Can we do something cool with gradients already?

S. Avidan and A. Shamir

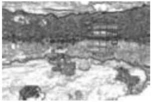
Seam Carving for Content-Aware Image Resizing

SIGGRAPH 2007

Paper: http://www.win.tue.nl/~wstahw/edu/2IV05/seamcarving.pdf

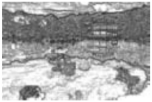
Simple Application: Seam Carving

- Imagine we want to rescale this by factor 2 in only one direction

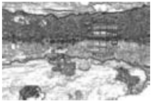




Simple Application: Seam Carving

- Content-aware resizing



- Find path from top to bottom row with minimum gradient energy
- Remove (or replicate) those pixels



Seam Carving

- A vertical seam $\mathbf{s}$ is a list of column indices, one for each row, where each subsequent column differs by no more than one slot.
- Let G denote the image gradient magnitude. Optimal 8-connected path:

$$\mathbf{s}^* = \mathrm{argmin}_{\mathbf{s}} E(\mathbf{s}) = \mathrm{argmin}_{\mathbf{s}} \sum_{i=1}^{n} G(s_i)$$

- Can be computed via dynamic programming
- Compute the cumulative minimum energy for all possible connected seams at each entry (i, j):

$$M(i,j) = G(i,j) + \min\left(M(i-1, j-1), M(i-1, j), M(i-1, j+1)\right)$$

- Backtrack from min value in last row of M to pull out optimal seam path.

Seam Carving – Examples

Edge Detection
State of The Art

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

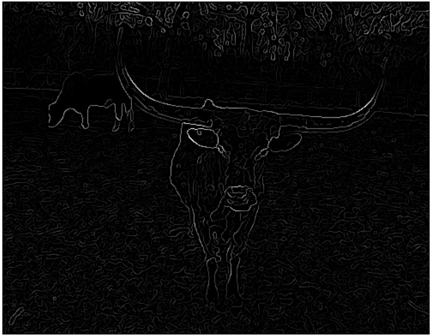


image gradients + NMS

Canny's edges

Testing the Canny Edge Detector

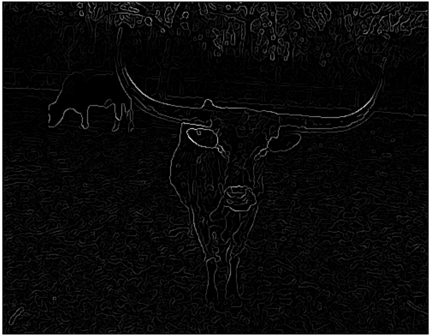


image gradients + NMS

Canny's edges

Testing the Canny Edge Detector

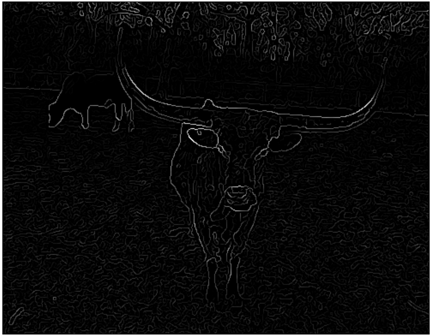


image gradients + NMS

Canny's edges

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

Annotate...

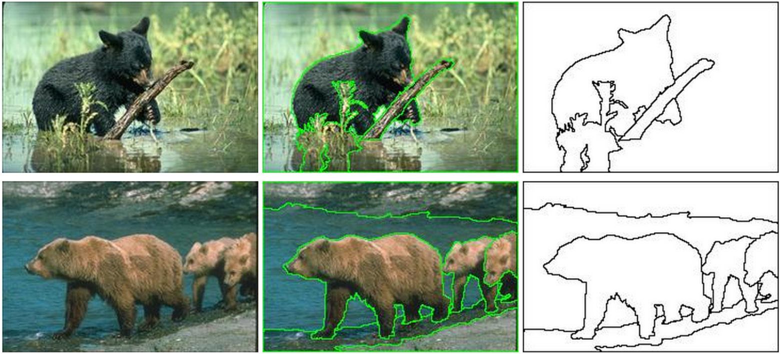
- Imagine someone goes and **annotates** which edges are **correct**
- ... and someone has:

Annotate...

