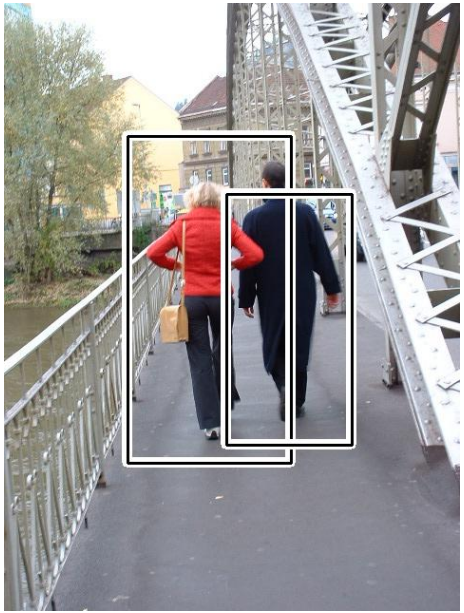
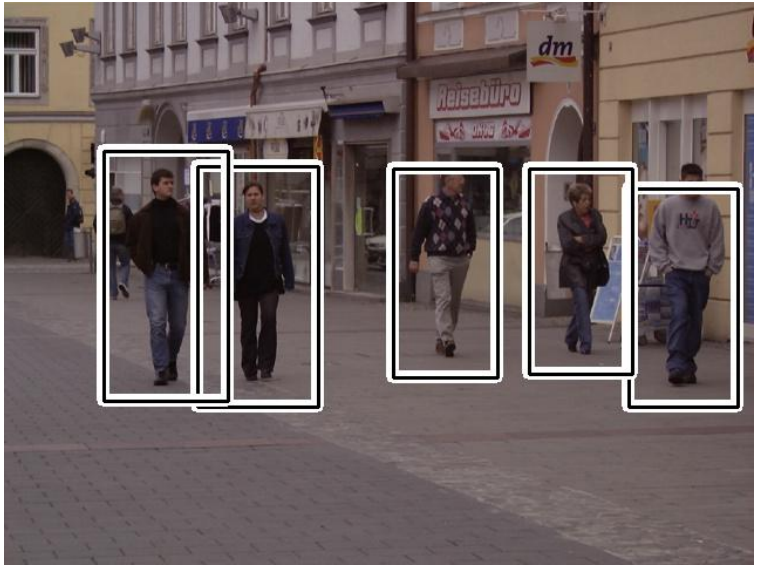
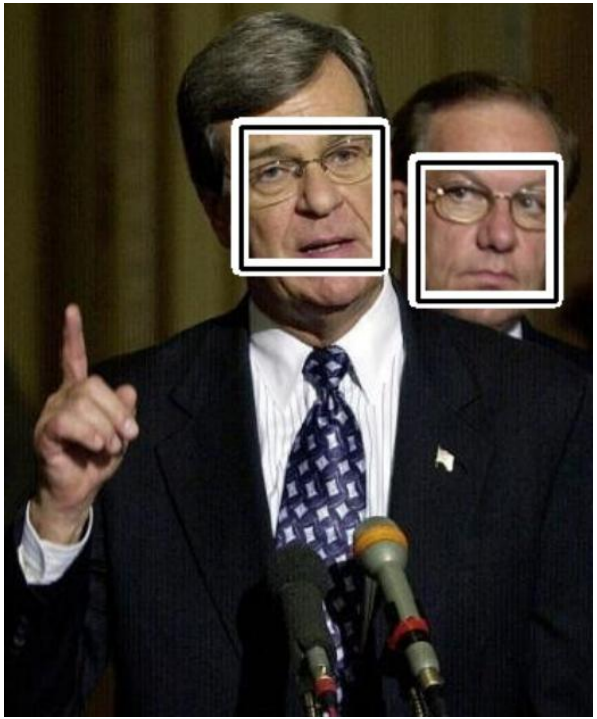
Object Detection

What is the output of object detection methods?

