Machine Vision for Crack Detection

Fault identification in solar panels and wind turbine blades

Jerome Wynne

July 26, 2017

University of Bristol, DNV-GL

Table of contents

Problem Description

Problem Description

DNV have recently begun inspecting wind turbines using drones.

The footage captured is manually reviewed in search of damage.

Our work

We sought to develop a computational model capable of identifying damage automatically.

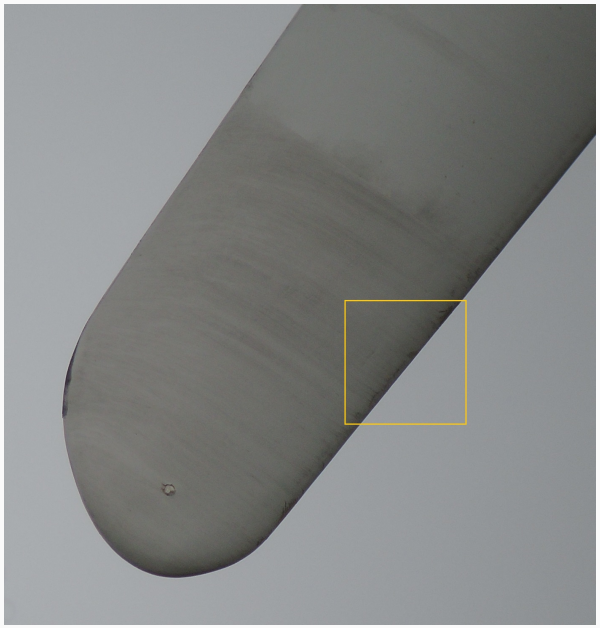
Figure 1: Automated turbine inspection can alleviate the personal risk, tedium, and expense of manual turbine inspection.

Problem Description

The drones captured visible and thermal video, along with point cloud data using an on-board lidar scanner.

Defects of interest

- Cracks
- Delaminated regions



(a) Image from the dataset.

(b) Crack on leading edge.

Problem Description

Complicating factors

- Redundant information (1400 $\times$ 1400 images)
- Dirt
- Background
- Lighting conditions
- Dynamic pathfinding
- Camera focus
- Junk on lens (e.g. water droplets, debris)

Exploratory work was conducted on a toy problem that was thought more tractable.

Problem Description

Toy problem

Detect cracks in images of solar panels.

