Latent Variable Methods Course

Learning from data

Instructor: Kevin Dunn
kevin.dunn@connectmv.com
http://connectmv.com

© Kevin Dunn, ConnectMV, Inc. 2011

Revision: 268:adfd compiled on 15-12-2011

Copyright, sharing, and attribution notice

This work is licensed under the Creative Commons Attribution-ShareAlike 3.0 Unported License. To view a copy of this license, please visit http://creativecommons.org/licenses/by-sa/3.0/

This license allows you:

- **to share** - to copy, distribute and transmit the work
- **to adapt** - but you must distribute the new result under the same or similar license to this one
- **commercialize** - you *are allowed* to create commercial applications based on this work
- **attribution** - you must attribute the work as follows:
  - "Portions of this work are the copyright of ConnectMV", *or*
  - "This work is the copyright of ConnectMV"

We appreciate:

- if you let us know about **any errors** in the slides
- **any suggestions to improve the notes**
- telling us if you use the slides, especially commercially, so we can inform you of major updates
- emailing us to ask about different licensing terms

All of the above can be done by writing us at

`courses@connectmv.com`

If reporting errors/updates, please quote the current revision number: `268:adfd`

Image analysis aided by multivariate methods

We will look at

1. Multivariate image analysis (MIA)
2. Also show how classical image analysis tools are combined with multivariate tools, especially PLS

Credits: MACC group: particularly

- Honglu Yu
- J. Jay Liu

We will study some applications from their PhD theses today.

References

- General image analysis book – Gonzalez and Woods: "Digital Image Processing"
- Good background: Multivariate image analysis: A review with applications
- General MIA technique: Geladi and Grahn, 1997 book
- Paper: Esbensen and Geladi - Strategy of MIA

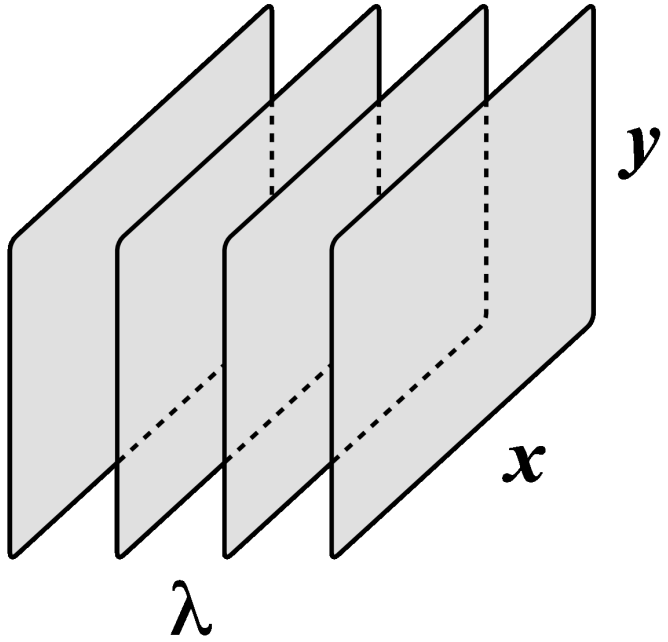
McMaster student applications:

- Lumber, pulp, steel: Manish Bharati's PhD thesis, 2002
- Snackfoods and flames: Honglu Yu's PhD thesis, 2003
- Texture, wavelets, appearance monitoring and optimization: June Liu's PhD thesis, 2004
- Medical images: Mark-John Bruwer's PhD thesis, 2006
- Wheat grading with NIR: Zheng Liu's Masters thesis, 2006
- Oats/groats detection: Emily Nichols' Masters thesis, 2011

Reasons for using image data

- Conventional sensors are expensive ($\sim$ \$ 5000 or more), but digital cameras are relatively cheap
- Image sensors work well for solids-processing systems, which generally have fewer sensors available
- Cameras take non-destructive samples
- Can be implemented on-line
- Often picks up information not available from other sensors

Conceptual storage of image data



The wavelength dimension (spectral dimension):

- 1 channel: grayscale
- 3 channels: colour image, RGB image
- multiple channels: hyperspectral image (e.g. NIR camera)

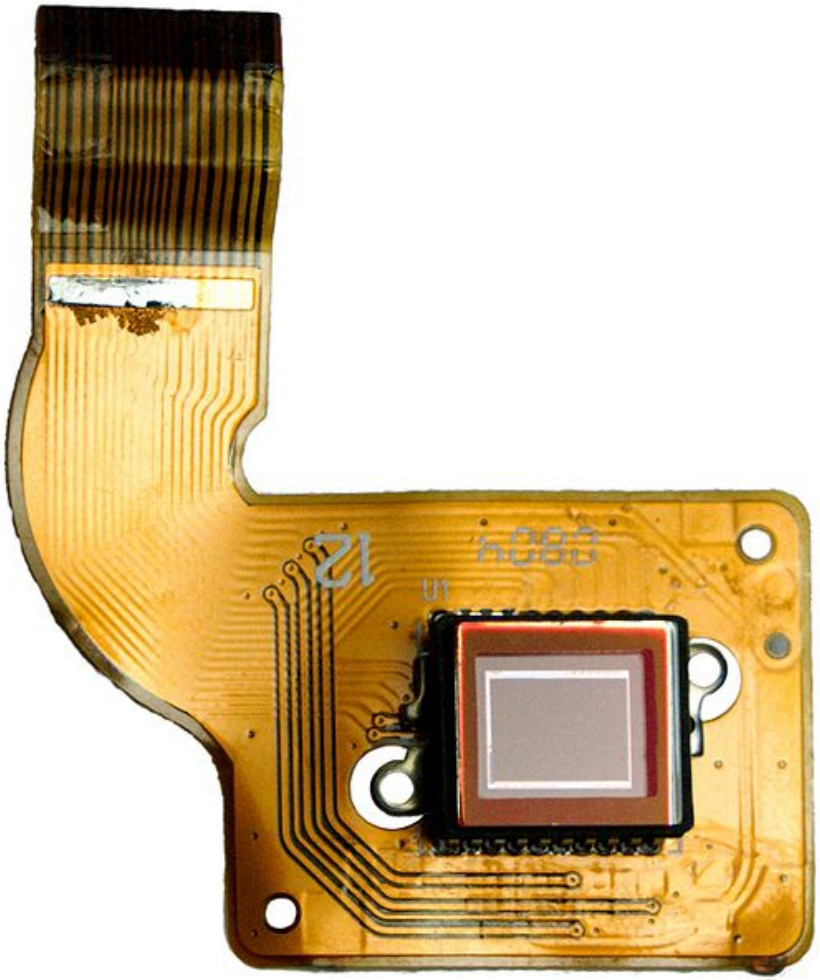
Tradeoff: *usually* we see lower spatial resolution as we acquire more channels, mainly due to cost of acquiring the additional wavelengths

What do we want from our images

We will see examples where we:

- Monitor for process problems
- Make predictions from the image data
- Optimize and improve our processes

How image data is acquired



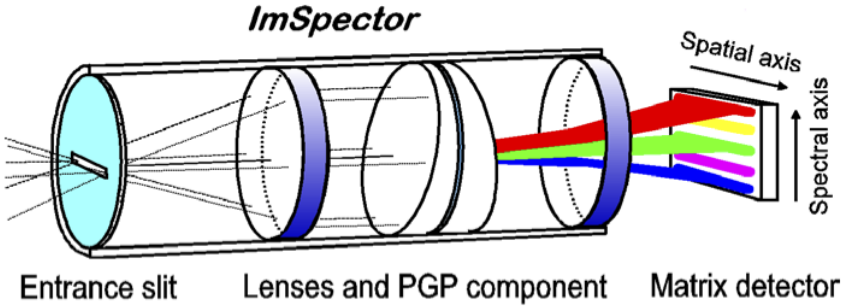
Charge-coupled device (CCD) senses the light.

Photons received → voltage → digitized and stored.

Image source: Wikipedia (CC-BY-SA)

How image data is acquired

NIR cameras (multiple channels) [1]

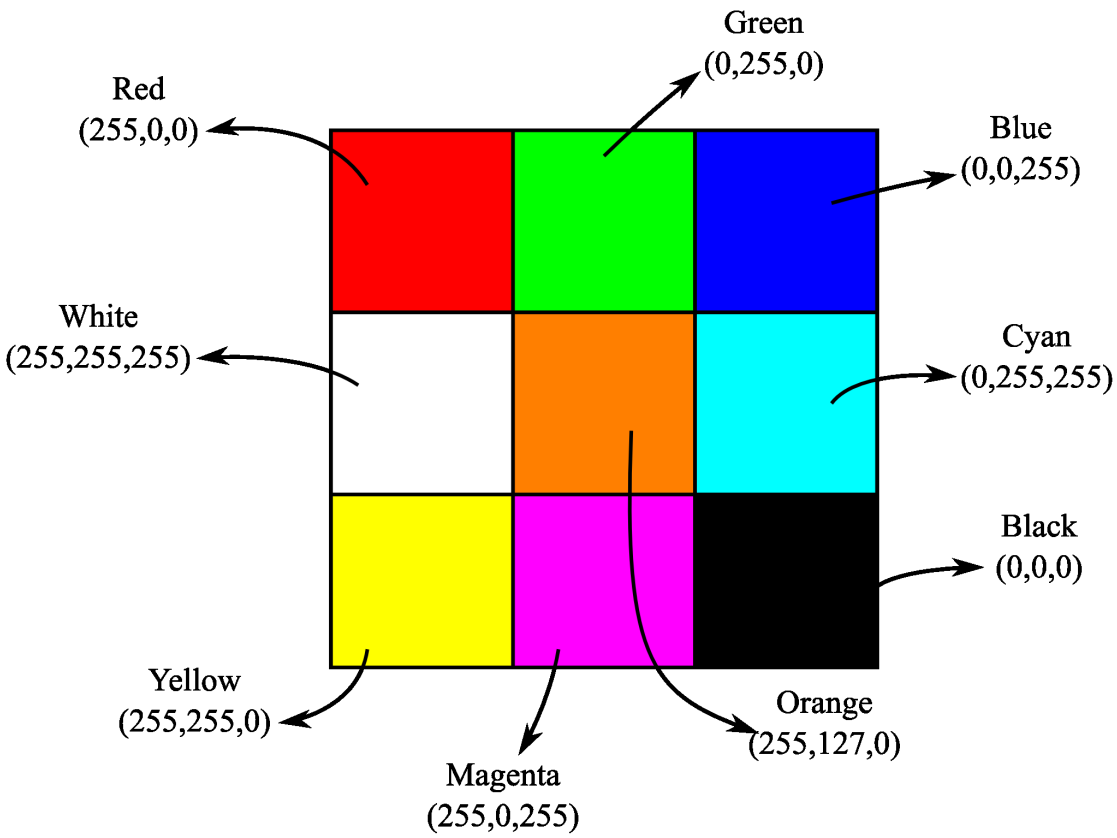


[1] From Emily Nichols' thesis (used with permission)

Actual electronic storage of image data

Pixels in the image are stored as a sequence of (usually) integer values

- RGB pixel with `(0, 0, 0)` is a black pixel
- RGB pixel with `(0, 0, 255)` is a blue pixel
- RGB pixel with `(255, 255, 255)` is a white pixel



Actual electronic storage of image data

- Integers from 0 to 255 are stored in 8 bits (one byte) per pixel
- Medical images require greater spectral resolution: often use 2 bytes: 0 to 65535 (16 bits, but use only 10 or 12 bits)

Tip

Never store your image data as JPEG $\leftarrow$ uses lossy compression; rather store as PNG (lossless compression)

Not all colour images are true colour

True colour images split the light, then use 3 CCD arrays for the R, G, and B channels.

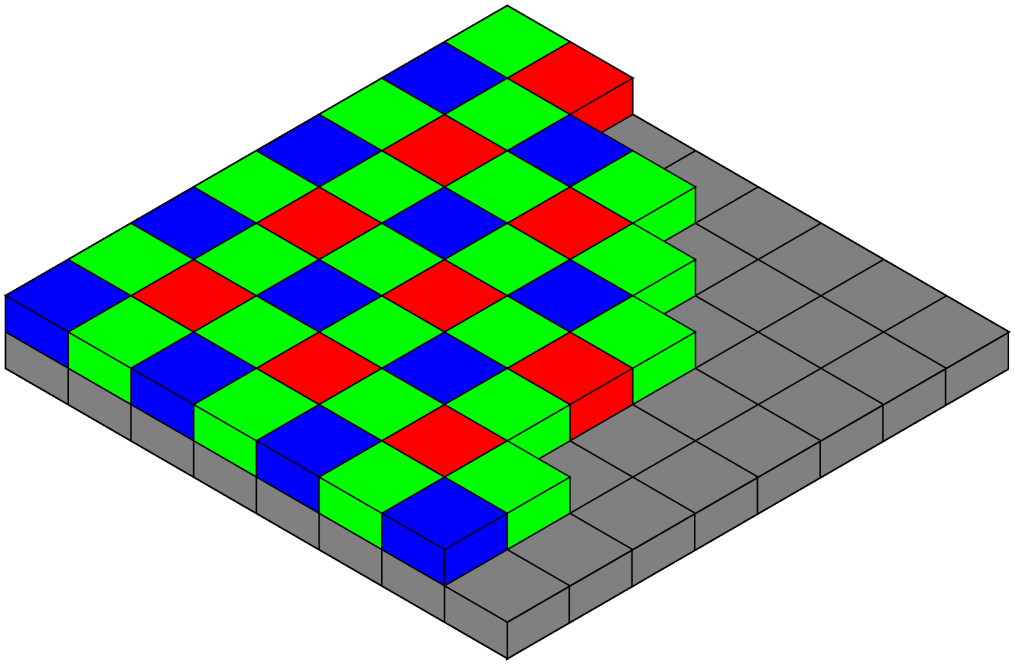


Image source: Wikipedia (CC-BY-SA)

Everyday colour images (cellphones, digital cameras) are in fact from a single CCD with a special array of colour filters: Bayer filter

The R, G, B layers we see in the digital file are algorithmically created from the single CCD matrix.

Viewing image data: grayscale

Computer monitors display RGB colour images.

Grayscale images displayed with equal weight in the 3 channels

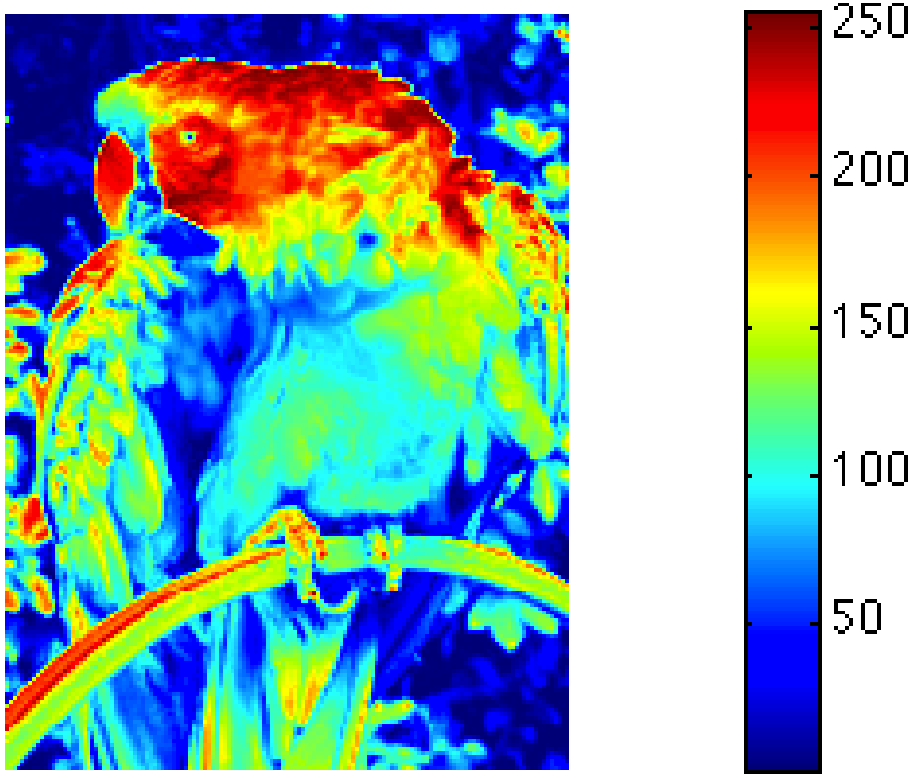
Convert RGB to grayscale:

Gray = 0.3R+0.59G+0.11B

which approximates the human eye's weighting of our light-sensitive cones.

Viewing image data: grayscale

A grayscale image has values between 0 and 255 – map each integer to a different colour: `colormap()`

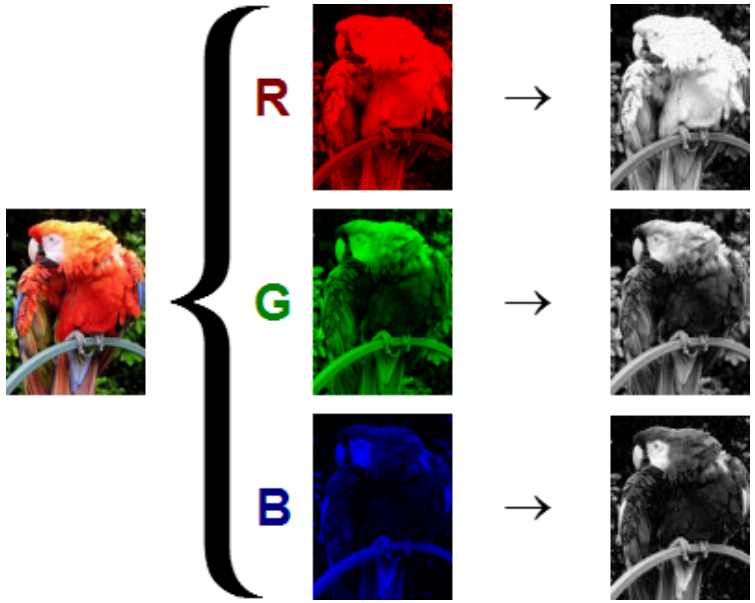


Creates a false-colour image from the grayscale image to help visualization. Many standard colour mappings are available.