- **Position of the object of interest**
- **Scale/size/bounding box of the object of interest**







Object Detection

- Haar

Cascade classifier in OpenCV

Paul Viola and Michael Jones
Rapid Object Detection using a Boosted Cascade of Simple Features

- HOG
- LBP

Traditional Approaches (sliding window)

- SIFT
- SURF

KeyPoints Approaches

- CNNs
- R-CNNs/YOLO/SSD

Deep Learning Approaches

Sliding Window

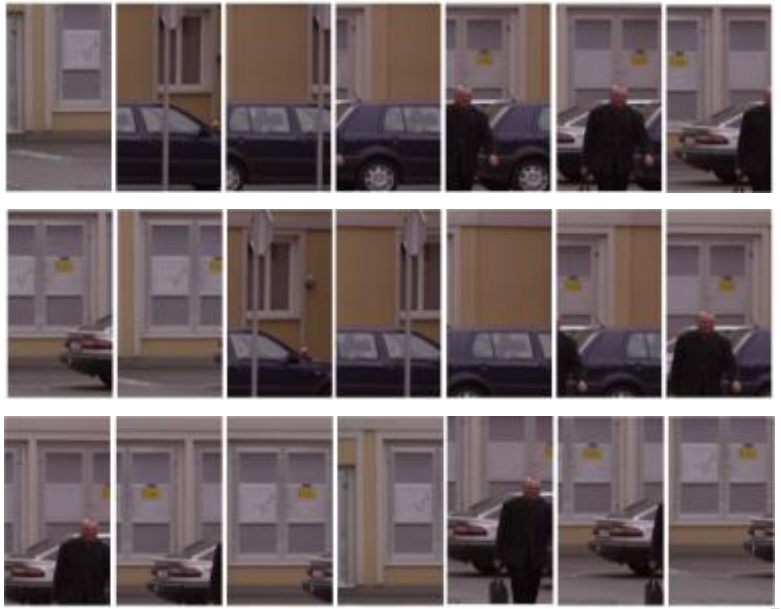
- In general, the sliding window technique represents the popular and successful approach for object detection. The main idea of this approach is that the input image is scanned by a rectangular window at multiple scales. The result of the scanning process is a large number of various sub-windows. A vector of features is extracted from each sub-window. The vector is then used as an input for the classifier (e.g. SVM classifier).

- During the classification process, some sub-windows are marked as the objects. Using the sliding window approach, the multiple positive detections may appear, especially around the objects of interest

Sliding Window

- These detections are merged to the final bounding box that represents the resulting detection.
- The classifier that determines each sub-window is trained over the training set that consists of positive and negative images.
- The key point is to find what values (features) should be used to effectively encode the image inside the sliding window.

Sliding Window



Sliding Window

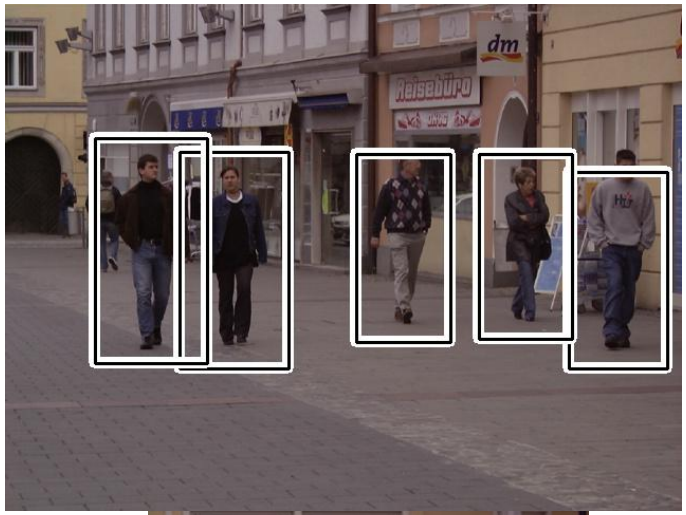
Sliding Window

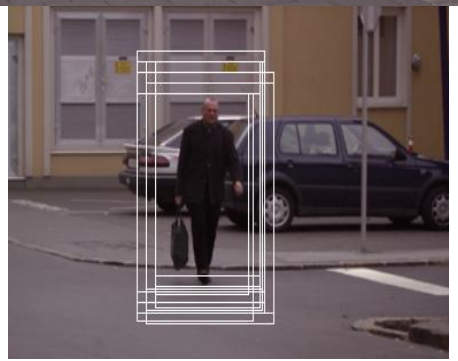
Feature Vector

(properties of object)

Trainable Classifier

(SVM, ANNs, …)

