Lecture 28: Object Detection
Menu for Today (March 29, 2021)

Topics:
— Object Detection
— Face Detection

Readings:
— Today’s Lecture: Forsyth & Ponce (2nd ed.) 17.1, 17.2
— Next Lecture: N/A

Reminders:
— Assignment 5: Scene Recognition with Bag of Words due Mar 31
Today’s “fun” Example: DensePose

DensePose:
Dense Human Pose Estimation In The Wild

Riza Alp Güler*
INRIA, CentraleSupélec

Natalia Neverova
Facebook AI Research

Iasonas Kokkinos
Facebook AI Research

* Riza Alp Güler was with Facebook AI Research during this work.
Today’s “fun” Example: Pose Estimation

[ Vondrak et al., CVPR 2008 ]
One common strategy to obtain a better classifier is to combine multiple classifiers.

A simple approach is to train an ensemble of independent classifiers, and average their predictions.

**Boosting** is another approach.
— Train an ensemble of classifiers sequentially.
— Bias subsequent classifiers to correctly predict training examples that previous classifiers got wrong.
— The final boosted classifier is a weighted combination of the individual classifiers.
Lecture 27: Re-cap

Figure credit: Paul Viola
Lecture 27: Re-cap

Weights Increased

Figure credit: Paul Viola
Lecture 27: Re-cap

Figure credit: Paul Viola
Lecture 27: Re-cap

Figure credit: Paul Viola
Lecture 27: Re-cap

Figure credit: Paul Viola
Final classifier is a combination of weak classifiers
Object Detection: Introduction

We have been discussing **image classification**, where we pass a whole image into a classifier and obtain a class label as output.

We assumed the image contained a single, central object.

The task of **object detection** is to detect and localize all instances of a target object class in an image.

— Localization typically means putting a tight bounding box around the object.
Sliding Window

Train an image classifier as described previously. ‘Slide’ a fixed-sized detection window across the image and evaluate the classifier on each window.

Image credit: KITTI Vision Benchmark
Train an image classifier as described previously. ‘Slide’ a fixed-sized detection window across the image and evaluate the classifier on each window.

Is there a car?

Image credit: KITTI Vision Benchmark
Train an image classifier as described previously. ‘Slide’ a fixed-sized detection window across the image and evaluate the classifier on each window.

Is there a car?

Image credit: KITTI Vision Benchmark
**Sliding Window**

Train an image classifier as described previously. ‘Slide’ a fixed-sized detection window across the image and evaluate the classifier on each window.

*Is there a car?*

*Image credit: KITTI Vision Benchmark*
Sliding Window

Train an image classifier as described previously. ‘Slide’ a fixed-sized detection window across the image and evaluate the classifier on each window.

Is there a car?

Image credit: KITTI Vision Benchmark
Train an image classifier as described previously. ‘Slide’ a fixed-sized detection window across the image and evaluate the classifier on each window.

Is there a car?

Image credit: KITTI Vision Benchmark
Sliding Window

Train an image classifier as described previously. ‘Slide’ a fixed-sized detection window across the image and evaluate the classifier on each window.

Is there a car?

Image credit: KITTI Vision Benchmark
**Sliding Window**

Train an image classifier as described previously. ‘Slide’ a fixed-sized detection window across the image and evaluate the classifier on each window.

Is there a car?

**Image credit:** KITTI Vision Benchmark
Train an image classifier as described previously. ‘Slide’ a fixed-sized detection window across the image and evaluate the classifier on each window.

Is there a car?

*Image credit: KITTI Vision Benchmark*
Sliding Window

Train an image classifier as described previously. ‘Slide’ a fixed-sized detection window across the image and evaluate the classifier on each window.

This is a search over location
— We have to search over scale as well
— We may also have to search over aspect ratios

Image credit: KITTI Vision Benchmark
What data we **train** a classifier on?

**Image** Classifiers

**Image** classifiers can be applied to regions/windows, but do not work so well in practice …
What data we **train** a classifier on?

**Image** Classifiers
What data we **train** a classifier on?

**Image** Classifiers

**Object** Classifiers
Today’s “fun” Example: Detection with No Data

Our **UniT**: Unified Knowledge Transfer

(Note, above result is obtained without human ever annotating bounding boxes or pixel-level segments for any of these objects)
Let’s assume we have **object** labeled data …

**Object** classifiers work a lot better … but require expensive bounding box annotations …
Let’s assume we have **object** labeled data …

**Object** classifiers work a lot better … but require expensive bounding box annotations …

**Object** Classifiers

(for convenience we will normalize patches to 64x64 … or 128x128)
**Example: Face Detection**

The **Viola-Jones** face detector is a classic sliding window detector that learns both efficient features and a classifier.

