Image filtering and image features

September 26, 2019

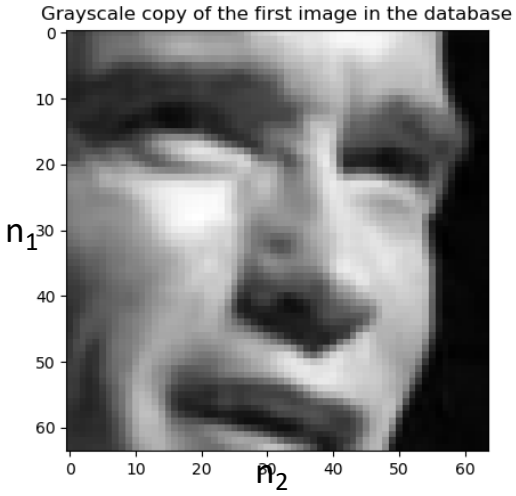
Outline: Image filtering and image features

- Images as signals
- Color spaces and color features
- 2D convolution
- Matched filters
- Gradient filters
- Separable convolution
- Accuracy spectrum of a 1-feature classifier

Images as signals

- x[n_1,n_2,c] = intensity in row n1, column n2, color plane c.
- Most image formats (e.g., JPG, PNG, GIF, PPM) distribute images with three color planes: Red, Green, and Blue (RGB)
- In this example (Arnold Schwarzenegger's face), the grayscale image was created as

$$\bar{x}\,[n_1, n_2] = \frac{1}{3} \sum_{c \in \{R,G,B\}} x[n_1, n_2, c]$$



Color spaces: RGB

- Every natural object reflects a continuous spectrum of colors.
- However, the human eye only has three color sensors:
  - Red cones are sensitive to lower frequencies
  - Green cones are sensitive to intermediate frequencies
  - Blue cones are sensitive to higher frequencies
- By activating LED or other display hardware at just three discrete colors (R, G, and B), it is possible to fool the human eye into thinking that it sees a continuum of colors.
- Therefore, most image file formats only code three discrete colors (RGB).

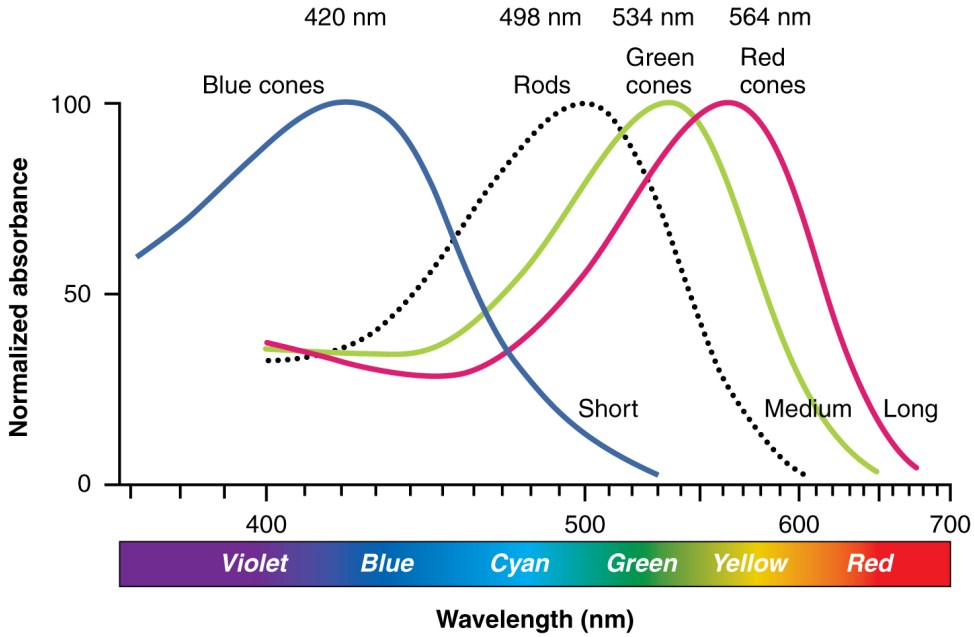


Illustration from Anatomy & Physiology, Connexions Web site. http://cnx.org/content/col11496/1.6/, Jun 19, 2013.

Color features: Luminance

- The "grayscale" image is often computed as the average of R, G, and B intensities, i.e., $\bar{x}\,[n_1, n_2] = \frac{1}{3}\sum_{c\in\{R,G,B\}} x[n_1, n_2, c]$.
- The human eye, on the other hand, is more sensitive to green light than to either red or blue.
- The intensity of light, as viewed by the human eye, is well approximated by the standard ITU-R BT.601:

$$x[n_1, n_2, Y] = 0.299x[n_1, n_2, R] + 0.587x[n_1, n_2, G] + 0.114x[n_1, n_2, B]$$

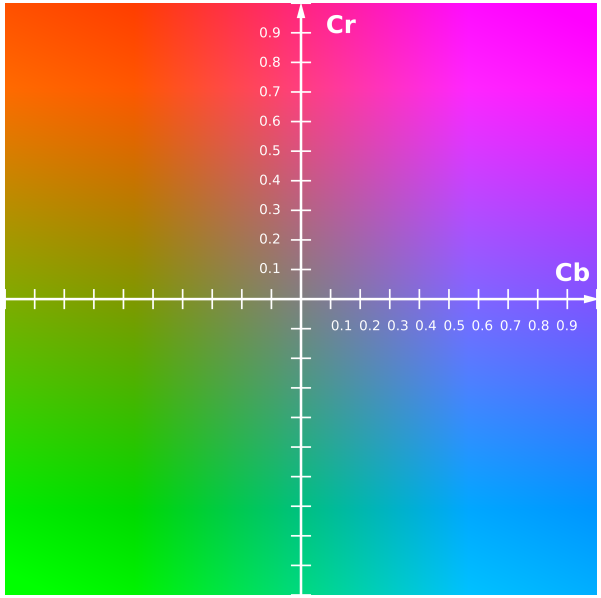
- This signal ($x[n_1, n_2, Y]$) is called the **luminance** of light at pixel $[n_1, n_2]$.

Color features: Chrominance

- Chrominance = color-shift of the image.
- We measure P_R=red-shift, and P_B=blue-shift, relative to luminance (luminance is sort of green-based, remember?)
- We want $P_R[n_1, n_2]$ and $P_B[n_1, n_2]$ to describe only the color-shift of the pixel, not its average luminance.
- We do that using

$$\begin{bmatrix} Y \\ P_B \\ P_R \end{bmatrix} = \begin{bmatrix} \vec{v}_Y \\ \vec{v}_B \\ \vec{v}_R \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix}$$

Where $sum(\vec{v}_R) = sum(\vec{v}_B) = 0$.