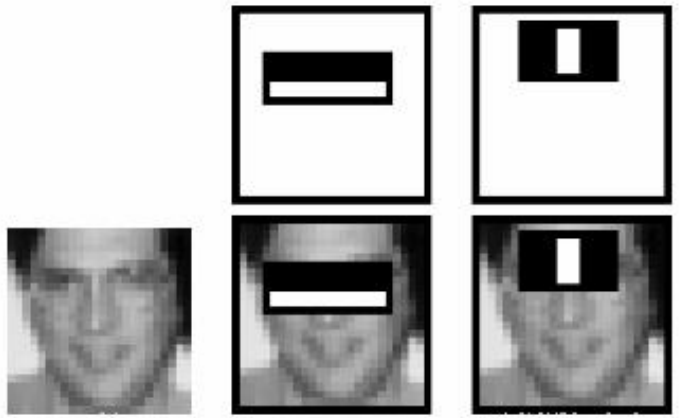




Face Detection

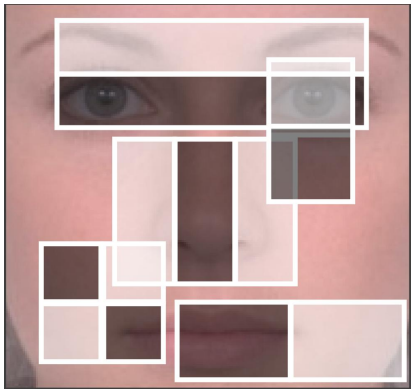
Face Detection (Viola-Jones Detector):

1. **Rectangle features (Haar features):**
- faces have similar properties
- eye regions are darker than the upper-cheeks
- the nose bridge region is brighter than the eyes
- thousands of possible features variations



2. **Integral Image**
- speed the computational process

3. **Cascade Classifier + AdaBoost**
- in an image, most of the image is non-face region
- reject the non-face region as soon as possible



David Gerónimo: Haar-like Features and Integral Image Representation, Master in Computer Vision and Artificial Intelligence, 18th December 2009
https://docs.opencv.org/4.5.5/db/d28/tutorial_cascade_classifier.html

Face Detection

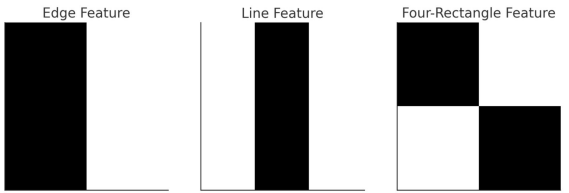
1. **Rectangle Features (Haar features):**

- Haar features are image descriptors used in object detection, particularly in the Viola-Jones detection framework. They work by comparing pixel intensity differences between adjacent rectangular regions within an image.

$$\textbf{Feature Value} = \sum(\text{pixels in black}) - \sum(\text{pixels in white})$$

- These features are designed to capture patterns such as edges, lines, and textures, which are essential for recognizing objects like faces. These features are placed at various positions and scales within a detection window (typically 24×24 pixels).

- The three primary types of Haar-like features are:
  - Edge features: Detect vertical or horizontal transitions, e.g. between forehead and eyes - A vertical feature with a dark region for the eyes and a bright region for the forehead.
  - Line features: Identify structures like the bridge of the nose. A feature with three bands (e.g. bright, dark, bright). Example: the brightness of the nose appears different from the surrounding areas.
  - Four-rectangle features: Capture finer patterns like cheekbones or jawlines.



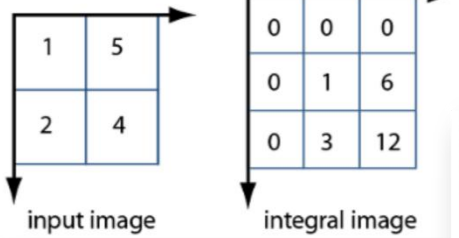
Face Detection

2. Integral Image

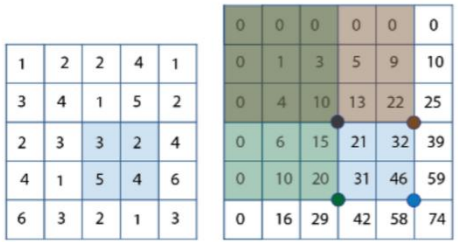
Viola-Jones algorithm needs to evaluate approximately 160,000+ Haar features efficiently. An integral image, also known as a summed-area table, is a technique used in computer vision to quickly compute the sum of pixel values within a rectangular region. It is widely used in Haar cascade classifiers for real-time face detection.

In an *integral image*, every pixel is the summation of the pixels above and to the left of it.