A key strategy is to use features that are fast to evaluate to reject most windows early.

The Viola-Jones detector computes ‘rectangular’ features within each window.
Example: Face Detection

A ‘rectangular’ feature is computed by summing up pixel values within rectangular regions and then differencing those region sums.

The integral image speeds up region summation, so computing Haar wavelets is fast.

Figure credit: P. Viola and M. Jones, 2001
Integral Image

\[ A(x, y) = \sum_{x' \leq x, y' \leq y} I(x', y') \]

**Original Image**

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>5</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

**Integral Image**

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>6</td>
<td>8</td>
</tr>
<tr>
<td>3</td>
<td>12</td>
<td>15</td>
</tr>
<tr>
<td>5</td>
<td>15</td>
<td>19</td>
</tr>
</tbody>
</table>

*Image Credit: Ioannis (Yannis) Gkioulekas (CMU)*
Integral Image

What is the sum of the bottom right 2x2 square?

$$I(x, y)$$

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>5</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>4</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>

original image

$$A(x, y)$$

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>6</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>12</td>
<td>15</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>15</td>
<td>19</td>
<td></td>
</tr>
</tbody>
</table>

integral image

Image Credit: Ioannis (Yannis) Gkioulkekas (CMU)
What is the sum of the bottom right 2x2 square?

$I(x, y)$

<table>
<thead>
<tr>
<th>$I(x, y)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1   5 2</td>
</tr>
<tr>
<td>2   4 1</td>
</tr>
<tr>
<td>2   1 1</td>
</tr>
</tbody>
</table>

$A(x, y)$

<table>
<thead>
<tr>
<th>$A(x, y)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1   6 8</td>
</tr>
<tr>
<td>3   12 15</td>
</tr>
<tr>
<td>5   15 19</td>
</tr>
</tbody>
</table>

Image Credit: Ioannis (Yannis) Gkioulekas (CMU)
What is the sum of the bottom right 2x2 square?

**Original Image**

\[
\begin{array}{ccc}
1 & 5 & 2 \\
2 & 4 & 1 \\
2 & 1 & 1 \\
\end{array}
\]

**Integral Image**

\[
\begin{array}{ccc}
1 & 6 & 8 \\
3 & 12 & 15 \\
5 & 15 & 19 \\
\end{array}
\]

\[
A(1, 1, 3, 3) = A(3, 3) - A(1, 3) - A(3, 1) + A(1, 1) \\
= 19 - 8 - 5 + 1 \\
= 7
\]

*Image Credit: Ioannis (Yannis) Gkioulkas (CMU)*
Integral Image

$$A(x, y) = \sum_{x' \leq x, y' \leq y} I(x', y')$$

<table>
<thead>
<tr>
<th>I(x, y)</th>
<th>A(x, y)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 5 2</td>
<td>1 6 8</td>
</tr>
<tr>
<td>2 4 1</td>
<td>3 12 15</td>
</tr>
<tr>
<td>2 1 1</td>
<td>5 15 19</td>
</tr>
</tbody>
</table>

Can find the **sum** of any block using 3 operations

$$A(x_1, y_1, x_2, y_2) = A(x_2, y_2) - A(x_1, y_2) - A(x_2, y_1) + A(x_1, y_1)$$

*Image Credit: Ioannis (Yannis) Gkioulkas (CMU)*
**Example: Face Detection**

Given an integral image, the sum within a rectangular region in $I$ can be computed with just 3 additions

$$\text{Sum} = A - B - C + D$$

**Constant time:** does not depend on the size of the region. We can avoid scaling images - just scale features directly (remember template matching!)
Integral Image Layer for Deep Neural Networks

In a classical paper [1] from 2001, Viola and Jones popularized the use of large rectangular image filters in order to obtain features for image recognition. The use of very large filters allowed Viola and Jones to compute features over very large receptive fields without blowing up the computation cost. For the next 10+ years, such features remained the staple of fast computer vision (e.g. [2]). The advent of deep learning made the use of integral-image features far less popular. Currently, state-of-the-art architectures invariably relying on very deep architectures. In these architectures sufficiently large receptive fields are obtained via the use of downsampling with subsequent upsampling [3] or via dilated convolutions [4]. All such tricks however have their downsides and usually necessitate the use of very deep networks.