Viewing image data: colour

Show each RGB channel individually, in only that channel (middle), or as grayscale (right)

- Low integer values will be dark
- High integer values will be light

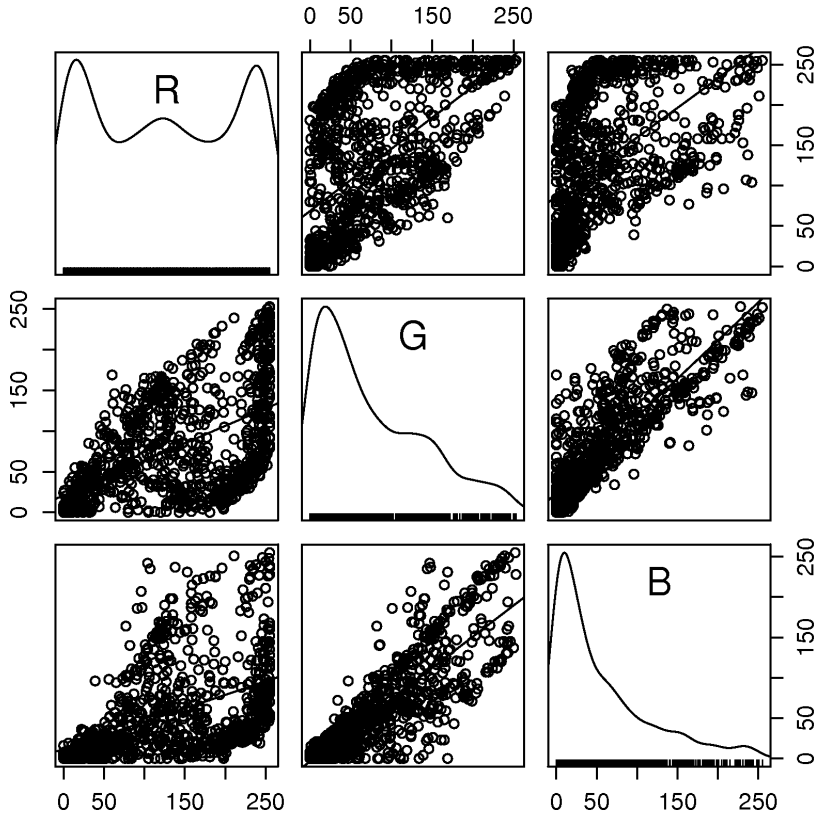


Viewing image data

Images with 4 or more wavelengths: create a *false-colour* composite image by picking any 3 channels.

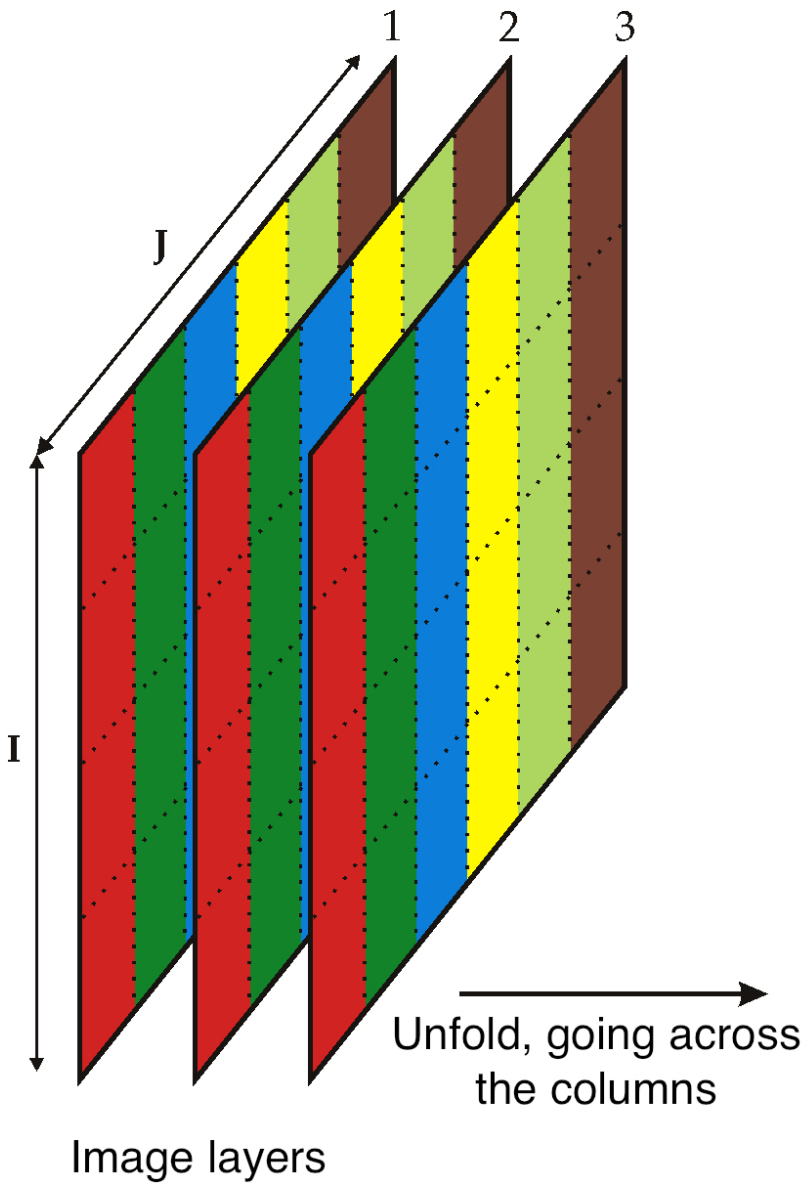
Using PCA for multivariate image analysis (MIA)

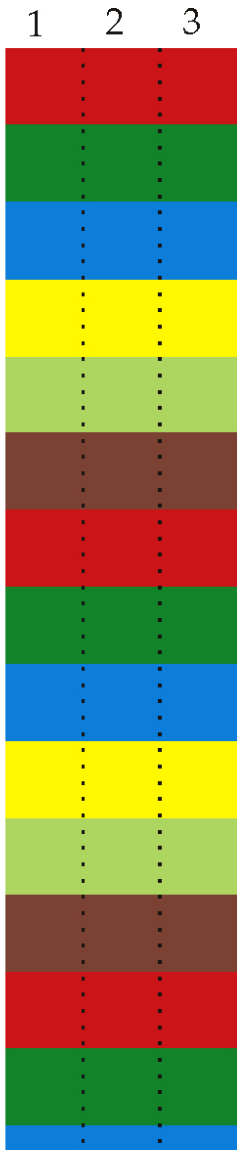
Multiple channels in the images are often strongly correlated, especially for NIR images



Strong green and blue interaction

Unfolding the image data



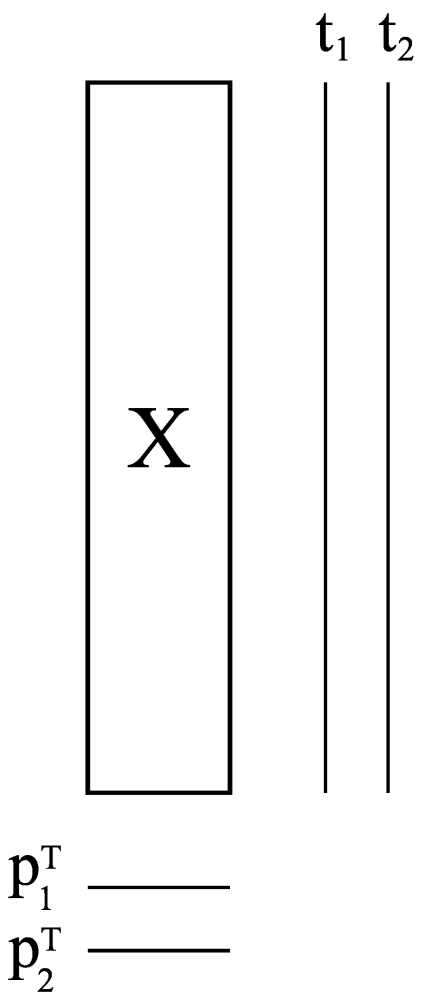


Other unfolding types possible, but this way makes sense to understand the correlation between the channels.

Sometimes called "*matricizing*" the image.

Calculating the PCA model

Once $\mathbf{X}$ is unfolded, we have a long, thin matrix.



Use the kernel PCA method on unfolded image data $\mathbf{X}$:

$$\begin{aligned} \mathbf{P} &= \text{eigenvectors of } \mathbf{X}^T\mathbf{X} \\ \mathbf{T} &= \mathbf{X}\mathbf{P} \end{aligned}$$

We may choose to use all $A = K$ eigenvectors in $\mathbf{P}$, or when K is large – e.g. NIR images – we only retain the first few ($A < K$):

$$\begin{aligned} \mathbf{X} &= \mathbf{T}\mathbf{P}^T + \mathbf{E} \\ \mathbf{E}_a &= \mathbf{X}_{a-1} - \mathbf{t}_a\mathbf{p}_a^T \\ \mathbf{X}_{a-1} &= \text{original image data} \end{aligned}$$

We don't usually preprocess the image (there isn't much difference if you do, but it is expensive).

Using the PCA model

We can investigate the model in the usual way:

- loadings
- binary score plots, e.g $\mathbf{t}_1$ vs $\mathbf{t}_2$: not practical
- SPE and Hotelling's T^2 (not practical)

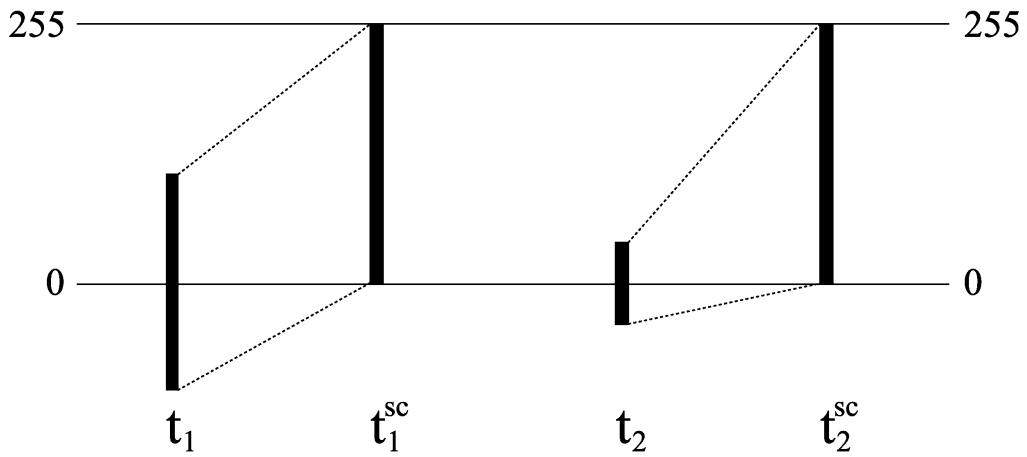
What do we want from the scores and SPE anyway?

Refolding the scores and residuals

Notice that each score $\mathbf{t}_a$ and each residuals in SPE have the same shape as the original image. It is natural to display these as grayscale images.

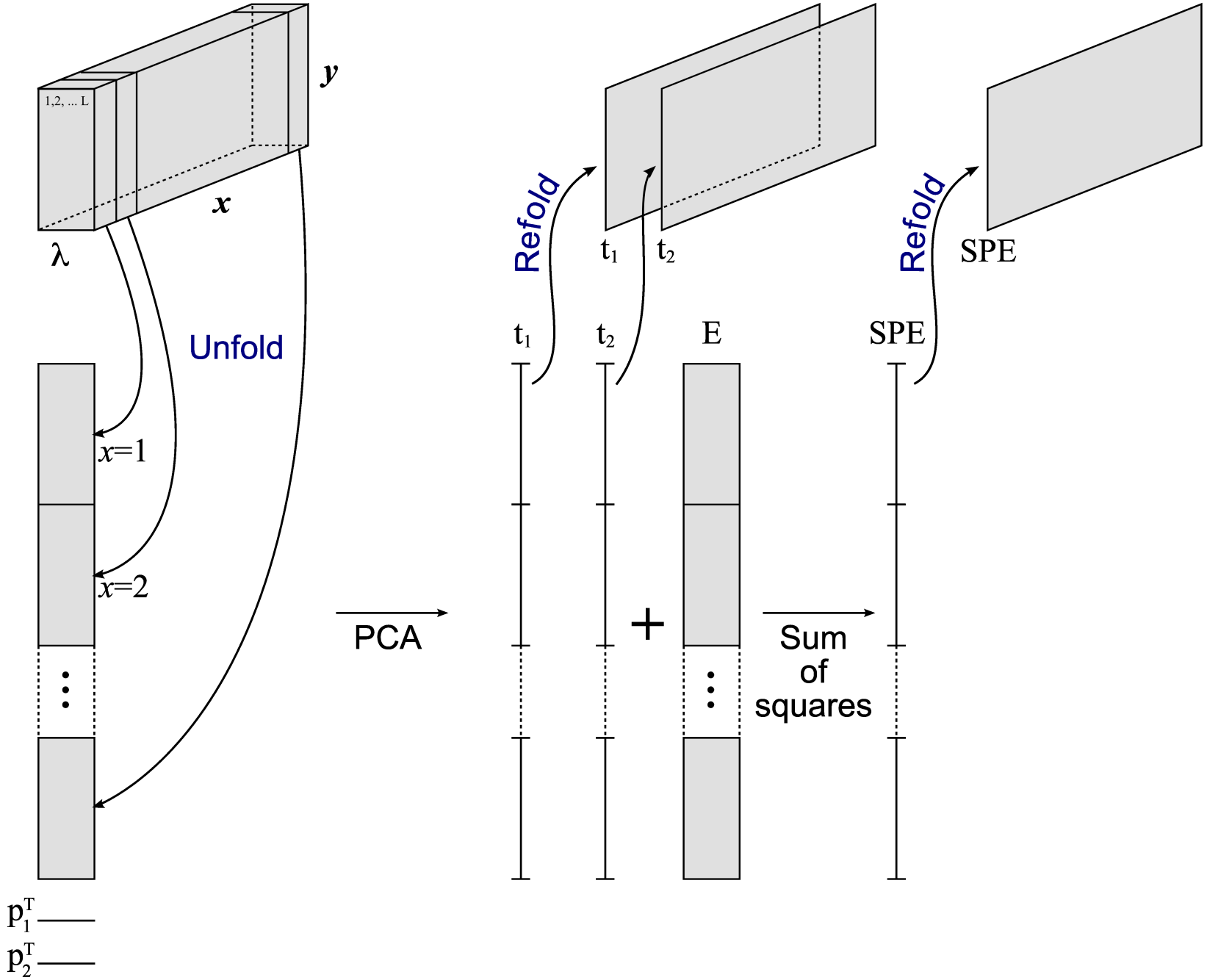
- grayscale displays require values between 0 and 255

The calculated $\mathbf{t}_a$ values have mean of zero. To display $\mathbf{t}_a^{\text{sc}}$ in the software:



$$\mathbf{t}_a^{\text{sc}} = \left\lfloor 255 \cdot \frac{\mathbf{t}_a - t_a^{\min}}{t_a^{\max} - t_a^{\min}} \right\rfloor$$

Rather display $\mathbf{t}_a^{\text{sc}}$ with colour coding.


1,2, ... L
y
x
λ
Unfold
x=1
x=2
PCA
Refold
t1
t2
t1
t2
E
+
Sum
of
squares
Refold
SPE
SPE
p1T
p2T

Binary score plots

After scores are scaled as integers between 0 and 255:

- fewer unique combinations of $\mathbf{t}_h$ vs $\mathbf{t}_v$
- show it as a 2D histogram instead
  - h = horizontal axis
  - v = vertical axis
  - height: use a colour proportional to number of pixels at (t_h, t_v) combination

Let's take a look at an example ...

An example

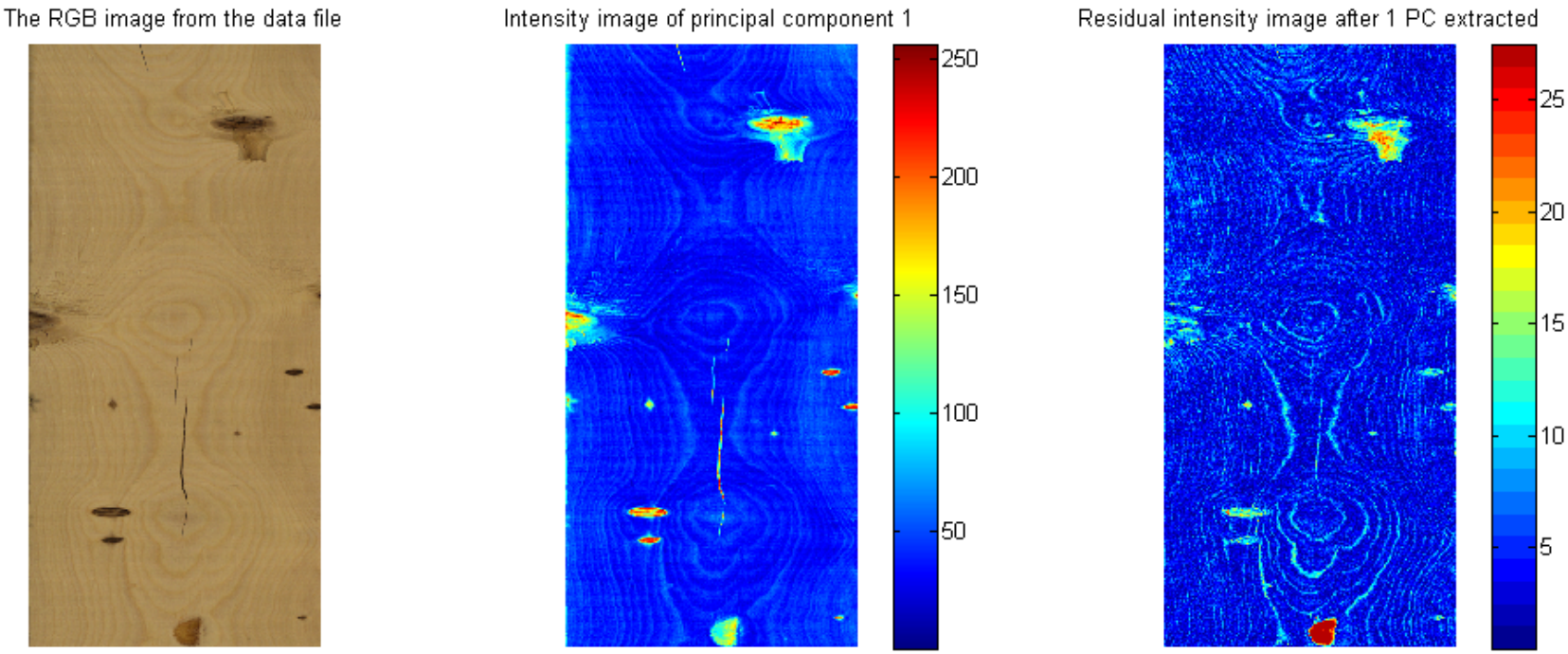
Original RGB image

Image is included with MACCMIA

You can follow along in class

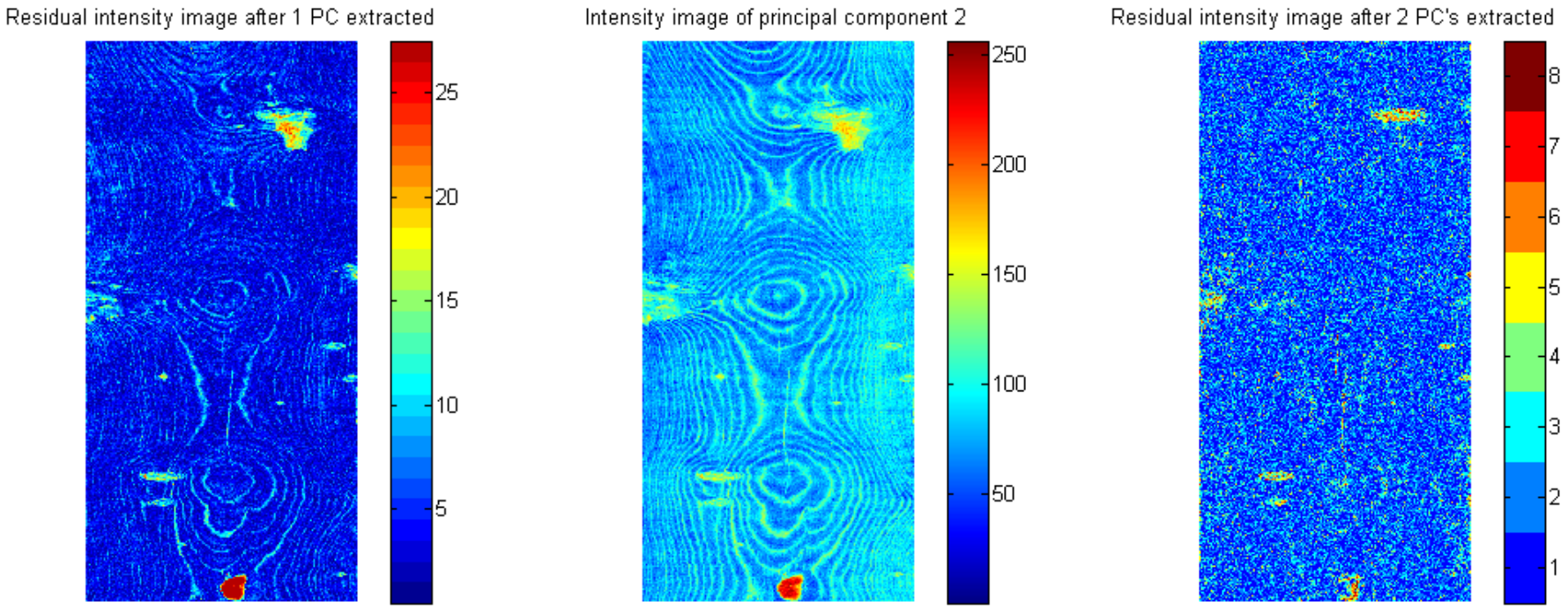
More details in Bharati, MacGregor and Tropper

