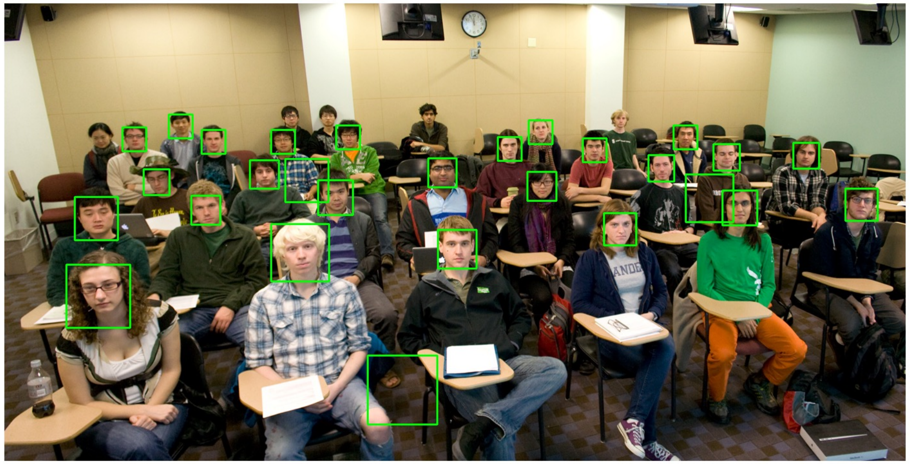
# AMME4710: COMPUTER VISION AND IMAGE PROCESSING WEEK 11

Dr. Mitch Bryson

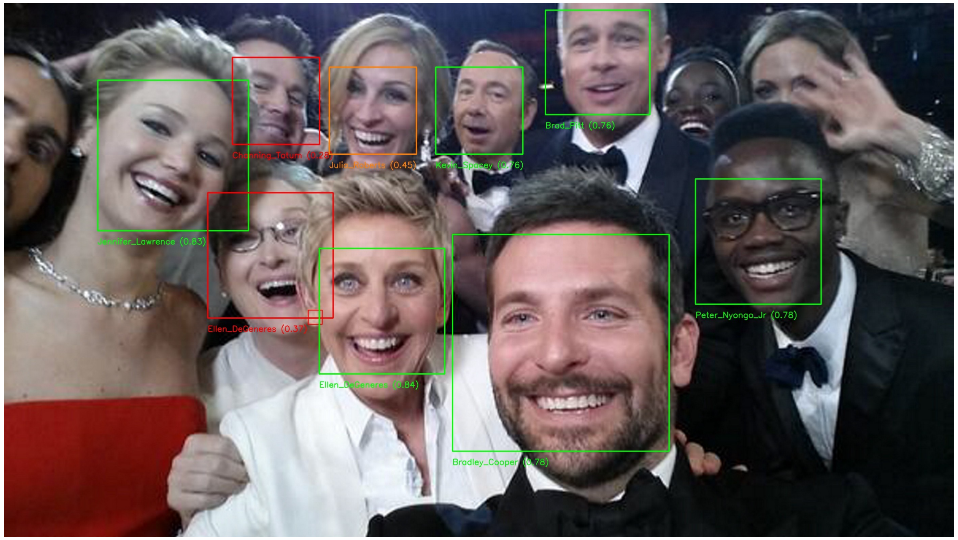
School of Aerospace, Mechanical and Mechatronic Engineering, University of Sydney

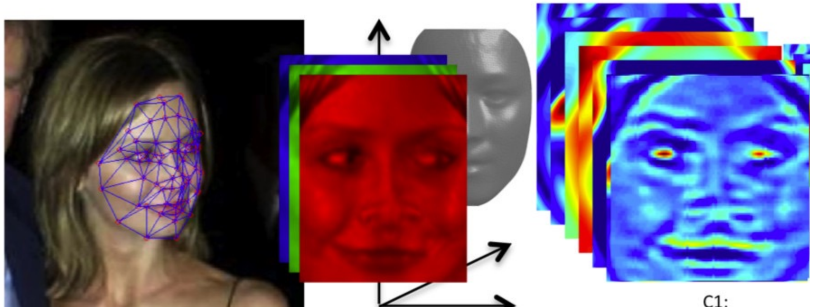
# Facial Detection and Recognition

- Face detection and recognition used in a range of applications including identity verification, security, social media



- Face Detection:
  - Detecting the presence (and position) of human faces in images
- Facial Recognition:
  - Determining that a face in an image belongs to a certain person X



