# Guerrilla Section #3

## Topics

- Kernels
- Decision Trees/Random Forest
- Convnets

## Kernels

## Kernels

A kernel is a function $k : D \times D \to R$, where $D \subseteq \mathcal{R}^d$ is the domain, which is

1. Symmetric: $k(x, x') = k(x', x)$ for all $x, x' \in D$
2. Positive Semi-definite: for every finite collection $\{x_1, \ldots, x_n\} \subseteq D$, the Gram matrix $K$ given by $K_{ij} = k(x_i, x_j)$ is positive semi-definite

## Kernel Trick

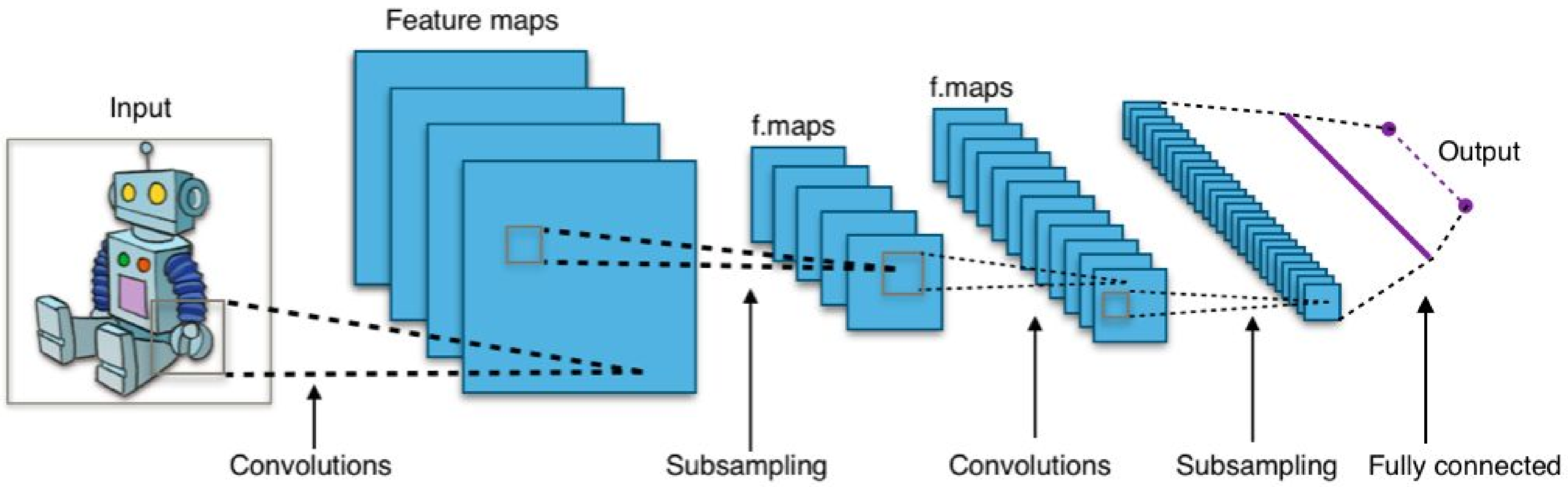
- Try to rewrite computations can be written in terms of inner products
- If so, we can replace the inner products with calls to a kernel function
- Now possible to implicitly perform inner product in some higher (possibly infinite) dimensional space

