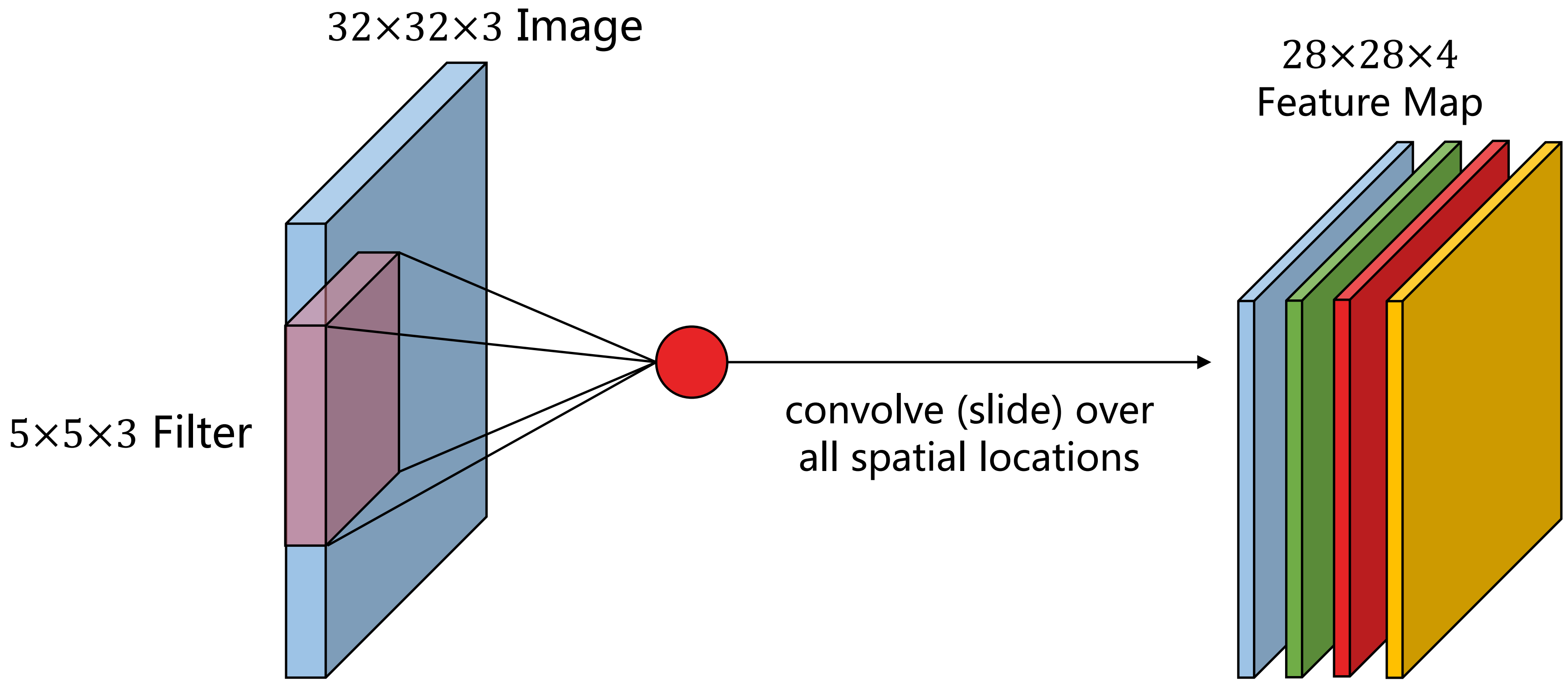
Convolve the filter with the image
i.e. “slide over the image spatially, computing dot products”

Convolutional Layer



1 number
the result of taking a dot product between the filter and a small 5×5×3 chunk of the image
(i.e. 75-dimensional dot product + bias)