*Calista_Flockhart_0002.jpg*
Detection & Localization

Frontalization:
@152X152x3

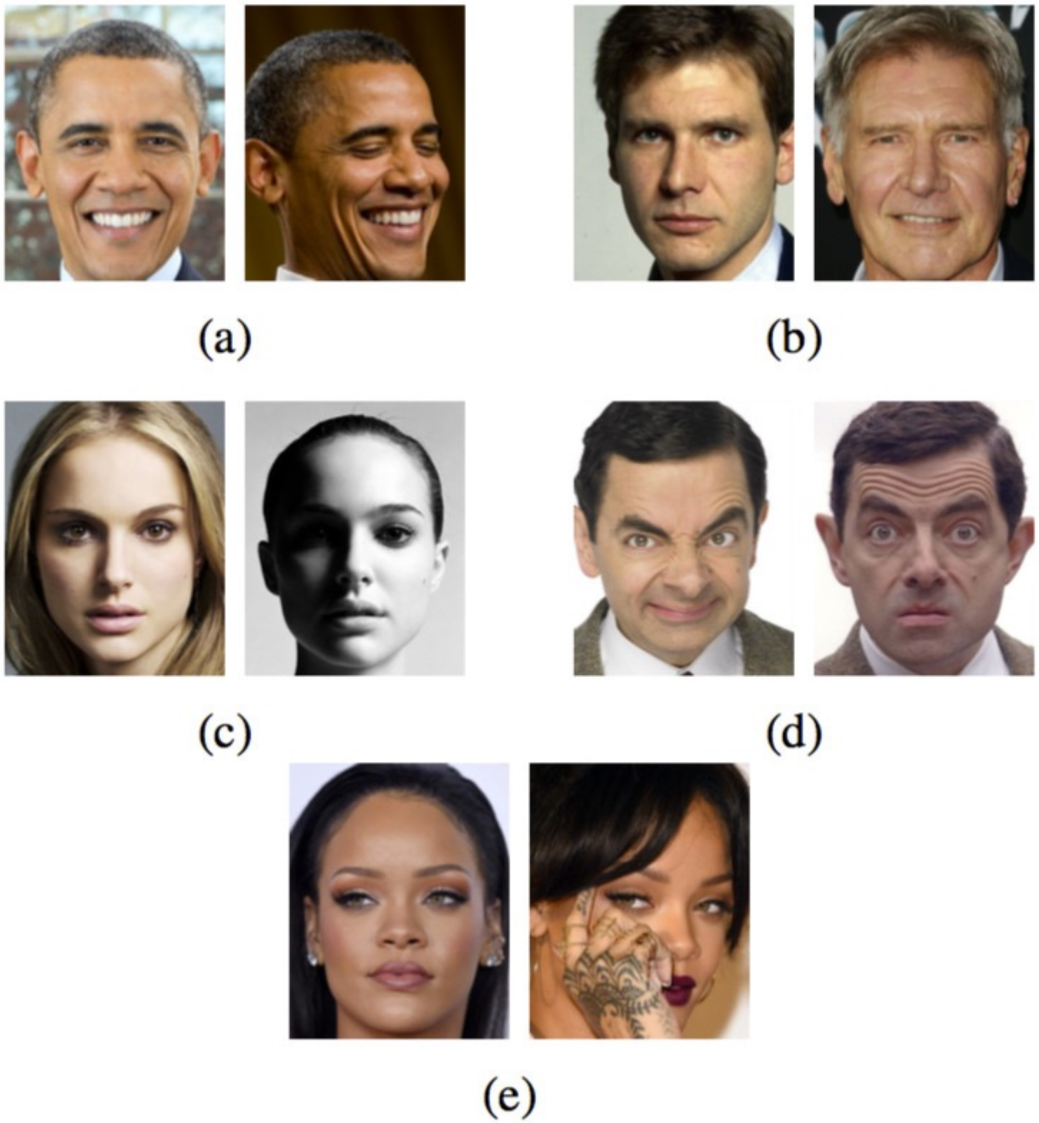
C1:
32x11x11x3
@142x142

# Applications/Examples

- Face Priority Auto-exposure/focus
- Advertising screens (e.g. Cooler Screens):
  - Detect/recognise faces for targeted advertising
- Apple Face ID:
  - Uses structured light depth sensing plus IR camera to measure face (can't be fooled by pictures, latex models etc.)
  - Built in gaze detection

# Challenges

- Variations that make face recognition challenging:
  - (a) Head pose
  - (b) Age
  - (c) Illumination changes
  - (d) Changes in expression
  - (e) Partial occlusion



D. Trigueros, L. Meng and M. Hartnett, “Face Recognition: From Traditional to Deep Learning Methods”, https://arxiv.org/abs/1811.00116

# Pipelines and Popular Algorithms

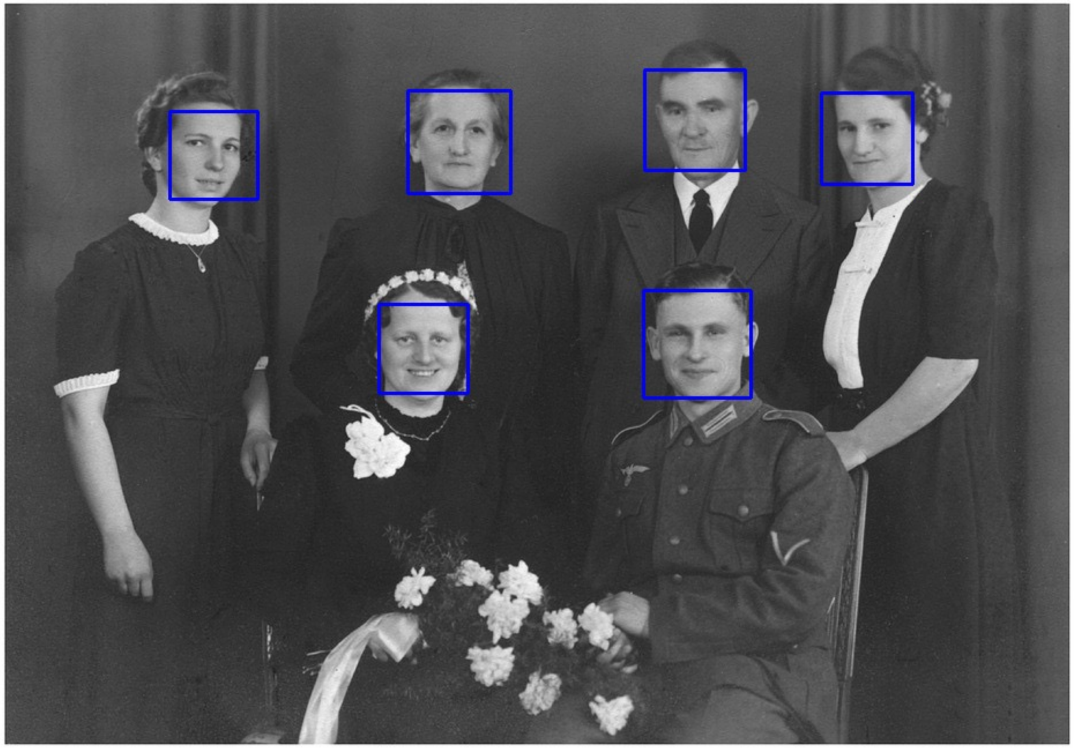
- Typical pipeline for facial recognition:
  - **Face detection**: detect instances of human faces for further analysis
  - **Face alignment**: use landmark features on the face to normalise the size and orientation
  - **Face recognition**: apply machine learning principles to classify face according to a specific person X

- Popular Algorithms:
  - Face Detection:
    - Viola-Jones face detector (Haar cascade classifiers)
  - Face Recognition:
    - Eigenfaces
    - Deepface

# Viola Jones Face Detection

- A seminal approach to face detection: takes input images and returns bounding box coordinates for detected faces

- Key properties of the approach include **robust detection** (low false positives/negatives) and **computationally fast detection** (at the cost that the algorithm is slow to train)



P. Viola and M. Jones, “Rapid Object Detection using a Boosted Cascade of Simple Features”, CVPR 2001.

# Viola Jones Face Detection

- Algorithm works by scanning a 24x24 pixel region over the image
- At each location, a detector is used to check if the region in question is a face
- The algorithm is then run at increasing scales (detector region size increased), until the region becomes the same size as the image

P. Viola and M. Jones, “Rapid Object Detection using a Boosted Cascade of Simple Features”, CVPR 2001.

# Viola Jones Face Detection

- Algorithm works by scanning a 24x24 pixel region over the image
- At each location, a detector is used to check if the region in question is a face
- The algorithm is then run at increasing scales (detector region size increased), until the region becomes the same size as the image

P. Viola and M. Jones, “Rapid Object Detection using a Boosted Cascade of Simple Features”, CVPR 2001.

# Viola Jones Face Detection

- Algorithm works by scanning a 24x24 pixel region over the image
- At each location, a detector is used to check if the region in question is a face
- The algorithm is then run at increasing scales (detector region size increased), until the region becomes the same size as the image

