Edge Detection Continued

Edge Detection

Idea: use machine learning!

P. Dollar and C. Zitnick

Structured Forests for Fast Edge Detection

ICCV 2013

Code:

`http://research.microsoft.com/en-us/downloads/389109f6-b4e8-404c-84bf-239f7cbf4e3d/default.aspx`

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

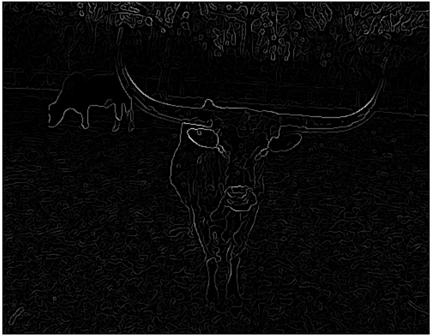


image gradients + NMS

Canny's edges

Testing the Canny Edge Detector

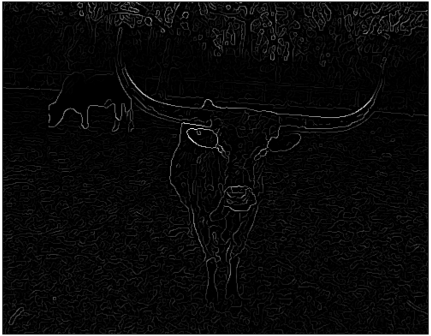


image gradients + NMS

Canny's edges

Testing the Canny Edge Detector

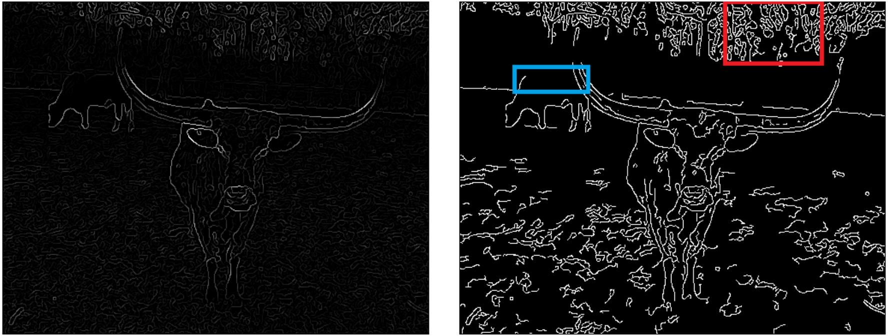


image gradients + NMS   Canny's edges

- Lots of "distractor" and missing edges
- Can we do better?

Annotate...

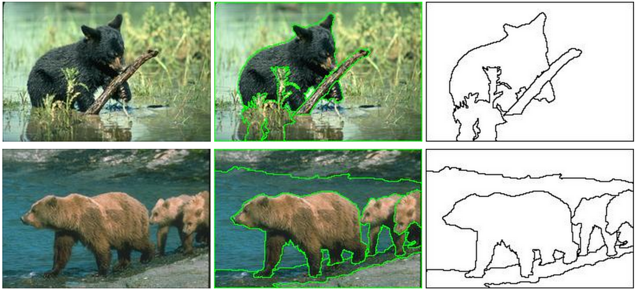
- Imagine someone goes and **annotates** which edges are **correct**
- ... and someone has:
  - 12,000 hand-labeled segmentations of 1,000 Corel dataset images from 30 human subjects

Annotate...

- Imagine someone goes and **annotates** which edges are **correct**
- ... and someone has:
  - 12,000 hand-labeled segmentations of 1,000 Corel dataset images from 30 human subjects

The Berkeley Segmentation Dataset and Benchmark

by D. Martin and C. Fowlkes and D. Tal and J. Malik



... and do Machine Learning

- 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

Classification Problems

Let's think a bit:

- Problem: I want to predict whether it will snow in Oct. What should I do?