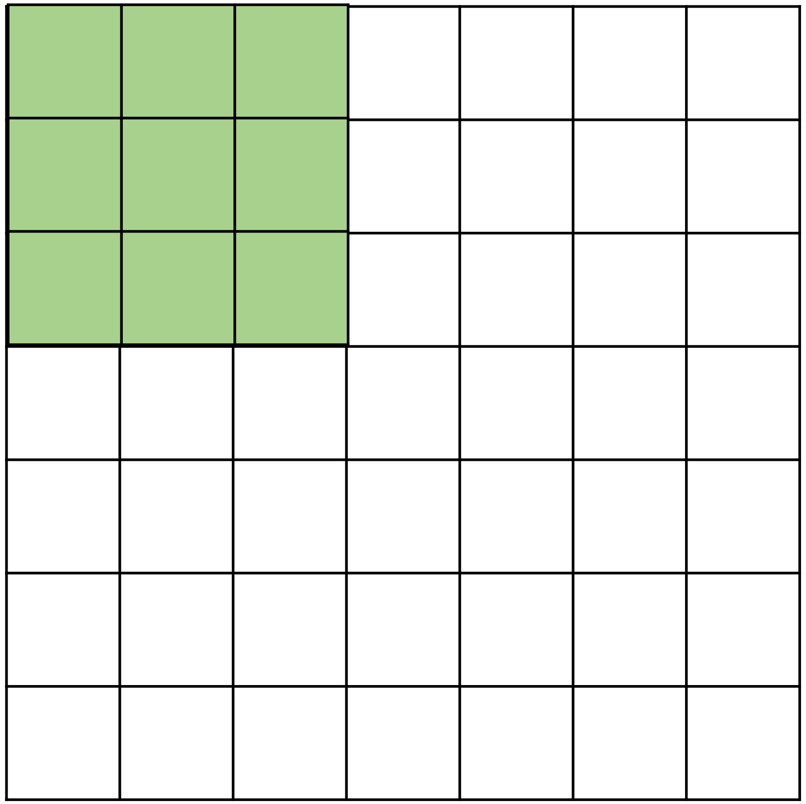
Convolutional Layer



Conclusion: If we have 4 $5\times5\times3$ filters, we can get 4 separate feature maps.

The number of parameters of the convolutional layer is $5\times5\times3\times4+5\times3\times4=360$.

Convolutional Layer



Assume the input is of size 7×7,

and the filter is of size 3×3.

With Stride = 1:

Output size is 5×5.

With Stride = 2:

Output size is 3×3.

With Stride = 3:

Cannot apply 3×3 filter on 7×7 input with stride 3.

Output Size= (Input Size – Filter Size) / Stride + 1

Convolutional Layer

0	0	0	0	0	0	0	0	0	0
0									0
0									0
0									0
0									0
0									0
0									0
0									0
0	0	0	0	0	0	0	0	0	0

Sometimes, we pad zeros to the border.

Assume the input is of size 7×7,

and the filter is of size 3×3.

With Stride = 3 and pad = 1

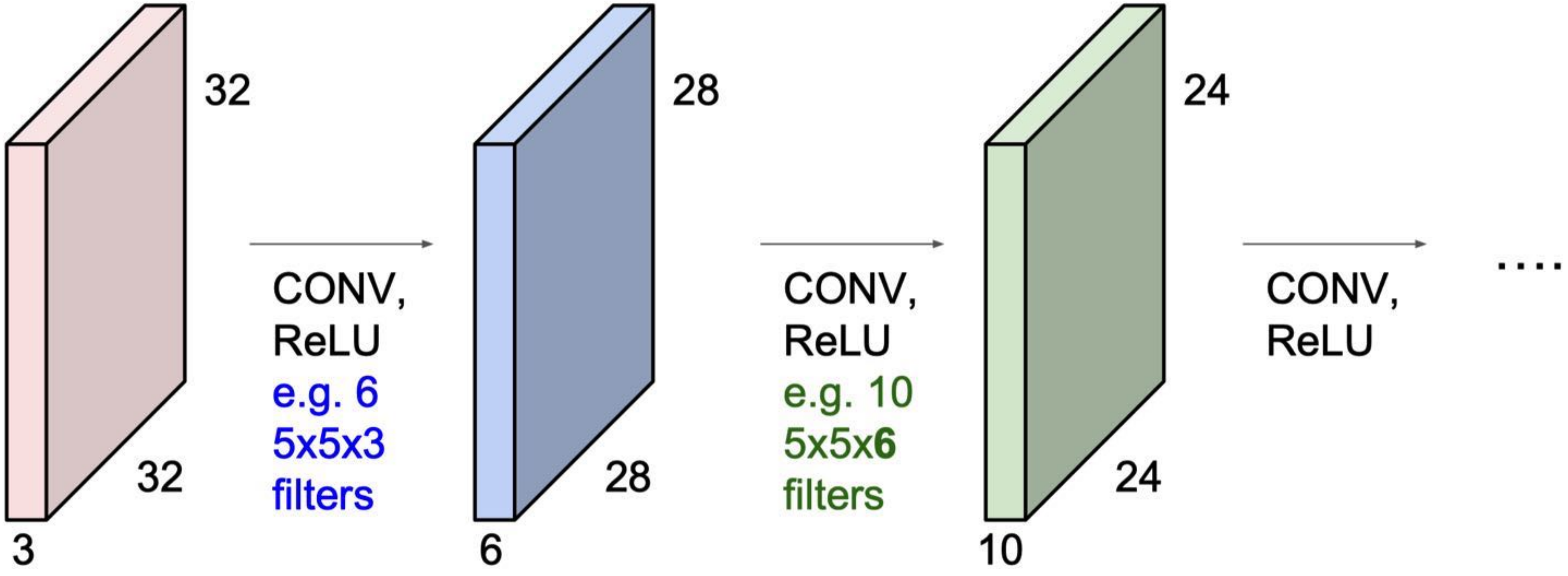
Output size is 3×3.