## Convnets

## Overview

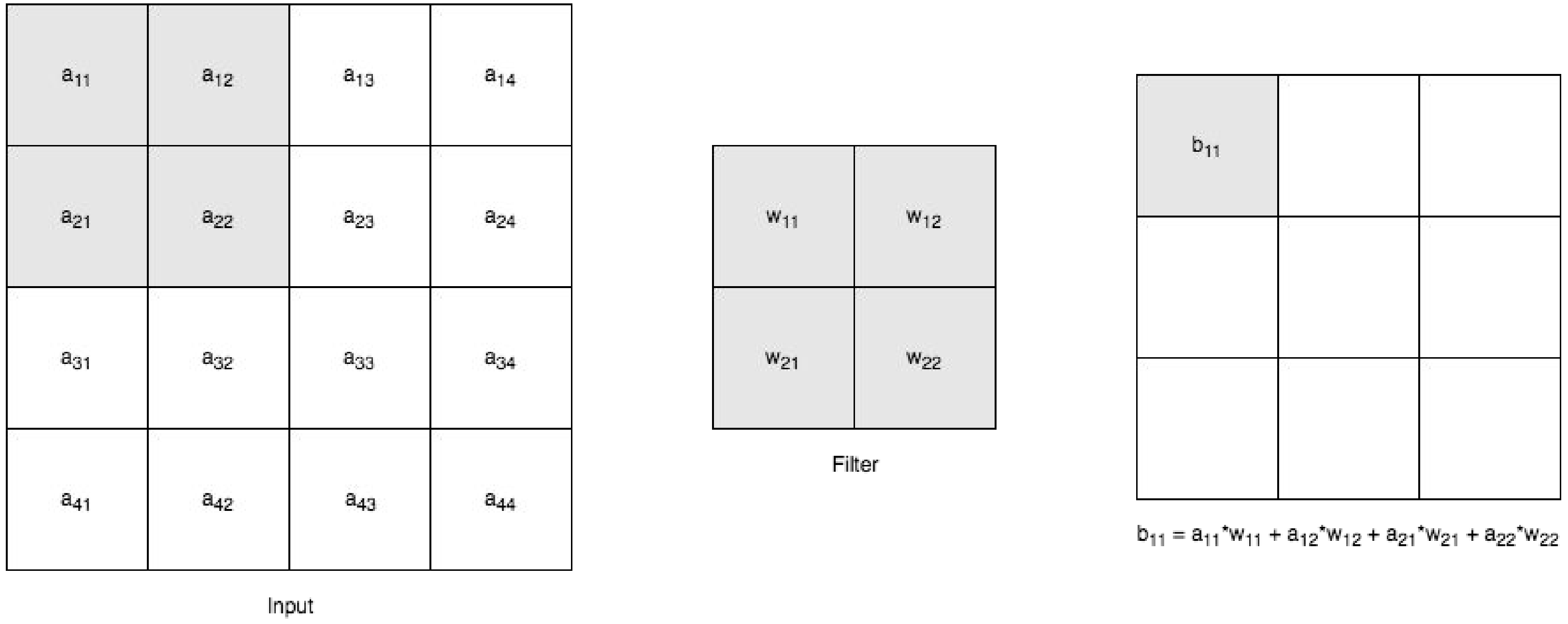
- Fully connected networks sometimes too big
- Rule of thumb: training points > number of weights
- Convnets good for images/videos
- Use notion of “shared” weights called filters applied to different patches of the images

## Overview


Feature maps
Input
f.maps
f.maps
Output
Convolutions
Subsampling
Convolutions
Subsampling
Fully connected

# 2d Convolution

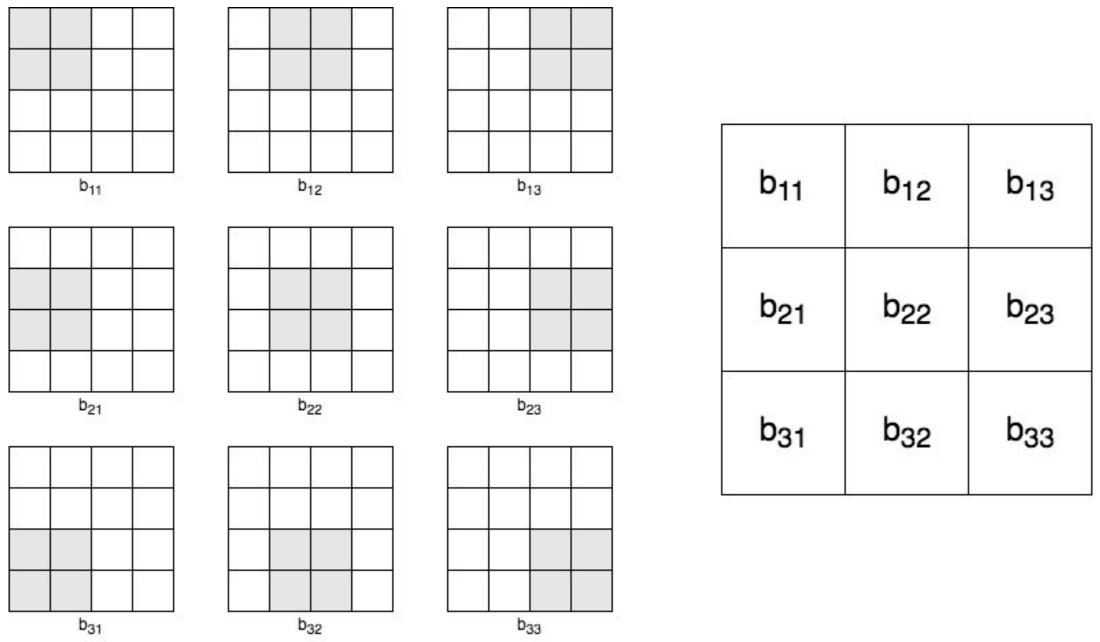
To obtain *one* square of our output, we take the “dot product” of the window (shaded region of the input) and the filter.