The first component



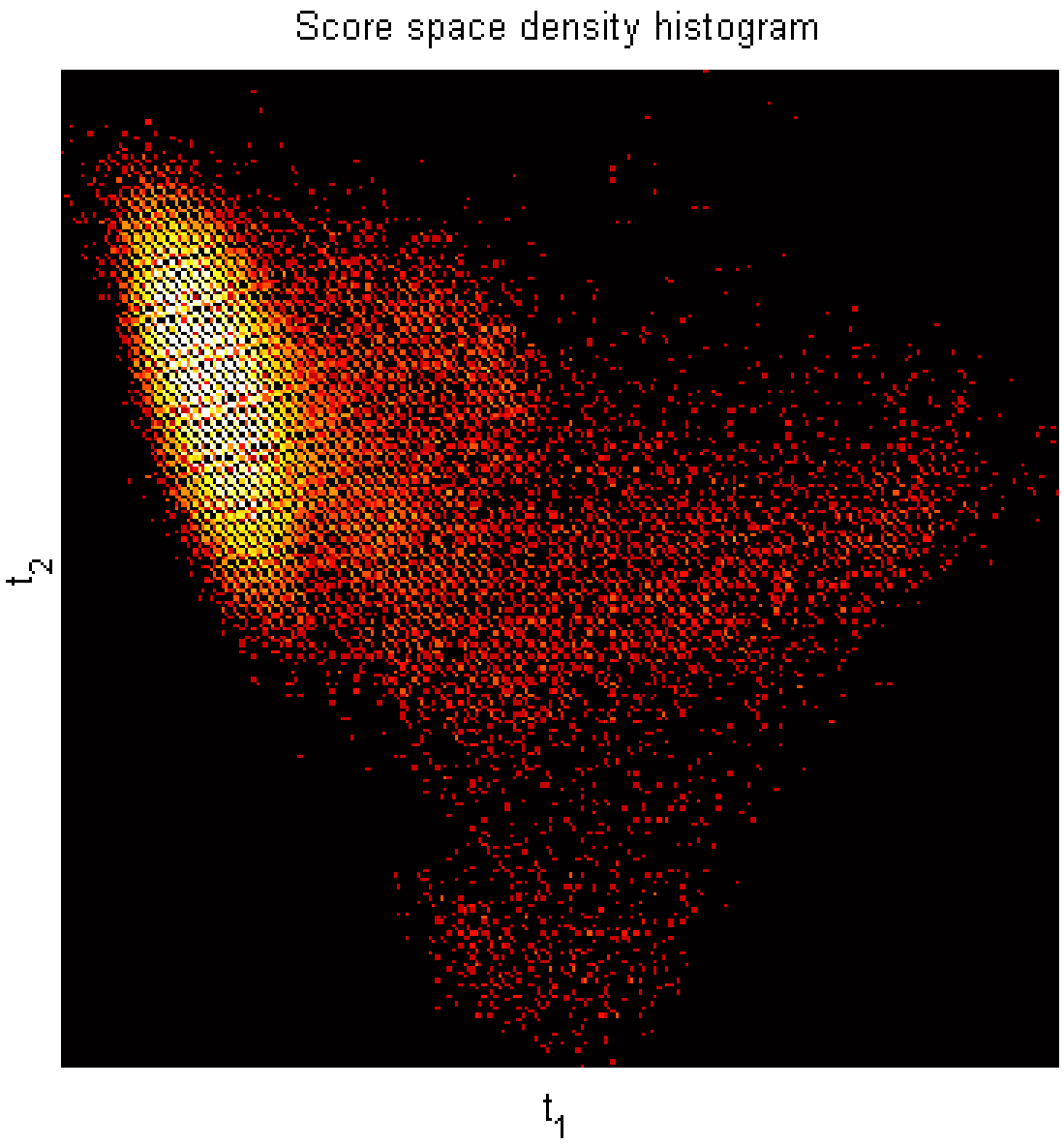
$$p_1 = [-0.72,\ -0.59,\ -0.37]$$

The second component



$$p_2 = [0.53,\ -0.13,\ -0.84]$$

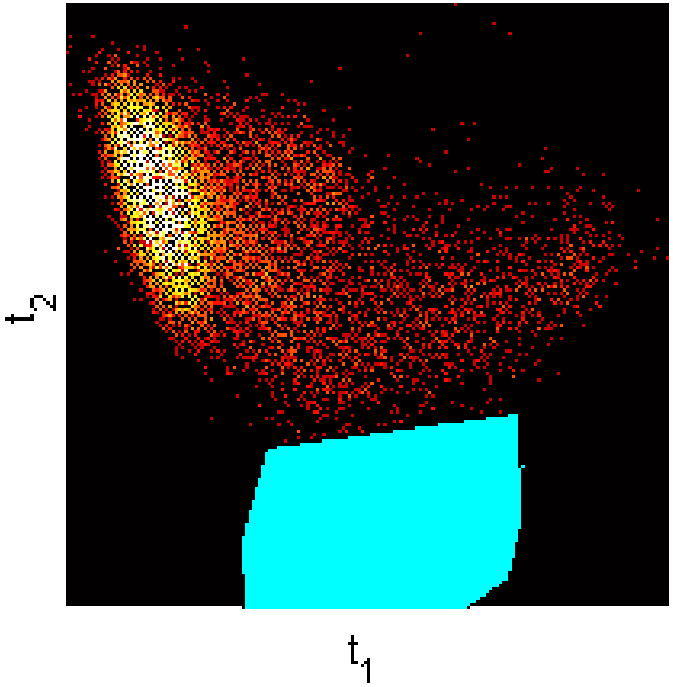
Score plot



Masking in the score space

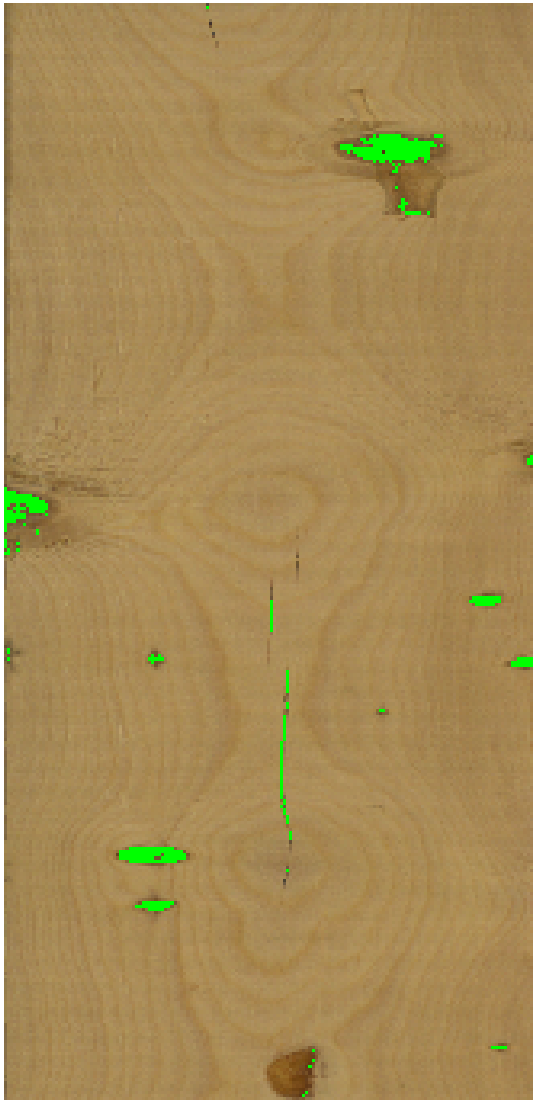
Image, with mask mapped back

Score space density histogram

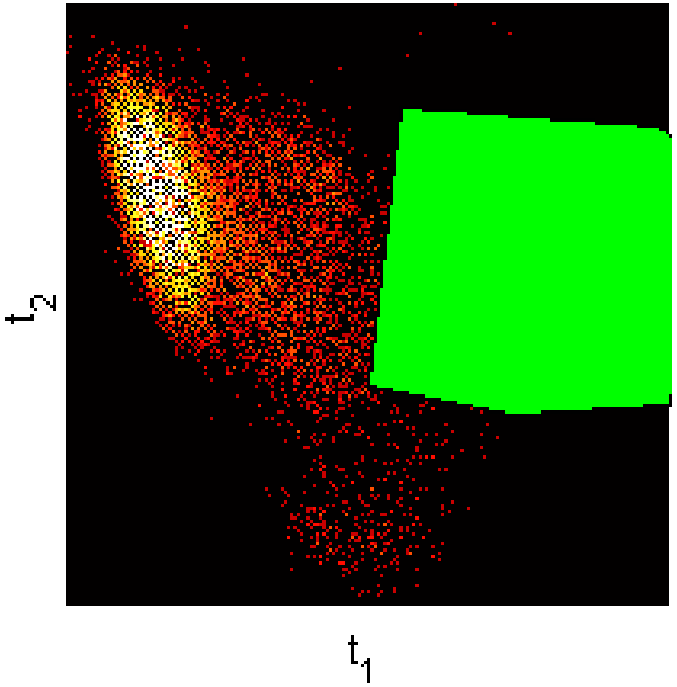


Masking in the score space

Image, with mask mapped back



Score space density histogram



Using the PCA model on a new image pixel

We have an existing PCA model; consists of

- loadings $\mathbf{P}$, an $K \times A$ matrix
- K = number of wavelengths (channels)
- $t_a^{\min}$ and $t_a^{\max}$
- one or more score-space masks (store the vertices)

A new image (of different size, even as small as a single pixel):

- Unfold the image as $\mathbf{X}_{\text{new}}$
- $\mathbf{X}_{\text{new}}$ has as many rows as there are pixels; has K columns
- $\mathbf{T}_{\text{new}} = \mathbf{X}_{\text{new}}\mathbf{P}$ = scores (has A columns)
- For each column in $\mathbf{T}_{\text{new}}$:
  - $\mathbf{t}_{a,\text{new}}^{\text{sc}} = \left\lfloor 255 \cdot \frac{\mathbf{t}_{a,\text{new}} - t_a^{\min}}{t_a^{\max} - t_a^{\min}} \right\rfloor$
- For each row in $\mathbf{T}_{a,\text{new}}^{\text{sc}}$ check whether the new score values lie under a mask

Other hints and techniques

We have a problem if image space objects are spread across more than 2 different scores:

- Resolved using automatic masking with support vector machines

Building a model from multiple images?

1. Just paste images side-by-side to create a big, single image
2. More elegantly:

$$\left(\mathbf{X}^T\mathbf{X}\right)_{\text{composite}} = \left(\mathbf{X}^T\mathbf{X}\right)_1 + \left(\mathbf{X}^T\mathbf{X}\right)_2 + \ldots + \left(\mathbf{X}^T\mathbf{X}\right)_{\text{last image}}$$

Calculate eigenvectors from $\left(\mathbf{X}^T\mathbf{X}\right)_{\text{composite}}$

Principle of Multivariate Image Analysis

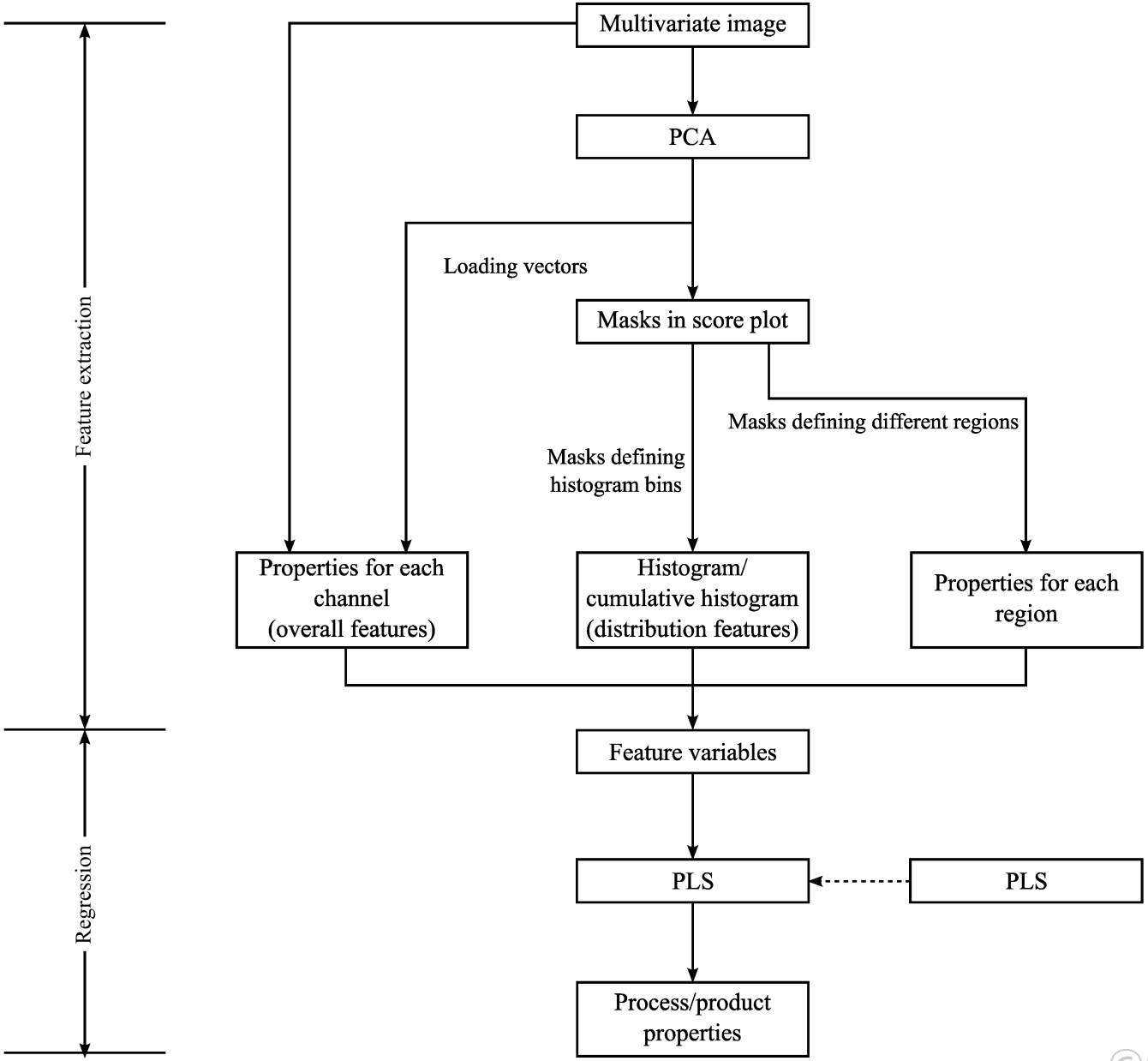
Multivariate Image Analysis (MIA)

Objects in the image with similar *spectral* signature, no matter where they are *spatially*, will appear clustered in the scores.

- There is a many-to-one mapping between the image space and score space
  - One location in the score space can appear in many spatial locations in the image space.

As a result, MIA is most useful when spatial information is not too important.