UPDATE:

Output Size =

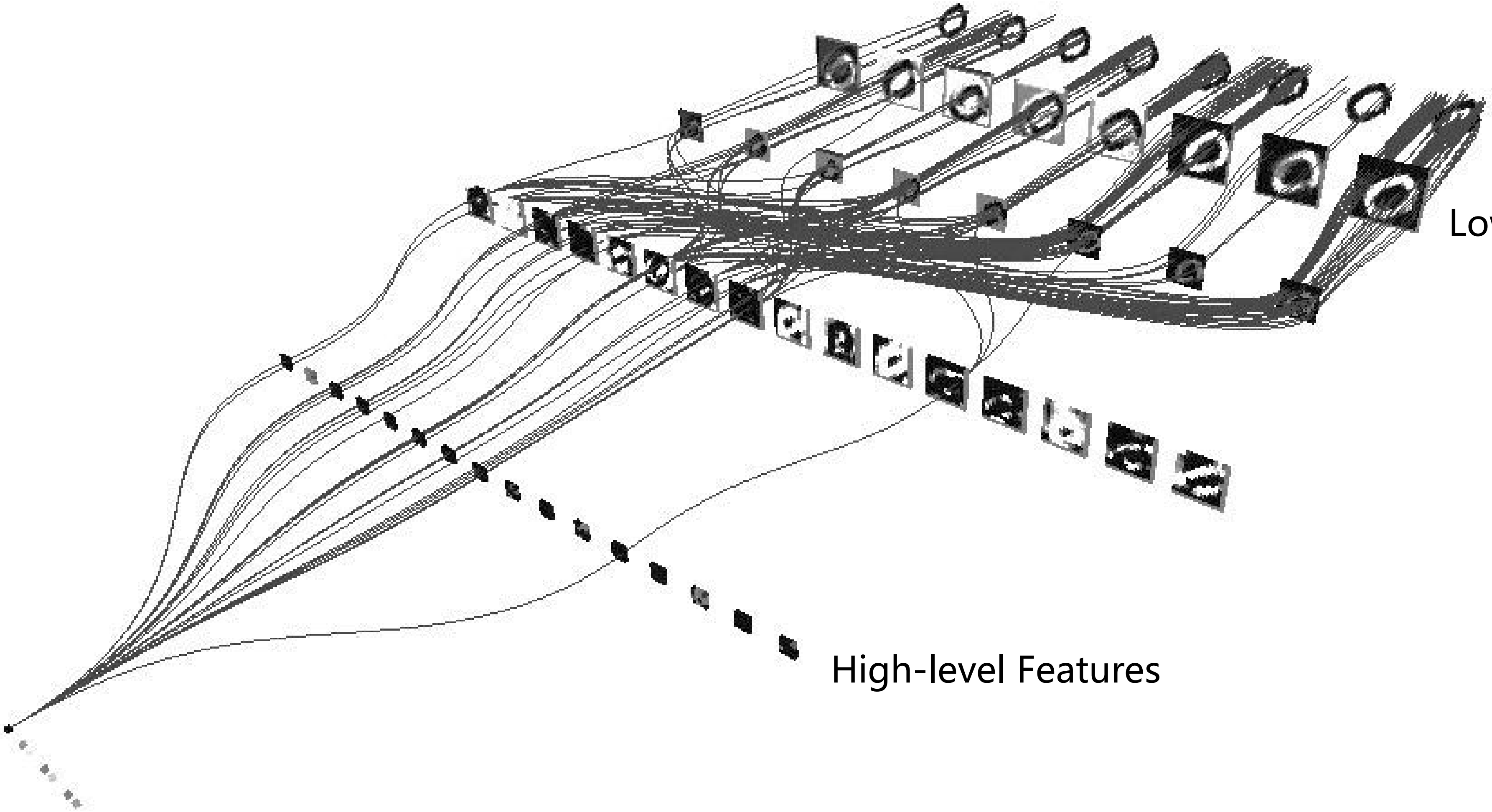
(Input Size – Filter Size + 2 * Padding) / Stride + 1

Convolutional Neural Networks



- ConvNet is a sequence of Convolutional Layers, interspersed with activation functions
- Shrinking too fast is not good, doesn’ t work well.