The goal of this project is to implement an integral image-based filtering as a layer for deep architectures in Torch deep learning package, and to evaluate it for the task of learning very fast object detectors (as an alternative to e.g. [5]) and semantic segmentation systems (as an alternative to e.g. [3,4]). The hope is to obtain much shallower architectures, which at least for simple classes (e.g. roadsigns or upright pedestrians) will approach the performance of much deeper ones.

The project is supervised by Victor Lempitsky at Skoltech, Moscow, Russia.

https://github.com/shrubb/integral-layer
Deep Neural Networks for Object Detection

Christian Szegedy  Alexander Toshev  Dumitru Erhan
Google, Inc.
{szegedy, toshev, dumitru}@google.com

Abstract

Deep Neural Networks (DNNs) have recently shown outstanding performance on image classification tasks [14]. In this paper we go one step further and address the problem of object detection using DNNs, that is not only classifying but also precisely localizing objects of various classes. We present a simple and yet powerful formulation of object detection as a regression problem to object bounding box masks. We define a multi-scale inference procedure which is able to produce high-resolution object detections at a low cost by a few network applications. State-of-the-art performance of the approach is shown on Pascal VOC.
Example: Face Detection

Weak classifier

\[ h_j(x) = \begin{cases} 
1 & \text{if } f_j(x) > \theta_j \\
0 & \text{otherwise}
\end{cases} \]
**Example:** Face Detection

Many possible rectangular features (180,000+ were used in the original paper)

*Figure credit:* B. Freeman
Example: Face Detection

Use **boosting** to both select the informative features and form the classifier. Each round chooses a weak classifier that simply compares a single rectangular feature against a threshold.

*Figure credit:* P. Viola and M. Jones, 2001
Example: Face Detection (Aside – math details not required)

2. Select best filter/threshold combination
   a. Normalize the weights
      $$w_{t,i} = \frac{w_{t,i}}{\sum_{j=1}^{n} w_{t,j}}$$
      $$h_j(x) = \begin{cases} 
          1 & \text{if } f_j(x) > \theta_j \\
          0 & \text{otherwise}
      \end{cases}$$
   b. For each feature, $j$
      $$\varepsilon_j = \sum_i w_i |h_j(x_i) - y_i|$$
   c. Choose the classifier, $h_t$ with the lowest error $\varepsilon_t$

3. Reweight examples
   $$w_{t+1,i} = w_{t,i} \beta_t^{1-|h_t(x_i) - y_i|}$$
   $$\beta_t = \frac{\varepsilon_t}{1 - \varepsilon_t}$$

Image Credit: Ioannis (Yannis) Gkioulekas (CMU)
Example: Face Detection

1. Select best filter/threshold combination
   a. Normalize the weights

   \[ w_{t,i} \leftarrow \frac{w_{t,i}}{\sum_{j=1}^{n} w_{t,j}} \]

   We start with all sample weights = 1

Image Credit: Ioannis (Yannis) Gkioulekas (CMU)
**Example:** Face Detection

1. Select best filter/threshold combination

   a. Normalize the weights
   \[ w_{t,i} \leftarrow \frac{w_{t,i}}{\sum_{j=1}^{n} w_{t,j}} \]

   b. For each feature, \( j \)
   \[ \epsilon_j = \sum_i w_i |h_j(x_i) - y_i| \]

   weighed sum of mis-classified training examples

   **Note:** the second term is 0/1
   — 0 predicted label and true label are same
   — 1 predicted label and true label are different (error)
Example: Face Detection

1. Select best filter/threshold combination

   a. Normalize the weights

   \[ w_{t,i} \leftarrow \frac{w_{t,i}}{\sum_{j=1}^{n} w_{t,j}} \]

   \[ h_j(x) = \begin{cases} 
   1 & \text{if } f_j(x) > \theta_j \\
   0 & \text{otherwise} 
   \end{cases} \]

   b. For each feature, \( j \)

   \[ \varepsilon_j = \sum_i w_i |h_j(x_i) - y_i| \]

   c. Choose the classifier, \( h_t \) with the lowest error \( \varepsilon_t \)

Image Credit: Ioannis (Yannis) Gkioulekas (CMU)
Example: Face Detection

1. Select best filter/threshold combination
   a. Normalize the weights
      \[ w_{t,i} \leftarrow \frac{w_{t,j}}{\sum_{j=1}^{n} w_{t,j}} \]
      \[ h_j(x) = \begin{cases} 1 & \text{if } f_j(x) > \theta_j \\ 0 & \text{otherwise} \end{cases} \]
   b. For each feature, \( j \)
      \[ \varepsilon_j = \sum_i w_i |h_j(x_i) - y_i| \]
   c. Choose the classifier, \( h_t \) with the lowest error \( \varepsilon_t \)