P. Viola and M. Jones, "Rapid Object Detection using a Boosted Cascade of Simple Features", CVPR 2001.

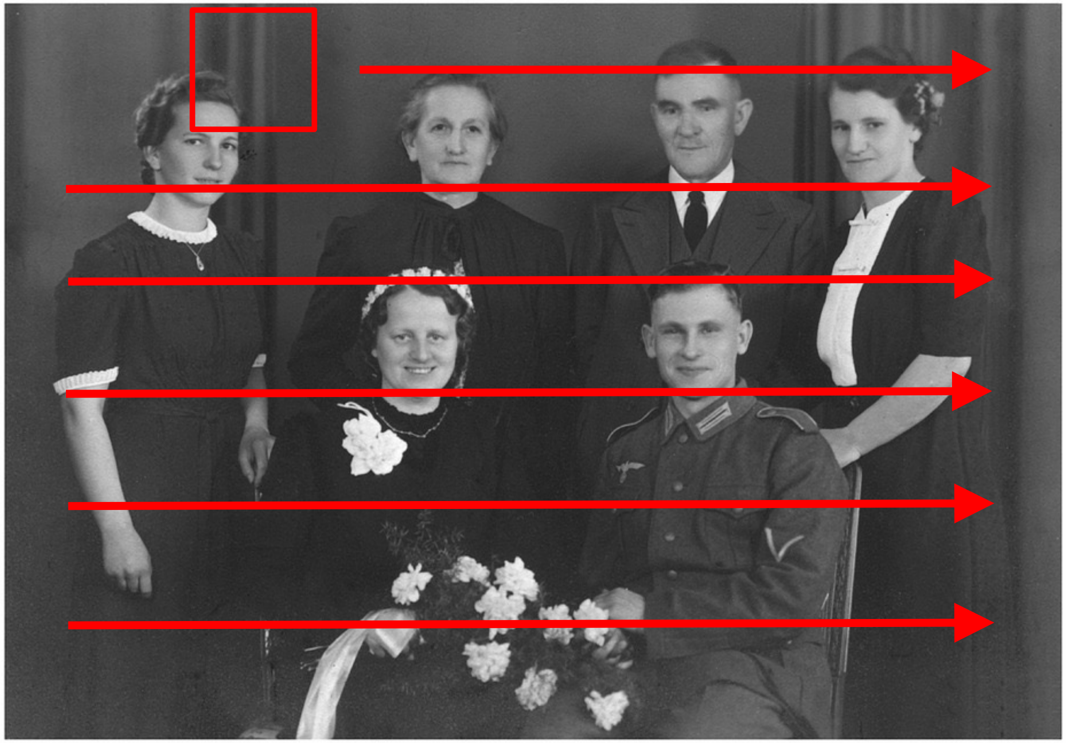
# Viola Jones Face Detection

- Algorithm works by scanning a 24x24 pixel region over the image
- At each location, a detector is used to check if the region in question is a face
- The algorithm is then run at increasing scales (detector region size increased), until the region becomes the same size as the image

P. Viola and M. Jones, “Rapid Object Detection using a Boosted Cascade of Simple Features”, CVPR 2001.

# Viola Jones Face Detection

- Algorithm works by scanning a 24x24 pixel region over the image
- At each location, a detector is used to check if the region in question is a face
- The algorithm is then run at increasing scales (detector region size increased), until the region becomes the same size as the image



P. Viola and M. Jones, "Rapid Object Detection using a Boosted Cascade of Simple Features", CVPR 2001.

# Viola Jones Face Detection

- Algorithm works by scanning a 24x24 pixel region over the image
- At each location, a detector is used to check if the region in question is a face
- The algorithm is then run at increasing scales (detector region size increased), until the region becomes the same size as the image

P. Viola and M. Jones, "Rapid Object Detection using a Boosted Cascade of Simple Features", CVPR 2001.

# Viola Jones Face Detection

- Algorithm works by scanning a 24x24 pixel region over the image
- At each location, a detector is used to check if the region in question is a face
- The algorithm is then run at increasing scales (detector region size increased), until the region becomes the same size as the image

P. Viola and M. Jones, “Rapid Object Detection using a Boosted Cascade of Simple Features”, CVPR 2001.

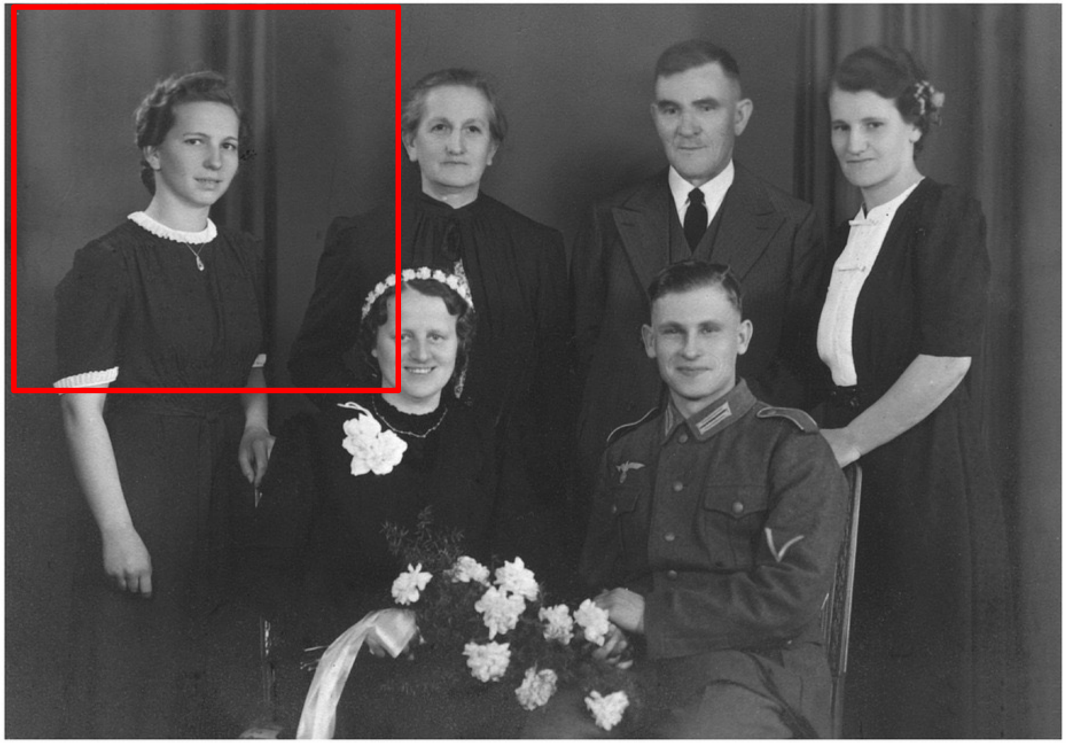
# Viola Jones Face Detection

- Algorithm works by scanning a 24x24 pixel region over the image
- At each location, a detector is used to check if the region in question is a face
- The algorithm is then run at increasing scales (detector region size increased), until the region becomes the same size as the image

P. Viola and M. Jones, "Rapid Object Detection using a Boosted Cascade of Simple Features", CVPR 2001.

# Viola Jones Face Detection

- Algorithm works by scanning a 24x24 pixel region over the image
- At each location, a detector is used to check if the region in question is a face
- The algorithm is then run at increasing scales (detector region size increased), until the region becomes the same size as the image

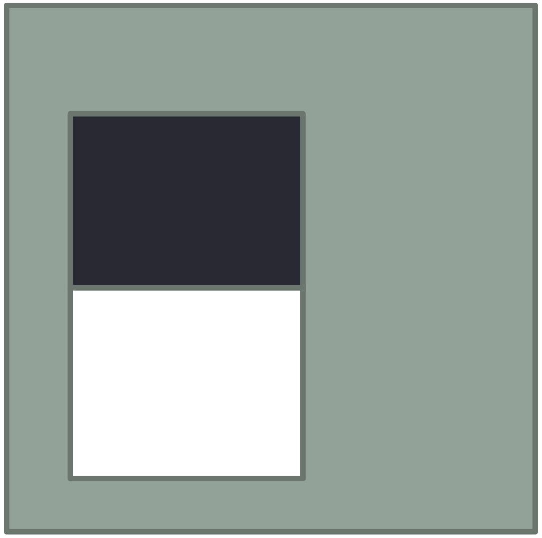


P. Viola and M. Jones, "Rapid Object Detection using a Boosted Cascade of Simple Features", CVPR 2001.

# Haar-like Features

- Viola-Jones face detection uses **Haar-like features** to build weak decision stumps that can detect a face
- A Haar-like feature is defined by a set of bounding box coordinates for a set of black or white regions that sit within a kernel that can be run over an image via filtering