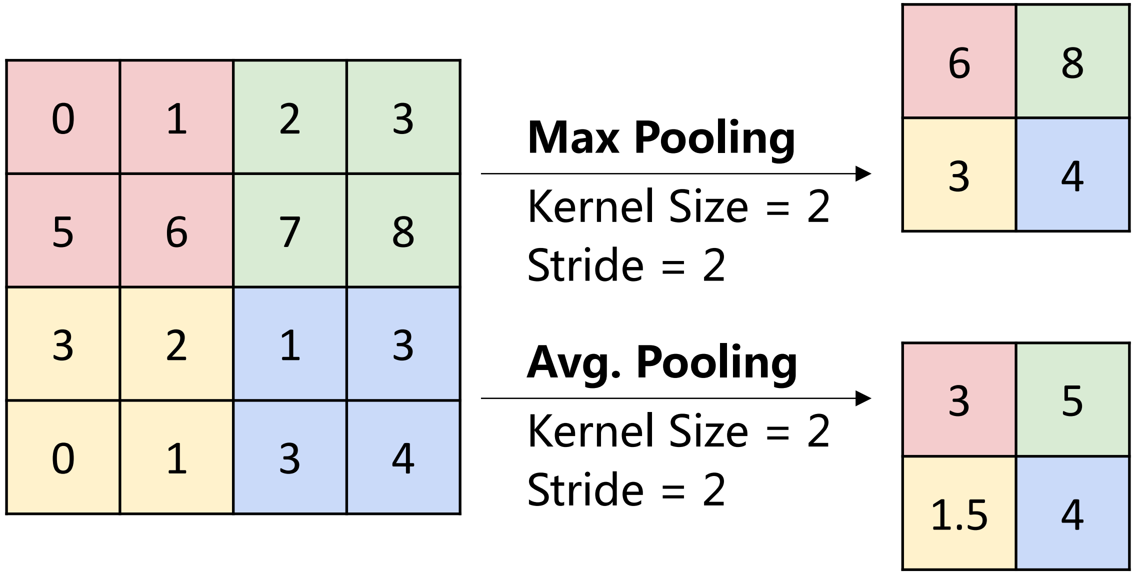
Convolutional Neural Networks

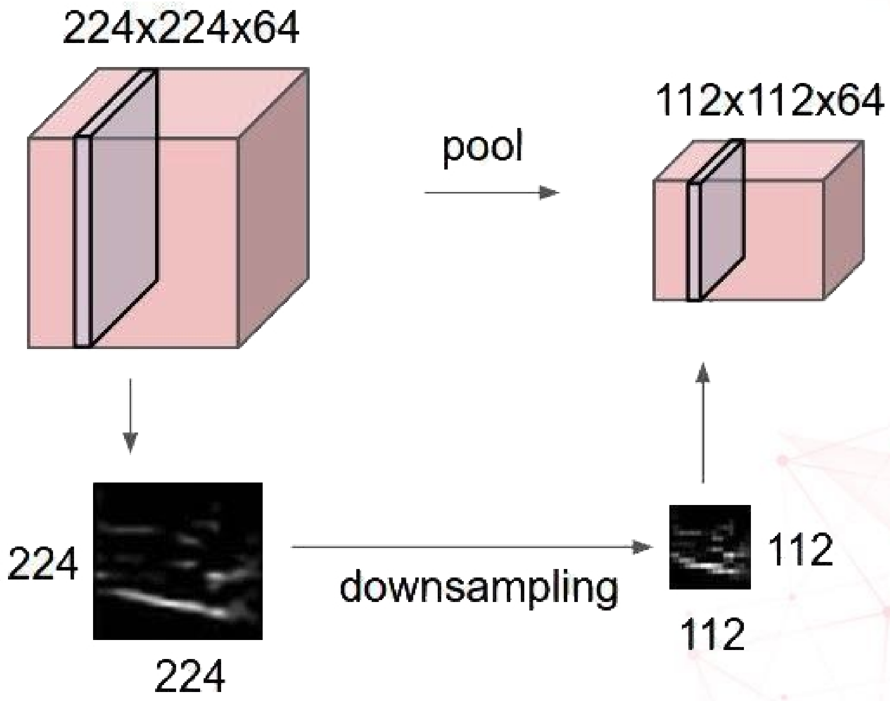


Low-level Features

High-level Features

Pooling Layer



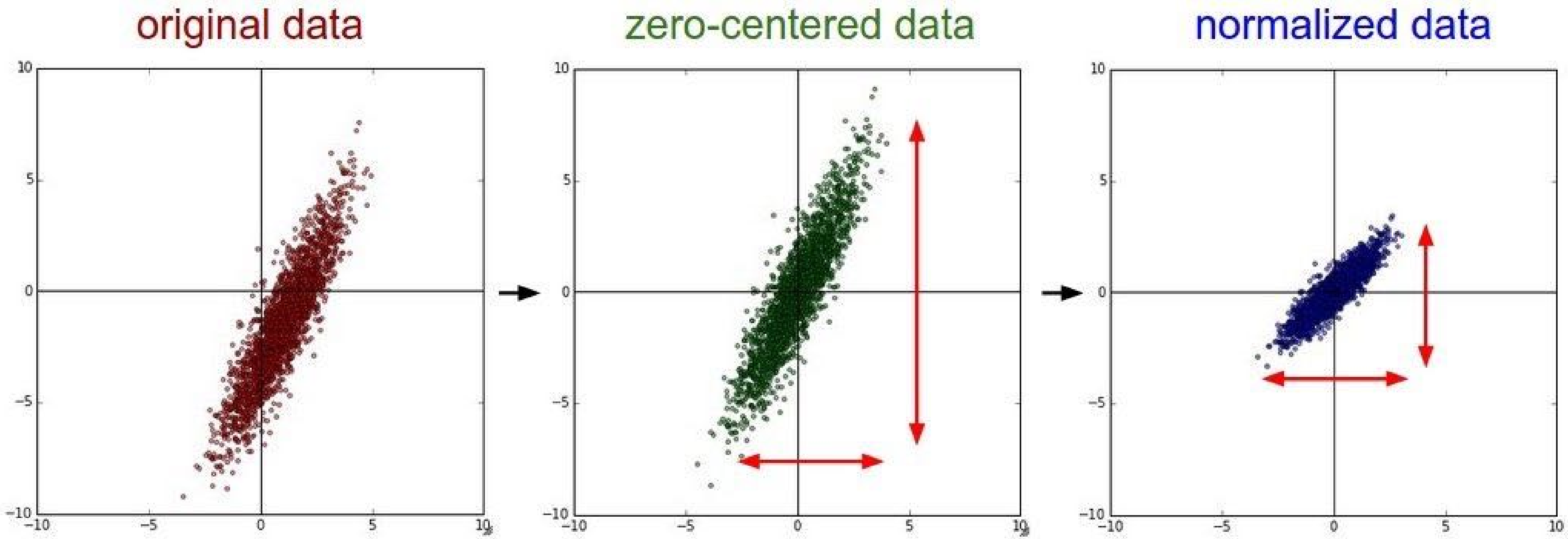


How to Train a Neural Network?

- Babysitting the Learning Process
  - Data Preprocessing
  - Choose the architecture (Convolutional Layers, Activation functions, Losses)
  - Weights initialization
  - Optimizers used for updating parameters
- Optional
  - Data Augmentation
  - Batch Normalization
  - Dropout

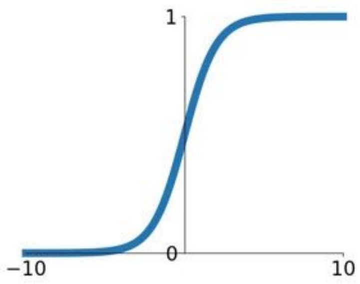
Data Preprocessing