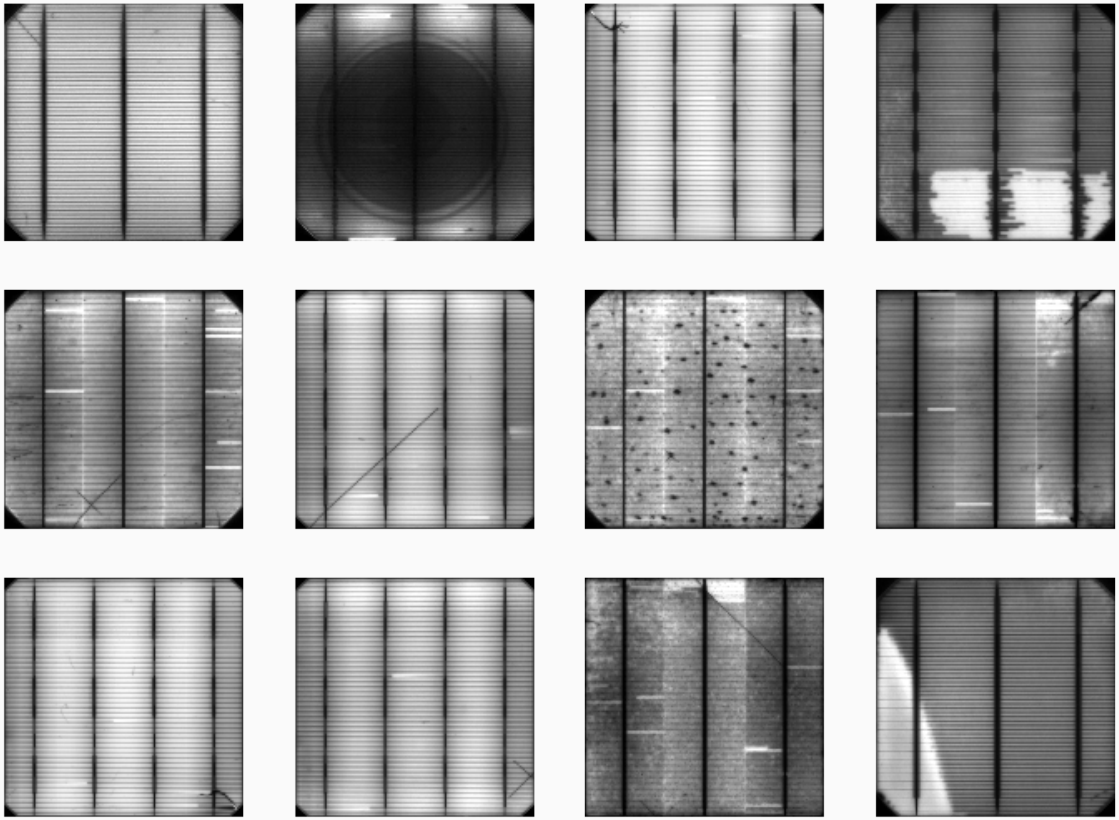


Figure 3: A sample of scanned panels. Cracks are dark lines oriented at $\approx 45^{\circ}$.

Problem Description

Complicating factors

- Handling large-ish images ($\approx 965 \times 965$)
- Low-frequency intensity variation due to delamination
- High-frequency intensity variation due to cells
- Other noise - hail damage, scratches, panel models, debris in scanner
- Variation between panel models
- Small dataset (52 images, not all of which were cracked)

Modeling Strategy

Modeling Strategy

Approach

- Supervised classification/regression problem.
- Manual and learned representations of the images were developed.
- A variety of discriminative and generative models considered.

Desirable model attributes

- Accurate
- Sensitive
- Probabilistic
- Localizes damage

Modeling Strategy

Labeling

Binary masks were created indicating which pixels constituted a crack segment.

Ground truth subjectivity

Cracks and scratches were difficult to distinguish: In general, the latter were fainter and had more curvature.

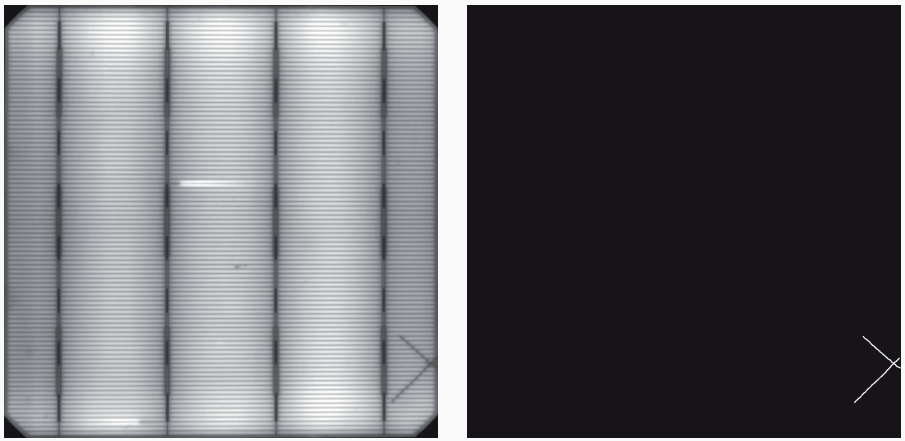


Figure 4: A panel and its associated binary mask.

Modeling Strategy

Loss function

Empirical per-pixel classification error rate:

$$J(\theta) = \mathbb{E}_{\mathbf{x},y\sim p_{train}} |f(\mathbf{x};\theta) - y|$$

Where p_{train} is the sampling distribution described by the training data, and f is our learned function mapping from a set of input features $\mathbf{x}$ to a class $y \in \{0, 1\}$, parameterized by the set θ.