Classification Problems

Let's think a bit:

- Problem: I want to predict whether some kid will grow over 2 meters when he grows up

Classification Problems

- You have a prediction problem, e.g. you want to predict whether a coin is a penny or a loonie

Classification Problems

- You have a prediction problem, e.g. you want to predict whether a coin is a penny or a loonie
- You collect **data points** (or examples) with **labels**, e.g. (coin 1, penny), (coin 2, loonie), etc

Classification Problems

- You have a prediction problem, e.g. you want to predict whether a coin is a penny or a loonie
- You collect **data points** (or examples) with **labels**, e.g. (coin 1, penny), (coin 2, loonie), etc
- (Thinking to ourselves: In order to use math, we should convert examples into numbers)

Classification Problems

- You have a prediction problem, e.g. you want to predict whether a coin is a penny or a loonie
- You collect **data points** (or examples) with **labels**, e.g. (coin 1, penny), (coin 2, loonie), etc
- (Thinking to ourselves: In order to use math, we should convert examples into numbers)
- Need to represent each example with a vector, e.g. $\mathbf{x}$. This vector should contain numbers that we quietly hope will be useful for our prediction

Classification Problems

- You have a prediction problem, e.g. you want to predict whether a coin is a penny or a loonie
- You collect **data points** (or examples) with **labels**, e.g. (coin 1, penny), (coin 2, loonie), etc
- (Thinking to ourselves: In order to use math, we should convert examples into numbers)
- Need to represent each example with a vector, e.g. $\mathbf{x}$. This vector should contain numbers that we quietly hope will be useful for our prediction
- Let's also assign a number to each possible label, e.g. -1 to *penny* and 1 to *loonie*

Classification Problems

- You have a prediction problem, e.g. you want to predict whether a coin is a penny or a loonie
- You collect **data points** (or examples) with **labels**, e.g. (coin 1, penny), (coin 2, loonie), etc
- (Thinking to ourselves: In order to use math, we should convert examples into numbers)
- Need to represent each example with a vector, e.g. $\mathbf{x}$. This vector should contain numbers that we quietly hope will be useful for our prediction
- Let's also assign a number to each possible label, e.g. -1 to *penny* and 1 to *loonie*
- We are ready for math

Classification Examples

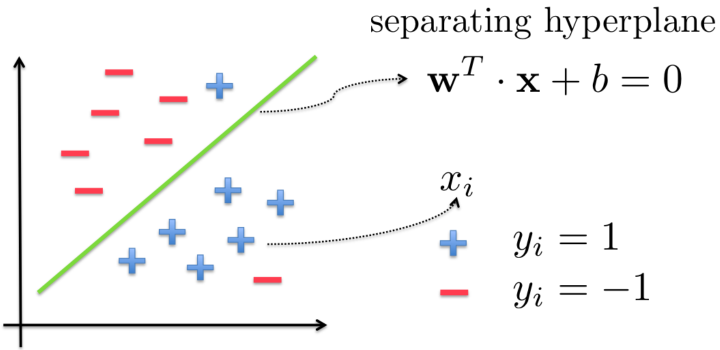
- Each data point **x** lives in a n-dimensional space, $x \in \mathbb{R}^n$
- For example, an n pixel image becomes a length n vector
- Or image is compressed into two features: object size, colour

Source: Wikipedia

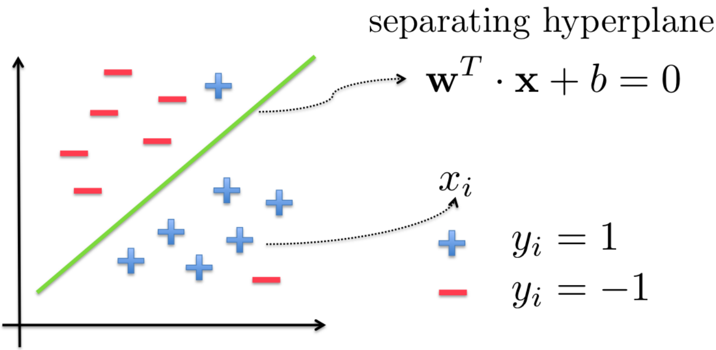
Classification Examples

- Each data point $\mathbf{x}$ lives in a n-dimensional space, $x \in \mathbb{R}^n$
- For example, an n pixel image becomes a length n vector
- A better idea might be to compress (project) into two features: object size, colour
- 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 – penny in our case), or -1 (negative example – loonie in our case)