2. Re-weight examples
   \[ w_{t+1,i} = w_{t,i} \beta_t^{1-|h_t(x_i)-y_i|} \]
   \[ \beta_t = \frac{\varepsilon_t}{1-\varepsilon_t} \]

Image Credit: Ioannis (Yannis) Gkioulekas (CMU)
**Example**: Face Detection

**Case 1**: Classification for the sample \( i \) is **correct**

\[
\mathbf{w}_{t+1,i} = \mathbf{w}_{t,i} \beta_t
\]

**Case 2**: Classification for the sample \( i \) is **incorrect**

\[
\mathbf{w}_{t+1,i} = \mathbf{w}_{t,i}
\]

2. Re-weight examples

\[
\mathbf{w}_{t+1,i} = \mathbf{w}_{t,i} \beta_t^{1 - |y_i(x_i) - y_i|} \quad \beta_t = \frac{\varepsilon_i}{1 - \varepsilon_t}
\]

**Image Credit**: Ioannis (Yannis) Gkioulekas (CMU)
Example: Face Detection

Case 1: Classification for the sample $i$ is **correct**

$$w_{t+1,i} = w_{t,i} \beta_t$$

Case 2: Classification for the sample $i$ is **incorrect**

$$w_{t+1,i} = w_{t,i}$$

2. Re-weight examples

**Note:** the Beta is $< 1$

$$w_{t+1,i} = w_{t,i} \beta_t^{1 - |h_t(x_i) - y_i|}$$

$$\beta_t = \frac{\varepsilon_i}{1 - \varepsilon_t}$$

Image Credit: Ioannis (Yannis) Gkioulekas (CMU)
Example: Face Detection

1. Select best filter/threshold combination
   
   a. Normalize the weights
   \[ w_{t,i} = \frac{w_{t,i}}{\sum_{j=1}^{n} w_{t,j}} \]
   
   \[ h_j(x) = \begin{cases} 
   1 & \text{if } f_j(x) > \theta_j \\
   0 & \text{otherwise} 
   \end{cases} \]

   b. For each feature, \( j \)
   \[ \epsilon_j = \sum_i w_i |h_j(x_i) - y_i| \]

   c. Choose the classifier, \( h_t \) with the lowest error \( \epsilon_t \)

2. Re-weight examples

   \[ w_{t+1,i} = w_{t,i} \beta_t^{1-|h_t(x_i) - y_i|} \]

   \[ \beta_t = \frac{\epsilon_t}{1 - \epsilon_t} \]

Image Credit: Ioannis (Yannis) Gkioulekas (CMU)
Example: Face Detection

Viola & Jones algorithm

3. The final strong classifier is

\[
h(x) = \begin{cases} 
1 & \sum_{t=1}^{T} \alpha_t h_t(x) \geq \frac{1}{2} \sum_{t=1}^{T} \alpha_t \\
0 & \text{otherwise}
\end{cases}
\]

\[\alpha_t = \log \frac{1}{\beta_t}\]

The final strong classifier is a weighted linear combination of the T weak classifiers where the weights are inversely proportional to the training errors.

Image Credit: Ioannis (Yannis) Gkioulekas (CMU)
Example: Face Detection Summary

Train cascade of classifiers with AdaBoost

Apply to each subwindow

Selected features, thresholds, and weights

New image

Figure credit: K. Grauman
**Example**: Face Detection Summary

**Main Issue**: Efficiency

**Figure credit**: K. Grauman
Observations:

— On average only 0.01% of all sub-windows are positive (faces)
— Equal computation time is spent on all sub-window
— Shouldn’t we spend most time only on potentially positive sub-windows?

Example: Face Detection
**Example**: Face Detection

**Observations**:

— On average only **0.01%** of all sub-windows are positive (faces)
— Equal computation time is spent on all sub-window
— Shouldn’t we spend most time only on *potentially positive* sub-windows?

A simple 2-feature classifier can achieve almost **100% detection rate** (0% false negatives) with **50% false positive rate**
**Observations:**

— On average only **0.01%** of all sub-windows are positive (faces)
— Equal computation time is spent on all sub-window
— Shouldn’t we spend most time only on **potentially positive** sub-windows?