Surrogate Loss Function

To improve our classifier's ability to discriminate between classes, we adopted cross-entropy as a surrogate measure of loss:

$$J^*(\theta) = \mathbb{E}_{\mathbf{x},y\sim p_{train}} - \Big[(1-y)\log[1 - q(\mathbf{x};\theta)] + y\log[q(\mathbf{x};\theta)]\Big]$$

Where $q(\mathbf{x};\theta)$ denotes the probability our learned function assigns to $\mathbf{x}$ being a positive instance.

Experimental Procedure

Experimental Procedure

Vanilla preprocessing procedure

- Resize the images ($965 \times 965 \rightarrow 200 \times 200$).
- Apply a global per-pixel mean subtraction and scaling by standard deviation (a.k.a. *whitening*)[1]

Training protocol

- Compute parameters with closed-form expressions over the entire training set.
- On-line minibatch training - gather a balanced minibatch across all pixels only as needed (1.5m+ negative examples, $<$ 10k positive examples).
- Apply basic patch augmentations (flipping, rotation, intensity shifts).

[1] whitening encourages numerical stability and suppresses common artifacts.

Experimental Procedure

Implementation

TensorFlow was chosen as a development platform. Very good reasons for this[2]:

- Flexible modeling environment
- Excellent diagnostic and logging tools
- GPU support

[2] at a cost of greater verbosity + cryptic error messages

Models & Representations

Models & Representations

Crack attributes

- Continuous
- Small aspect ratio
- Dark (constant intensity)
- Generally propagates diagonally
- Near borders
- ...

Non-crack attributes

- Horizontal/vertical gradients, or a mush of gradients (in the case of circular elements)
- Higher intensity values than crack
- Patterned
- Variable intensity
- ...

Models & Representations

Models implemented

- Gaussian discriminant analyzers.
- Convolutional neural networks.
  - Batch normalization units
  - Spatial transformer subnetworks
  - Expanded input feature spaces
- Canned `sklearn` classifiers.

Models explored

- Relaxation labelers
- Hidden Markov models
- Markov random fields

Experimental Results

Experimental Results

Gaussian discriminant analyzer

As a point of reference, we attempted to model class-conditional pixel intensity variation using a multivariate normal distribution over a patch of pixels:

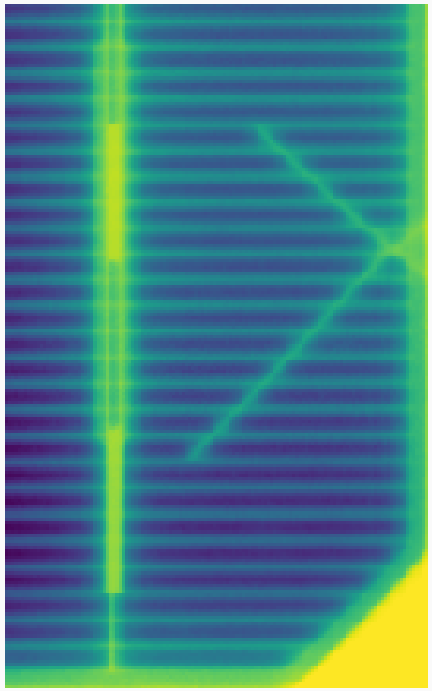
$$\Pr(\mathbf{x}|y = k, \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k) = \text{Norm}_{\mathbf{x}}[\mathbf{x}, \boldsymbol{\mu}, \boldsymbol{\Sigma}]$$

Where $\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k$ are the class mean and covariance respectively. A Bernoulli prior was used, parameterized by the base rate λ. The maximum likelihood estimate of these parameters was taken.