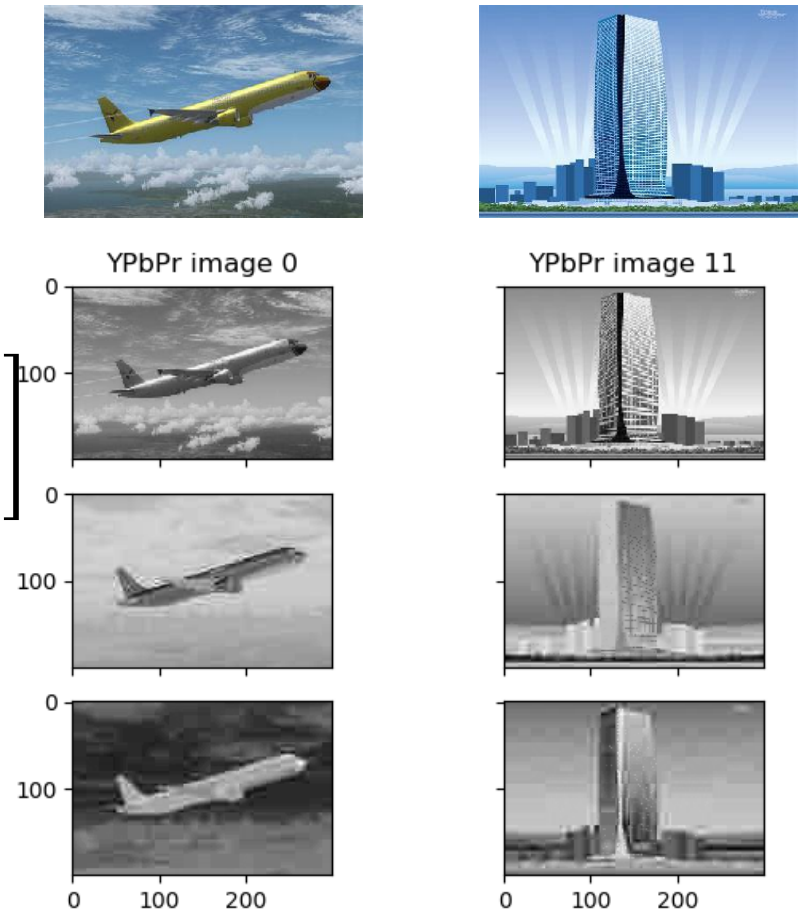


Cr and Cb, at Y=0.5
Simon A. Eugster, own work.

Color features: Chrominance

$$\begin{bmatrix} Y \\ P_B \\ P_R \end{bmatrix} = \begin{bmatrix} 0.299 & 0.587 & 0.114 \\ -0.168736 & -0.331264 & 0.5 \\ 0.5 & -0.418688 & -0.081312 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix}$$

gives $sum(\vec{v}_R) = sum(\vec{v}_B) = 0.$



Color features: Chrominance

- Some images are obviously red! (e.g., fire, or wood)
- Some images are obviously blue! (e.g., water, or sky)
- Average(Pb)-Average(Pr) should be a good feature for distinguishing between, for example, ”fire” versus “water”

Color features: norms

- The average Pb value is $\bar{P}_B = \frac{1}{N_1 N_2} \sum_{n_1=0}^{N_1-1} \sum_{n_2=0}^{N_2-1} P_B[n_1, n_2]$.
  - The problem with this feature is that it gives too much weight to small values of $P_B[n_1, n_2]$, i.e., some pixels might not be all that bluish – as a result, some "water" images have low average-pooled Pb.
- The max Pb value is $\hat{P}_B = \max_{n_1} \max_{n_2} P_B[n_1, n_2]$.
  - The problem with this feature is that it gives too much weight to LARGE values of $P_B[n_1, n_2]$, i.e., in the "fire" image, there might be one or two pixels that are blue, even though all of the others are red --- as a result, some "fire" images might have an unreasonably high max-pooled Pb.
- The Frobenius norm is $\|P_B\| = \left(\sum_{n_1=0}^{N_1-1} \sum_{n_2=0}^{N_2-1} P_B^2[n_1, n_2] \right)^{1/2}$
  - The Frobenius norm emphasizes large values, but it doesn't just depend on the LARGEST value – it tends to resemble an average of the largest values.
- In MP3, Frobenius norm seems to be work better than max-pooling or average-pooling. For other image processing problems, you might want to use average-pooling or max-pooling instead.

Outline: Image filtering and image features

- Images as signals
- Color spaces and color features
- 2D convolution
- Matched filters
- Gradient filters
- Separable convolution
- Accuracy spectrum of a 1-feature classifier

2D convolution

The 2D convolution is just like a 1D convolution, but in two dimensions.

$$x[n_1, n_2, c] ** h[n_1, n_2, c] = \sum_{m_1=0}^{N_1-1} \sum_{m_2=0}^{N_2-1} x[m_1, m_2, c] h[n_1 - m_1, n_2 - m_2, c]$$

Note that we don't convolve over the color plane – just over the rows and columns.

Full, Valid, and Same-size convolution outputs

$$y[n_1, n_2, c] = \sum_{m_1=0}^{N_1-1} \sum_{m_2=0}^{N_2-1} x[m_1, m_2, c] h[n_1 - m_1, n_2 - m_2, c]$$

Suppose that x is an N_1xN_2 image, while h is a filter of size M_1xM_2. Then there are three possible ways to define the size of the output:

- "Full" output: Both $x[m_1, m_2]$ and $h[m_1, m_2]$ are zero-padded prior to convolution, and then $y[n_1, n_2]$ is defined wherever the result is nonzero. This gives $y[n_1, n_2]$ the size of (N_1+M_1-1)x(N_2+M_2-1).
- "Same" output: The output, $y[n_1, n_2]$, has the size N_1xN_2. This means that there is some zero-padding.
- "Valid" output: The summation is only performed for values of (n1,n2,m1,m2) at which both x and h are well-defined. This gives $y[n_1, n_2, c]$ the size of (N_1-M_1+1)x(N_2-M_2+1).

Example: differencing

Suppose we want to calculate the difference between each pixel, and its second neighbor:

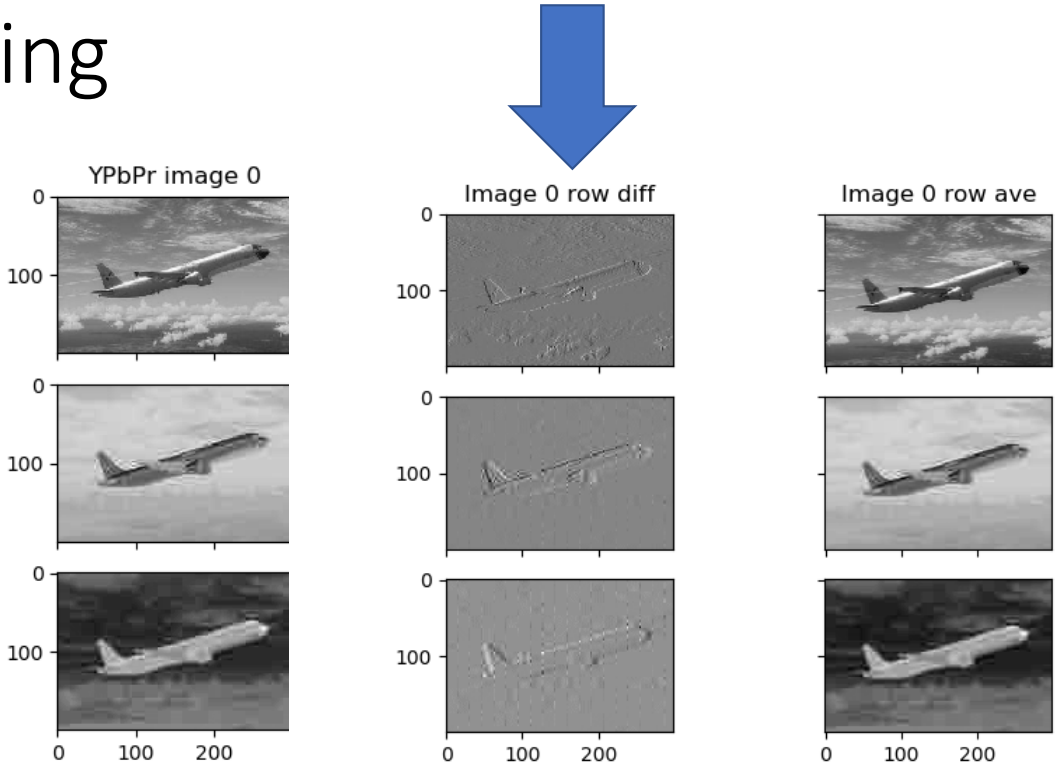
$$y[n_1, n_2] = x[n_1, n_2] - x[n_1, n_2 - 2]$$

We can do that as

$$y = \sum_{m_1=0}^{N_1-1} \sum_{m_2=0}^{N_2-1} x[m_1, m_2] h[n_1 - m_1, n_2 - m_2]$$

where

$$h[n_1, n_2] = \begin{cases} 1 & n_1 = 0, n_2 = 0 \\ -1 & n_1 = 0, n_2 = 2 \\ 0 & else \end{cases}$$



…we often will write this as h=[1,0,-1].