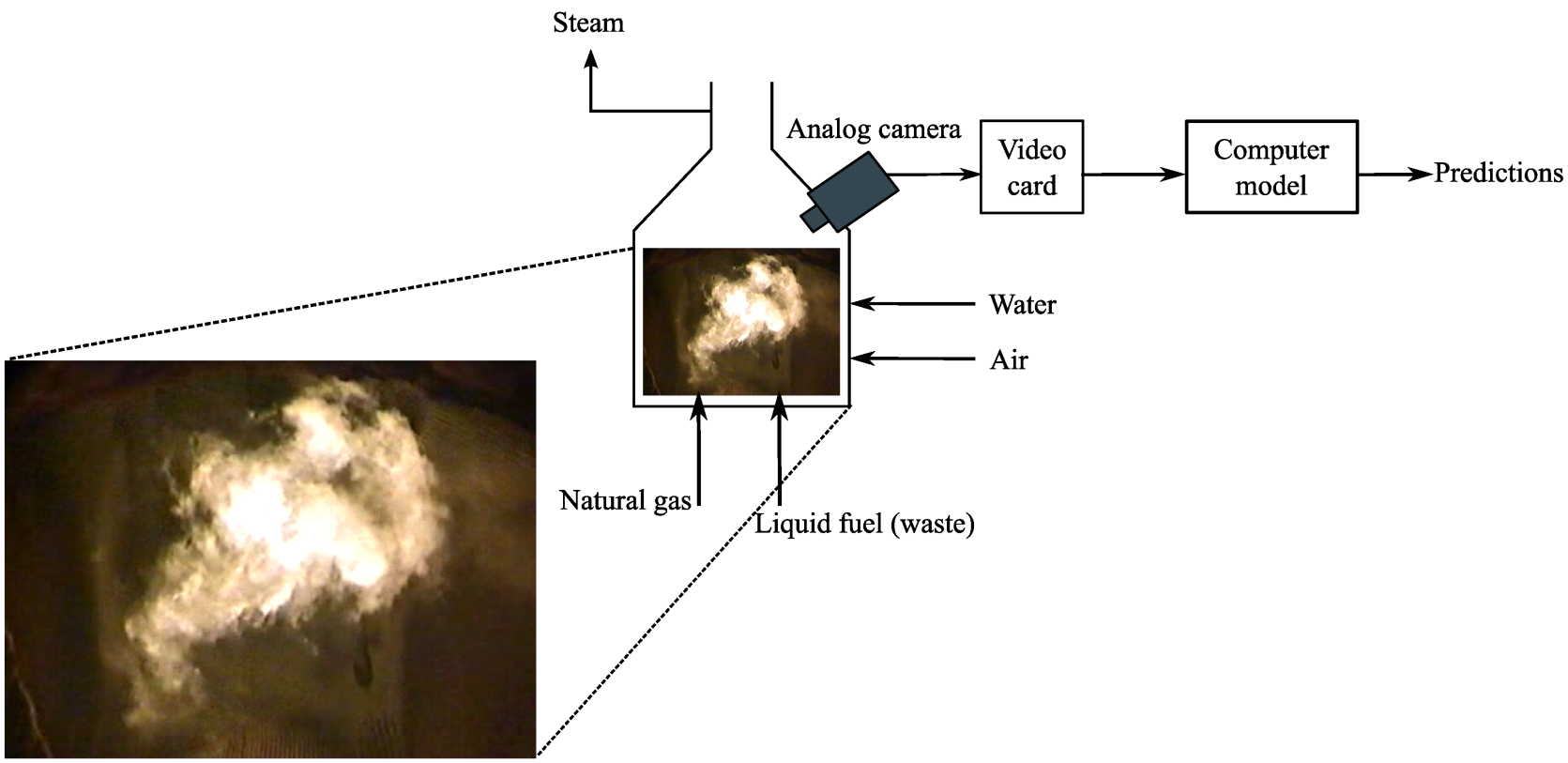
General strategy for MIA



Applications of image analysis

1. Flame monitoring and prediction (in depth)
   - Credit: thanks to Honglu Yu for permission to use her slides
   - PhD thesis
2. Snack food seasoning prediction, and monitoring (brief)
   - Credit: Honglu Yu
3. Automated judging steel sheets
   - Credit: thanks to J. Jay Liu for permission to use his slides
   - PhD thesis

Flame: system overview

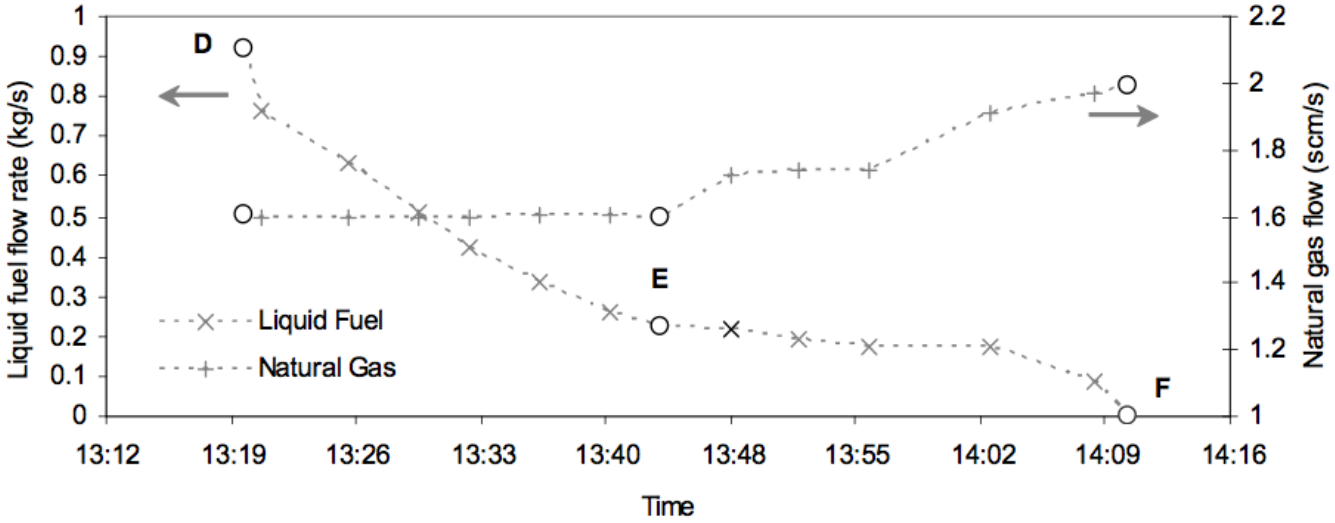


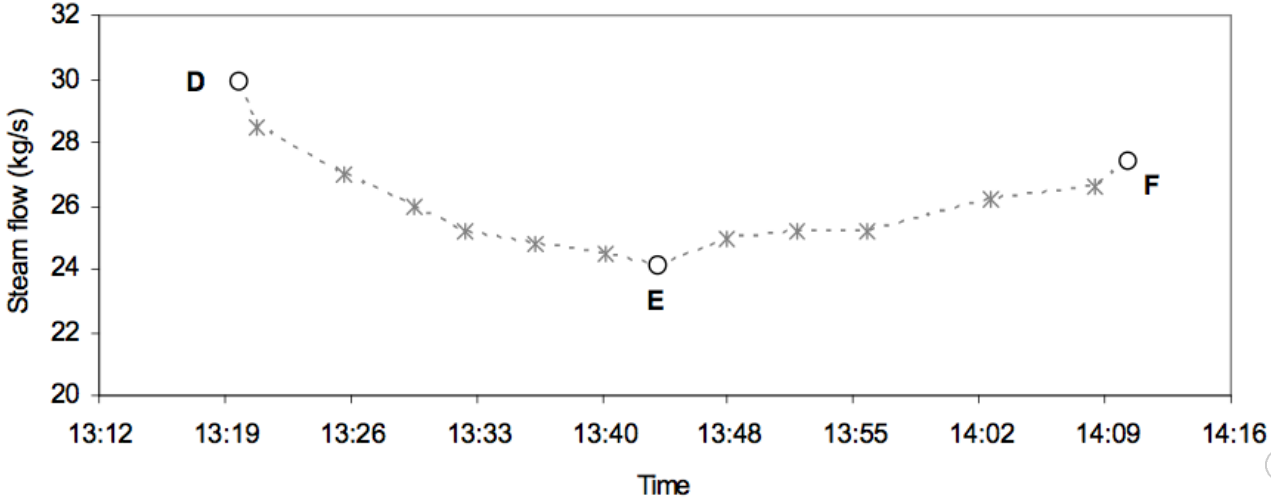
- Liquid waste stream has energy content
- Perhaps predict steam flow rate from image?

Data available

Process data: average value over $\sim$ 3 minute interval

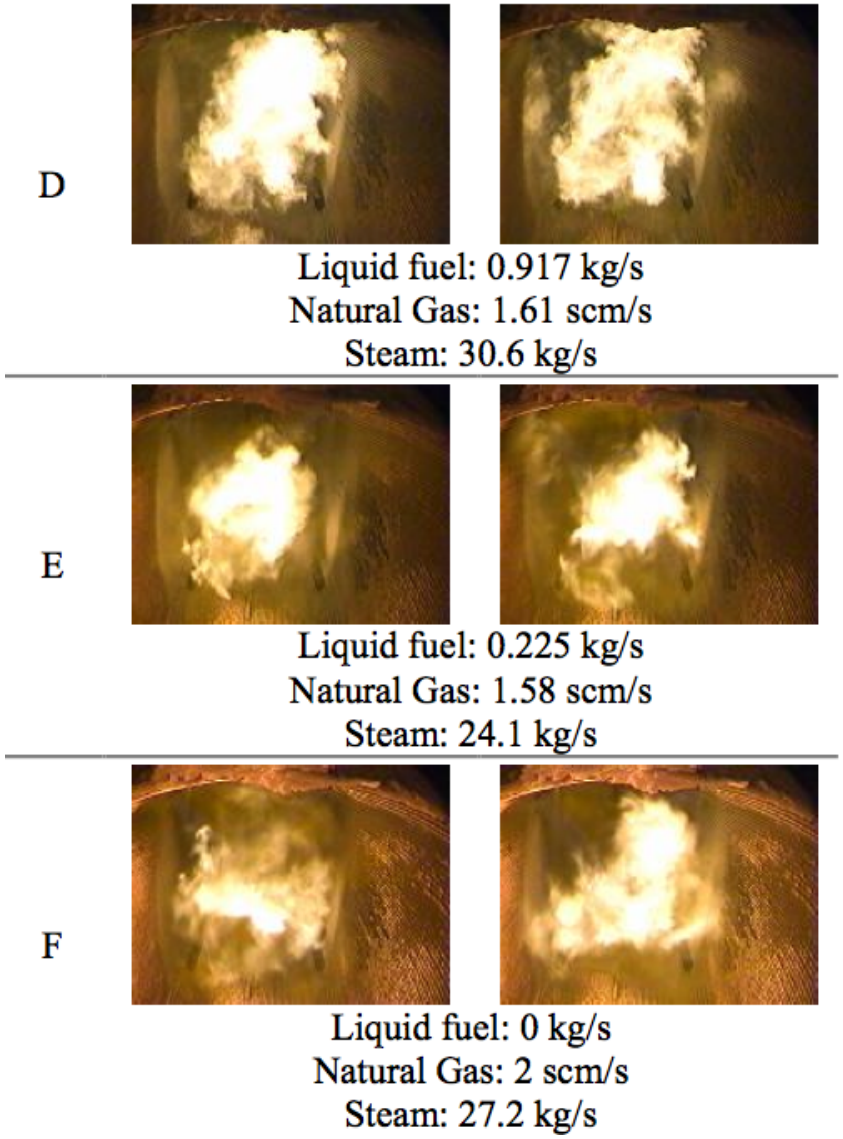
- steam flow (y variable)
- natural gas and waste stream flows





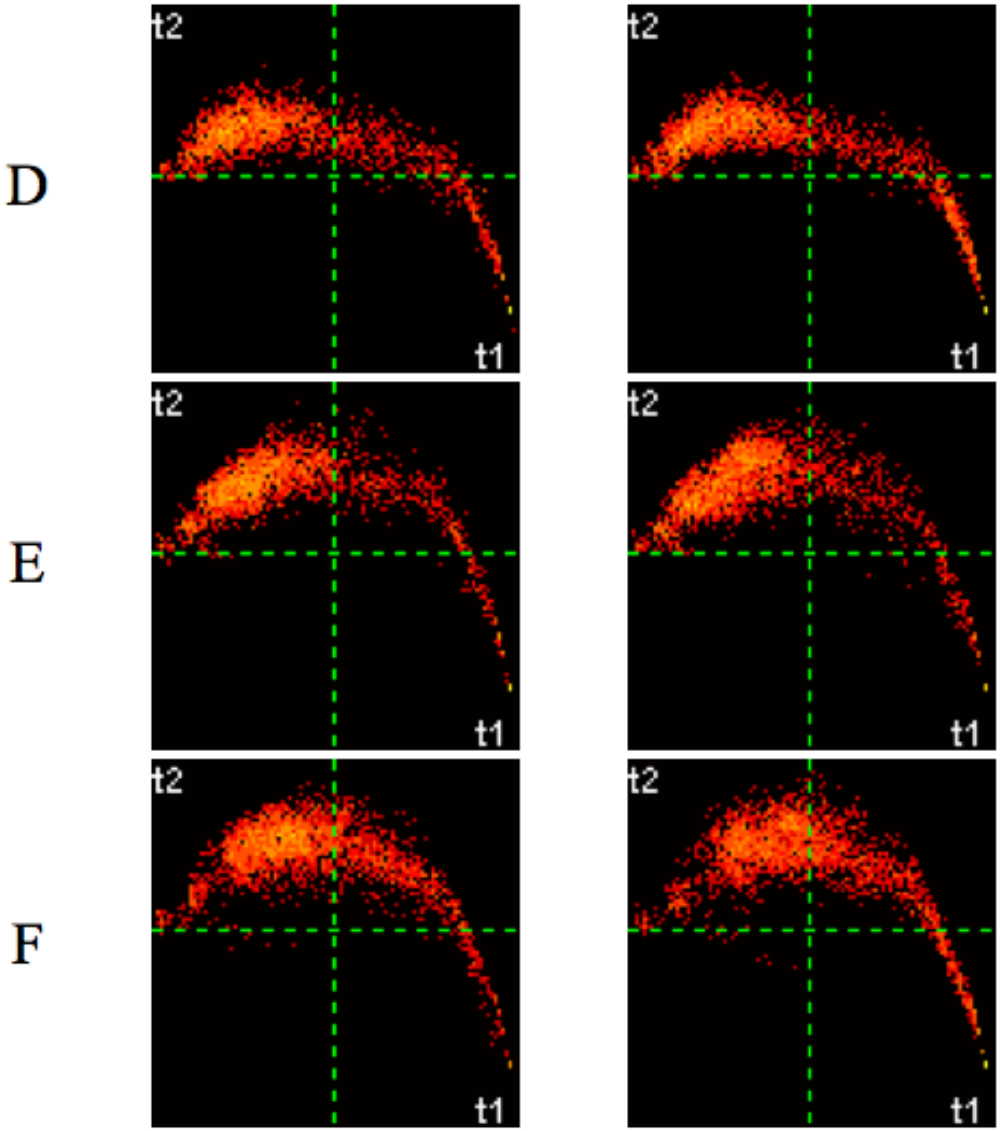
Data available

Video data: images digitized 1 second apart. Examples at D, E, F:



- Each frame is $120 \times 160 \times 3 = 56$ kilobytes

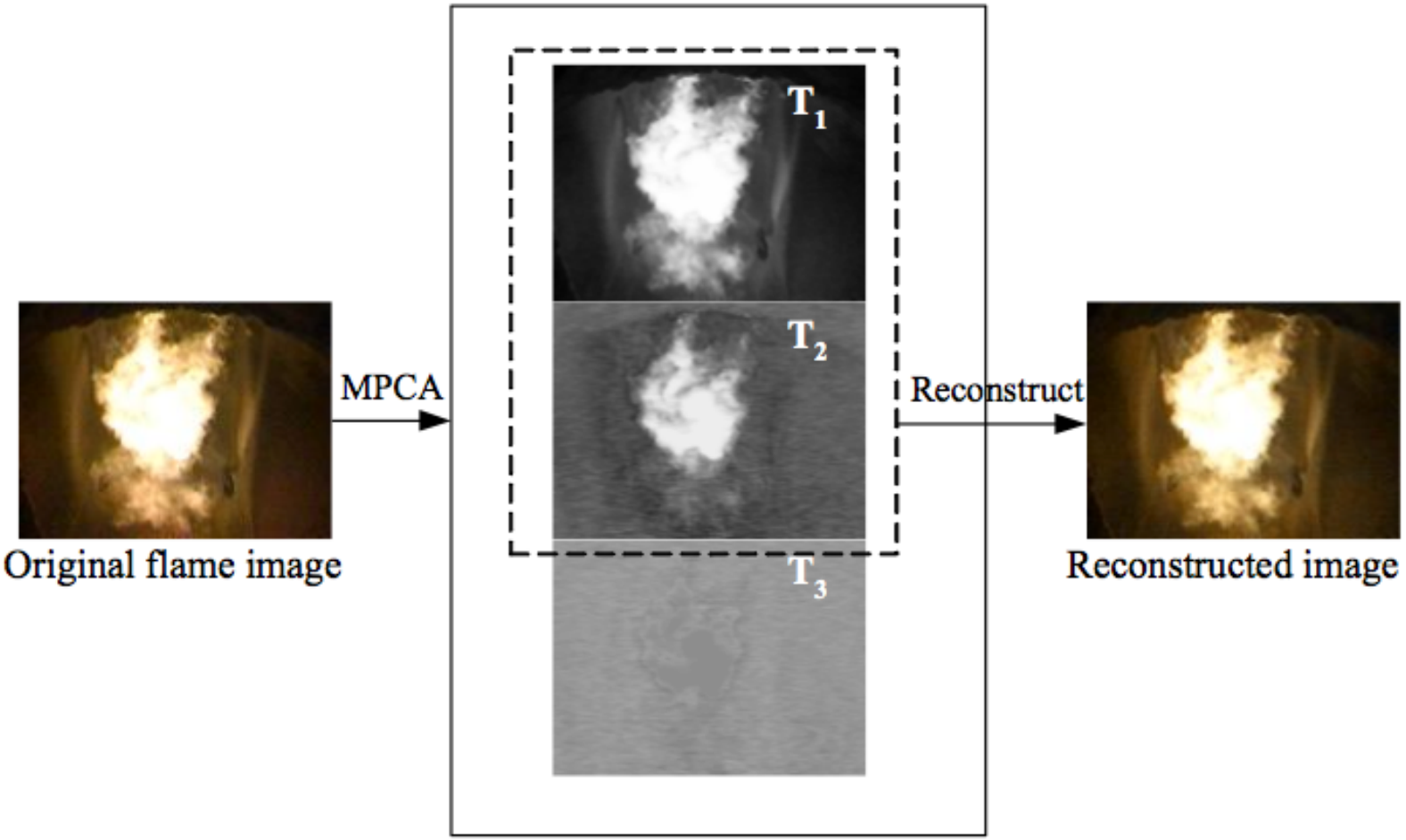
PCA on the flame images



Show video

PCA notes

- Use only 2 components: third PC mainly noise
- Reconstruct the denoised image: $\hat{\mathbf{X}} = \mathbf{t}_1\mathbf{p}_1^T + \mathbf{t}_2\mathbf{p}_2^T$