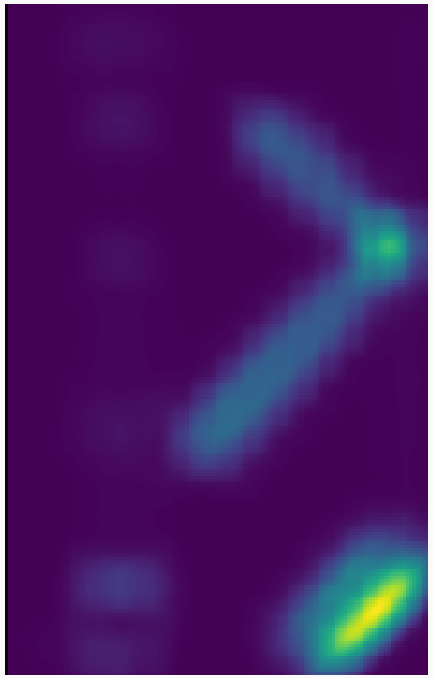
For inference, the maximum *a posteriori* class was taken:

$$\hat{y} = \arg\max_k \left[\Pr(\mathbf{x}|y = k, \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k) \cdot \Pr(y = k, \lambda_k)\right]$$

Experimental Results



(a) Query image.



(b) Crack log-odds.

Experimental Results

Why the Gaussian model was unreasonable

The Gaussian model by itself is inappropriate for several reasons:

- It is only capable of modeling local variations (covariance matrix constrains patch size to 20×20)
- It struggles to represent crack orientations simultaneously - the cross-shaped mean intensity is vulnerable to circular non-crack elements (i.e. hail damage).
- A pixel's intensity does not have a symmetric distribution.
- The classes probably aren't separable by a hyperparaboloid.

Experimental Results

Convolutional neural networks

We used a convolutional neural network to learn a nonlinear transformation that improved class separation. This transformation vastly improved the efficacy of simple classifiers (e.g. logistic regression).

(brief summary of convnet working principle, suitability for the problem)

Various architectures were tested. These will now be described (*pending*).

Experimental Results

Gabor Filtering

A Gabor filter is an image processing tool for extracting edges from an image. For our purposes, it consists of convolving an input image with an oriented 2D decaying sinusoidal kernel.

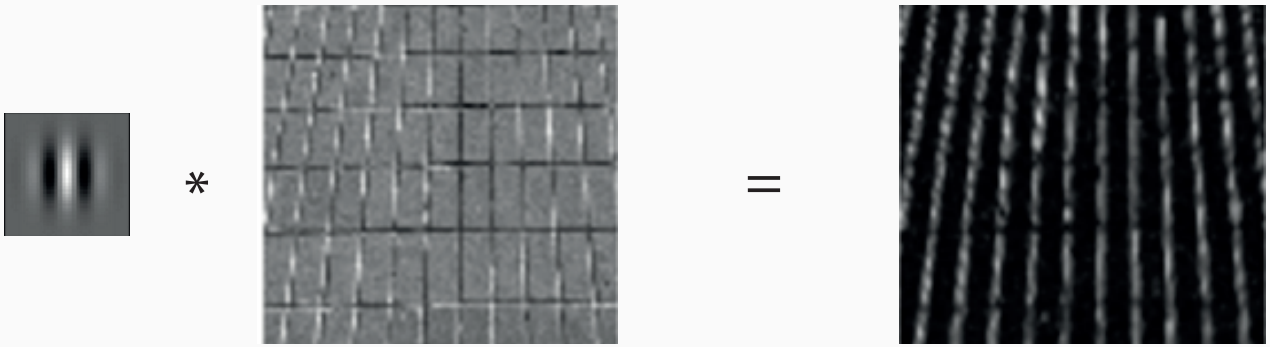


Figure 6: We convolve a 2D sinusoidal kernel (*left*) with an image (*right*) to yield an edge-based representation of the image (*right*)

.

Experimental Results

Sobel and Gabor filtering

By computing the intensity derivatives[3] in the horizontal direction, then taking the result's derivatives in the vertical direction, it was possible to suppress a large proportion of the noise.

Figure 7: (*Left*) Input image; (*Center*) Derivatives followed by a difference of Gabor filters; (*Right*) One of the Gabor kernels used - the other is flipped across the vertical axis.

[3] via a Sobel filter

Experimental Results

Histograms of oriented gradients

A weakness of the Sobel+Gabor composition is that it does not suppress unwanted image artifacts with slanted edges, such as other panel defects. To circumvent this issue, we introduced a HOG descriptor that binned gradient orientations over image cells.