Example: averaging

Suppose we want to calculate the average between each pixel, and its two neighbors:

$$y[n_1, n_2] = x[n_1, n_2] + 2x[n_1, n_2 - 1] + x[n_1, n_2 - 2]$$

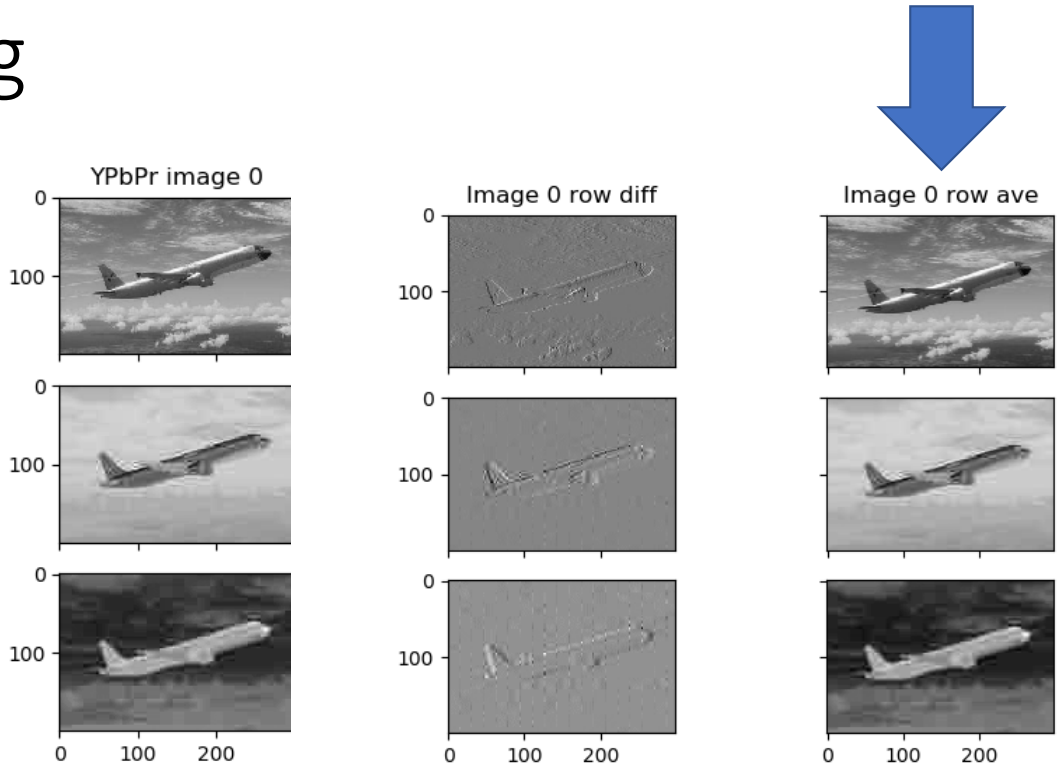
We can do that as

$$y = \sum_{m_1=0}^{N_1-1} \sum_{m_2=0}^{N_2-1} x[m_1, m_2] h[n_1 - m_1, n_2 - m_2]$$

where

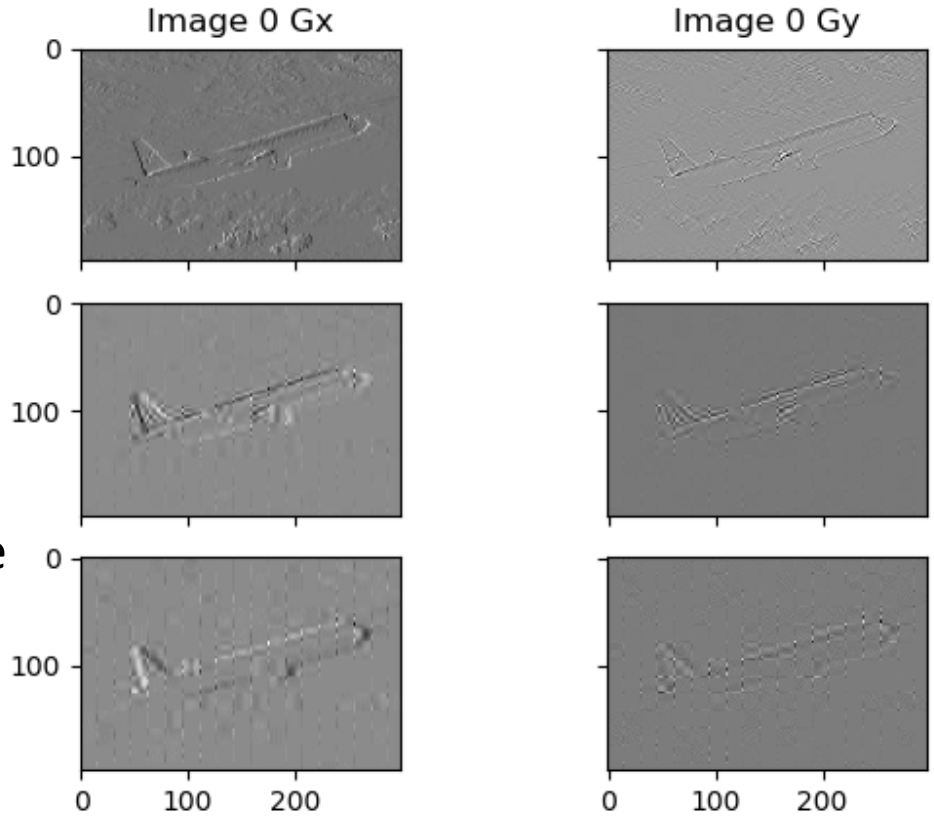
$$h[n_1, n_2] = \begin{cases} 1 & n_1 = 0, n_2 \in \{0,2\} \\ 2 & n_1 = 0, n_2 = 1 \\ 0 & else \end{cases}$$

…we often will write this as h=[1,2,1].



The two ways we'll use convolution in mp3

1. **Matched filtering**: The filter is designed to pick out a particular type of object (e.g., a bicycle, or a Volkswagon beetle). The output of the filter has a large value when the object is found, and a small random value otherwise.
2. **Gradient**: Two filters are designed, one to estimate the horizontal image gradient $G_x[n_1, n_2, c] = \frac{\delta}{\delta n_2} x[n_1, n_2, c]$, and one to estimate the vertical image gradient $G_y[n_1, n_2, c] = \frac{\delta}{\delta n_1} x[n_1, n_2, c]$



Outline: Image filtering and image features

- Images as signals
- Color spaces and color features
- 2D convolution
- Matched filters
- Gradient filters
- Separable convolution
- Accuracy spectrum of a 1-feature classifier

Matched filter is the solution to the “signal detection” problem.

Suppose we have a noisy signal, x[n]. We have two hypotheses:

- H0: x[n] is just noise, i.e., x[n]=v[n], where v[n] is a zero-mean, unit-variance Gaussian white noise signal.
- H1: x[n]=s[n]+v[n], where v[n] is the same random noise signal, but s[n] is a deterministic (non-random) signal that we know in advance.

We want to create a hypothesis test as follows:

1. Compute y[n]=h[n]*x[n]
2. If y[0] > threshold, then conclude that H1 is true (signal present). If y[0] < threshold, then conclude that H0 is true (signal absent).

Can we design h[n] in order to maximize the probability that this classifier will give the right answer?