- Normalize

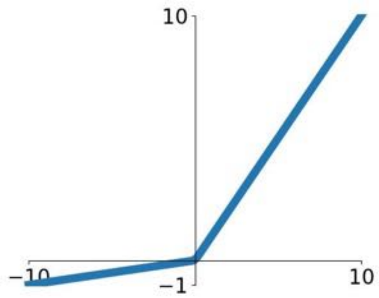


Activation Functions

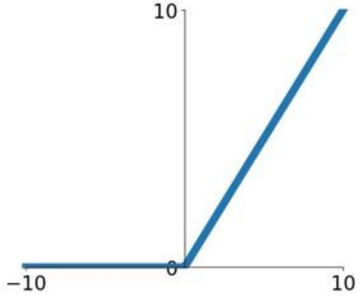
- Sigmoid
  - $\sigma(x) = \frac{1}{1+e^{-x}}$
- ReLU
  - $f(x) = \max(0, x)$
- Leaky ReLU
  - $f(x) = \max(0.01x, x)$
- ELU
  - $f(x) = \begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$



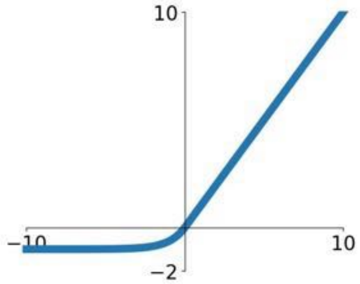
Sigmoid



Leaky ReLU



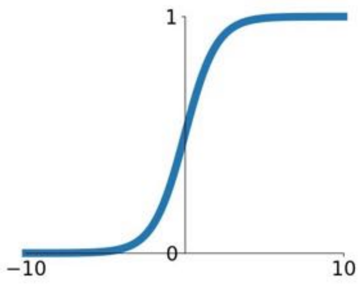
ReLU



ELU

Activation Functions

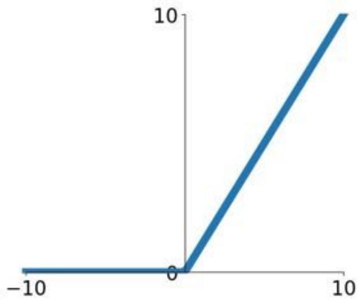
- Sigmoid
  - $\sigma(x) = \frac{1}{1+e^{-x}}$
  - Squashes numbers to range $[0,1]$