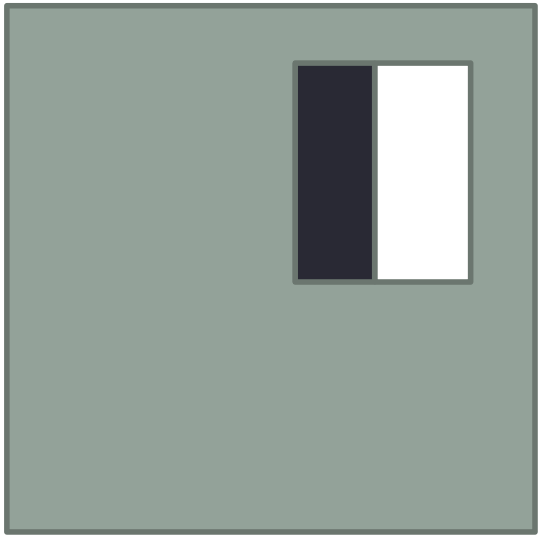
- The response of a feature is the sum of all pixel intensities in white regions minus sum of pixel intensities in the black regions



Kernel

# Haar-like Features

- Viola-Jones face detection uses **Haar-like features** to build weak decision stumps that can detect a face
- A Haar-like feature is defined by a set of bounding box coordinates for a set of black or white regions that sit within a kernel that can be run over an image via filtering

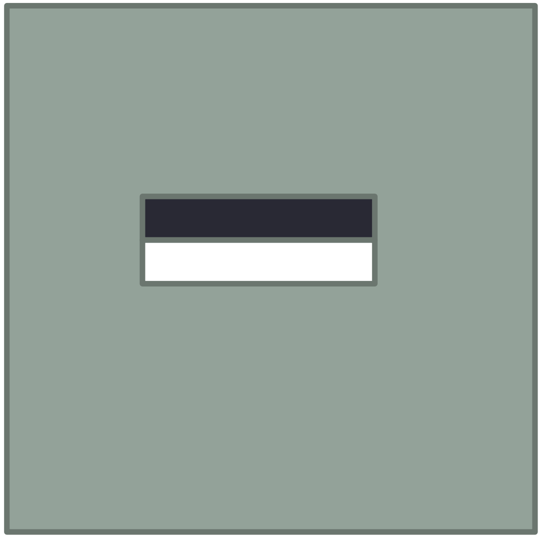


Kernel

- The response of a feature is the sum of all pixel intensities in white regions minus sum of pixel intensities in the black regions

# Haar-like Features

- Viola-Jones face detection uses **Haar-like features** to build weak decision stumps that can detect a face
- A Haar-like feature is defined by a set of bounding box coordinates for a set of black or white regions that sit within a kernel that can be run over an image via filtering
- The response of a feature is the sum of all pixel intensities in white regions minus sum of pixel intensities in the black regions



Kernel

# Haar-like Features

- Viola-Jones face detection uses **Haar-like features** to build weak decision stumps that can detect a face
- A Haar-like feature is defined by a set of bounding box coordinates for a set of black or white regions that sit within a kernel that can be run over an image via filtering

Kernel

- The response of a feature is the sum of all pixel intensities in white regions minus sum of pixel intensities in the black regions

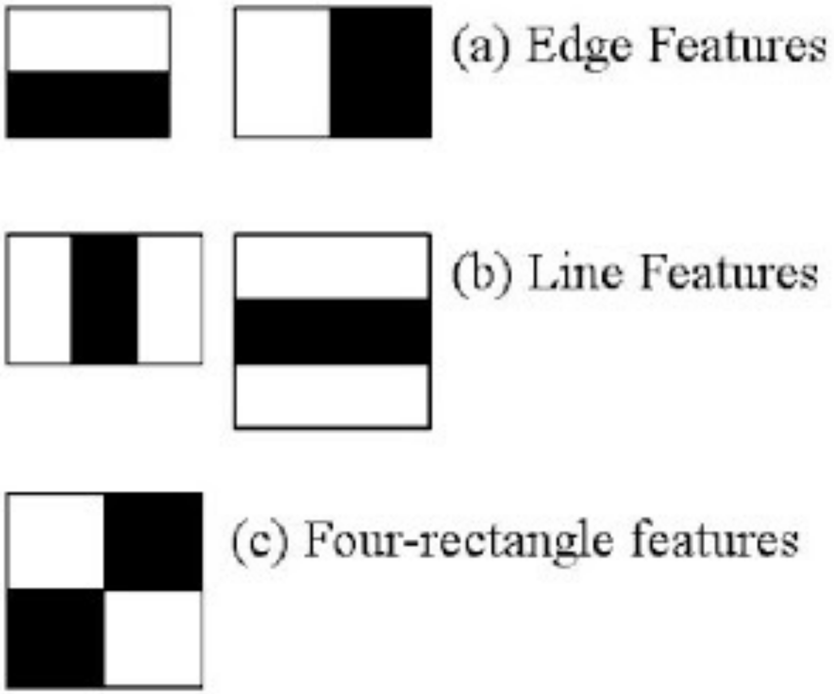
# Haar-like Features

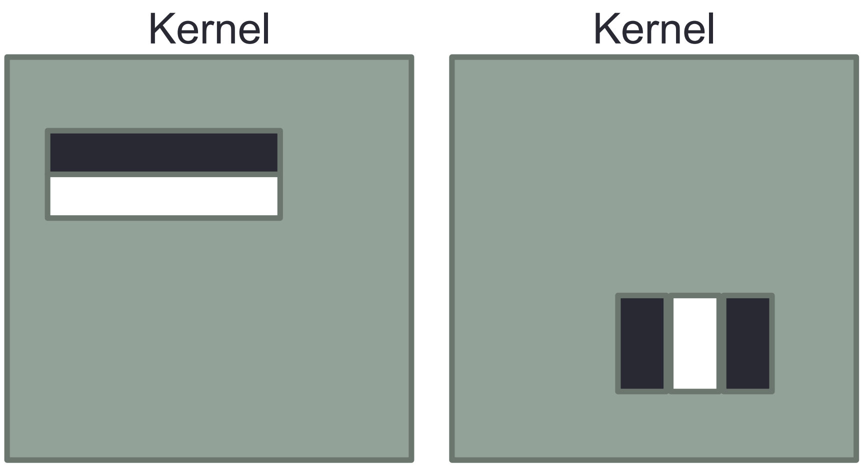
- Viola-Jones face detection uses **Haar-like features** to build weak decision stumps that can detect a face
- A Haar-like feature is defined by a set of bounding box coordinates for a set of black or white regions that sit within a kernel that can be run over an image via filtering
- The response of a feature is the sum of all pixel intensities in white regions minus sum of pixel intensities in the black regions

$$R = \sum I_1 - \sum I_2$$

# Haar-like Features

- Haar-like features can be defined in various arrangements: 3 fundamental types used in Viola-Jones
- For face detection, features “respond” to different aspects of shading present in faces (i.e. dark eyes and light brow, brightness on top of nose, shading either side)



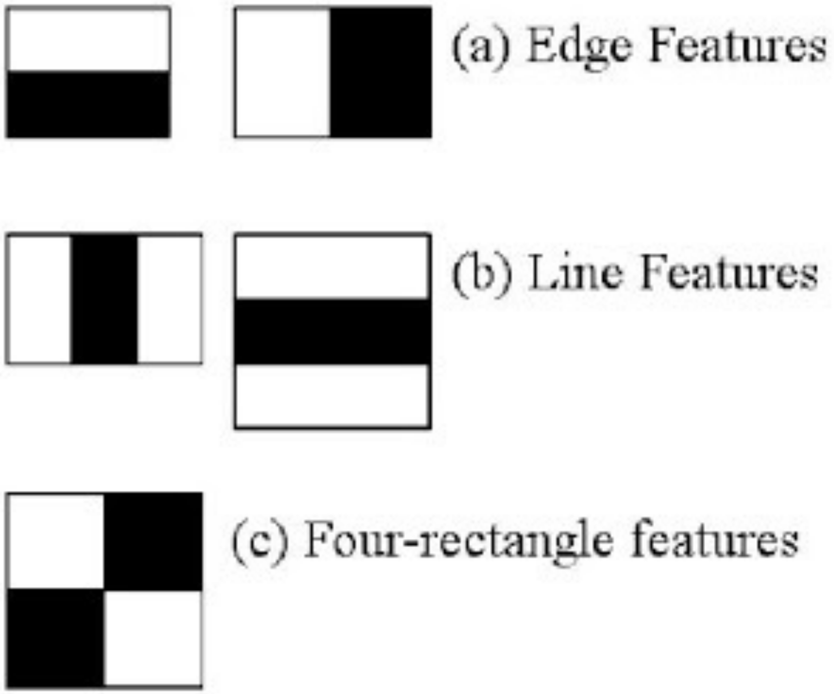




Example Haar-like features

# Haar-like Features

- Haar-like features can be defined in various arrangements: 3 fundamental types used in Viola-Jones