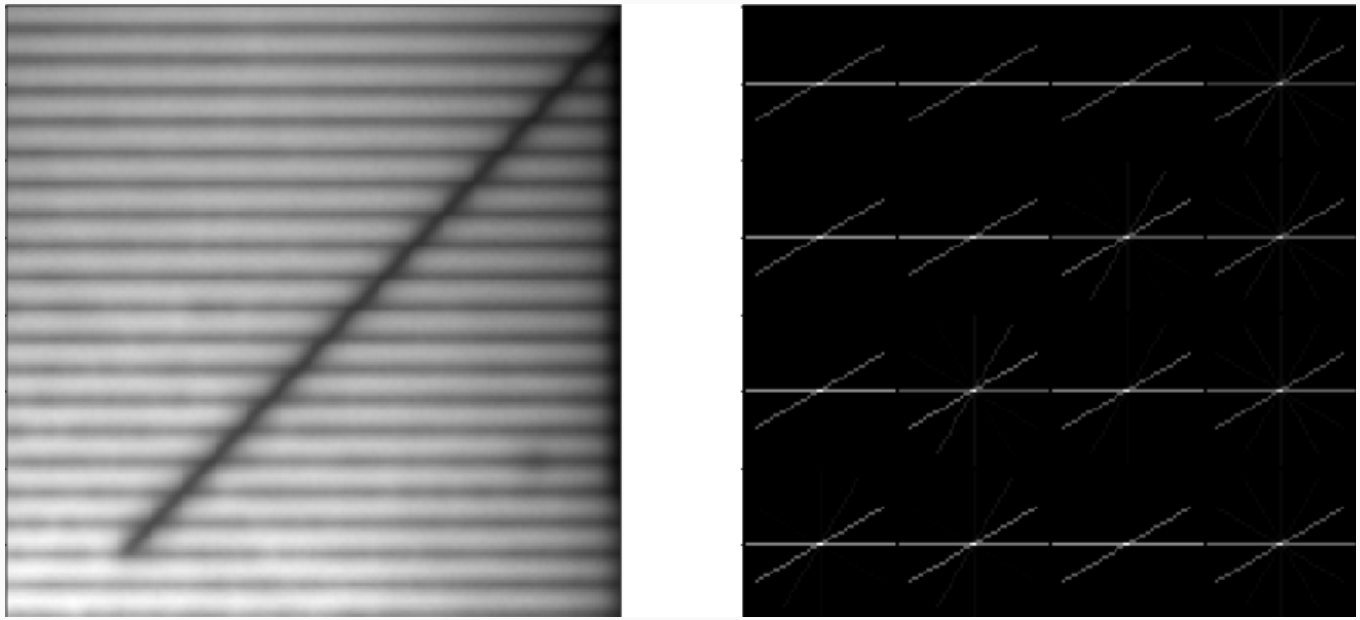


Figure 8: (*Left*) Image patch; (*Right*) Visualization of orientation histograms: Large gradients are in the slant-wise direction only.

Experimental Results

Histograms of oriented gradients contd.