- 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



... and do Machine Learning

- How can we make use of such data to **improve** our edge detector?

... 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

... 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 – a Disney edition (pictures only)

- You have a prediction problem, e.g. you want to predict whether a prof will be late for class

Classification – a Disney edition (pictures only)

- You have a prediction problem, e.g. you want to predict whether a prof will be late for class
- You collect **data points** (or training examples) with **labels**, e.g. (Sanja-Lecture 1, not late), (Sanja-Lecture 2, late), etc

Classification – a Disney edition (pictures only)

- You have a prediction problem, e.g. you want to predict whether a prof will be late for class
- You collect **data points** (or training examples) with **labels**, e.g. (Sanja-Lecture 1, not late), (Sanja-Lecture 2, late), etc
- (Thinking to ourselves: In order to use math, we should convert examples into numbers)

Classification – a Disney edition (pictures only)

- You have a prediction problem, e.g. you want to predict whether a prof will be late for class
- You collect **data points** (or training examples) with **labels**, e.g. (Sanja-Lecture 1, not late), (Sanja-Lecture 2, late), 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. $\mathbf{x}$, typically called a **feature vector**. This vector should contain numbers that we quietly hope will be useful for our prediction

Classification – a Disney edition (pictures only)

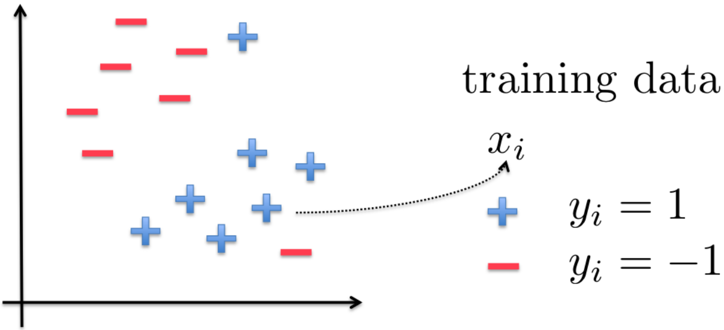
- You have a prediction problem, e.g. you want to predict whether a prof will be late for class
- You collect **data points** (or training examples) with **labels**, e.g. (Sanja-Lecture 1, not late), (Sanja-Lecture 2, late), etc
- (Thinking to ourselves: In order to use math, we should convert examples into numbers)
- Need to represent each example with a vector, e.g. $\mathbf{x}$, typically called a **feature vector**. 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 *not-late* and 1 to *late*

Classification – a Disney edition (pictures only)

- You have a prediction problem, e.g. you want to predict whether a prof will be late for class
- You collect **data points** (or training examples) with **labels**, e.g. (Sanja-Lecture 1, not late), (Sanja-Lecture 2, late), 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. $\mathbf{x}$, typically called a **feature vector**. 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 *not-late* and 1 to *late*
- We are ready for math

Classification – a Disney edition (pictures only)

- Each data point **x** lives in a *n*-dimensional space, $x \in \mathbb{R}^n$
- We have many 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)

Let's think a bit:

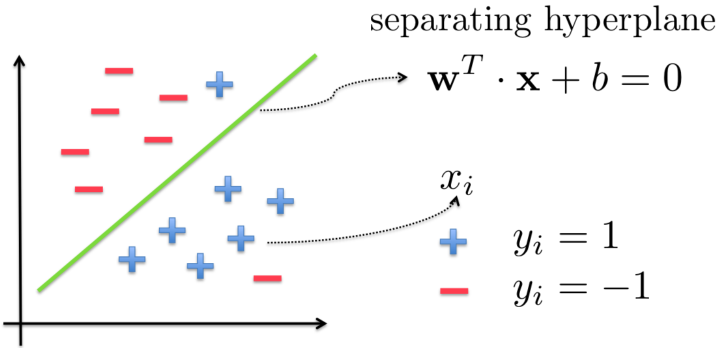
- Problem: I want to predict whether it will snow tomorrow. What should I do?