**Solution:**

— A simple 2-feature classifier can act as a 1st layer of a series to filter out most negative (clearly non-face) windows
— 2nd layer with 10 features can tackle “harder” negative-windows which survived the 1st layer, and so on…

**Example:** Face Detection

A simple 2-feature classifier can achieve almost 100% detection rate (0% false negatives) with 50% false positive rate.
Cascading Classifiers

To make detection faster, features can be reordered by increasing complexity of evaluation and the thresholds adjusted so that the early (simpler) tests have few or no false negatives.

Any window that is rejected by early tests can be discarded quickly without computing the other features.

This is referred to as a cascade architecture.

Figure credit: P. Viola
Example: Face Detection Summary

Figure credit: K. Grauman
**Hard Negative Mining**

1. **Pool of Negative Samples**
   - Randomly draw $M^-$ samples

2. **Select $M_h^- (<< M^-)$ samples with highest $f^+$ scores**

3. **A MINIBATCH**
   - Randomly draw $M^+$ samples

4. **Training CNN**

*Image From: Jamie Kang*
Example: Face Detection

Just for fun:

"CV Dazzle, a project focused on finding fashionable ways to thwart facial-recognition technology"

Figure source: Wired, 2015
Today’s “fun” Example: Fooling Face Detection

Fools Viola-Jones detector
Viola-Jones in Action

https://vimeo.com/12774628
Recall: Sliding Window

Train an image classifier as described previously. ‘Slide’ a fixed-sized detection window across the image and evaluate the classifier on each window.

Image credit: KITTI Vision Benchmark
Recall: Sliding Window

Train an image classifier as described previously. ‘Slide’ a fixed-sized detection window across the image and evaluate the classifier on each window.

This is a lot of possible windows! And most will not contain the object we are looking for.

Image credit: KITTI Vision Benchmark
**Object Proposals**

**Object proposal** algorithms generate a short list of regions that have generic object-like properties.

— These regions are likely to contain some kind of foreground object instead of background texture.

The object detector then considers these candidate regions only, instead of exhaustive sliding window search.
Object Proposals

First introduced by Alexe et al., who asked ‘what is an object?’ and defined an ‘objectness’ score based on several visual cues

Figure credit: Alexe et al., 2012

Blue (proposals), Green (ground truth), Red (Stuff – non-objects)
First introduced by Alexe et al., who asked ‘what is an object?’ and defined an ‘objectness’ score based on several visual cues

This work argued that objects typically
— are unique within the image and stand out as salient
— have a contrasting appearance from surroundings and/or
— have a well-defined closed boundary in space

Figure credit: Alexe et al., 2012
**Object Proposals**

Multiscale **Saliency**
— Favours regions with a unique appearance within the image

---

**High scale**

**Low scale**

**Successful Case**
- Cars not salient

**Failure Case**
- Cars not salient

---

**Figure credit**: Alexe et al., 2012
Object Proposals

Colour Contrast
— Favours regions with a contrasting colour appearance from immediate surroundings

Successful Cases
Objects (cyan) have high colour contrast with their surrounding ring.

Failure Case
Colour is not sufficient, here.

Figure credit: Alexe et al., 2012
Object Proposals

Superpixels Straddling
— Favors regions with a well-defined closed boundary
— Measures the extent to which superpixels (obtained by image segmentation) contain pixels both inside and outside of the window

Figure credit: Alexe et al., 2012
**Object Proposals**

**Superpixels** Straddling
- Favors regions with a well-defined closed boundary
- Measures the extent to which superpixels (obtained by image segmentation) contain pixels both inside and outside of the window

![Successful Cases](image1.png) ![Failure Case](image2.png)

**Figure credit**: Alexe et al., 2012
Object Proposals

TABLE 2: For each detector [11, 18, 33] we report its performance (left column) and that of our algorithm 1 using the same window scoring function (right column). We show the average number of windows evaluated per image #win and the detection performance as the mean average precision (mAP) over all 20 classes.

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>mAP</td>
<td>0.186</td>
<td>0.162</td>
<td>0.268</td>
</tr>
<tr>
<td>#win</td>
<td>79945</td>
<td>1349</td>
<td>18562</td>
</tr>
</tbody>
</table>

mAP = mean Average Precision

Table credit: Alexe et al., 2012

Summary

The Viola-Jones face detector constructs a detector using Boosting, which combines weak detectors by re-weighting mis-classified detections. Then a cascade of detectors makes it real-time.

An object proposal algorithm generates a short list of regions with generic object-like properties that can be evaluated by an object detector in place of an exhaustive sliding window search.