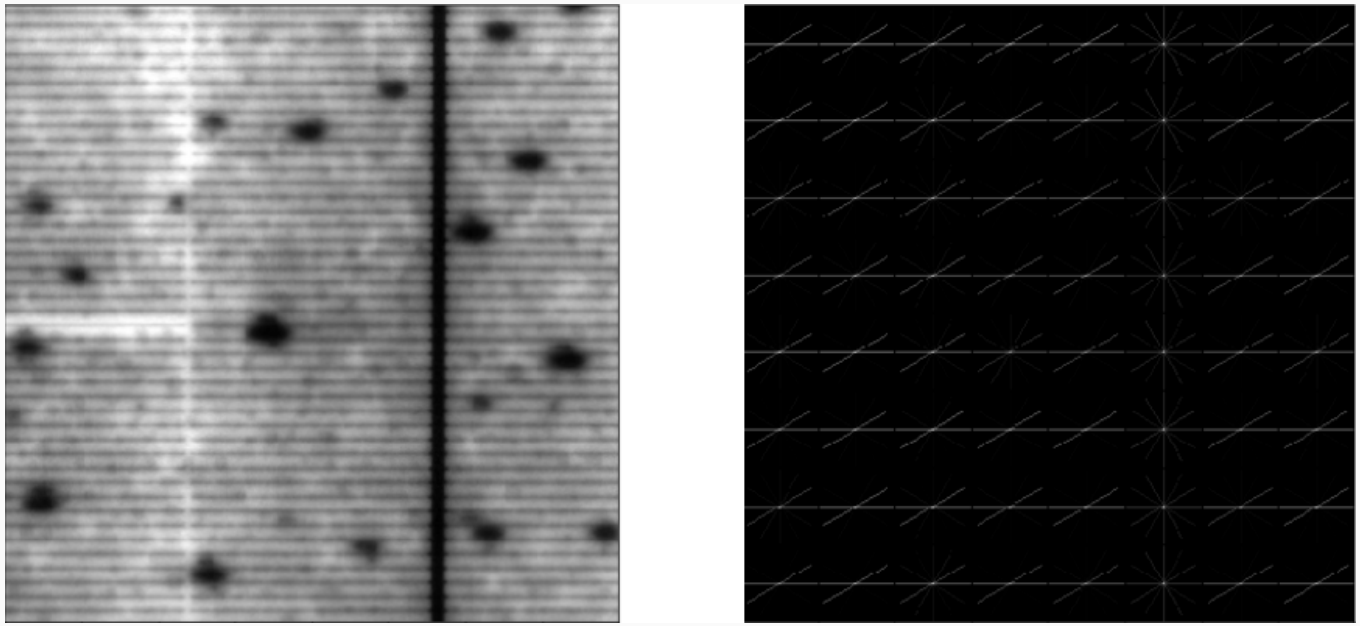


Figure 9: Another patch. Large gradients in all directions $\Longrightarrow$ network may be able to learn a dynamic threshold that suppresses this sort of thing.

Experimental Results

Passing alternative representations into the network

Filter representations were fed into the network by appending them as additional channels to the pixel intensities.

Each channel of the HOG descriptor corresponds to a gradient orientation[4].

Figure 10: Filtered images were stacked with the original grayscale intensities to pass the feature set to the network.

[4] we project the HOG tensor onto a space of the same dimensionality as the input patch.

Experimental Results

Batch normalization

We normalize the output of each layer so that the input distribution to each layer is of a constant magnitude during training. This serves to avoid very large/small gradients. It also properly conditions the second derivative of the loss surface, encouraging stable convergence. *Present results*.

Spatial transformer networks

We plan to introduce a subnetwork that learns the parameters of an affine transform. This affine transform is applied to the input image to produce a cropped version of the original image, potentially reducing panel processing time.

Image and patch size

See results

Questions?

References

Pending []

Appendix: Panel Extraction

Part-way through the project, it was necessary to adopt a new dataset, consisting of individual panels extracted from a grid. The extraction procedure required designing a set of deterministic filters. The procedure was as follows:

1. Equalize then threshold in favor of dark pixels.
2. Crop image borders (i.e. all-black columns/rows at edges).
3. Deskew and center the grid of panels.
   - 3.1 Estimate the bounding rectangle's orientation and position using a Hough transform.
   - 3.2 Apply an affine transform to this rectangle.
4. Detect horizontal gaps between panels: Use these to estimate the panel width.
5. Use the panel width to estimate the position of the vertical gaps between panels.
6. Use the indices defined by these gaps to crop each panel from the grid.

Appendix: Panel Extraction

Histogram equalization and thresholding

To emphasize the dark gaps between panels the images were first equalized and thresholded.