Classification – a Disney edition (pictures only)

Let's think a bit:

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

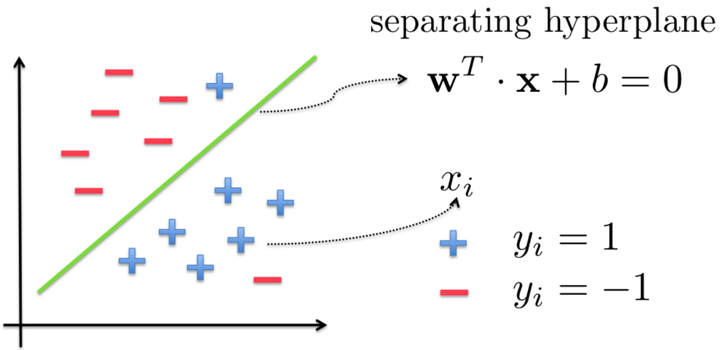
Classification – a Disney edition (pictures only)



- We define a **model**, for example a linear function:

$$f(\mathbf{x}; \mathbf{w}) = \mathbf{w}^T \mathbf{x} + b$$

Classification – a Disney edition (pictures only)



At training time:

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

Classification – a Disney edition (pictures only)

Training:

- We typically have an objective or **loss function** that measures how well our model fits the data:

$$\mathrm{loss}(\{\mathbf{x}, y\}_i; \mathbf{w})$$

- A very simple loss function is:

$$\mathrm{loss}(\{\mathbf{x}, y\}_i; \mathbf{w}) = \sum_{i=1}^{N} (f(\mathbf{x}_i; \mathbf{w}) - y_i)^2$$

  but this one is typically not well suited for classification (why?). Better loss functions: cross-entropy loss, hinge loss (used for SVMs), etc

Classification – a Disney edition (pictures only)

Training:

- We typically have an objective or **loss function** that measures how well our model fits the data:

$$\text{loss}(\{\mathbf{x}, y\}_i; \mathbf{w})$$

- A very simple loss function is:

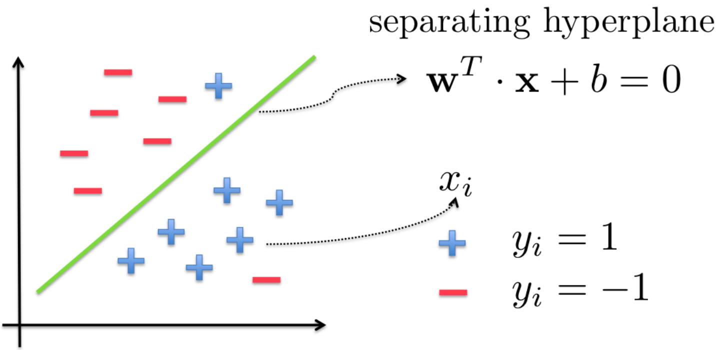
$$\text{loss}(\{\mathbf{x}, y\}_i; \mathbf{w}) = \sum_{i=1}^{N} (f(\mathbf{x}_i; \mathbf{w}) - y_i)^2$$

but this one is typically not well suited for classification (why?). Better loss functions: cross-entropy loss, hinge loss (used for SVMs), etc

- We are trying to find:

$$\min_{\mathbf{w}} \ \text{loss}(\{\mathbf{x}, y\}_i; \mathbf{w})$$

Classification – a Disney edition (pictures only)



At **test** time:

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

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

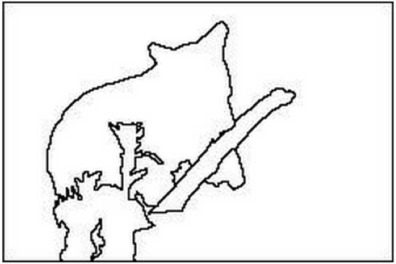
Training an Edge Detector

- How should we do this?

Training an Edge Detector

- How should we do this?