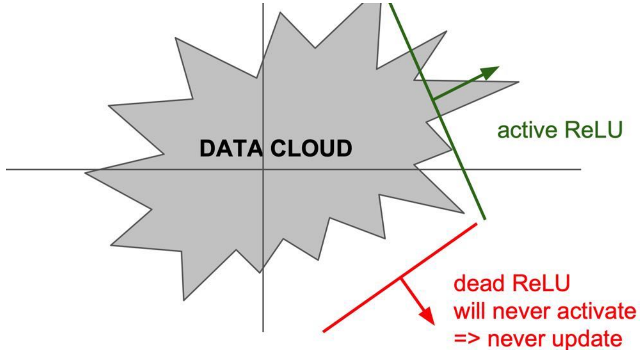
- Issues
  - Saturated neurons "kill" the gradients
  - Sigmoid outputs are not zero-centered
  - exp() is a bit compute expensive

Activation Functions

- ReLU
  - $f(x) = \max(0, x)$

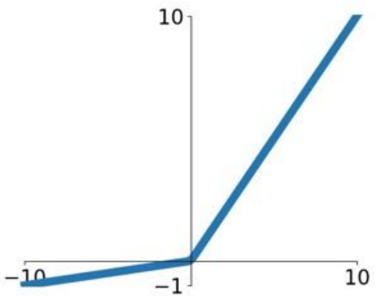


- Characteristics
  - Very computationally efficient
  - Converges much faster than sigmoid/tanh in practice (e.g. 6x)
- Issues
  - Not zero-centered output
  - dead ReLU



Activation Functions

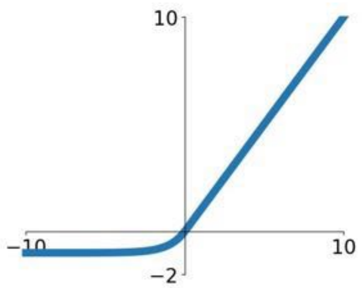
- Leaky ReLU
  - $f(x) = \max(0.01x, x)$



- Characteristics
  - Does not saturate
  - Very computationally efficient
  - Converges much faster than sigmoid/tanh in practice (e.g. 6x)
  - will not “die”

Activation Functions

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



- Characteristics
  - All benefits of ReLU
  - Closer to zero mean outputs
  - Negative saturation regime compared with Leaky ReLU adds some robustness to noise
- Issues
  - Computation requires exp()

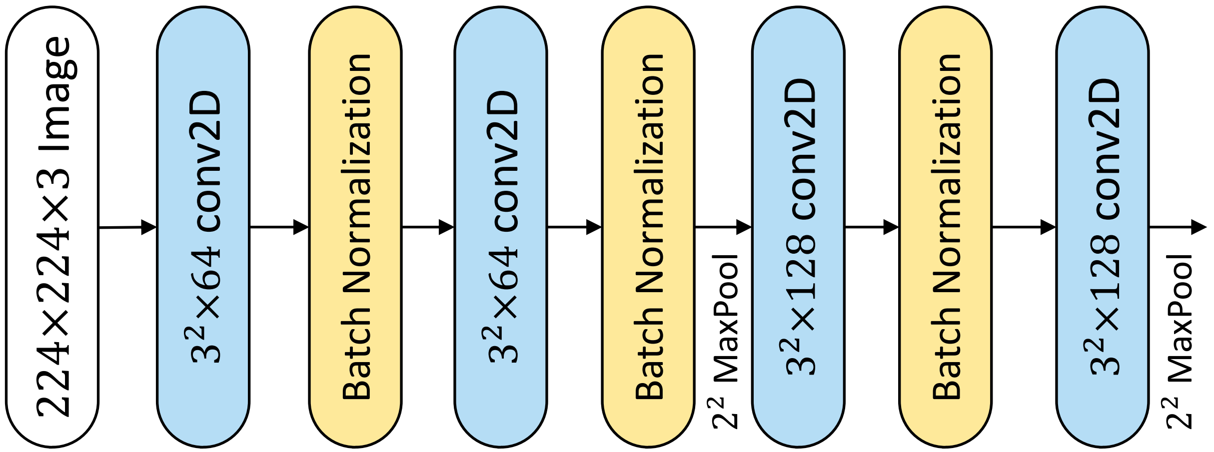
Weights Initialization

- What if all the initial values are set to 0?
  - Output the same thing and have the same gradient.
- What if initial values are samples from a Gaussian distribution $\sigma(0, 0.01)$?
  - Works small for small networks, but problems with deeper networks.
- How to solve the problem?
  - Xavier initialization
  - Kaiming initialization
  - …

Batch Normalization

- "Do you want unit gaussian activations? just make them so."
- Consider a batch of activations at some layer. To make each dimension unit gaussian, apply:

$$\hat{x}^k = \frac{x^k - \mathrm{E}[x^k]}{\sqrt{\mathrm{Var}[x^k]}}$$



Note: BNs are usually inserted after Fully Connected or Convolutional layers, and before nonlinearity.