The "signal detection" problem

$$y[n] = x[n] * h[n] = s[n] * h[n] + v[n] * h[n]$$

- Call it w[n]: $\mathrm{w[n]} = v[n] * h[n] = \sum v[m]\, h[n-m]$ is a Gaussian random variable with zero average.
  - The weighted sum of Gaussians is also a Gaussian
  - $\mathrm{E}[w[m]] = 0$ because $\mathrm{E}[v[m]] = 0$
- The variance is $\sigma_w^2 = \sum \sigma_v^2 h^2\ [n-m] = \sum h^2\ [n-m]$
  - (because we assumed that $\sigma_v^2 = 1$).
  - Suppose we constrain h[n] as $\sum h^2\ [n-m] = 1$. Then we have $\sigma_w^2 = 1$.
- So under H0 (signal absent), y[n] is a zero-mean, unit-variance Gaussian random signal.

The “signal detection” problem

$$y[n] = x[n] * h[n] = s[n] * h[n] + w[n]$$

So w[0] is a zero-mean, unit-variance Gaussian random variable.

We have two hypotheses:

- H0: $y[0] = w[0]$
- H1: $y[0] = w[0] + \sum s[m]h[0 - m]$

Goal: we know s[m]. We want to design h[m] so that $\sum s[m]h[-m]$ is as large as possible, subject to the constraint that $\sum h^2\,[n - m] = 1$.

The solution: matched filters

Goal: we know s[m]. We want to design h[m] so that $\sum s[m]h[-m]$ is as large as possible, subject to the constraint that $\sum h^2\,[n-m] = 1$.

The solution: $h[m] \propto s[-m]$.

(Specifically, $h[m] = s[-m]/\sqrt{\sum s^2\,[m]}$)

Under H0 (signal absent), y[0] is a zero-mean unit-variance Gaussian (ZMUVG):

$$y[0] = w[0]$$

Then under H1 (signal present), y[0] is a ZMUVG + 1:

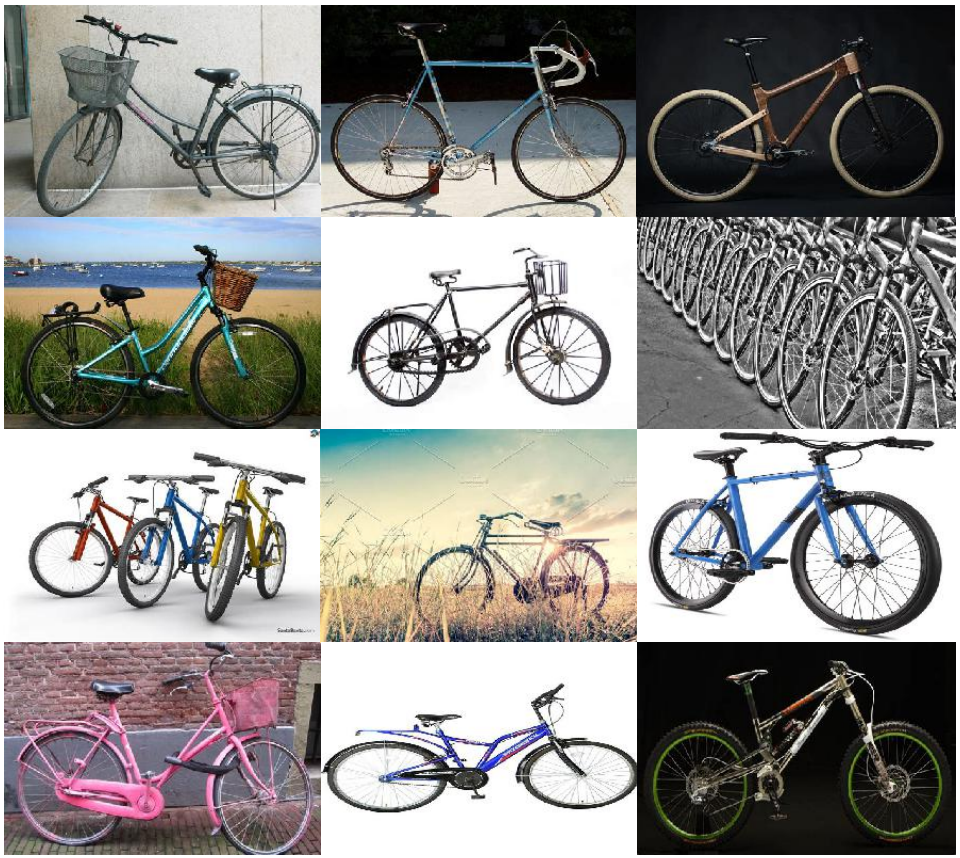
$$y[0] = w[0] + \sum s[m]h[-m] = w[0] + \frac{\sum s^2\,[m]}{\sqrt{\sum s^2\,[m]}} = w[0] + \sqrt{\sum s^2\,[m]}$$

The solution: matched filters

The solution: $h[m] = s[-m]/\sqrt{\sum s^2\,[m]}$.

1. Compute y[n]=h[n]*x[n]
    1. If y[0] > 0.5 $\sqrt{\sum s^2\,[m]}$, then conclude that H1 is true (signal present).
    2. If y[0] < 0.5 $\sqrt{\sum s^2\,[m]}$, then conclude that H0 is true (signal absent).

Example: beetles versus bicycles



Designing a matched filter by averaging all of the input data

- Given: 12 example images of beetles, $x_d[n_1, n_2, c]$, for $0 \leq d \leq 11$.
- Goal: design a matched filter $h[n_1, n_2, c]$ that will maximize

$$y_d[0,0,c] = \sum_{m_1} \sum_{m_2} x_d[m_1, m_2, c]\, h[-m_1, -m_2, c]$$

…on average for $0 \leq d \leq 11$, subject to $\sum_{m_1} \sum_{m_2} h^2[m_1, m_2, c] = 1$.
Solution:

$$h[m_1, m_2, c] \propto \frac{1}{12} \sum_{d} x_d[-m_1, -m_2, c]$$

Designing a matched filter by averaging all of the input data

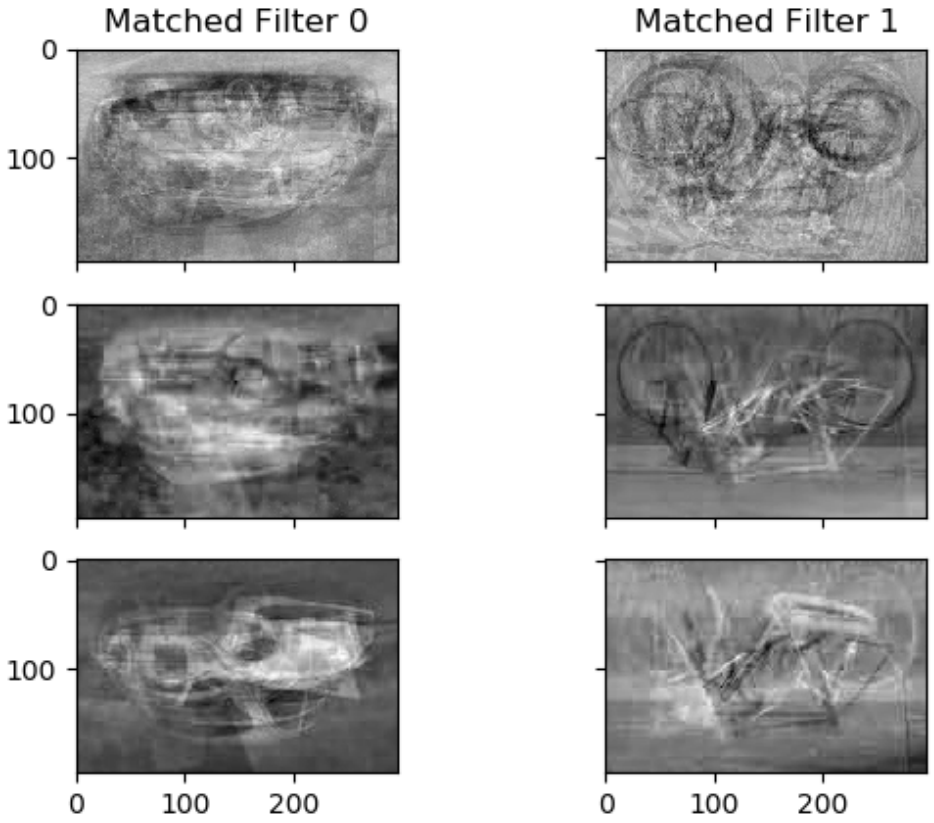
Solution:

$$h[m_1, m_2, c] \propto \frac{1}{12} \sum_d x_d[-m_1, -m_2, c]$$

- Flip each image left-to-right ($-m_2$)
- Flip each image top-to-bottom ($-m_1$)
- Take the average, across all of the training images
- To make the output of this filtering more interesting: throw away the first two rows, last two rows, first two columns, and last two columns of each input image, the result will be that $h[m_1, m_2, c]$ has a size $M_1=N_1-4$, $M_2=N_2-4$, so the "valid" output will be 5x5.