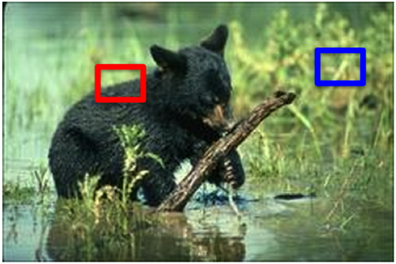
image

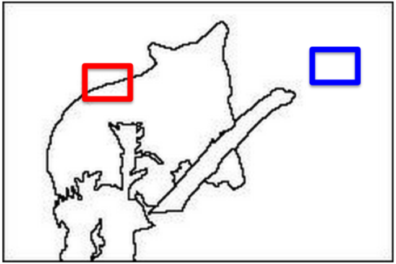


annotation

Training an Edge Detector

- We extract lots of image patches

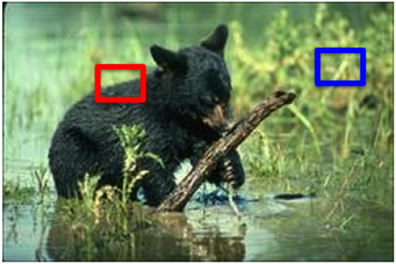


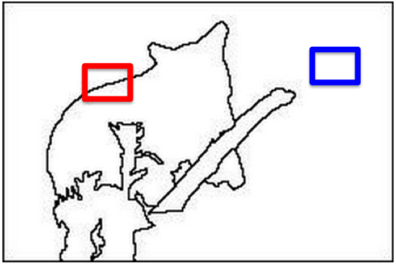


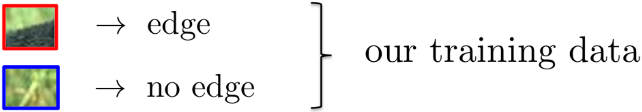
We call each such crop an **image patch**

Training an Edge Detector

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







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?