Important note

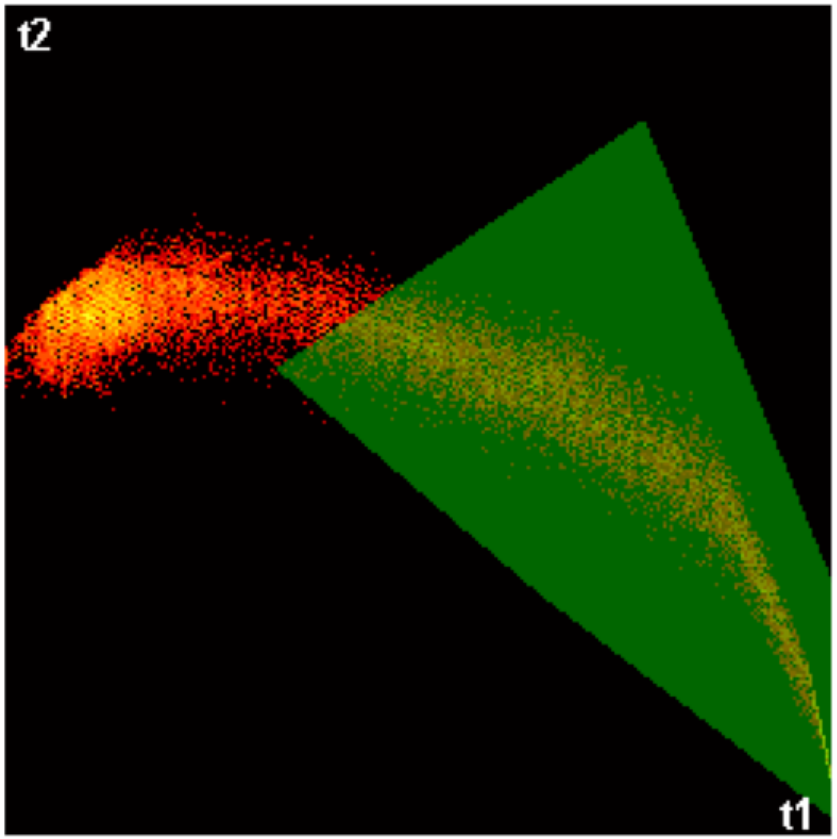
Based on the video and previous images:

- The image space *changes rapidly* from frame-to-frame
- Classical approaches to flame monitoring extract features directly from the image. For example:
  - perimeter
  - area
  - sphericity
  - luminosity
  - entropy (sum of absolute values)
  - maximum and minimum intensity
- The score space is much more stable
  - Use the score space to extract features

First segment the images

Def: *segmentation* – to divide the image into 2 or more regions

- Separate "flame" from "non-flame"
- Hard and slow to do in the image space (try it!)
- Much easier in the score space



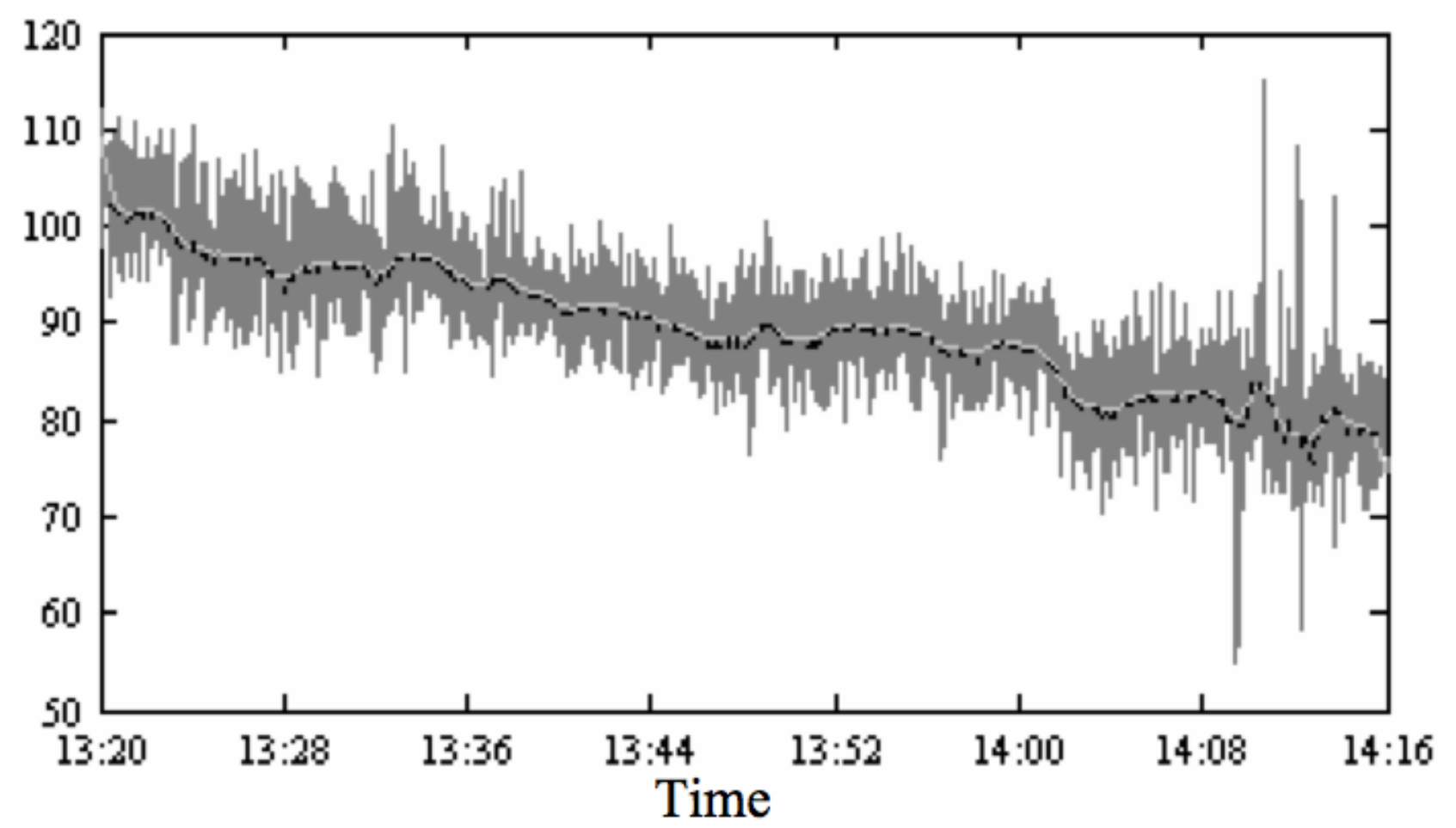
Features extracted by Honglu

Extract these features from the **entire image**:

- **A**: number pixels in flame region (area)
- **B**: average intensity of flame pixels after converting to grayscale (brightness)
- **U**: standard deviation of grayscale flame pixels (uniformity)
- **W**: average intensity of background pixels
- $\mathbf{s}_{1,m}, \mathbf{s}_{2,m} = \left[\ \overline{R},\ \overline{G},\ \overline{B}\ \right] \mathbf{P} =:$ average colour of entire image projected onto the PCA model
- $\mathbf{s}_{1,f}, \mathbf{s}_{2,f}$: as above, except for average flame colour
- $\mathbf{N_c}$: number of unique colours in the flame region

i.e. generate a feature vector once per second

Stability of features?

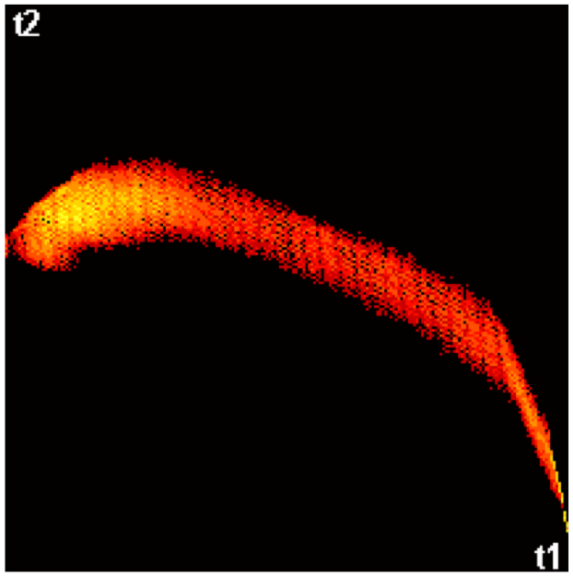


The features are not quite as stable as hoped for. Use smoothing

- dark-gray: feature from each image (noisy)
- light-gray: smooth the features after calculating
- black: smooth in the score space ... described next

Smoothing

1. Average 60 consecutive images, then calculate features (poor)
2. Calculate features, then use the average (e.g. MA, or EWMA)
3. Calculate 60 consecutive score images, *average in the score space*, then reconstruct image: $\hat{\mathbf{X}} = \mathbf{t}_1\mathbf{p}_1^T + \mathbf{t}_2\mathbf{p}_2^T$ and calculate features on reconstructed image.



Option 2 and 3 give similar performance (previous slide)

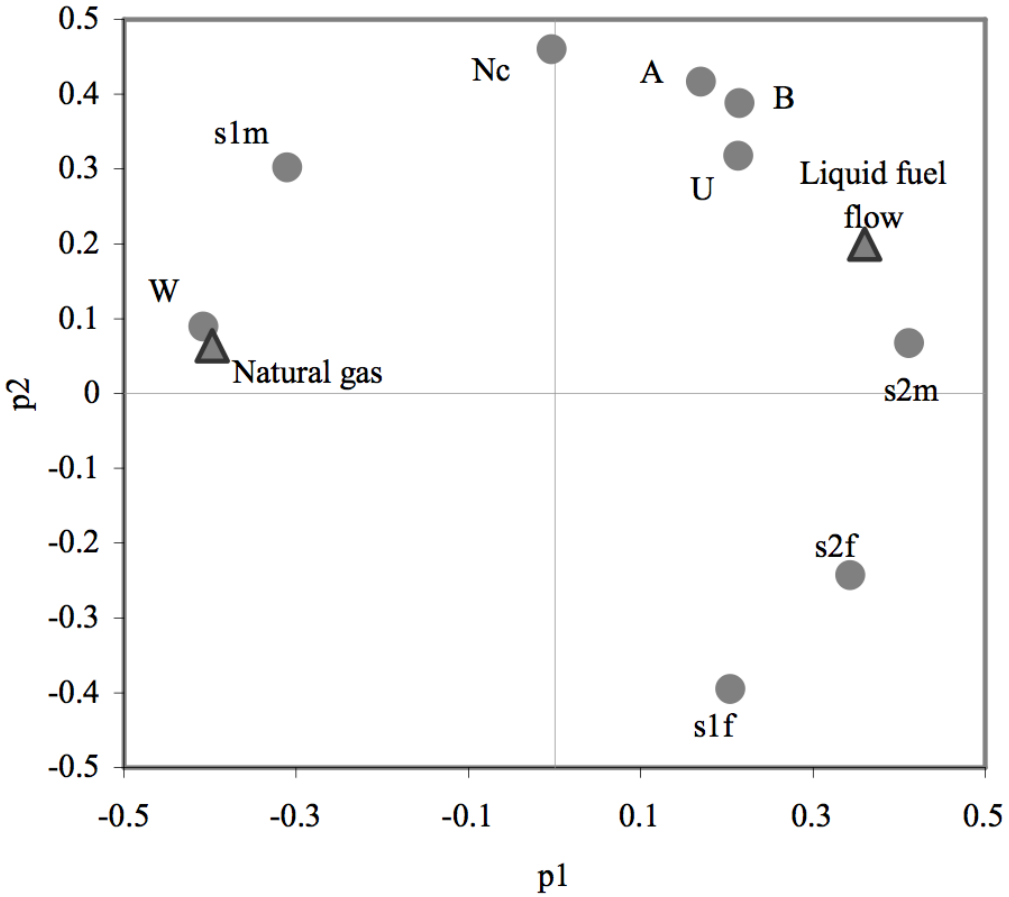
PCA on the features

Waste fuel:

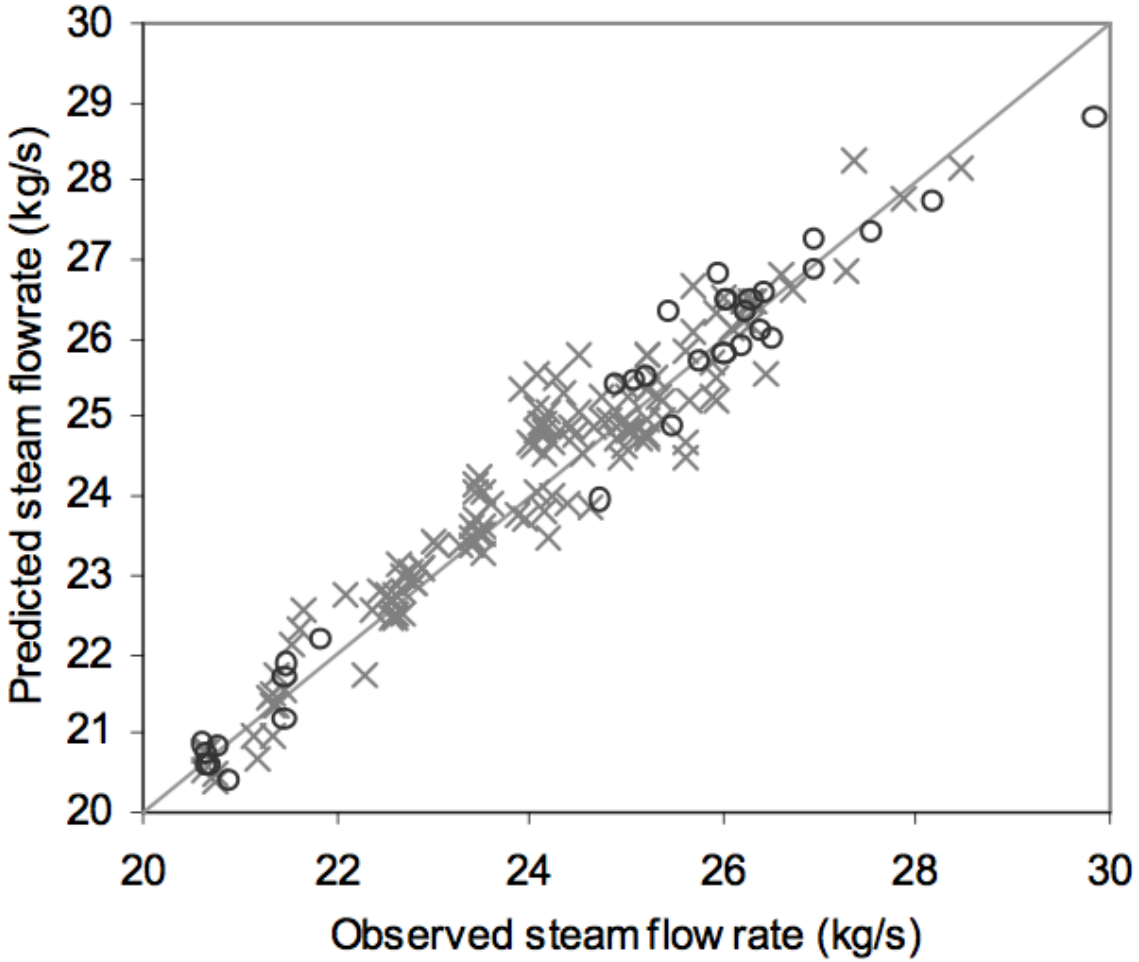
- bigger flame, brigher, and higher standard deviation (more variable)
- fewer non-flame pixels
- higher t_2 values for flame and non-flame regions (shift up-down)

Natural gas:

- more reflectance off boiler walls (more non-flame pixels)
- higher t_1 values (shifts left-right)



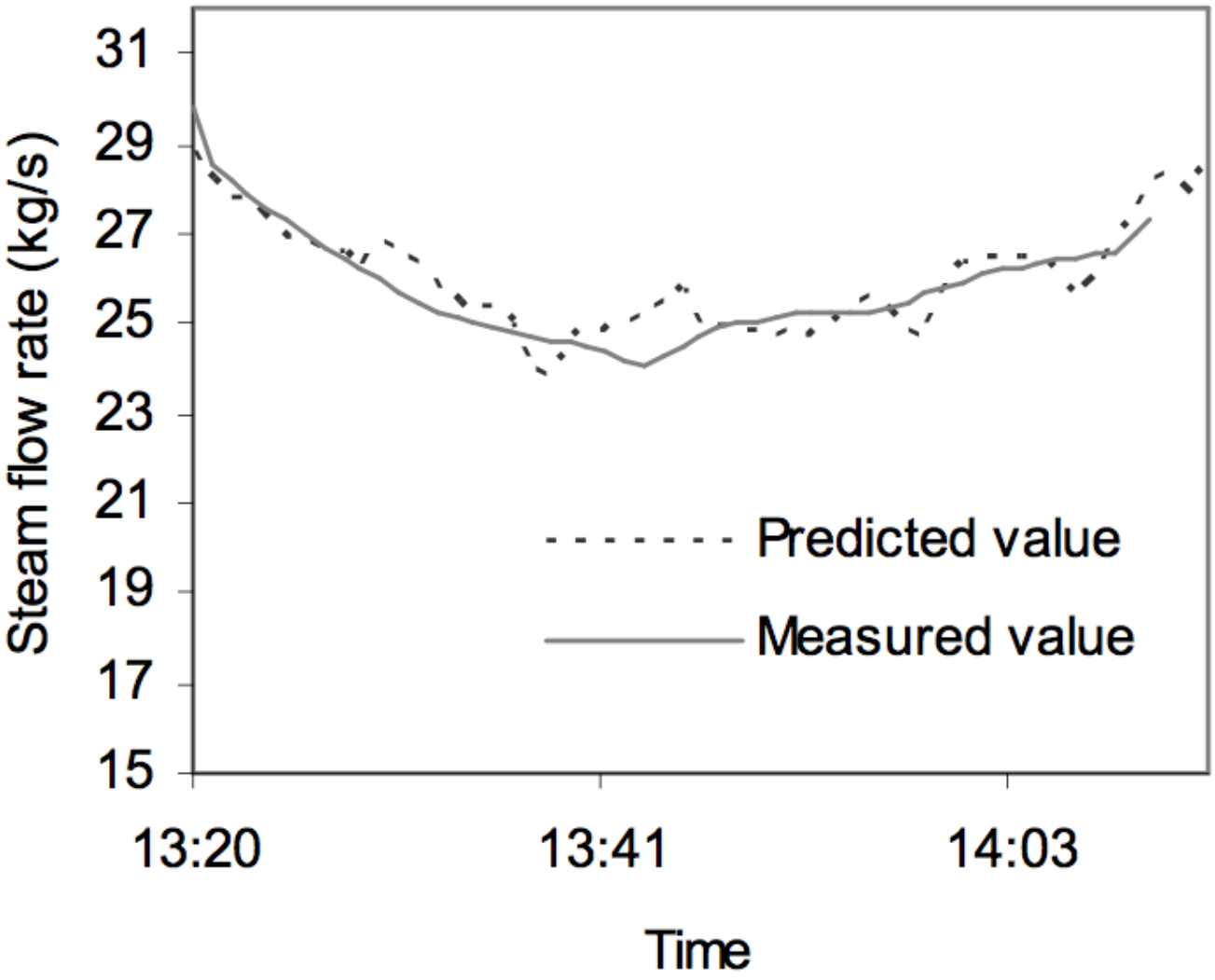
Prediction results



o = training data (RMSEP=0.4 kg/s)

x = testing data (RMSEE = 0.5 kg/s)

Prediction results



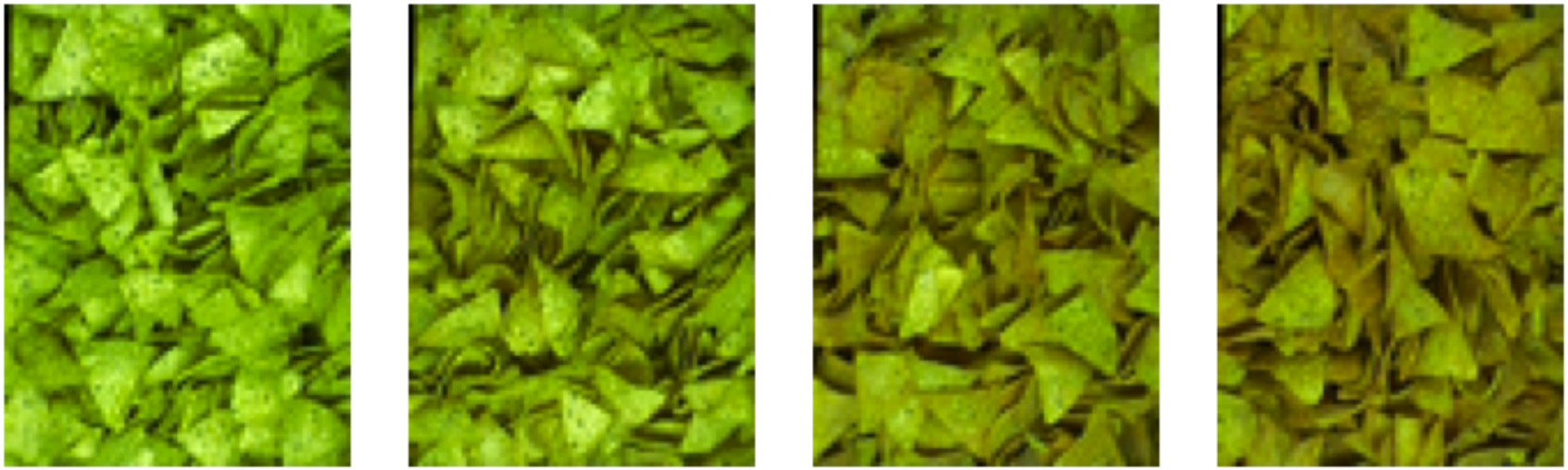
Definitely would require more in-depth study to prove long-term reliability, but shows good promise.