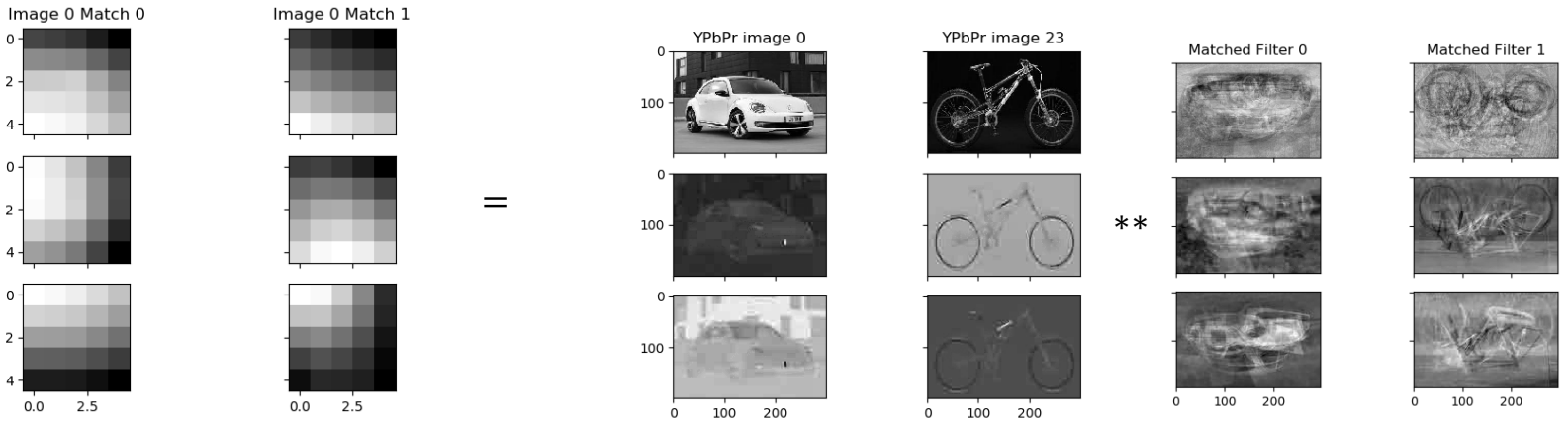
Matched filters for beetles and bicycles

- Flip each image left-to-right ($-m_2$)
- Flip each image top-to-bottom ($-m_1$)
- Take the average, across all of the training images.
- Shown here: three color planes, Y, Pb, and Pr.



Match = input image, convolved with matched filter

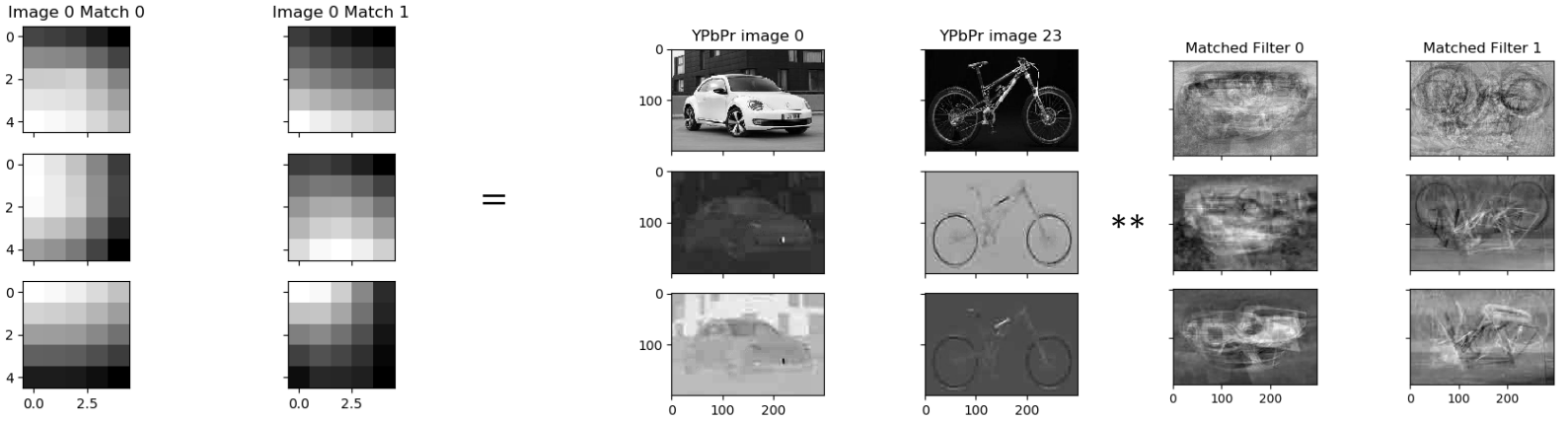
$$y_d[n_1, n_2, c] = x_d[n_1, n_2, c] ** h[n_1, n_2, c]$$



A note about ”valid” output pixels:

$$y_d[n_1, n_2, c] = \sum_{m_1} \sum_{m_2} x_d[m_1, m_2, c]\, h[n_1 - m_1, n_2 - m_2, c]$$

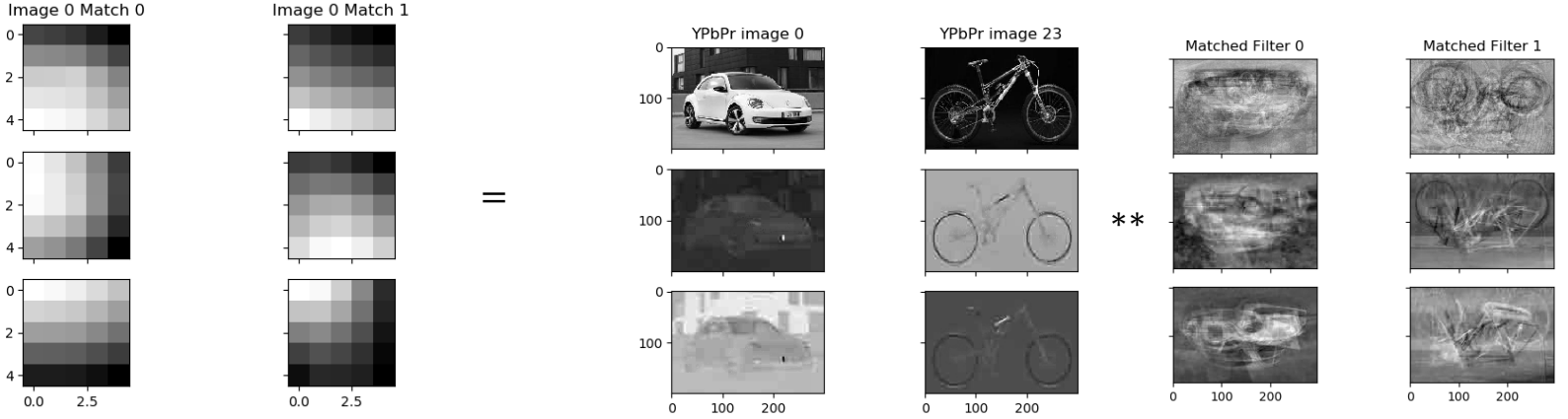
<u>Valid output pixels</u> are the values of $y_d[n_1, n_2, c]$ whose summations include only valid pixels of $x_d[m_1, m_2, c]$ and $h[n_1 - m_1, n_2 - m_2, c]$. The result has size $(N_1\text{-}M_1\text{+}1)\text{x}(N_2\text{-}M_2\text{+}1)\text{=}5\text{x}5$.