Classification Problems)



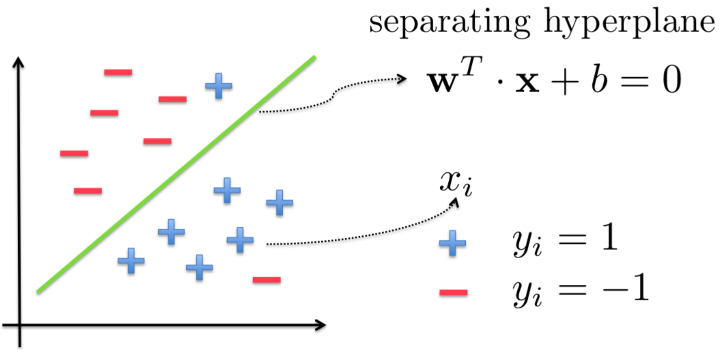
Classification Problems)



At training time:

Finding **weights w** so that positive and negative examples are optimally separated

Classification Problems)



At test time:

$\mathbf{w}^T \cdot \mathbf{x} + b > 0 \rightarrow \mathbf{x}$ is a positive example

$\mathbf{w}^T \cdot \mathbf{x} + b < 0 \rightarrow \mathbf{x}$ is a negative example

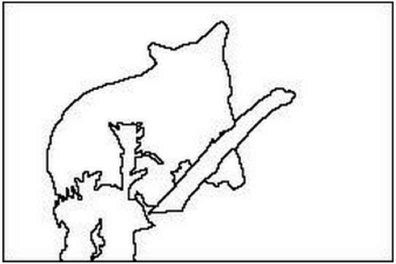
Training an Edge Detector

- How should we do this?

Training an Edge Detector

- How should we do this?

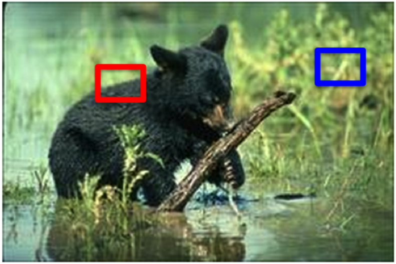
image

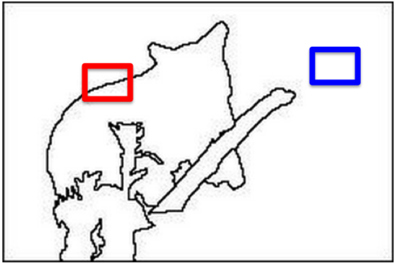


annotation

Training an Edge Detector

- We extract lots of image patches

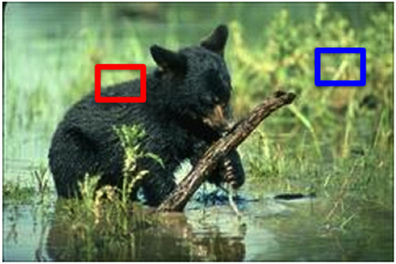


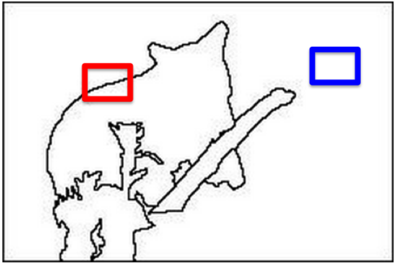


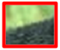
We call each such crop an **image patch**

Training an Edge Detector

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





 $\rightarrow$ edge

 $\rightarrow$ no edge

} our training data

Training an Edge Detector

- We extract lots of image patches
- These are our training data
- We need to do something with each of our data samples (image patches **P**) to represent each one with a vector (representing measurements about the patch) **x**. The simplest possibility in our case would be to just vectorize an image patch. Any problems with this?