- For face detection, features “respond” to different aspects of shading present in faces (i.e. dark eyes and light brow, brightness on top of nose, shading either side)

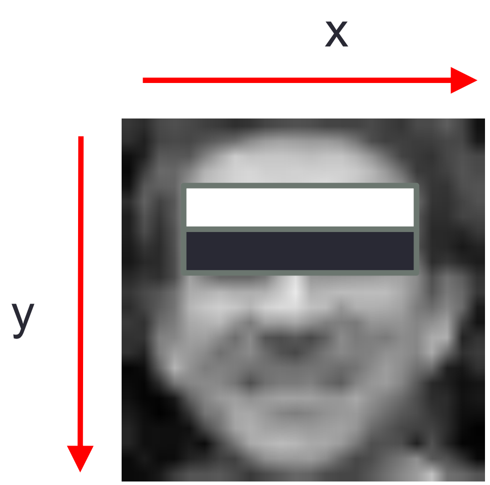



# Making Haar features fast: Integral Images

- Calculating feature response by summing intensity data in image regions can be computationally expensive for high-resolution images
- Integral images are used to speed up response calculation: integral image is pre-computed once for each image (grayscale)
- Calculating the sum of intensities in a given bounding box can then be computed in constant time from 4 values
- This significantly speeds up computational time for real-time detection and is part of the motivation for the use of Haar-like features

# Making Haar features fast: Integral Images

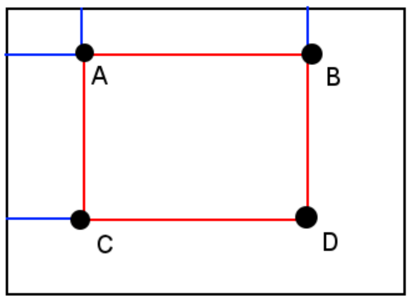
- Calculating feature response by summing intensity data in image regions can be computationally expensive for high-resolution images
- Integral images are used to speed up response calculation: integral image is pre-computed once for each image (grayscale)
- Calculating the sum of intensities in a given bounding box can then be computed in constant time from 4 values
- This significantly speeds up computational time for real-time detection and is part of the motivation for the use of Haar-like features



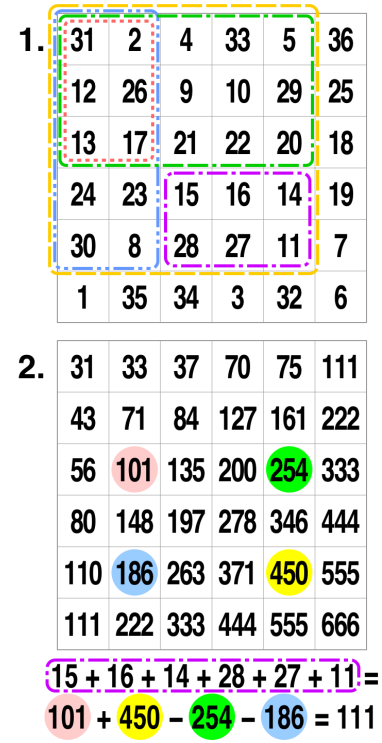

$$I(x,y) = \sum_{\substack{x' \le x \\ y' \le y}} i(x', y')$$

# Making Haar features fast: Integral Images

- Calculating feature response by summing intensity data in image regions can be computationally expensive for high-resolution images
- Integral images are used to speed up response calculation: integral image is pre-computed once for each image (grayscale)
- Calculating the sum of intensities in a given bounding box can then be computed in constant time from 4 values
- This significantly speeds up computational time for real-time detection and is part of the motivation for the use of Haar-like features



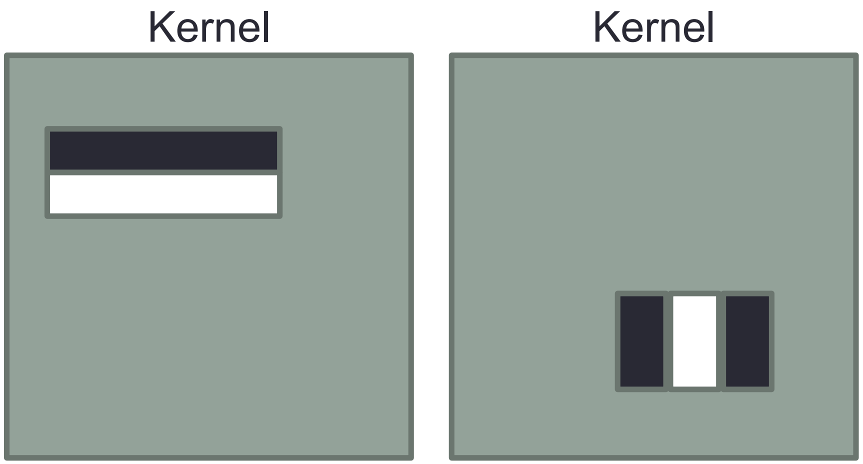

Sum = D - B - C + A




$$\sum_{\substack{x_0 < x \leq x_1 \\ y_0 < y \leq y_1}} i(x,y) = I(D) + I(A) - I(B) - I(C)$$

# Boosting

- For a 24x24 image, and four different Haar types, there are approx. 160k different potential feature types
- Even the best Haar features form a “weak classifier” of a face (i.e. detection performance just better than random) when taken alone

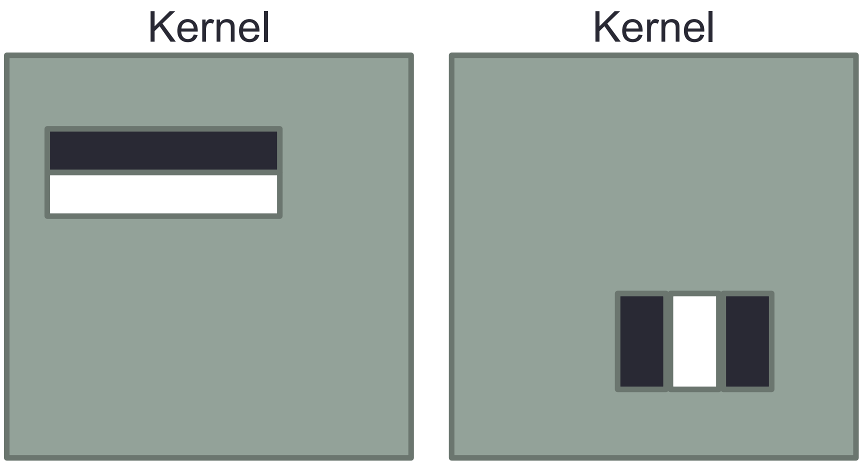



Example Haar-like features

# Boosting

- For a 24x24 image, and four different Haar types, there are approx. 160k different potential feature types
- Even the best Haar features form a "weak classifier" of a face (i.e. detection performance just better than random) when taken alone