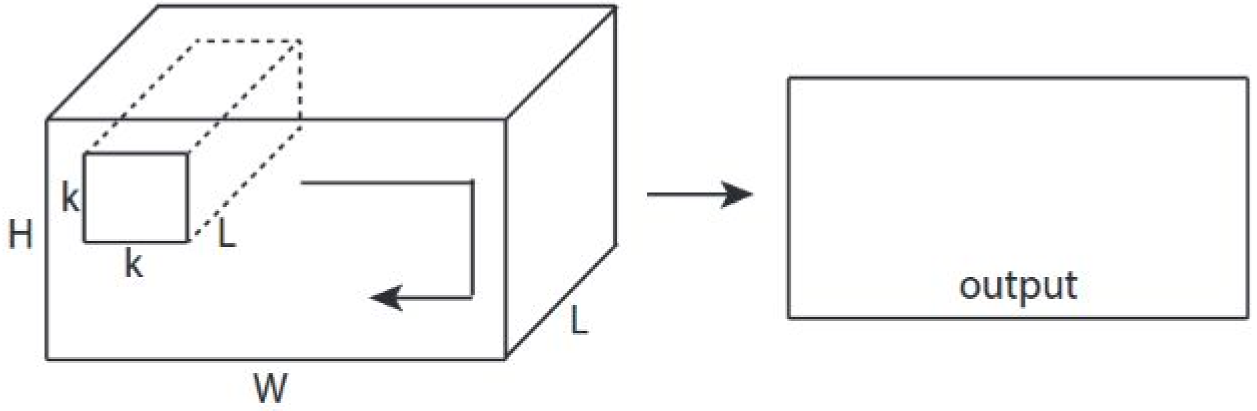
$b_{11} = a_{11}{}^*w_{11} + a_{12}{}^*w_{12} + a_{21}{}^*w_{21} + a_{22}{}^*w_{22}$

# 2d Convolution

We move the window in our input to get every combination. How much we move our window is known as the stride. Stride is usually 1, which is true for the case below.


$b_{11}$
$b_{12}$
$b_{13}$
$b_{21}$
$b_{22}$
$b_{23}$
$b_{31}$
$b_{32}$
$b_{33}$
$b_{11}$ $b_{12}$ $b_{13}$
$b_{21}$ $b_{22}$ $b_{23}$
$b_{31}$ $b_{32}$ $b_{33}$