matrix $\mathbf{P}$

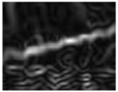
$$\rightarrow \quad \mathbf{x} = \mathbf{P}(:)$$

Training an Edge Detector

- We extract lots of image patches
- These are our training data
- This works better: Extract meaningful **image features** such as gradients, a color histogram, etc, representing each patch

matrix $\mathbf{P}$

$\xrightarrow{\text{compute gradients}}$



matrix $\mathbf{G}$

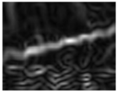
$$\mathbf{x} = \mathbf{G}(:)$$

Training an Edge Detector

- We extract lots of image patches
- These are our training data
- This works better: Extract meaningful **image features** such as gradients, a color histogram, etc, representing each patch
- Image features are mappings from images (or patches) to other (vector) meaningful representations.

matrix $\mathbf{P}$

$\xrightarrow{\text{compute gradients}}$



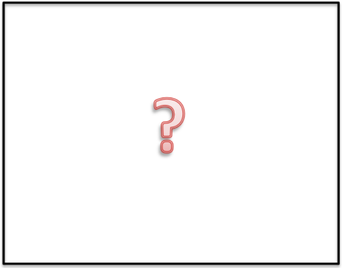
matrix $\mathbf{G}$

$$\mathbf{x} = \mathbf{G}(:)$$

$\xrightarrow{\text{compute color}}$

- Once trained, **how can we use** our new edge detector?

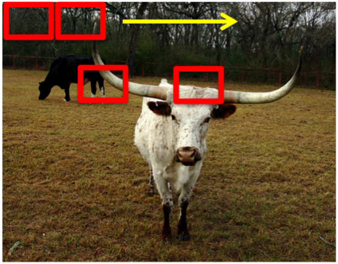

image



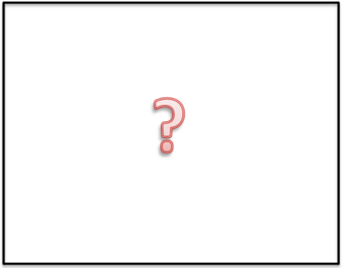
prediction

Using an Edge Detector

- We extract all (overlapping) image patches



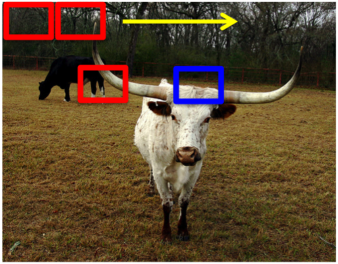
image



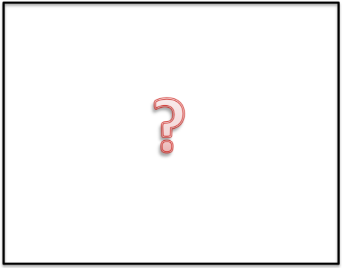
prediction

Using an Edge Detector

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



image

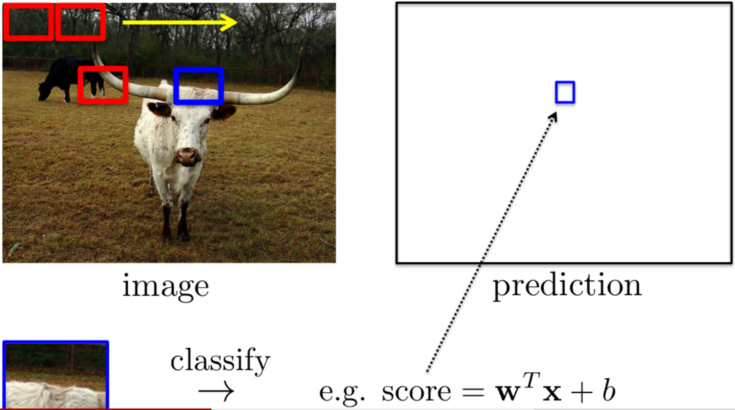


prediction

$$\xrightarrow{\text{classify}} \qquad \text{e.g. score} = \mathbf{w}^T\mathbf{x} + b$$