Batch Normalization

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1...m}\}$;
Parameters to be learned: γ, β

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

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

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

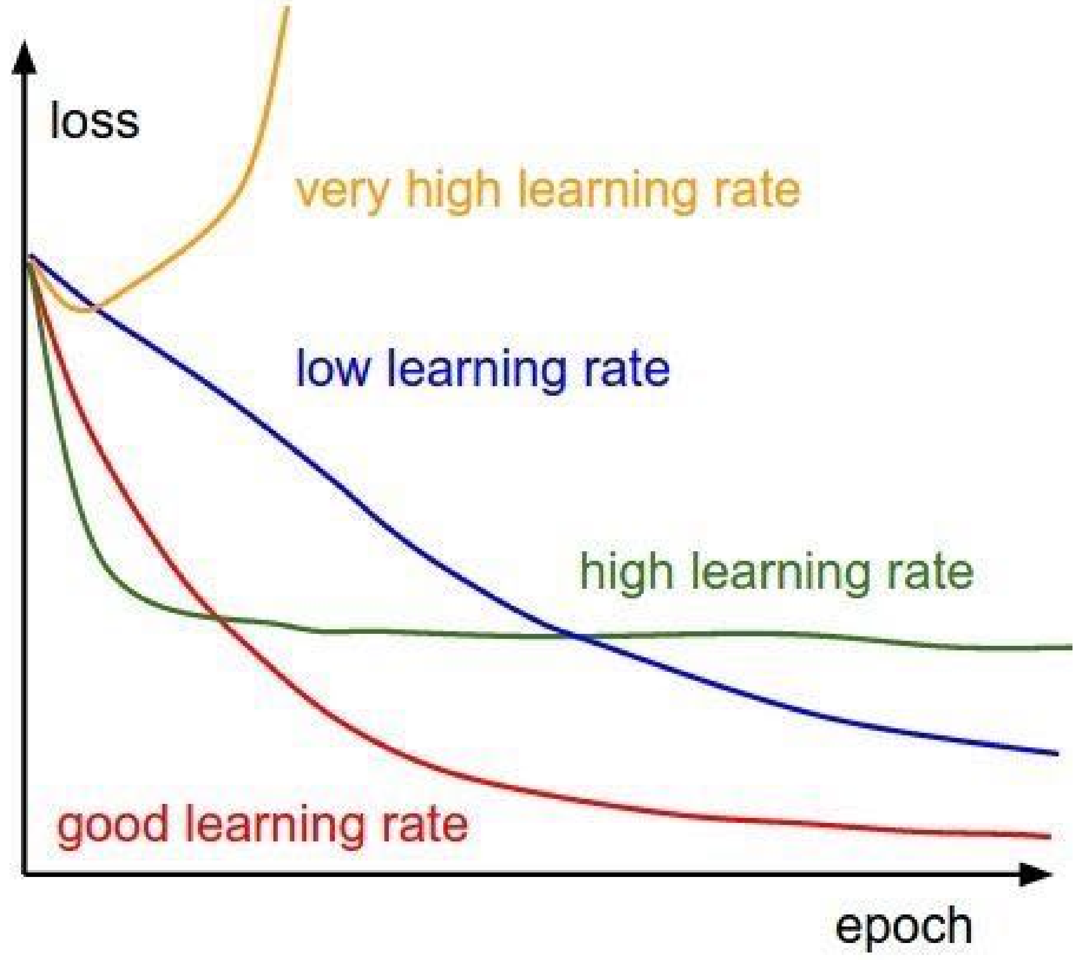
$$\widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}} \qquad \text{// normalize}$$

