Edge Detection
State of The Art

P. Dollar and C. Zitnick
Structured Forests for Fast Edge Detection
ICCV 2013


(Time stamp: Sept 15, 2014)
Testing the Canny Edge Detector

- Let's take this image
- Our goal (a few lectures from now) is to detect objects (cows here)
Testing the Canny Edge Detector

image gradients + NMS

Canny’s edges
Testing the Canny Edge Detector

image gradients + NMS

Canny’s edges
Testing the Canny Edge Detector

- image gradients + NMS
- Canny’s edges

- Lots of “distractor” and missing edges
- Can we do better?
Annotate...

- Imagine someone goes and **annotates** which edges are **correct**
- ... and someone has:
Imagine someone goes and annotates which edges are correct...

... and someone has:

The Berkeley Segmentation Dataset and Benchmark

by D. Martin and C. Fowlkes and D. Tal and J. Malik
How can we make use of such data to improve our edge detector?
How can we make use of such data to **improve** our edge detector?

We can use Machine Learning techniques to:

- **Train classifiers!**

Please learn what a classifier / classification is

- In particular, learn what a **Support Vector Machine** (SVM) is (some links to tutorials are on the class webpage)
- With each week it’s going to be more important to know about this
- You don’t need to learn all the details / math, but to understand the concept enough to know what’s going on
... and do Machine Learning

- How can we make use of such data to improve our edge detector?
- We can use Machine Learning techniques to:

  **Train classifiers!**

- Please **learn what a classifier / classification is**
- In particular, learn what a **Support Vector Machine (SVM)** is (some links to tutorials are on the class webpage)
- With each week it’s going to be more important to know about this
- You don’t need to learn all the details / math, but to understand the concept enough to know what’s going on
Each data point \( \mathbf{x} \) lives in a \( n \)-dimensional space, \( \mathbf{x} \in \mathbb{R}^n \).

We have a bunch of data points \( \mathbf{x}_i \), and for each we have a label, \( y_i \).

A label \( y_i \) can be either 1 (positive example — correct edge in our case), or \(-1\) (negative example — wrong edge in our case).
Classification – a Disney edition (pictures only)

Separating hyperplane:

\[ \mathbf{w}^T \cdot \mathbf{x} + b = 0 \]

\[ x_i \]

\[ y_i = 1 \]

\[ y_i = -1 \]
At training time:

Finding weights $w$ so that positive and negative examples are optimally separated
Classification – a Disney edition (pictures only)

At test time:

\[ w^T \cdot x + b > 0 \rightarrow x \text{ is a positive example} \]
\[ w^T \cdot x + b < 0 \rightarrow x \text{ is a negative example} \]
Training an Edge Detector

- How should we do this?
Training an Edge Detector

- How should we do this?
Training an Edge Detector

- We extract lots of image patches

We call each such crop an **image patch**
Training an Edge Detector

- We extract lots of image patches
- These are our training data

\[\begin{align*}
\text{edge} & \rightarrow \text{no edge} \\
\text{red box} & \rightarrow \text{our training data}
\end{align*}\]
Training an Edge Detector

- We extract lots of image patches
- These are our training data
- We convert each image patch $\mathbf{P}$ (a matrix) into a vector $\mathbf{x}$

$$\Rightarrow \quad \mathbf{x} = \mathbf{P}(,:)$$

matrix $\mathbf{P}$
Training an Edge Detector

- We extract lots of image patches
- These are our training data
- We convert each image patch $\mathbf{P}$ (a matrix) into a vector $\mathbf{x}$
- Well... This works better: Extract image features for each patch

$$\text{compute gradients} \quad \rightarrow \quad \begin{array}{c}
\text{matrix } \mathbf{P} \\
\text{matrix } \mathbf{G}
\end{array} \quad x = \mathbf{G}( :)$$
Training an Edge Detector

- We extract lots of image patches
- These are our training data
- We convert each image patch $P$ (a matrix) into a vector $x$
- Well... This works better: Extract image features for each patch
- Image features are mappings from images (or patches) to other (vector) meaningful representations. More on this in the next class!

\[
x = G(:)
\]

matrix $P$ $\rightarrow$ compute gradients $\rightarrow$ matrix $G$

compute color
Once trained, **how can we use** our new edge detector?
Using an Edge Detector

- We extract all image patches

![Image](image.png)

image

prediction
Using an Edge Detector

- We extract all image patches
- Extract features and use our trained classifier

\[ \text{classify} \quad \rightarrow \quad \text{e.g. score} = \mathbf{w}^T \mathbf{x} + b \]
Using an Edge Detector

- We extract all image patches
- Extract features and use our trained classifier
- Place the predicted value (score) in the output matrix

\[ \text{score} = w^T \mathbf{x} + b \]
Comparisons: Canny vs Structured Edge Detector

- Image
- Image gradients
- Gradients + NMS

“Edginess score”

Score + NMS
Comparisons: Canny vs Structured Edge Detector

- **Canny**
  - Image
  - Image gradient
  - "Edgeness" score

- **Structured Edge Detector**
  - Image
  - Image gradients
  - Gradients + NMS
  - "Edgeness" score
  - Score + NMS
Comparisons: Canny vs Structured Edge Detector

- Image
- Image gradients
- Gradients + NMS

“Edgeness” score
Score + NMS
Comparisons: Canny vs Structured Edge Detector

image

image gradients

gradients + NMS

“edgeness” score

score + NMS
Comparisons: Canny vs Structured Edge Detector

- Image
- Image gradients
- Gradients + NMS

- Image gradient
- “Edgeness” score
- Score + NMS
Figure: green=correct, blue=wrong, red=missing, green+blue=output edges
Evaluation

- **Recall**: How many of all annotated edges we got correct (best is 1)
- **Precision**: How many of all output edges we got correct (best is 1)

Recall = \[
\frac{\# \text{ of green (correct edges)}}{\# \text{ of all edges in ground-truth (second picture)}}
\]
Evaluation

- **Recall**: How many of all annotated edges we got correct (best is 1)
- **Precision**: How many of all output edges we got correct (best is 1)

Precision = \( \frac{\text{\# of green (correct edges)}}{\text{\# of all edges in output (second picture)}} \)
Evaluation

- **Recall**: How many of all annotated edges we got correct (best is 1)
- **Precision**: How many of all output edges we got correct (best is 1)

![Graph showing recall vs precision for different methods with human agreement indicated.](image-url)
Lesson 1

- **Trained detectors** (typically) perform better (true for all applications)

- In this case, the method seems to work better for finding object boundaries (edges) than finding text boundaries. Any idea **why**?

- What would you do if you wanted to detect text (e.g., licence plates)?

- **Think about your problem**, don’t just use code as a black box.