- **Boosting** is a classification technique that uses an ensemble of weak classifiers to create a stronger one



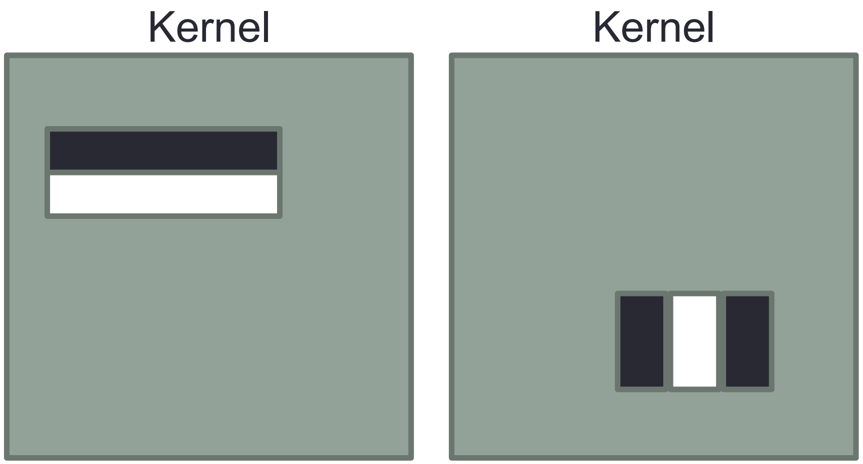
Example Haar-like features

$$h(\mathbf{x}) = \operatorname{sgn}\left(\sum_{j=1}^{M} \alpha_j h_j(\mathbf{x})\right)$$

# Boosting

- For a 24x24 image, and four different Haar types, there are approx. 160k different potential feature types
- Even the best Haar features form a "weak classifier" of a face (i.e. detection performance just better than random) when taken alone

- **Boosting** is a classification technique that uses an ensemble of weak classifiers to create a stronger one




Example Haar-like features

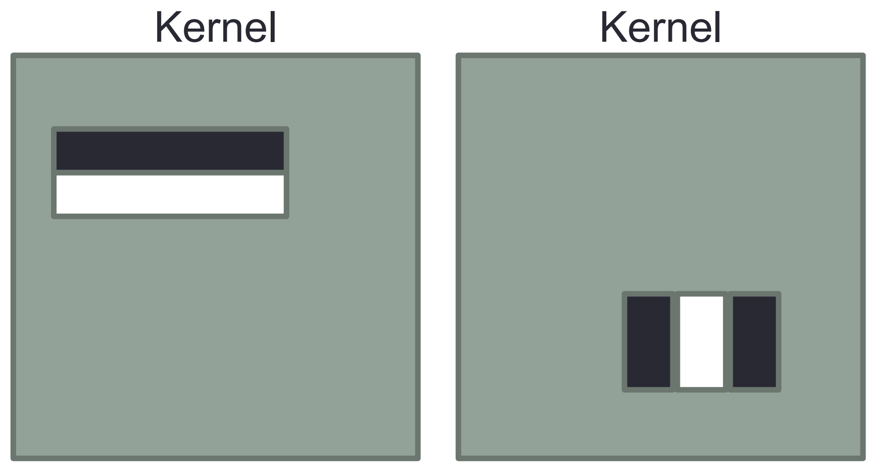
Boosted output

Input data

$$h(\mathbf{x}) = \operatorname{sgn}\left(\sum_{j=1}^{M} \alpha_j h_j(\mathbf{x})\right)$$

# Boosting

- For a 24x24 image, and four different Haar types, there are approx. 160k different potential feature types
- Even the best Haar features form a "weak classifier" of a face (i.e. detection performance just better than random) when taken alone



Example Haar-like features

- **Boosting** is a classification technique that uses an ensemble of weak classifiers to create a stronger one

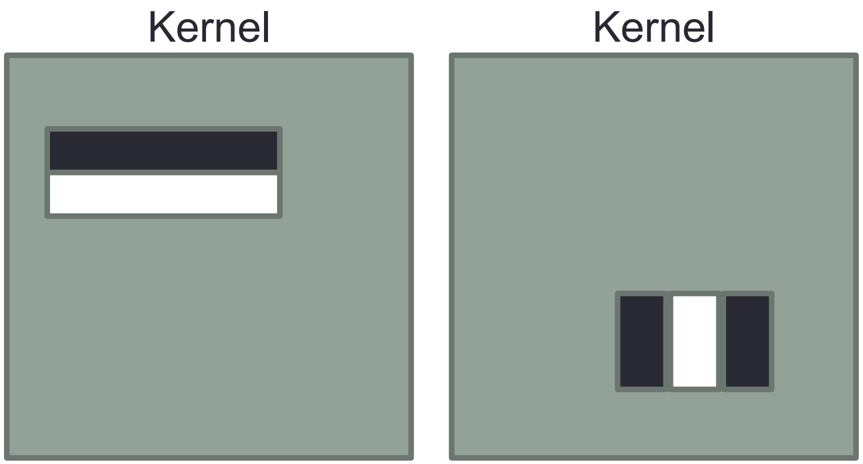
$$h(\mathbf{x}) = \operatorname{sgn}\left(\sum_{j=1}^{M} \alpha_j h_j(\mathbf{x})\right)$$

Weight for j

Weak classifier j

# Boosting

- For a 24x24 image, and four different Haar types, there are approx. 160k different potential feature types
- Even the best Haar features form a "weak classifier" of a face (i.e. detection performance just better than random) when taken alone




Example Haar-like features

- **Boosting** is a classification technique that uses an ensemble of weak classifiers to create a stronger one
- Viola-Jones face detection uses the Adaboost algorithm, with Haar features/associated thresholds acting as weak classifiers, to create a robust detector

$$h(\mathbf{x}) = \operatorname{sgn}\left(\sum_{j=1}^{M} \alpha_j h_j(\mathbf{x})\right)$$

# Adaboost and VJ Face Detection

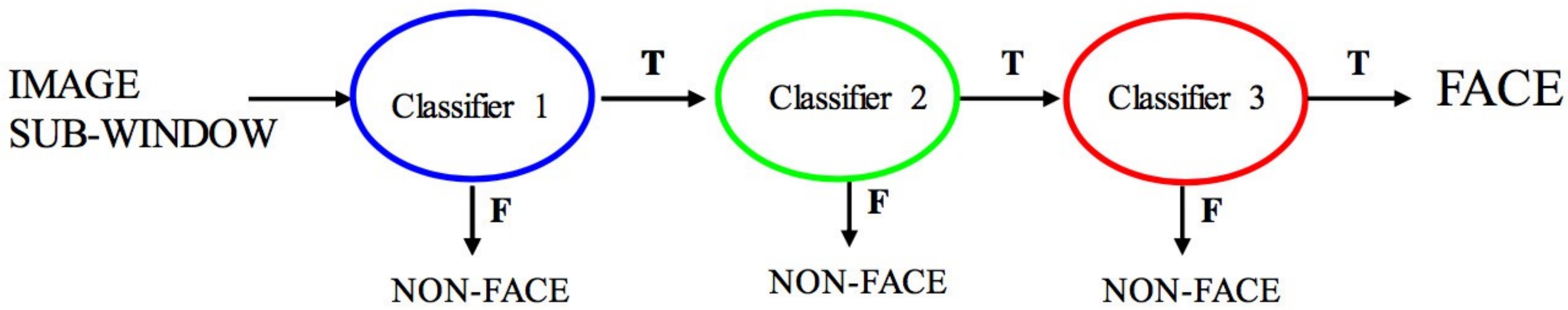
- The final classifier is trained for a set of example image patches of real faces, and other non-face objects:

1. Each training example is given a weight $w_i = 1/N$
2. For each Haar feature j:
   1. Calculate the feature response on each training example, then calculate a threshold value that optimally splits face/no-face based on the weighted error for each training example
   2. Assign a weight $\alpha_j$ which is inversely proportionate to the average error for this feature
   3. Reduce weights $w_i$ for correctly classified examples, and re-normalise weights (sum to 1)

- Each subsequent feature uses a threshold that picks up the mistakes made by the previous feature, so the combined response becomes complimentary

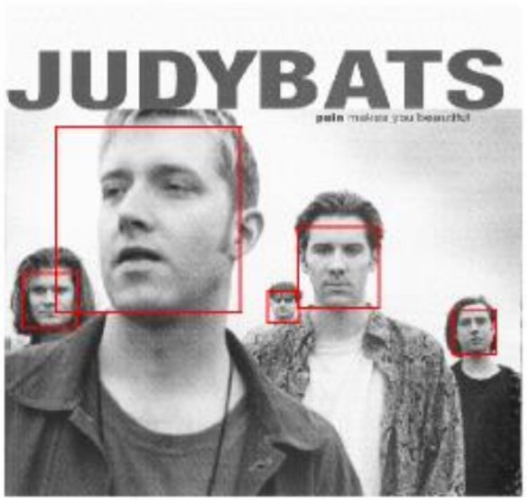
# Cascade Classifiers

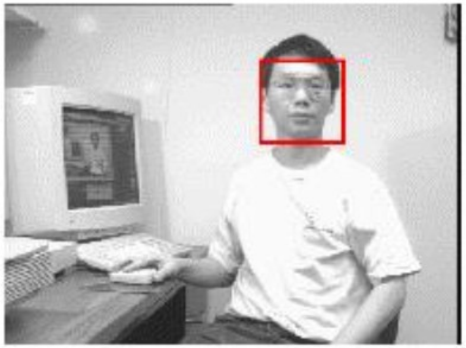
- For real face detection, most analysed sub-windows are not faces: to speed-up detection, VJ face detection uses an attentional cascade:

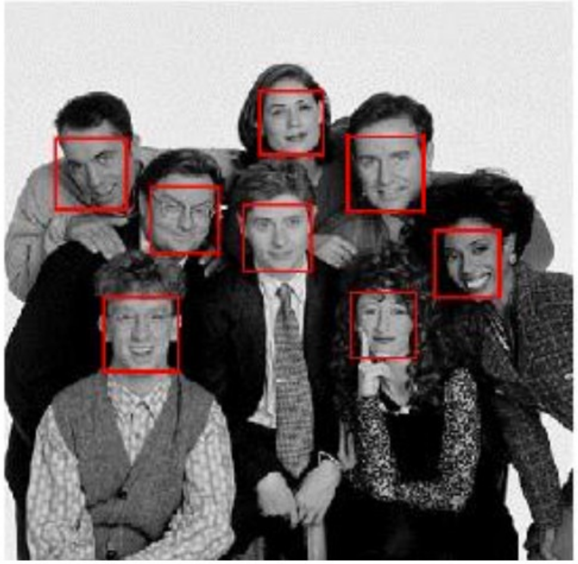


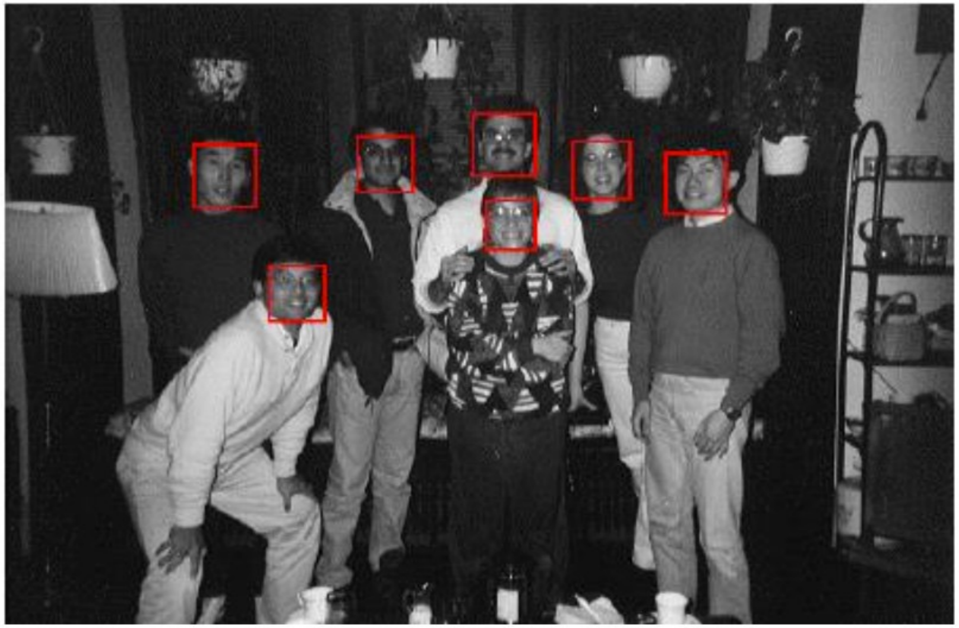

- The highest weighted features are applied in an ensemble that has a low false negative (but high false positive)
- Windows that pass this classifier are sent to subsequent stages: faces are detected when the image is labelled as a face by all stages in the cascade
- The original VJ uses 38 stages with over 6000 features to achieve almost zero false negatives and 93% detection rate