Using an Edge Detector

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



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


“edgeness” score


score + NMS

Comparisons: Canny vs Structured Edge Detector

image

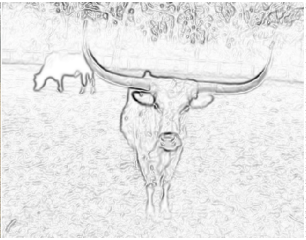
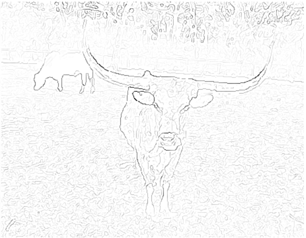
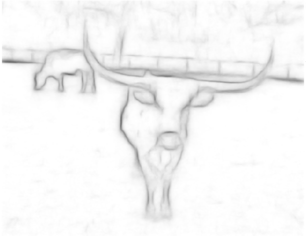


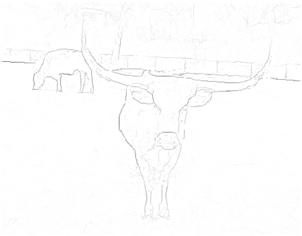
image gradients



gradients + NMS



“edgeness” score



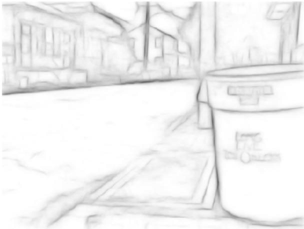
score + NMS

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

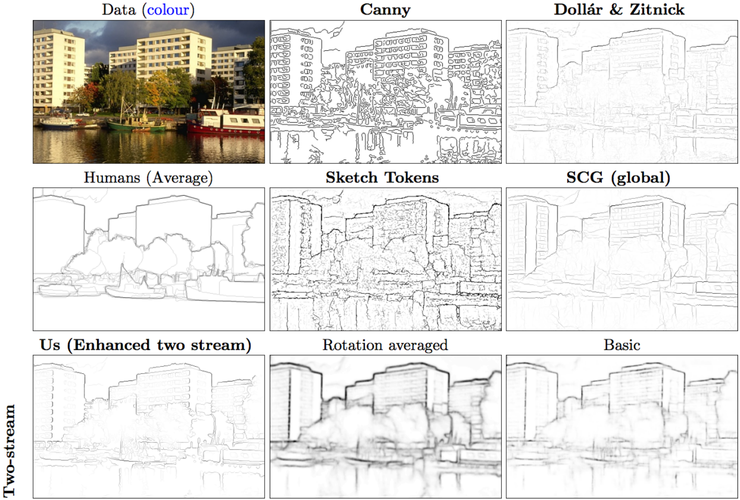
“edgeness” score

“edgeness” score

score + NMS

Deep Approach

- You can use more fancy classifiers (e.g., Neural Networks)



[Kivien, Williams, Hees. Visual Boundary Prediction: A Deep Neural Prediction Network and Quality Dissection. AISTATS'2014]