## 3d Convolution - 3d input to 2d output

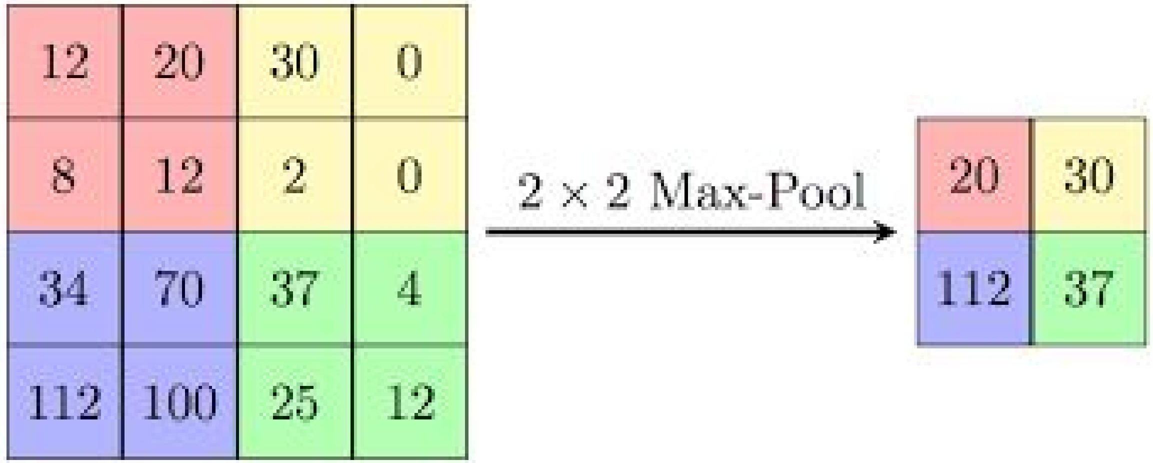


input = [W,H,L], filter = [k,k,L] output = [W,H]

output-shape is 2D Matrix

Usage: L sets of WxH filters, result of applying these filters to an image is 3d, use 3d convolution rather than convolving 1 layer at a time

## Max Pooling


12
20
30
0
8
12
2
0
34
70
37
4
112
100
25
12
$2 \times 2$ Max-Pool
20
30
112
37

## Decision Trees

# Decision Tree

- How to choose the variable to split on?

  1. Try all splits

     - All features
     - All splits within a feature

  2. For a set S, let J(S) be the cost of S and choose the split that minimizes the weighted average

$$\frac{|S_l|J(S_l)+|S_r|J(S_r)}{|S_l|+|S_r|}$$

# Decision Tree Definitions

- **Surprise**: How likely an event is
  - P(C) = 1, surprise = __
  - P(C) = 0, surprise = __

$$-\log_2 p_C.$$

- **Entropy:** Average surprise over all possible classes
  - Rank the following from smallest to greatest:
    - Entropy if all points fall in one class
    - Entropy if all points fall equally into 2 classes
    - Entropy if points fall randomly into n classes

$$H(S) = -\sum_{C} p_C \log_2 p_C.$$

- **Information Gain**: Difference between prior entropy and 'new' entropy
  - Do you maximize or minimize information gain?

$$H(S) - H_{\text{after}}$$

## Decision Tree Definitions

- **Surprise**: How likely an event is
  - P(C) = 1,  surprise = 0
  - P(C) = 0,  surprise = 1

$$-\log_2 p_C.$$

- **Entropy:** Average surprise over all possible classes
  - Rank the following from smallest to greatest:
    - Entropy if all points fall in one class: 0
    - Entropy if all points fall equally into 2 classes:  1
    - Entropy if points fall randomly into n classes: > 1

$$H(S) = -\sum_{C} p_C \log_2 p_C.$$

- **Information Gain**: Difference between prior entropy and ‘new’ entropy

$$H(S) - H_{\text{after}}$$

## Decision Tree Generalization

Overfitting (low bias, high variance)

Decision Trees can obtain 0 training error if trained to infinite depth. This is true if there are no conflicting points (exact same training point, different labels). Though this sounds appealing, it overfits to the training set.

Regularization

Types of regularization include limiting max depth (increases training error) and using boosting/bagging methods with random forests.

## Bagging (Bootstrap Aggregation), Random Forests

Given training set $D$ with size $n$, generate $m$ new sets $D_i$ each of size $n'$ by sampling from $D$ uniformly with replacement. Train each of the $m$ models individually, and have them vote on the correct label during classification.

If $n'=n$, then you can expect for each D_i to have a fraction $1-1/e=63.2\%$ of unique samples in each training set.

Random forests use data bagging, as well as “feature bagging”: for each node, uniformly pick $d'$ of the $d$ features, and choose best split from these $d'$ features. Typically choosing $d'=\sqrt{d}$ yields good results.