What was learned

- score space has no concept of *spatial* relationships
- scores only capture *spectral similarities*
- masking is effective at spectral segmentation (very rapid)
- hardest part: **good feature extraction**
  - extract features relevant to goal
  - features should be as stable as the underlying process

Seasoning application

We can see a change in product appearance with more seasoning: [2]



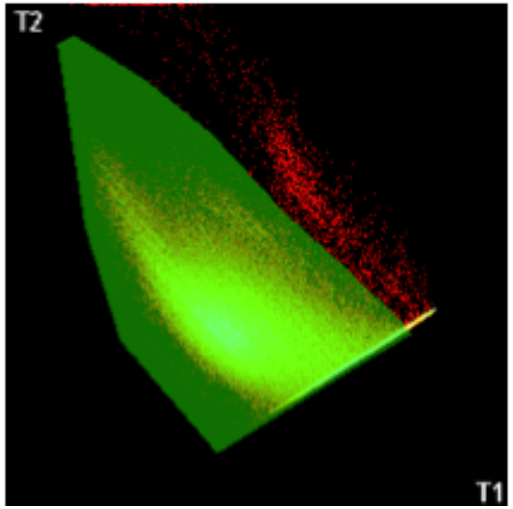
Increased seasoning from left to right

[2] From Honglu Yu's PhD work (used with permission)

Background: segment "background" vs "product" pixels

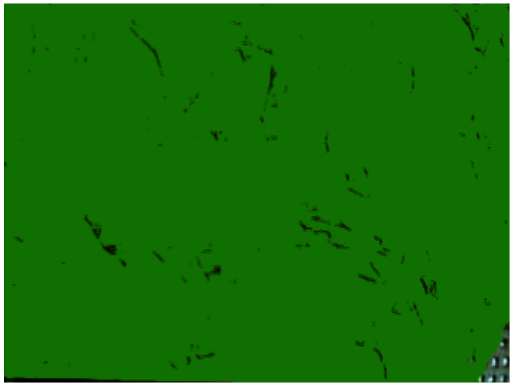
Training image

$\Longrightarrow$



Find suitable mask

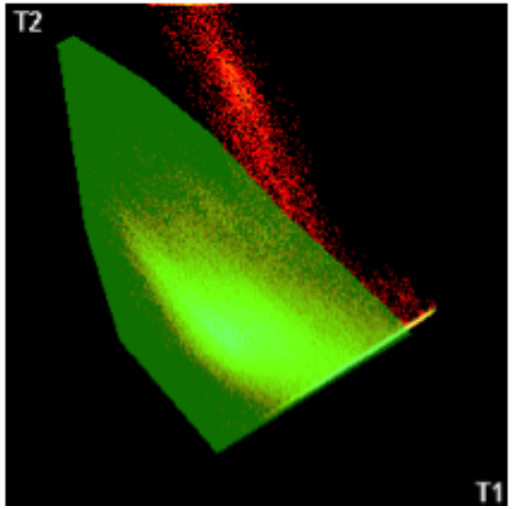
$\Longrightarrow$



Segmented result

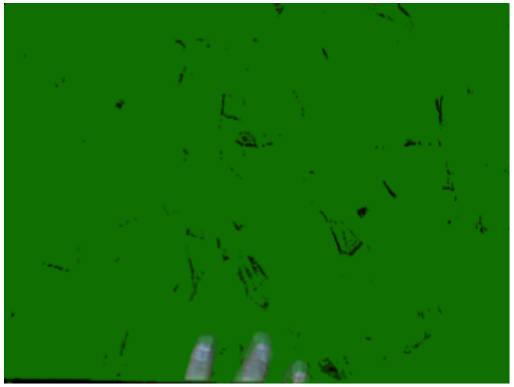
Future **testing** image

$\Longrightarrow$



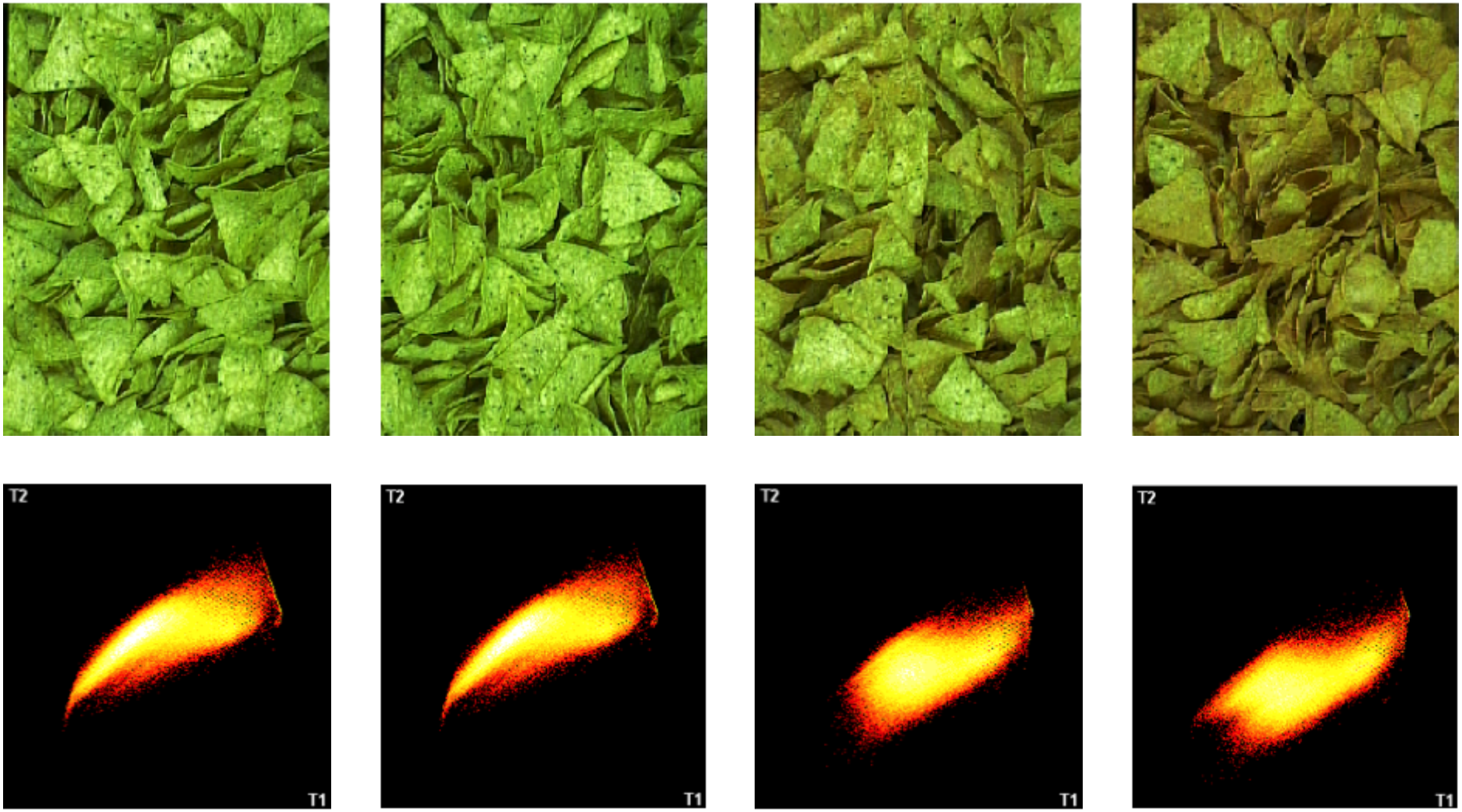
Use the same mask

$\Longrightarrow$



Segmented result on test image

Score images

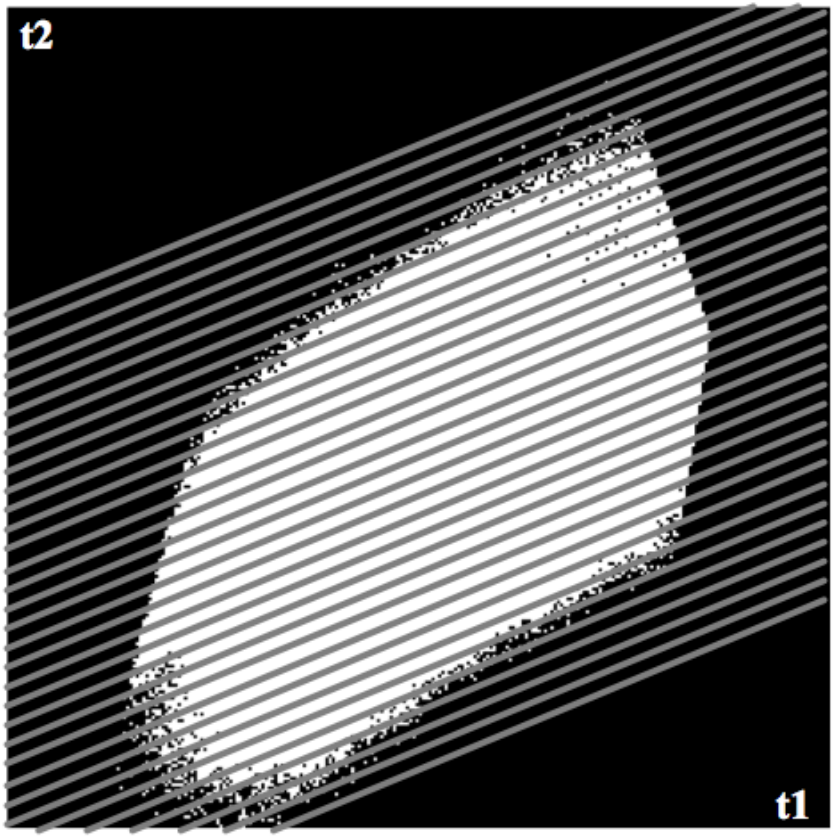


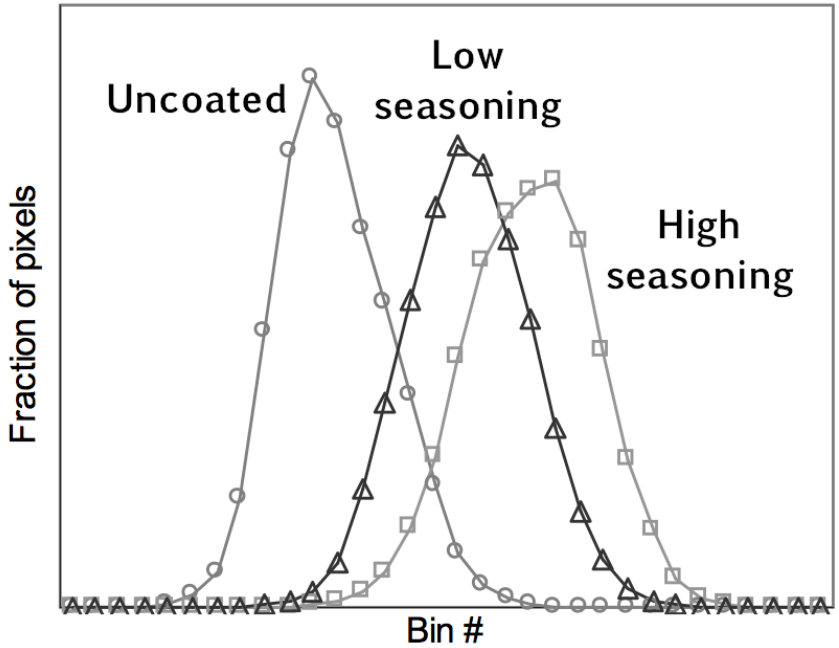
No seasoning | **No** seasoning | **Low** seasoning | **High** seasoning

- Image space changes, while score space remains stable
- Scores change for different seasoning (spectral features)

Feature extraction in the score space

Create bins in the score space and count pixels





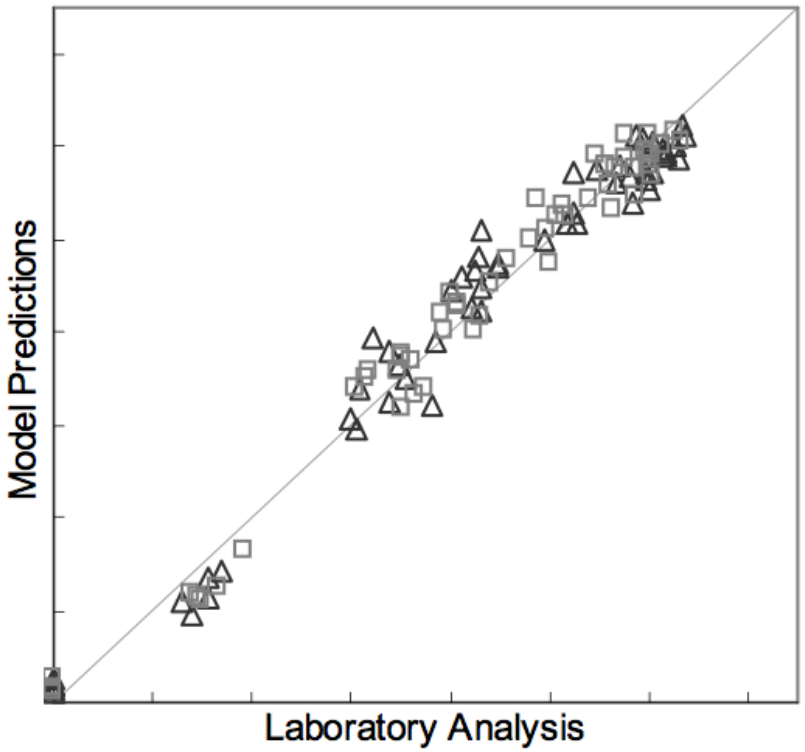
Direction of bins: direction in which seasoning level shifts the score plot [3]

[3] Other binning methods described in journal publications and thesis

Model building and predictions

PLS model:

- **X**-space: cumulative histogram of *fractional* bin counts
  - Note the **X**-space will be very collinear
- y-variable: seasoning level

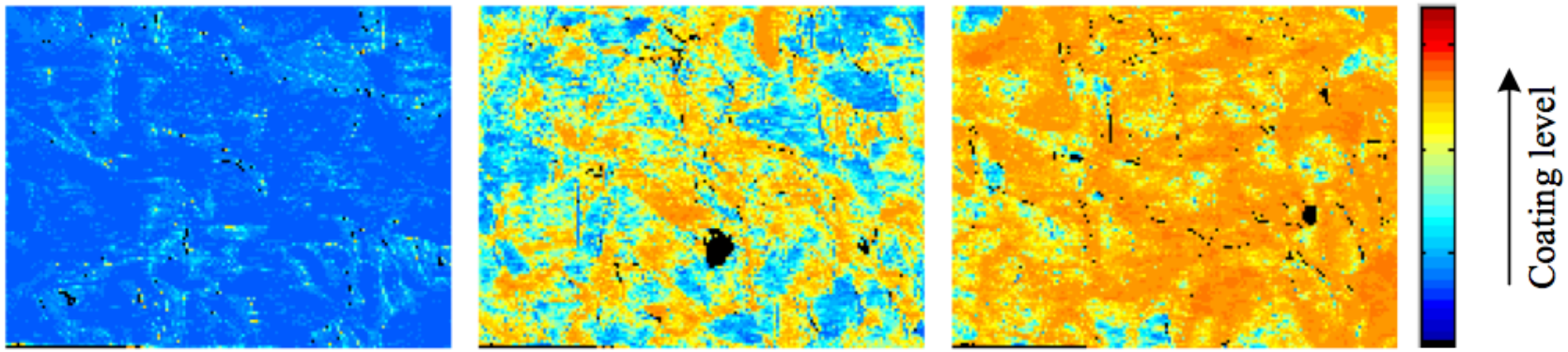


triangles = training data; squares = testing data

Apply the procedure to new images

Notice that none of the steps taken are a function of the number of pixels.