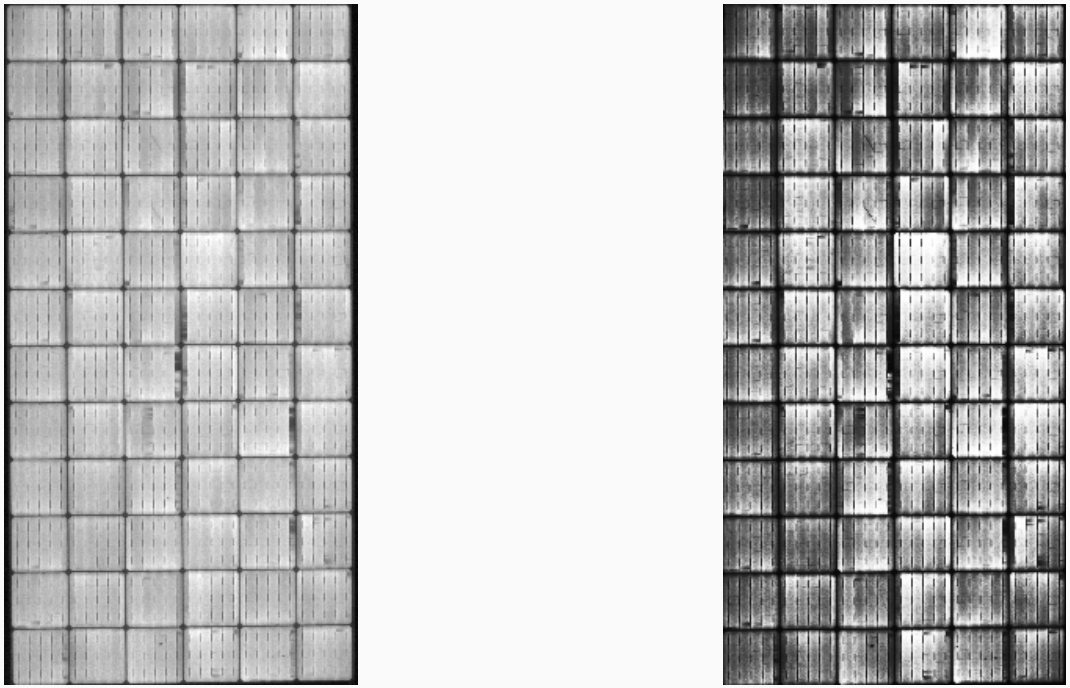


Figure 11: (*Left*) A grid of panels before preprocessing. (*Right*) A grid of panels after preprocessing.

Appendix: Panel Extraction

Hough and affine transform

A high-resolution Hough transform was then applied to the thresholded image.

This produced estimates of the orientation and position of the grid's edges.

These values could then be used to shift the grid into the center of the image frame.

Appendix: Panel Extraction

Line detection

After thresholding and cropping the images, a line detector was applied. The detector was a product over the image axes. For vertical lines, this was:

$$\text{Line score} = \prod_{i=1}^{W} \gamma^{1-x_i}$$

Where W denotes the image's width in pixels, γ allows us to tune the detector's sensitivity, and x_i is the binary ith binary pixel value. Note that:

$$0.99^{1-x_i} = \begin{cases} 1 & \text{if} \quad x_i = 1 \\ 0.99 & \text{if} \quad x_i = 0 \end{cases}$$

Where we've adopted $\gamma = 0.99$ for the purposes of example.

This function was chosen to penalize zero pixels exponentially - the more pixels that are zero, the greater the cost of an additional zero pixel.

Appendix: Panel Extraction

Line detection

The response of the scoring function on the previous slide is shown below.

The peaks in this plot correspond to the dark lines between the panels on the previous slide.