Evaluation

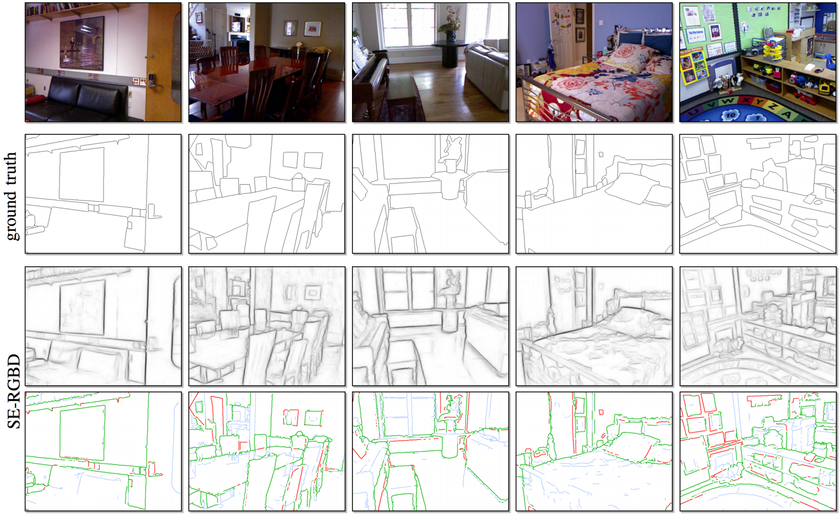
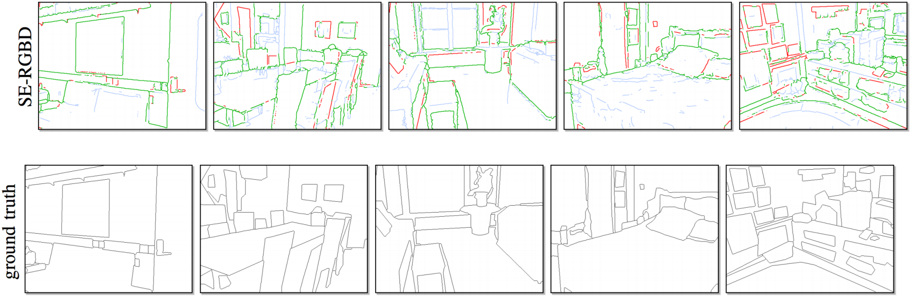


Figure: green=correct, blue=wrong, red=missing, green+blue=output edges

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)

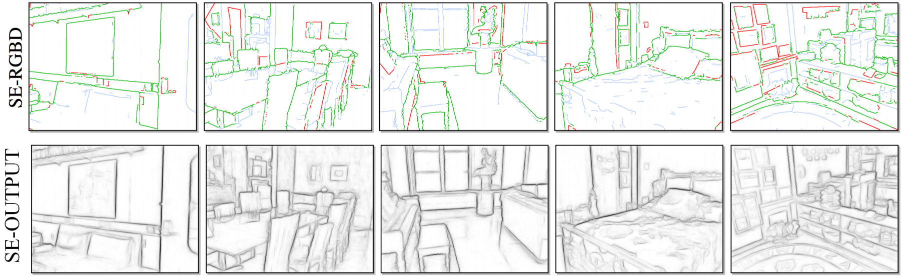
$$\textbf{Recall} = \frac{\#\text{ of green (correct edges)}}{\#\text{ of all edges in ground-truth (second Row)}}$$



Evaluation

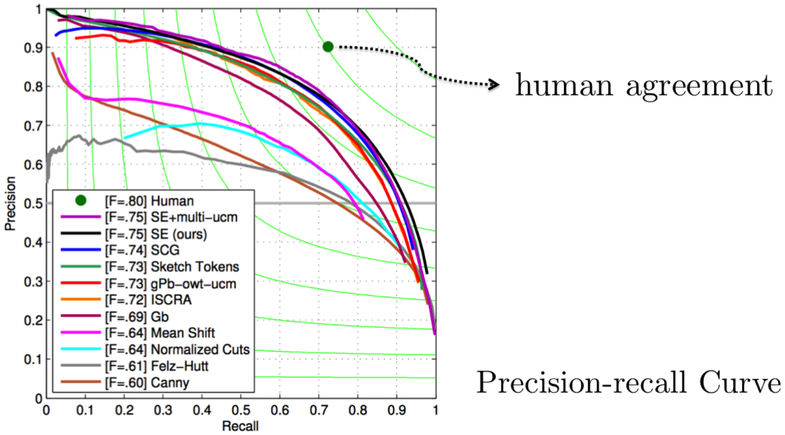
- **Recall:** How many of all **annotated** edges we got correct (best is 1)
- **Precision** How many of all **output** edges we got correct (best is 1)

$$\textbf{Precision} = \frac{\text{\# of green (correct edges)}}{\text{\# of all edges in output (second Row)}}$$



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)



Lesson 1

- **Trained detectors** (typically) perform better (not 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

Lesson 1

- **Trained detectors** (typically) perform better (not 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
- **Great news:** This type of approach can also be used to detect objects (cars, cows, people, etc)! More about it later in class