Match outputs

$$y_d[n_1, n_2, c] = \sum_{m_1} \sum_{m_2} x_d[m_1, m_2, c]\, h[n_1 - m_1, n_2 - m_2, c]$$

The best match should occur at n1=0, n2=0, which is sort of the pixel in the middle of the output. However, that middle pixel is rarely the best match. Instead, the best match often occurs a few pixels to the right, left, up, or down, implying that this particular image is shifted relative to the mean-image.



Outline: Image filtering and image features

- Images as signals
- Color spaces and color features
- 2D convolution
- Matched filters
- Gradient filters
- Separable convolution
- Accuracy spectrum of a 1-feature classifier

Computing the gradient of image pixels

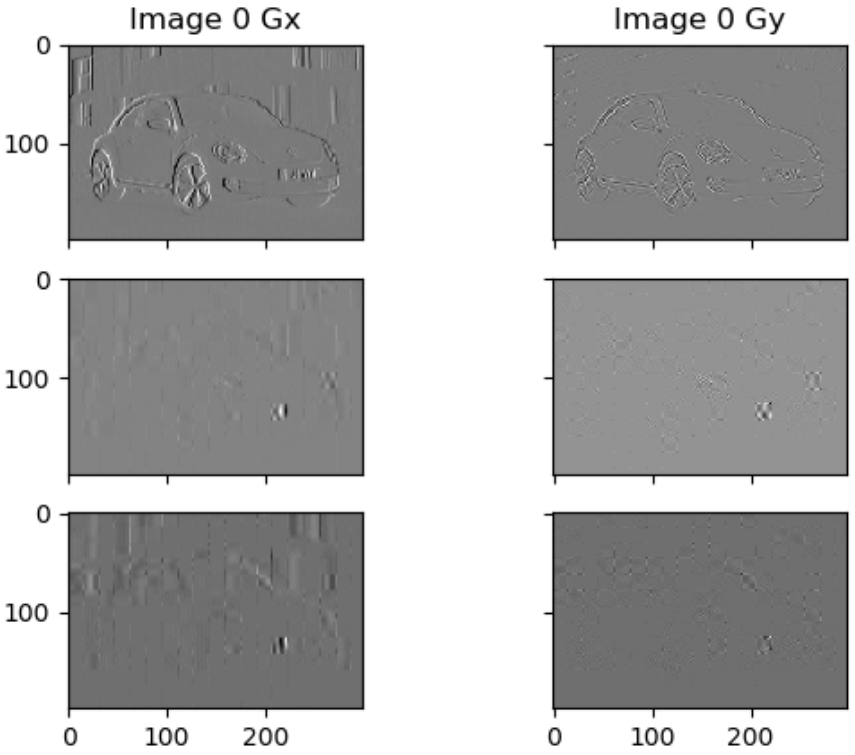
The gradient of an image turns each image plane, c, into a pair of image planes:

$$\nabla x[n_1, n_2, c] = \left[\frac{\delta}{\delta n_1} x[n_1, n_2, c], \frac{\delta}{\delta n_2} x[n_1, n_2, c]\right]$$

We usually divide the gradient into two sub-images, the horizontal gradient Gx, and the vertical gradient Gy:

$$G_x[n_1, n_2, c] = \frac{\delta}{\delta n_2} x[n_1, n_2, c]$$

$$G_y[n_1, n_2, c] = \frac{\delta}{\delta n_1} x[n_1, n_2, c]$$

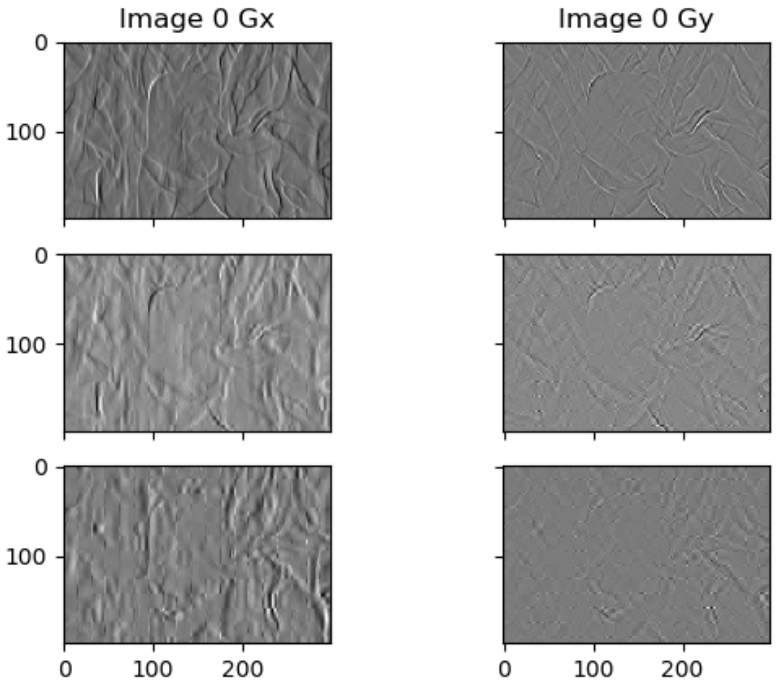


Computing the gradient of image pixels

Of course we can't really calculate the derivative of a discrete image. So we approximate it using filters

$$G_x[n_1, n_2, c] = x[n_1, n_2, c] ** h_x[n_1, n_2]$$
$$\approx \frac{\delta}{\delta n_2} x[n_1, n_2, c]$$

$$G_y[n_1, n_2, c] = x[n_1, n_2, c] ** h_y[n_1, n_2]$$
$$\approx \frac{\delta}{\delta n_1} x[n_1, n_2, c]$$

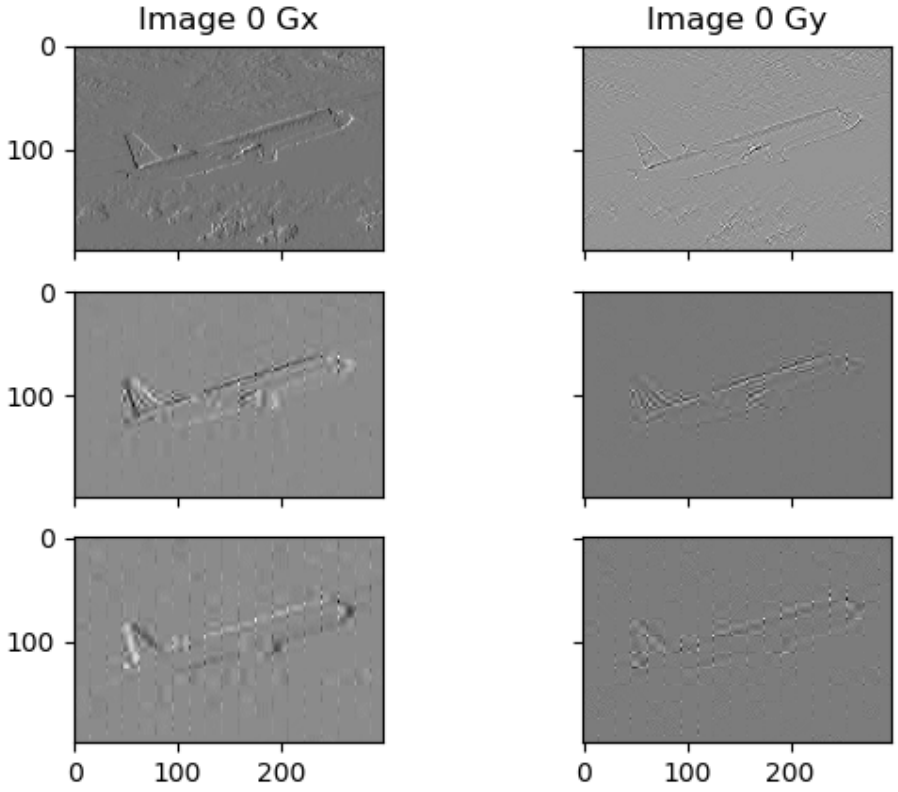


The Sobel mask

The Sobel mask is a particularly simple approximation to the gradient – it takes the difference in one direction, then averages in the other direction:

$$h_x[n_1, n_2] = \begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix}$$

$$h_y[n_1, n_2] = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}$$



Splitting the Sobel mask into separable filters

The Sobel mask is very popular, in part, because each of the 2D filters can be separated into a row-filter, followed by a column-filter:

$$h_x[n_1, n_2] = \begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix} = \begin{bmatrix} 1 \\ 2 \\ 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & -1 \end{bmatrix}$$

$$h_y[n_1, n_2] = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ -1 \end{bmatrix} \begin{bmatrix} 1 & 2 & 1 \end{bmatrix}$$

Outline: Image filtering and image features

- Images as signals
- Color spaces and color features
- 2D convolution
- Matched filters
- Gradient filters
- Separable convolution
- Accuracy spectrum of a 1-feature classifier

Separable filters

A “separable filter” is one that can be written as the product of a row-filter, times a column-filter:

$$h[n_1, n_2] = h_1[n_1]h_2[n_2]$$

If the filter can be separated, then the convolution can also be separated:

$$\sum_{m_1}\sum_{m_2} x_d[m_1, m_2]\, h[n_1 - m_1, n_2 - m_2]$$

$$= \sum_{m_2}\left(\sum_{m_1} x_d[m_1, m_2]\, h_1[n_1 - m_1]\right) h_2[n_2 - m_2]$$

Separable filters

This operation requires a double-summation, which has a computational complexity equal to (# rows)X(# columns):

$$y[n_1, n_2] = \sum_{m_1} \sum_{m_2} x_d[m_1, m_2]\, h[n_1 - m_1, n_2 - m_2]$$

This operation requires a single summation, which has a computational complexity equal to (# rows):

$$v[n_1, m_2] = \sum_{m_1} x_d[m_1, m_2]\, h_1[n_1 - m_1]$$

Separable filters

This operation requires a double-summation, which has a computational complexity equal to (# rows)X(# columns):

$$y[n_1, n_2] = \sum_{m_1} \sum_{m_2} x_d[m_1, m_2]\, h[n_1 - m_1, n_2 - m_2]$$

This operation requires a single summation, which has a computational complexity equal to (# rows):

$$v[n_1, m_2] = \sum_{m_1} x_d[m_1, m_2]\, h_1[n_1 - m_1]$$

This operation requires a single summation, which has a computational complexity equal to (# columns):

$$y[n_1, n_2] = \sum_{m_2} v[n_1, m_2]\, h_2[n_2 - m_2]$$

Separable filters

This operation requires a double-summation, which has a computational complexity equal to (# rows)X(# columns):

$$y[n_1, n_2] = \sum_{m_1} \sum_{m_2} x_d[m_1, m_2]\, h[n_1 - m_1, n_2 - m_2]$$

This operation requires two single summations, with a computational complexity equal to (# rows) + (# columns):

$$y[n_1, n_2] = \sum_{m_2} \left(\sum_{m_1} x_d[m_1, m_2]\, h_1[n_1 - m_1] \right) h_2[n_2 - m_2]$$

Usually, a computational complexity of (# rows) + (# columns) is much, much less than (# rows)X(# columns)!!!

Outline: Image filtering and image features

- Images as signals
- Color spaces and color features
- 2D convolution
- Matched filters
- Gradient filters
- Separable convolution
- Accuracy spectrum of a 1-feature classifier

What is a “1-feature classifier”?

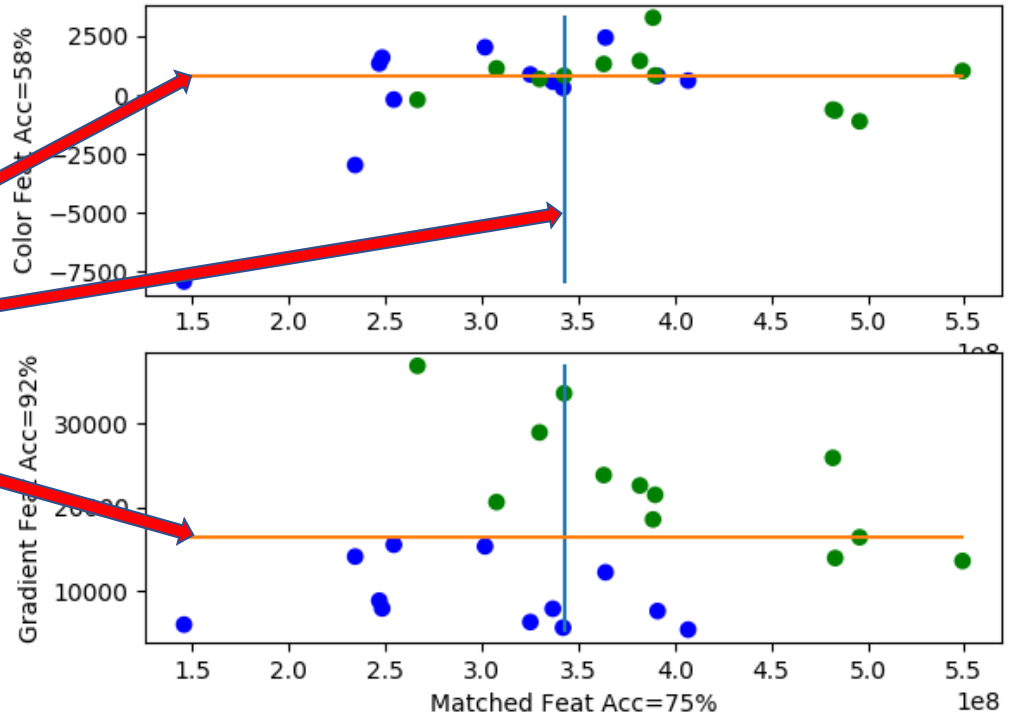
Test some feature, f[k]. Say that the test image is “class 1” if and only if f[k] >= threshold.