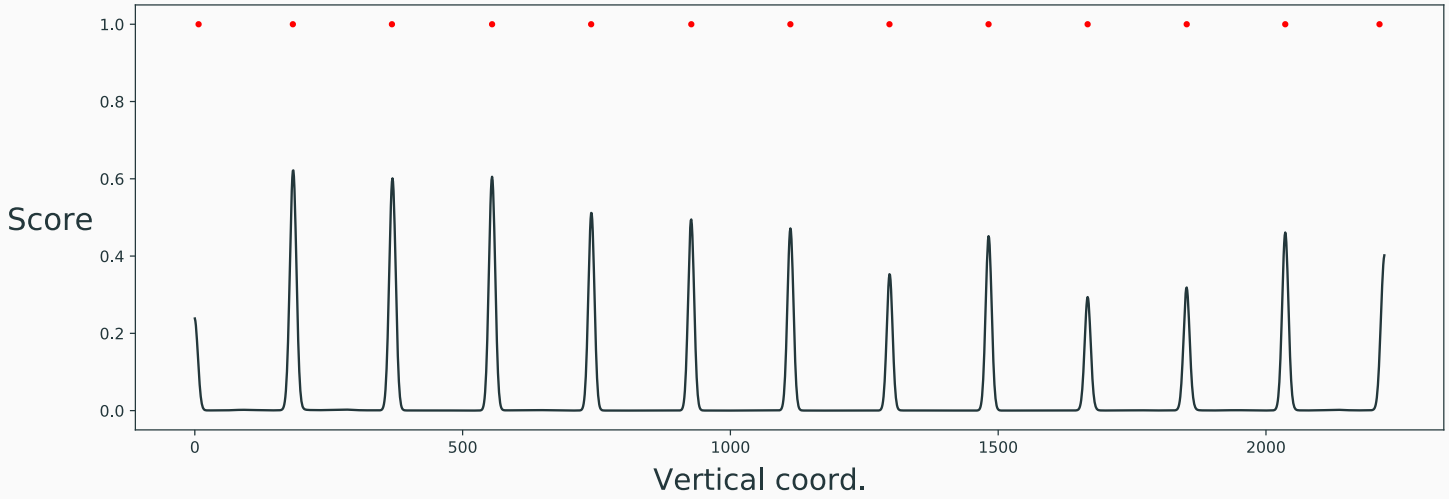
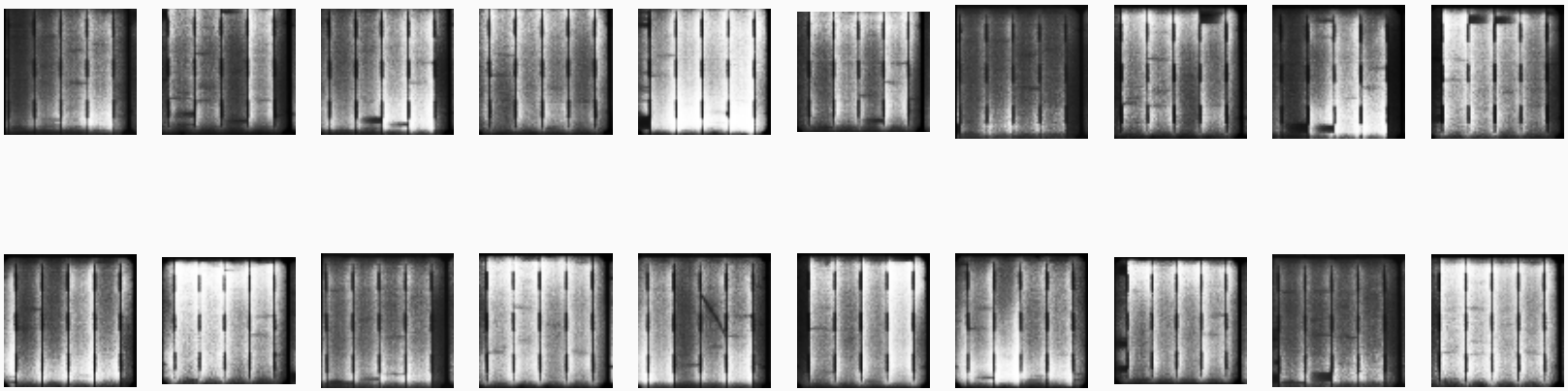


Figure 12: Horizontal line scores for the image on the previous slide. The red points indicate indices at which the detector was triggered.

Appendix: Panel Extraction

Results

A sample of the panels extracted from the grid shown previously is displayed below. There were some unusual (i.e. rectangular) panels for which the detector did not work, however these were rare.



References i