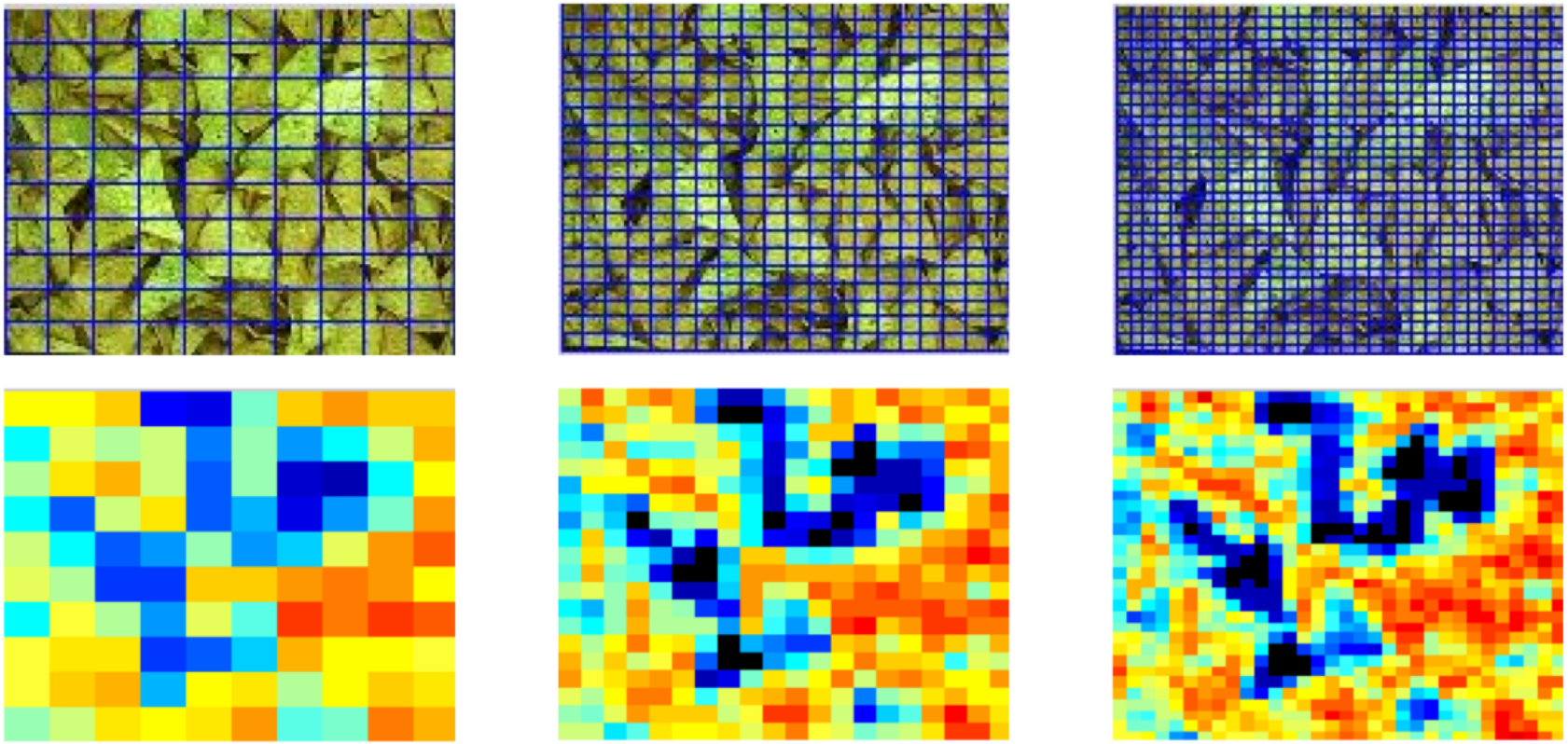
- Can apply the procedure to every pixel in an image



- Used for visual display at the manufacturing line

Apply the procedure to new images

- Apply procedure to subset of pixels: "small-window strategy"



- Calculate seasoning variance: monitoring chart

Seasoning application: summary

- Image segmentation again: to remove background
  - background pixels have different spectral signature
- Features extracted in the score space this time
  - score space is stable, while the image space varies
- Use PLS to regress y = seasoning onto $\mathbf{X}$ = features
  - the features are extremely collinear (MLR would fail)
- Prediction method is not dependent on the image size
  - use the "small-window strategy" to predict seasoning variability
- The seasoning prediction is now used for feedback control at many of the company's manufacturing facilities
- The seasoning variance can be used in monitoring charts

Flame application: references

- Honglu Yu PhD thesis: describes all the details
- Yu and MacGregor: journal publication
- Szatvanyi et al. - predict exit temperature from a kiln

Concepts can be applied to any situation where:

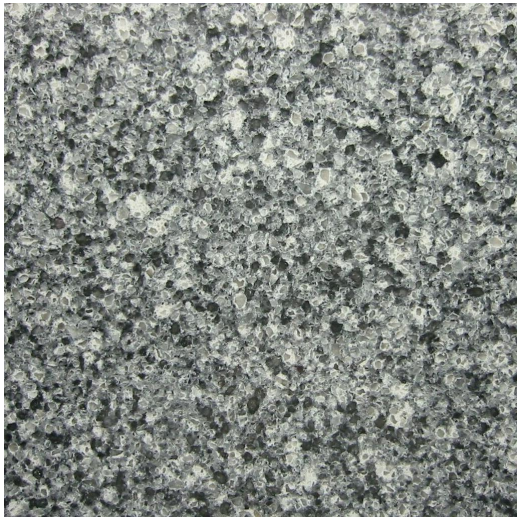
- fast moving and dynamic images, but
- the process underlying it is inherently slow
  - Engines, boilers, furnaces, kilns

Intro to appearance monitoring and control

Manufactured countertops [4]

Too fine

Too high contrast

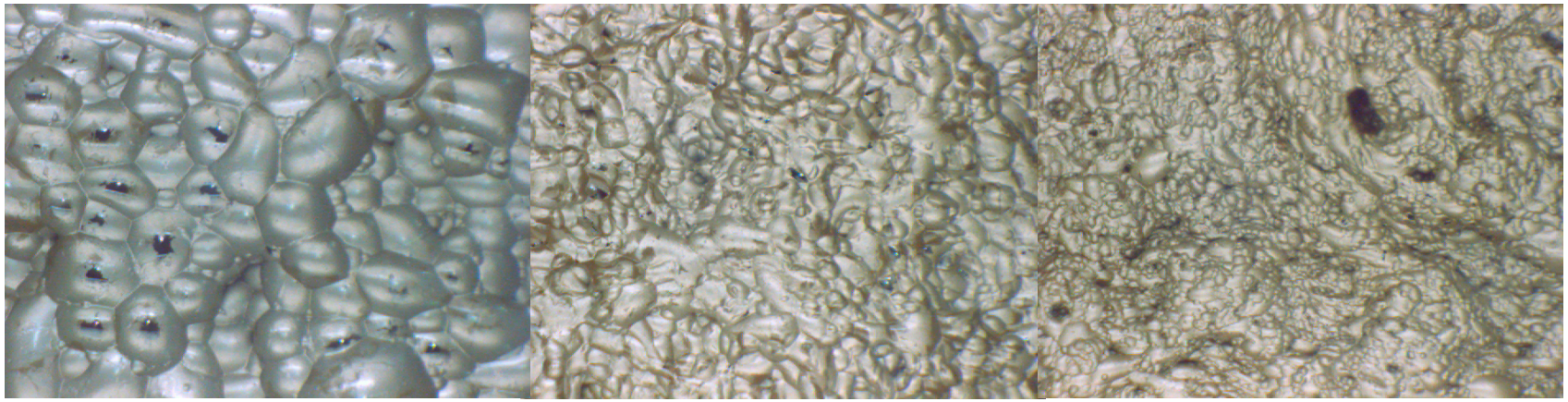


Too coarse

[4] From J. Jay Liu's thesis [Ch 8] (used with permission)

Intro to appearance monitoring and control

Froth flotation [5]

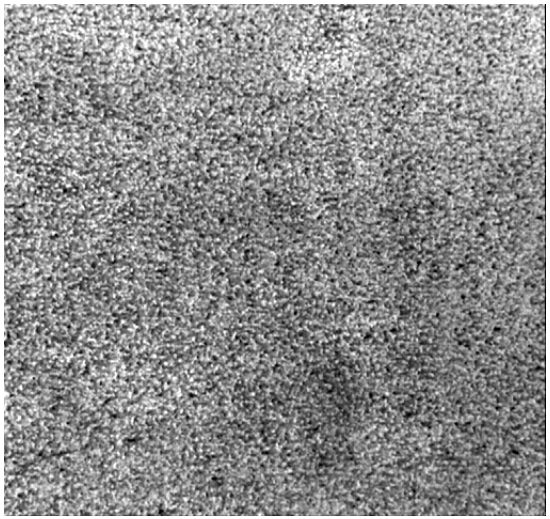


Poorly-loaded bubbles | Well-loaded bubbles | Overloaded bubbles

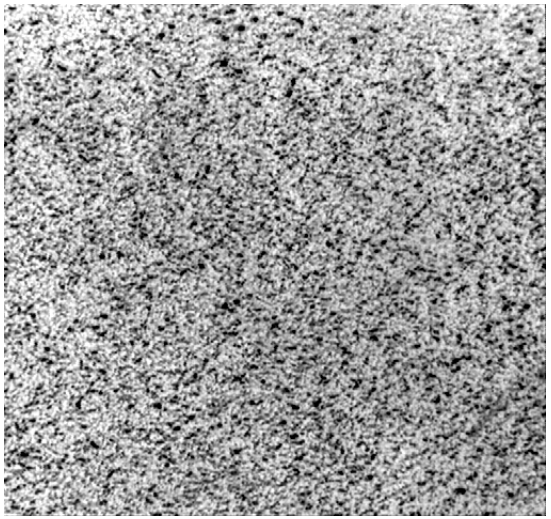
[5] From June Liu's thesis [Ch 6 and 7] (used with permission)

Intro to appearance monitoring and control

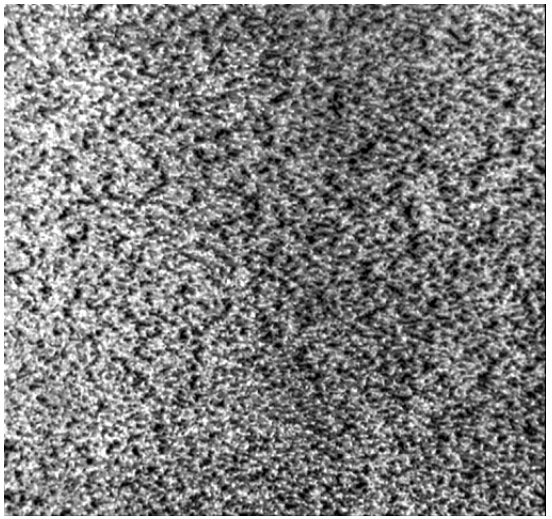
Surface quality of rolled steel sheets [6]



Good



Medium



Bad

[6] From June Liu's thesis [Ch 3] (used with permission)

Steel application details

Steel sheets are classified: excellent, good, medium, and bad

- size and severity of pits
- number of pits

Excellent and good categories: have only few, shallow pits

Data available

- 35 grayscale images from 4 groups (ideally we would like images for testing)
- black ink is poured on and wiped off: to enhance pits

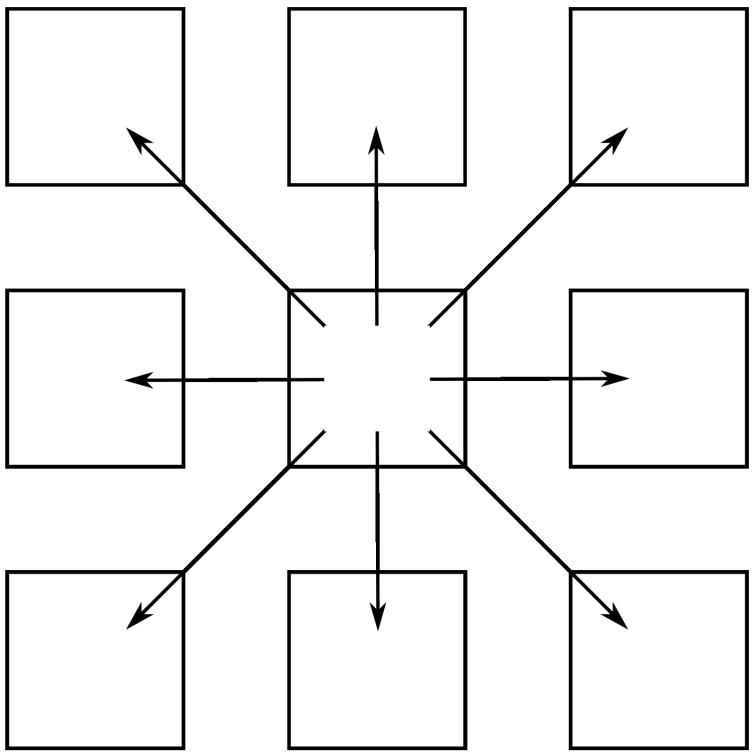
Suitable features?

It is the visual appearance (particularly texture) that is important. How to quantify this?

Extracting texture features: one way

Take the original image and shift it 8 times by 1 or more pixels:

- Repeat for each colour layer
- Creates a very correlated cube of data
- e.g. RGB image: unfolds into $9 \times 3 = 27$ columns



Some of the scores will capture "directional derivatives".

More details in review paper by Prats-Montalbán et al.

Wavelet texture analysis: another way

Alternative to shifting image is to extract wavelet features.
Decomposes image into 4 parts:

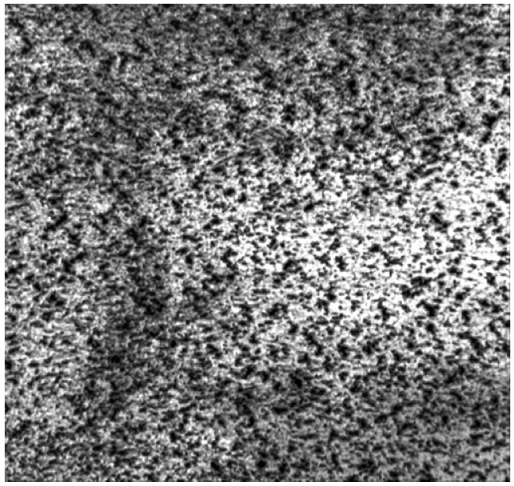
- a sub-sampled "approximation" image (lowpass, then lowpass filter)
- horizontal details (lowpass, then highpass filter)
- vertical details (highpass, then lowpass filter)
- diagonal details (highpass, then highpass filter)

We will combine the 3 "detail" images into one for display

$$\mathbf{X} = \textbf{approximation} + \textbf{detail}$$

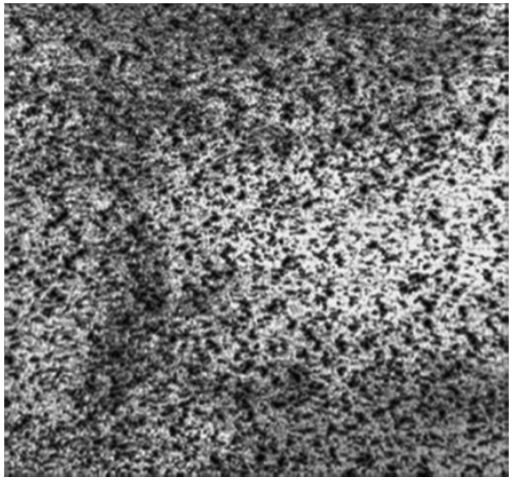
Caution: I'm an extreme novice wrt wavelets ... use this section with caution.