Threshold for the color feature

Threshold for the matched-filtering feature

Threshold for the gradient feature

Example: airplanes (blue) vs. skyscrapers (green)

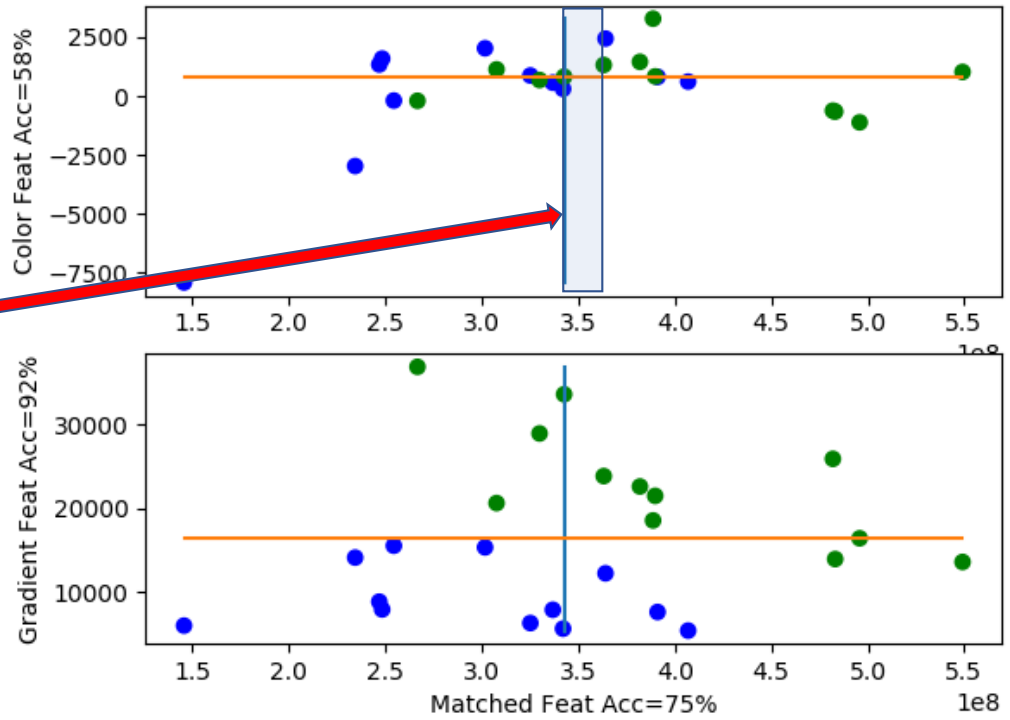


What are the possible thresholds?

Notice that the only thresholds that are worth testing are the values of feature[k] that are actually measured values, for at least one datum!

Varying the threshold from one datum to the next makes no change, at all, in the accuracy, so it's not useful to test those thresholds.

Example: airplanes (blue) vs. skyscrapers (green)

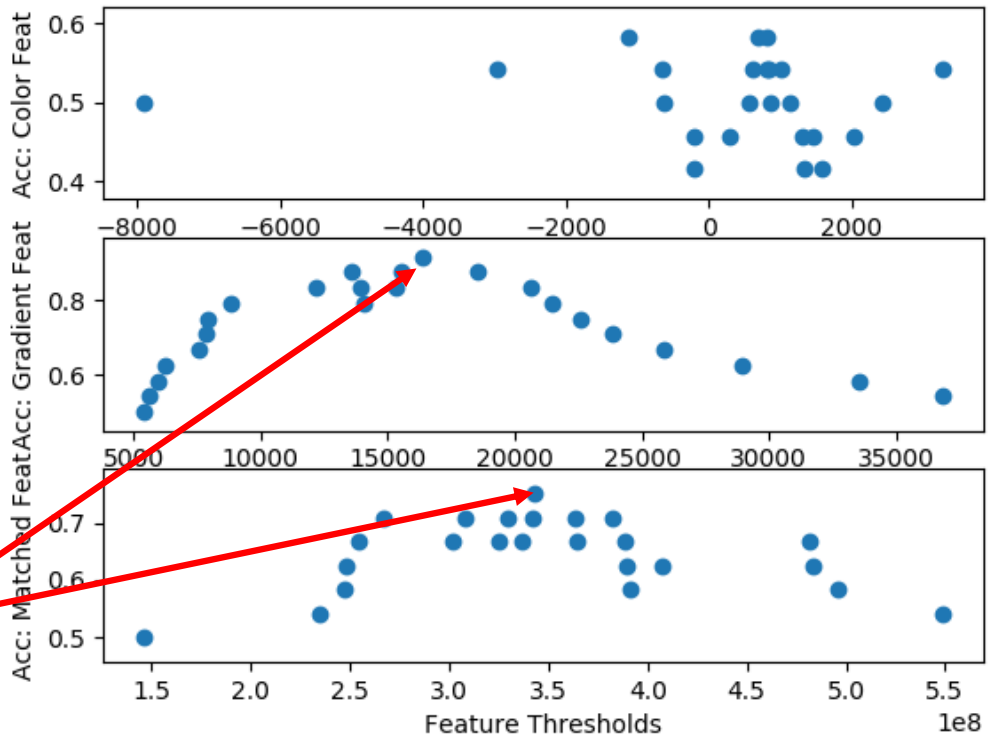


Accuracy spectrum

The “accuracy spectrum” for a particular feature is the list of all possible accuracies that could be achieved by any one-feature classifier.

- Test, as possible threshold, every value of feature[k] observed for any training image.
- List the resulting classifier accuracies.
- For each feature, find the best threshold.

Example: airplanes vs. skyscrapers



Negative polarity

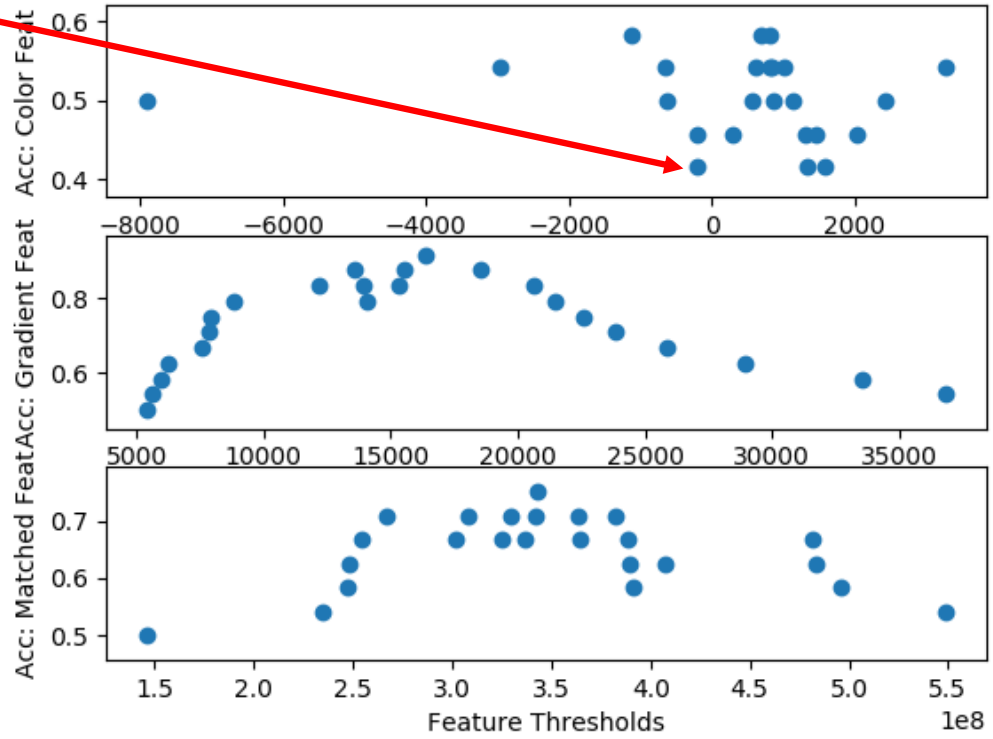
What happens if some accuracies are below 50%?

That just means that you should use, instead, a “negative polarity” classifier:

Call an image “class 0” if feature[k]>=threshold.

The accuracy of the “negative polarity” classifier is 1 minus the accuracy of the “positive polarity” classifier. So you want max(accuracy,1-accuracy), maximum over all possible thresholds.

Example: airplanes vs. skyscrapers



A few final thoughts

- Fire vs. Water: you should find that color is the best classifier
- Airplanes vs. Skyscrapers: gradient feature gets 92% accuracy! Skyscrapers have a lot of vertical edges (Gx is large), while airplanes have a lot of horizontal edges (Gy is large).
- Beetles vs. Bicycles: the matched filter does well in this case, because beetles and bicycles have pretty matchable shapes.