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

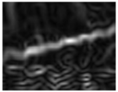
matrix $\mathbf{P}$

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 $\mathbf{P}$

compute gradients $\longrightarrow$



matrix $\mathbf{G}$

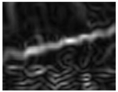
$$\mathbf{x} = \mathbf{G}(:)$$

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 $\mathbf{P}$

compute gradients $\longrightarrow$



matrix $\mathbf{G}$

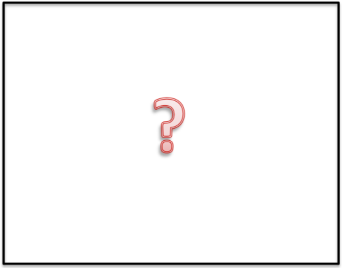
$$\mathbf{x} = \mathbf{G}(:)$$

compute color $\longrightarrow$

Using an Edge Detector

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

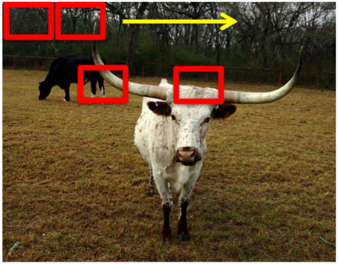
image



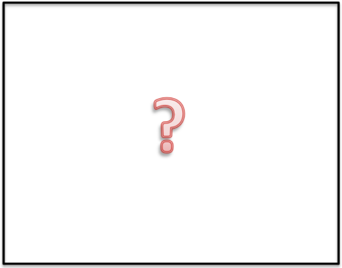
prediction

Using an Edge Detector

- We extract all image patches



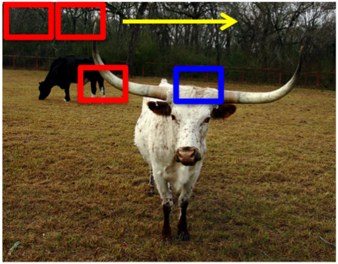
image



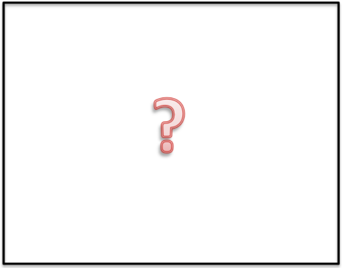
prediction

Using an Edge Detector

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



image

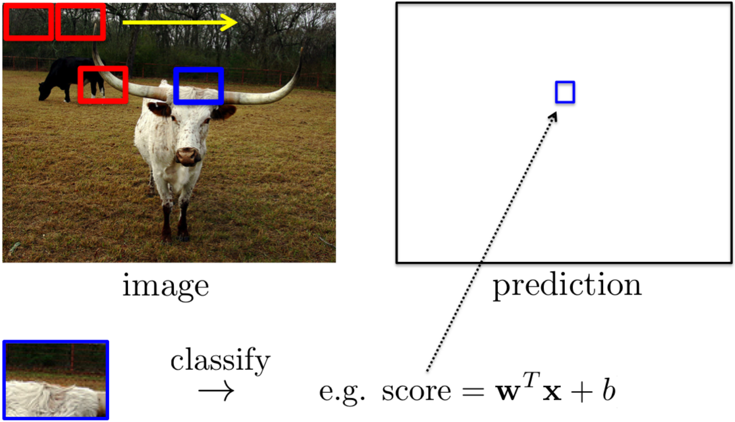


prediction

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

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



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

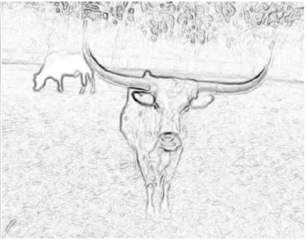
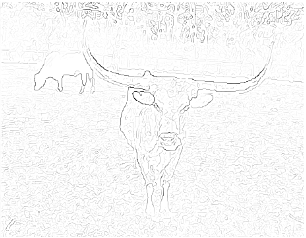
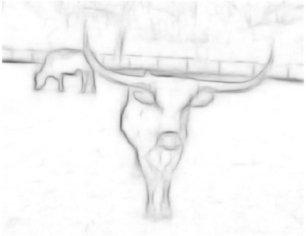