Wavelet texture breakdown on bad steel image



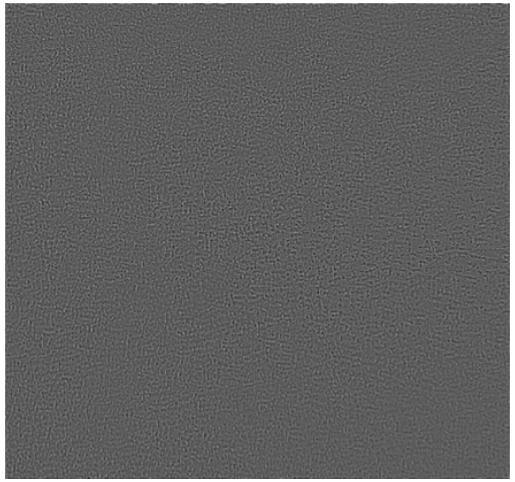
Original

=



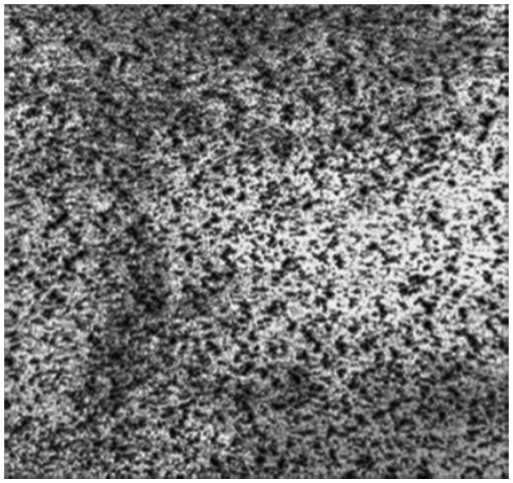
A1

+



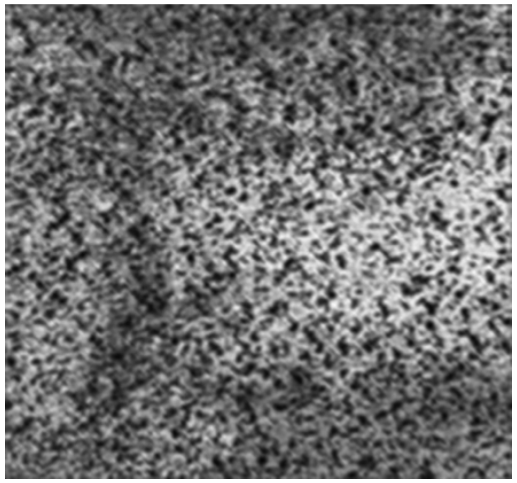
D1

Wavelet texture breakdown on bad steel image



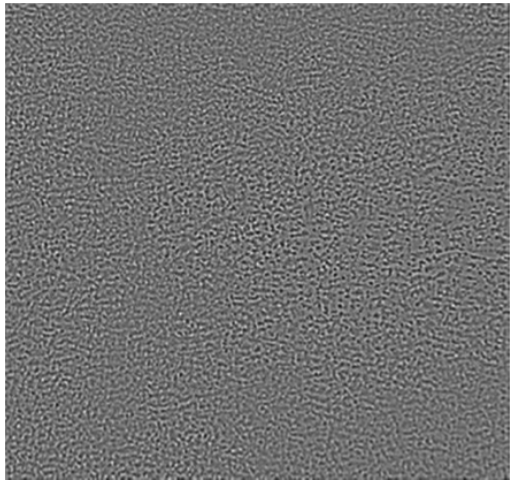
A1

=



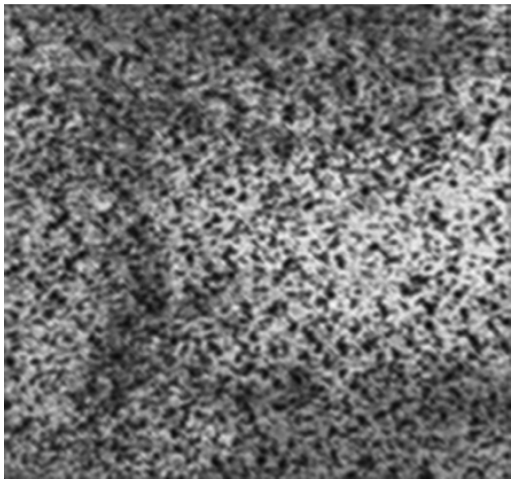
A2

+



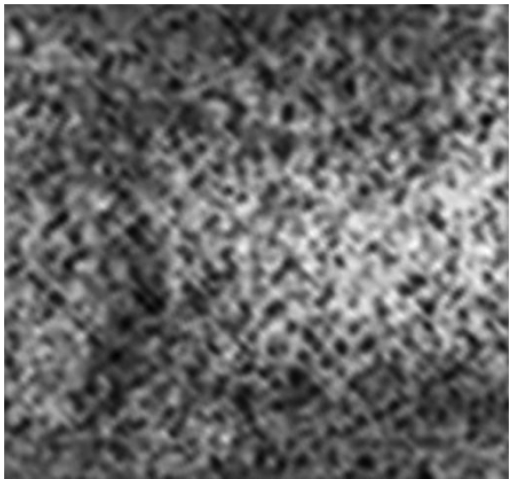
D2

Wavelet texture breakdown on bad steel image



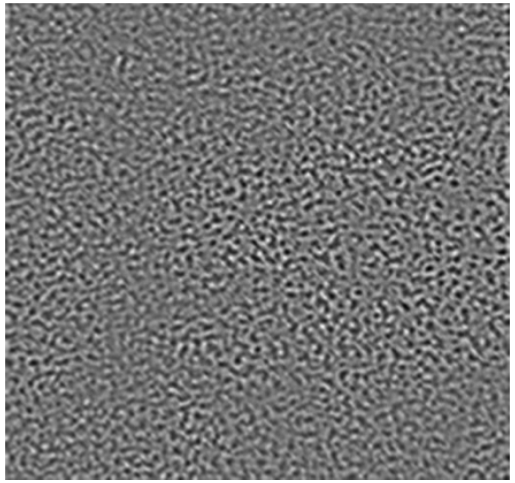
A2

=



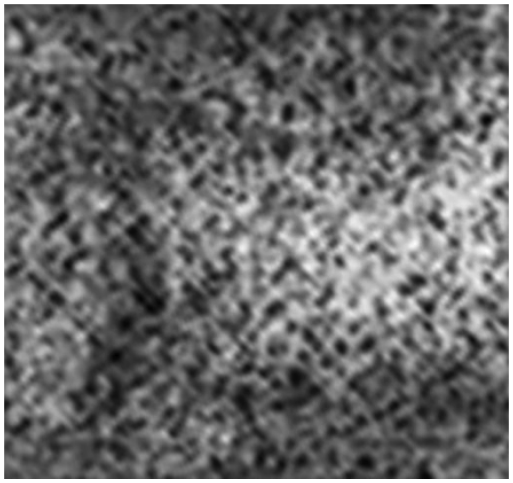
A3

+



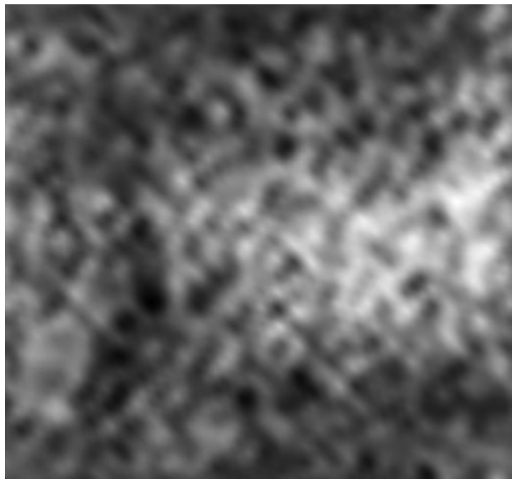
D3

Wavelet texture breakdown on bad steel image



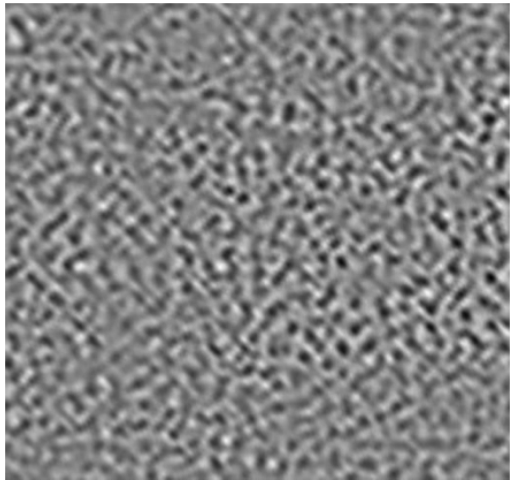
A3

=



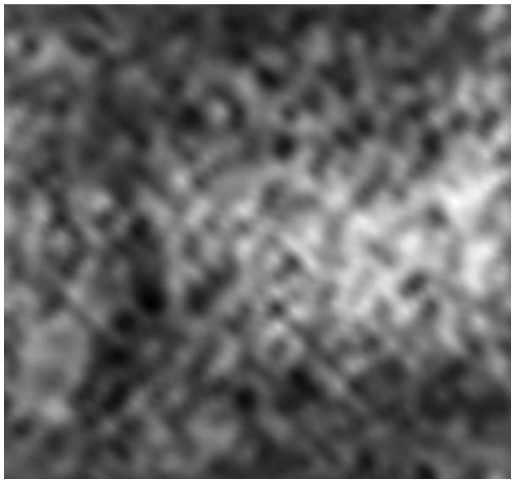
A4

+



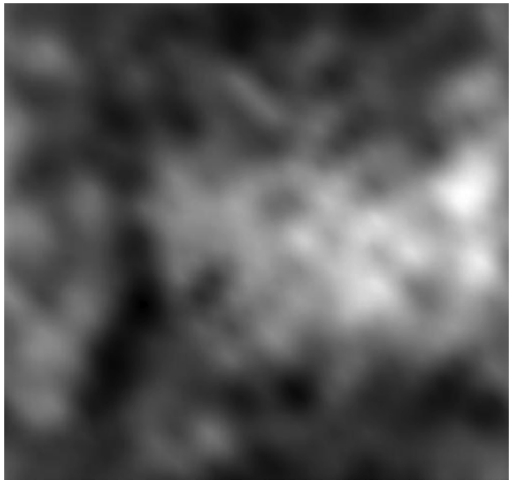
D4

Wavelet texture breakdown on bad steel image



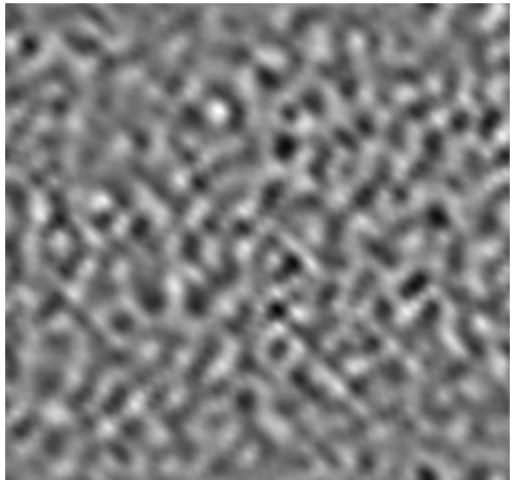
A4

$=$



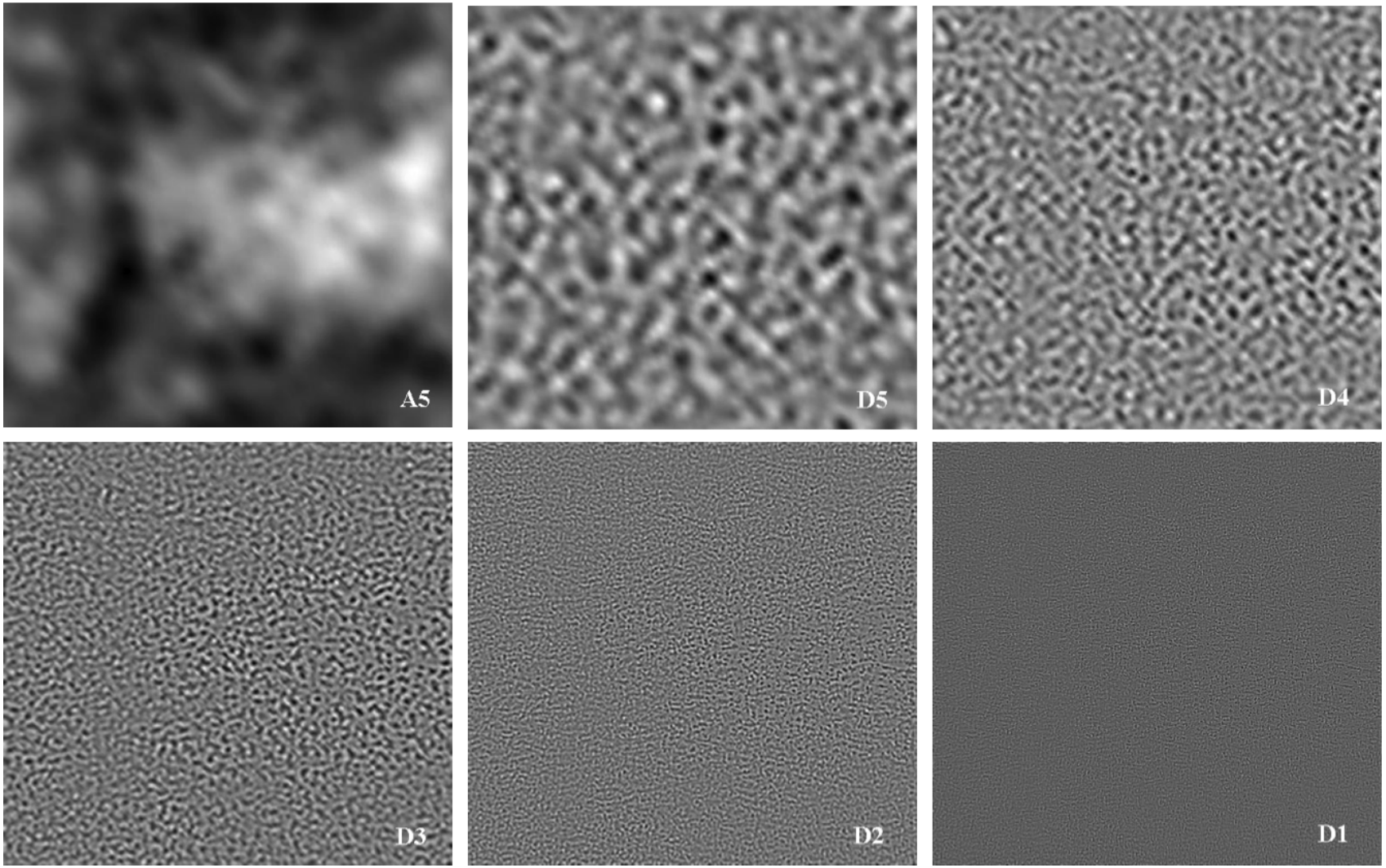
A5

$+$



D5

Wavelet texture summary



A complete summary of the original image at different frequencies.

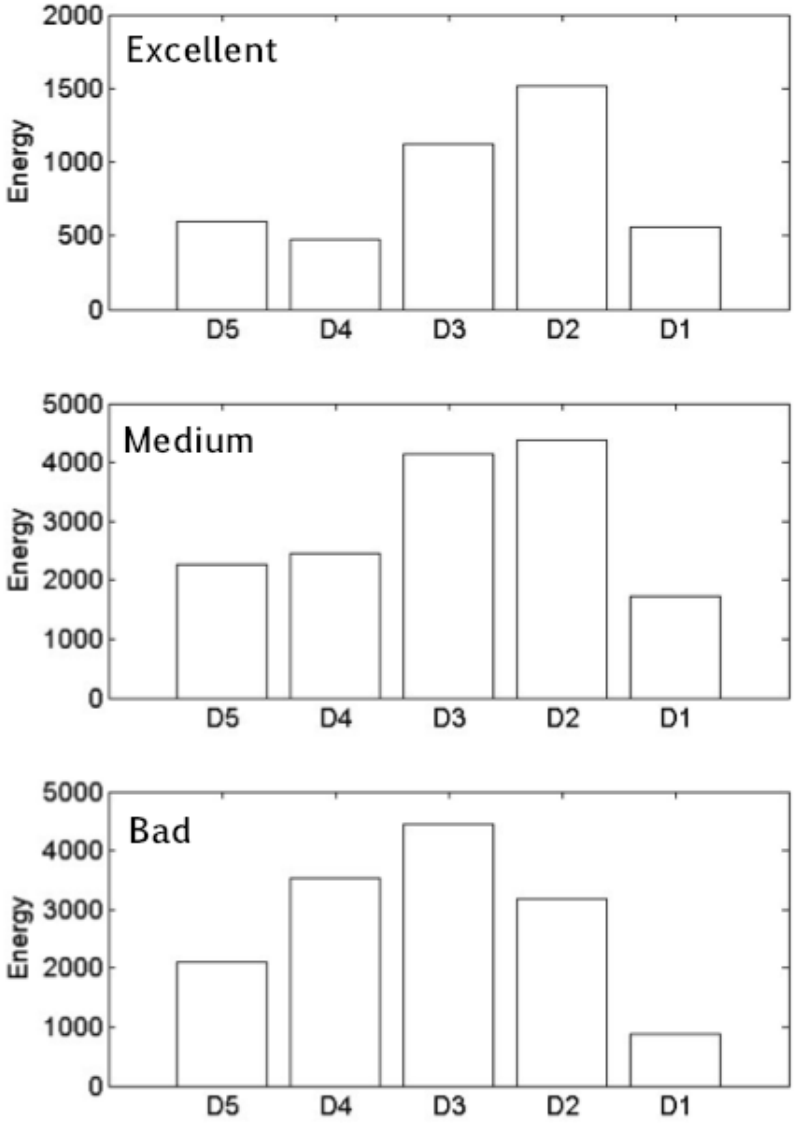
Feature extraction

- D1, D2, ... D5: capture lower and lower frequencies
- D1: captures really fine, high frequency details
- Larger pits are captured in higher D_i images

Potential features to extract from D_i images:

- "energy" $= \sqrt{\text{sum of squares of pixel values}} =$ variability
- "entropy" = sum of absolute values ($0 \Longrightarrow$ a solid colour)
- "maximum" and "minimum"

Examples of features extracted



As expected, bad surfaces have more energy in the higher details (larger pits)

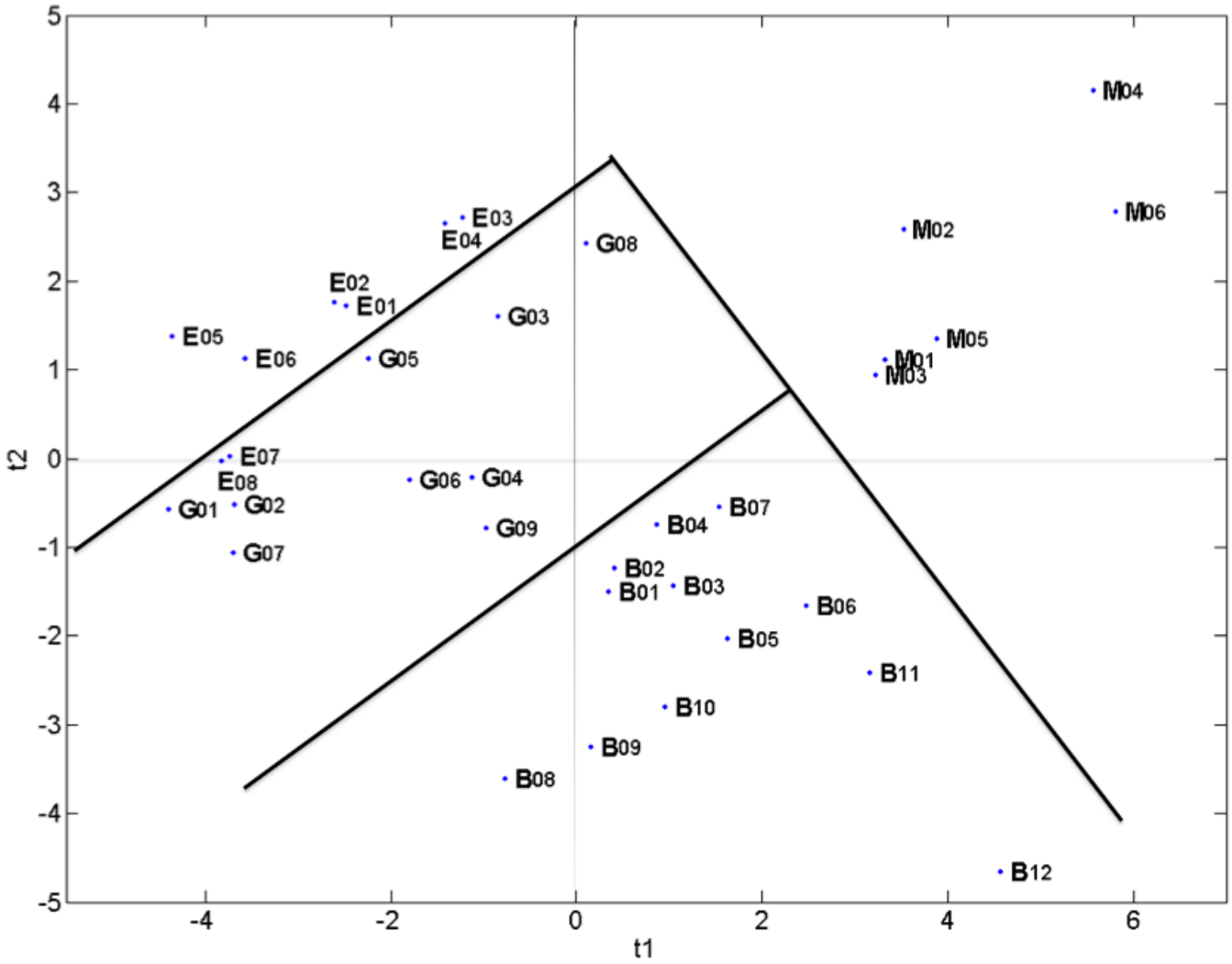
June Liu extracted 15 *energy* features: 5×3 details:

1. horizontal
2. vertical
3. diagonal

The sum of the energy in all 3 directions is shown on the left.

Calculations in MATLAB; used Coiflet wavelet.

Separating grades of steel



- PCA on the 15 features show good, natural separation into the 4 categories: **E**xcellent, **M**edium, **G**ood and **B**ad
- Separating lines are added manually

Practical issues

Lighting, lighting, lighting

- surface pits on steel: how to emphasize pits?
- snack food: performance degrades dramatically if ambient light changes

Practical issues

Drift in ambient lighting (day/night)

- Recalibrate with a neutral background periodically
  - background belt (e.g. seasoning application)
  - colour cards of known spectral intensity

Correct for drift. More details in Zheng Liu's Masters thesis.

Similar issue: different cameras give different pixels values on same image scene

Summary

Feature extraction step is critical:

- Assumption is that image which you extract features from is entirely related to y
- That is only an approximation
  - seasoning in image not all the same
  - surface appearance not all the same
- The stronger this assumption is met, the better the predictions
- Extract features from the space that varies with the y-variable
  - flame application: y was stable in one frame to the next, feature space should also be