# Examples

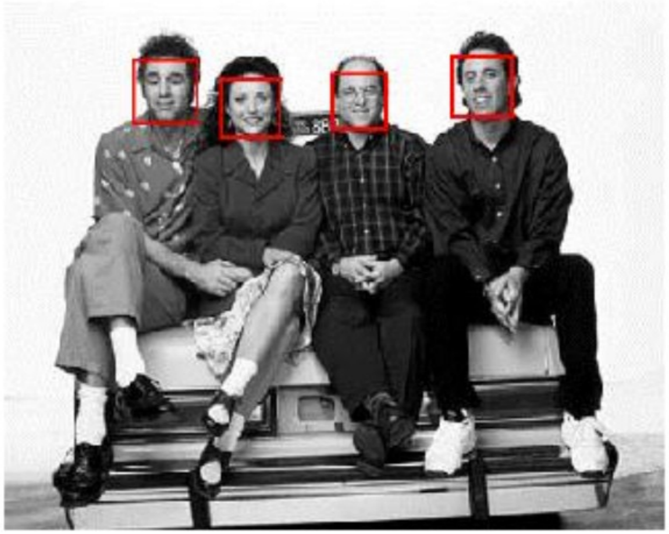


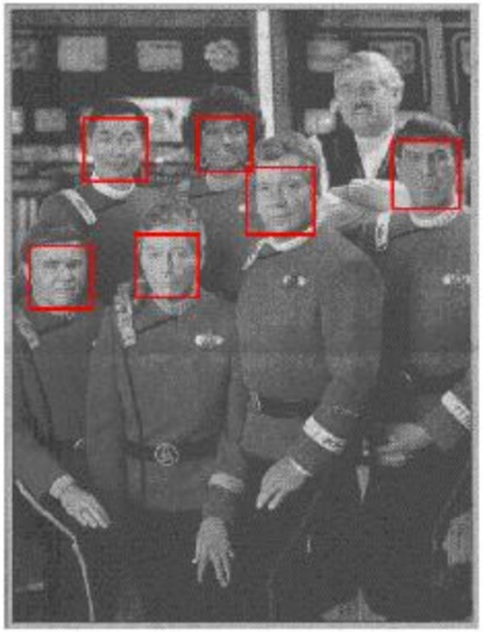



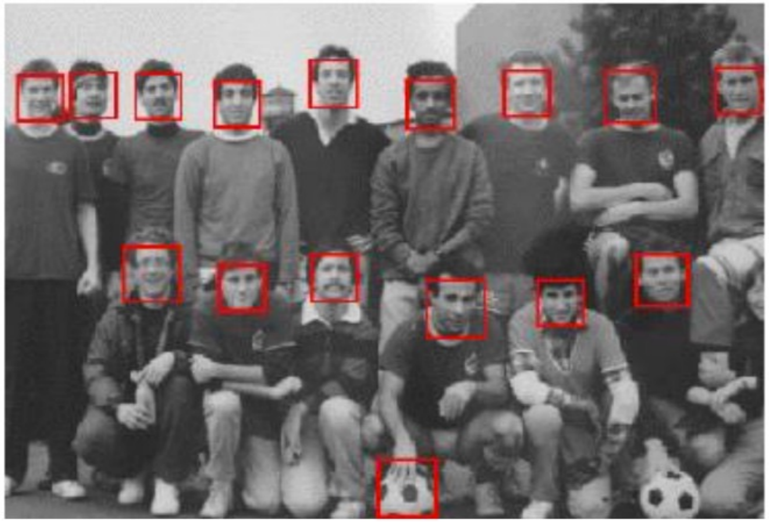


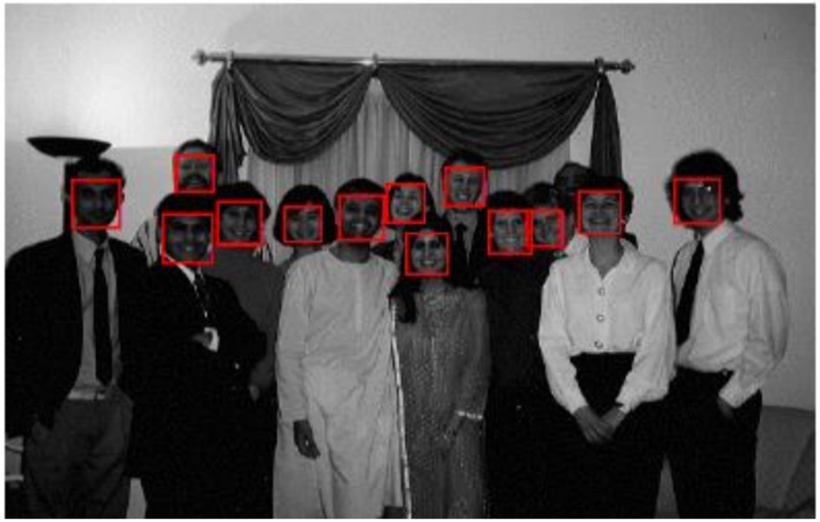


# Examples



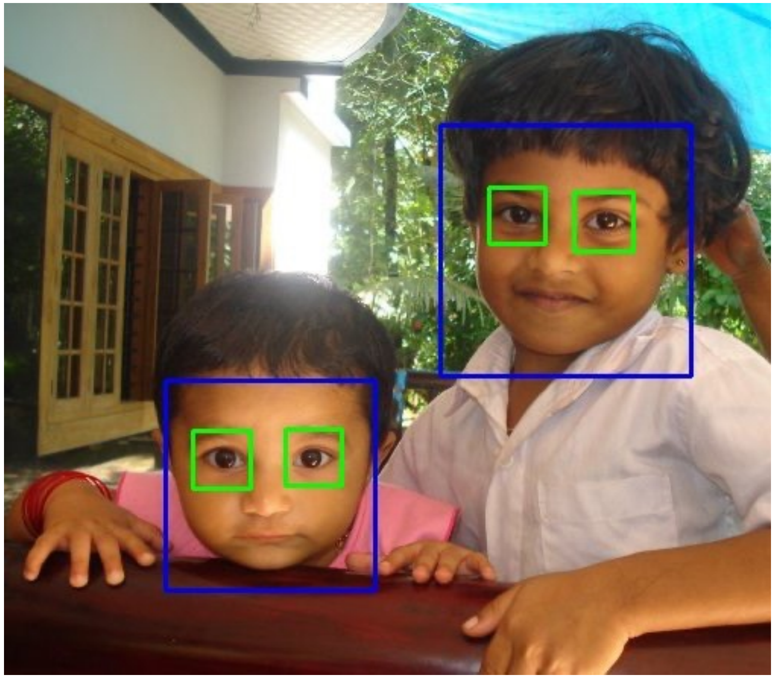


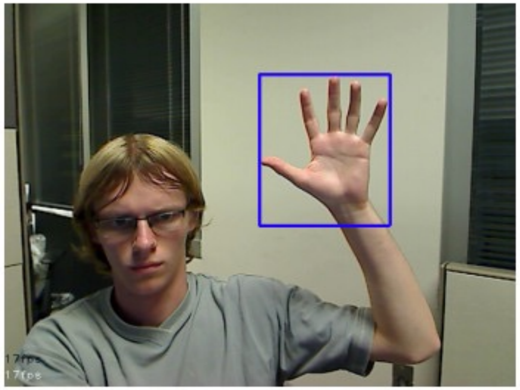


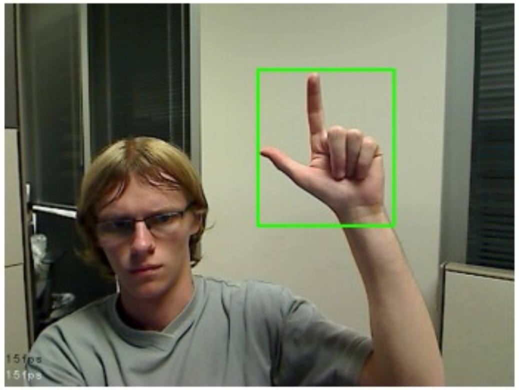


# Haar Cascade Classifiers

- Although popularised for face detection, the same principles can be applied to many different applications including eye detection, hand and gesture detection etc.
- Implementations of Haar Cascade Classifiers:
  - Python (OpenCV): https://docs.opencv.org/2.4/doc/user_guide/ug_traincascade.html
  - MATLAB: https://www.mathworks.com/help/vision/ref/vision.cascadeobjectdetector-system-object.html







# Haar Cascade Classifier Demo

- Live online Notebook via Google Colab (*required Google account to run):
- https://colab.research.google.com/drive/10zCvW_7p9ReIfi5p7YfOVBWN_pm46x1j?usp=sharing

- Download the Python Code and run on your own device ("face_track_demo.zip" under Modules/Week 11)
  - Requires you first install python (https://www.python.org/)
  - Then install modules for OpenCV:
    - pip install opencv-python
  - You can run the demo from the command line/terminal/shell using "python run_face_track.py"

# Further Reading and Next Week

- References:
  - R. Szeliski, “Computer Vision: Algorithms and Applications”, Springer, 2010 (Chapter 14)
  - P. Viola and M. Jones, "Rapid object detection using a boosted cascade of simple features", CVPR, 2001.
  - Face detection using Haar cascades, OpenCV Notes, https://docs.opencv.org/3.0-beta/doc/py_tutorials/py_objdetect/py_face_detection/py_face_detection.html