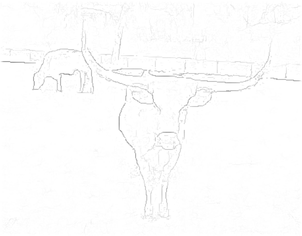
image gradients



gradients + NMS



"edgeness" score



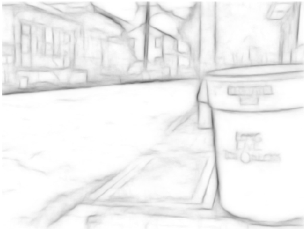
score + NMS

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

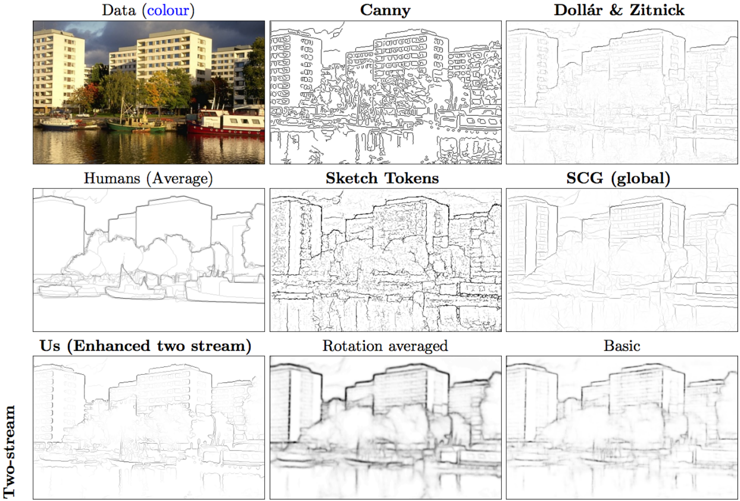
"edgeness" score

"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

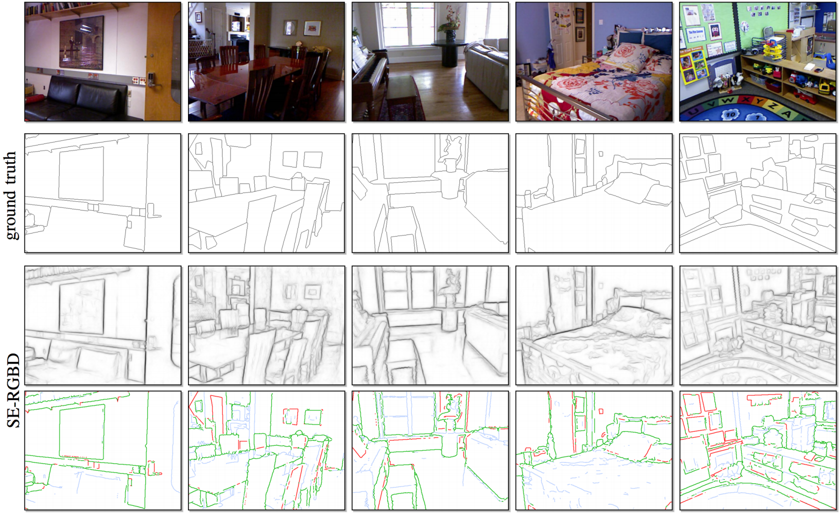
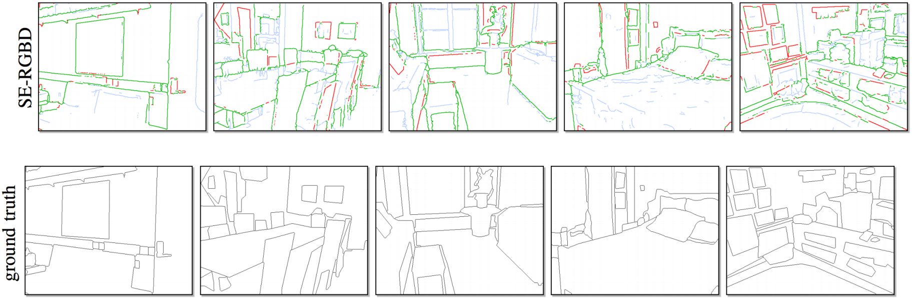


Figure: green=correct, blue=wrong, red=missing, green+blue=output edges

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)

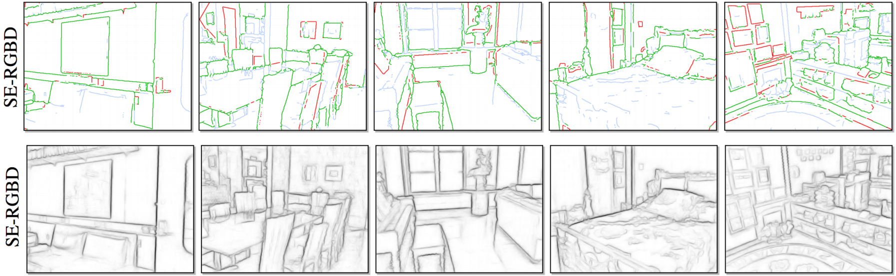
$$\textbf{Recall} = \frac{\#\text{ of green (correct edges)}}{\#\text{ of all edges in ground-truth (second picture)}}$$



Evaluation

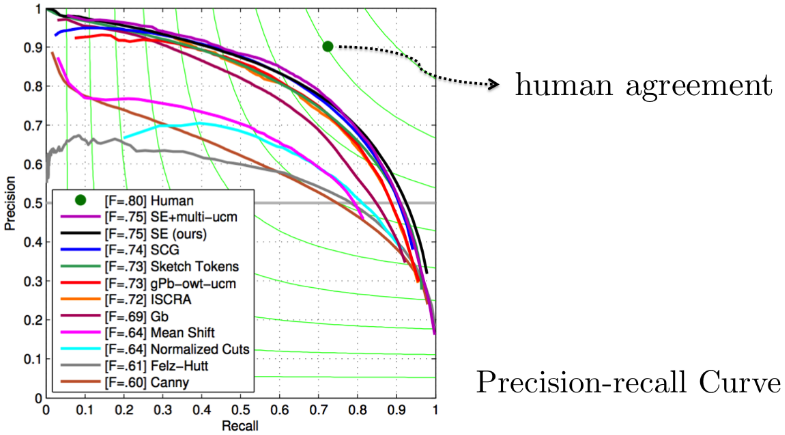
- **Recall:** How many of all **annotated** edges we got correct (best is 1)
- **Precision** How many of all **output** edges we got correct (best is 1)

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



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)



Precision-recall Curve

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

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
- **Great news:** This type of approach can also be used to detect objects (cars, cows, people, etc)! More about it later in class