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

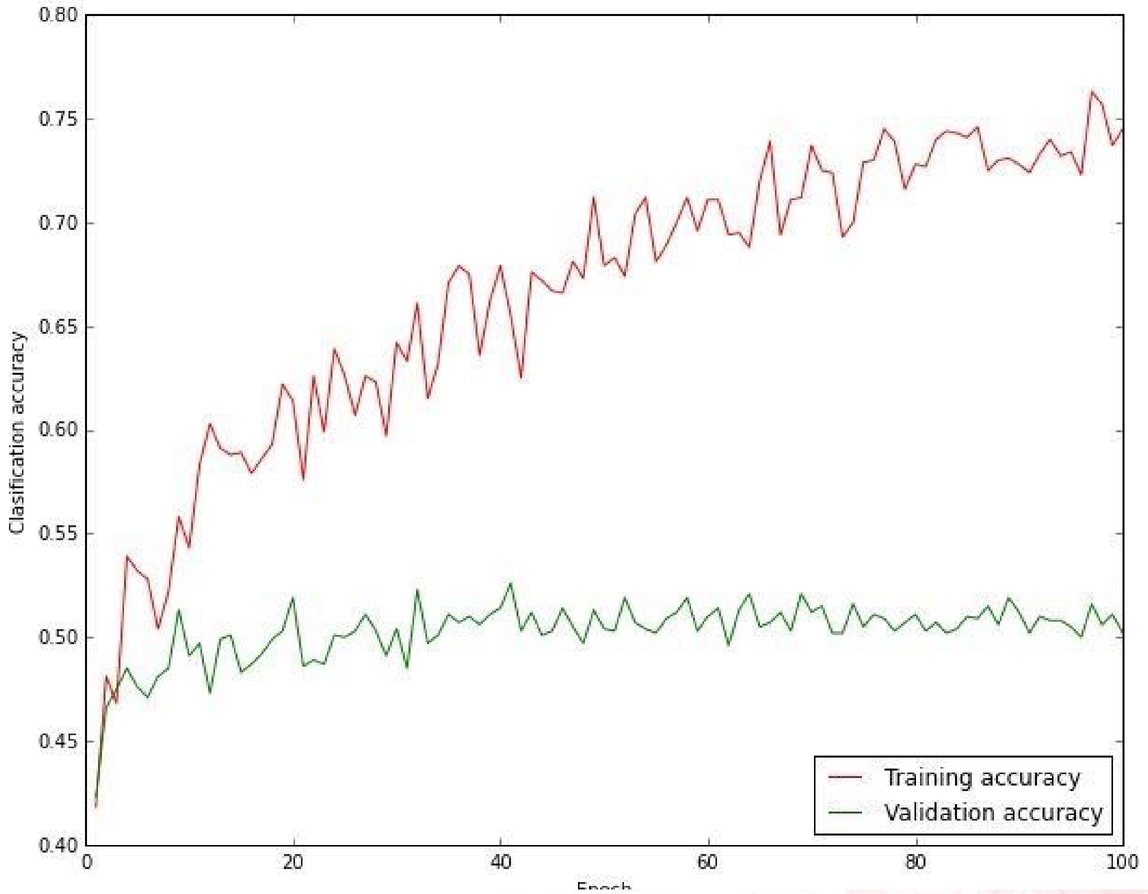
- Improves gradient flow through the network
- Allows higher learning rates
- Reduces the strong dependence on initialization
- Acts as a form of regularization in a funny way, and slightly reduces the need for dropout, maybe

Babysitting the Training Process

- Choose a Proper Learning Rate

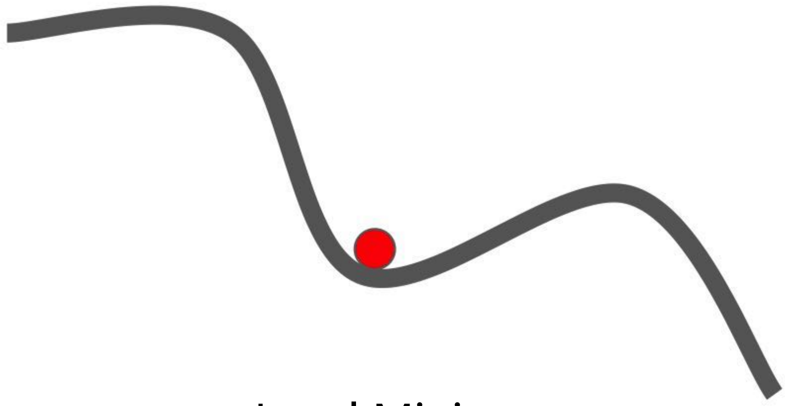


- Overfitting



Optimizers

- SGD
  - $g_t = \nabla_{\theta_{t-1}} f(\theta_{t-1})$
  - $\nabla_{\theta_t} = -\eta g_t$
- SGD + Momentum
  - $m_t = \mu m_{t-1} + g_t$
  - $\nabla \theta_t = -\eta m_t$
  - Typically, $\mu = 0.9$ or 0.99
- Problems with SGD



Local Minima

Saddle Point

Optimizers

- AdaGrad
  - $n_t = n_{t-1} + g_t^2$
  - $\nabla\theta_t = -\frac{\eta}{\sqrt{n_t + \epsilon}} \cdot g_t$
- RMSProp
  - $n_t = v n_{t-1} + (1 - v) g_t^2$
  - $\nabla\theta_t = -\frac{\eta}{\sqrt{n_t + \epsilon}} \cdot g_t$
- Adam
  - $m_t = \mu m_{t-1} + (1 - \mu) g_t$
  - $n_t = v n_{t-1} + (1 - v) g_t^2$
  - $\widehat{m_t} = \frac{m_t}{1 - \mu^t}$
  - $\widehat{n_t} = \frac{n_t}{1 - v^t}$
  - $\nabla\theta_t = -\frac{\widehat{m_t}}{\sqrt{\widehat{n_t} + \epsilon}} \cdot \eta$

Thank You!