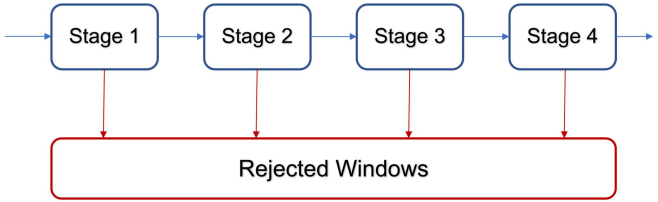
To illustrate, the following shows and image and its corresponding integral image. The integral image is padded to the left and the top to allow for the calculation. The pixel value at (2, 1) in the original image becomes the pixel value (3, 2) in the integral image after adding the pixel value above it (2+1) and to the left (3+0). Similarly, the pixel at (2, 2) in the original image with the value 4 becomes the pixel at (3, 3) in the integral image with the value 12 after adding the pixel value above it (4+5) and adding the pixel to the left of it (9+3).



Using an integral image, you can rapidly calculate summations over image subregions. Integral images facilitate summation of pixels and can be performed in constant time, regardless of the neighborhood size. The following figure illustrates the summation of a subregion of an image, you can use the corresponding region of its integral image. For example, in the input image below, the summation of the shaded region becomes a simple calculation using four reference values of the rectangular region in the corresponding integral image. The calculation becomes, 46 − 22 − 20 + 10 = 14. The calculation subtracts the regions above and to the left of the shaded region. The area of overlap is added back to compensate for the double subtraction.



https://www.mathworks.com/help/images/integral-image.html



3. Cascade Classifier + AdaBoost

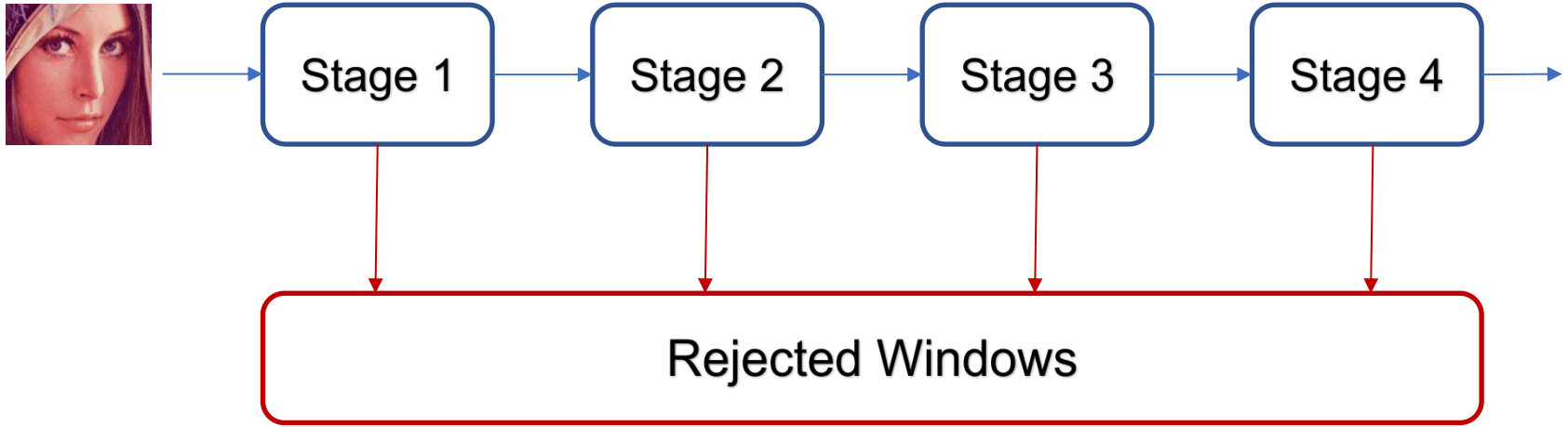
Many Haar features (approximately 160,000+) are generated for various scales and positions within the image (e.g., 24x24 windows). Since most features are irrelevant, a selection process is necessary, this is done using **Cascade Classifier + AdaBoost**.

The cascade classifier in the Viola-Jones framework is a hierarchical structure of classifiers designed to efficiently detect objects (e.g., faces) in an image by progressively eliminating non-object regions. It uses Haar features to evaluate patterns in the image and applies a series of increasingly complex classifiers to focus computational resources only on promising regions.

- The cascade consists of multiple stages. At each stage, regions that are unlikely to contain the object (e.g., faces) are quickly rejected. Only regions passing all stages are classified as containing the object.
- Each stage is trained using AdaBoost, which selects the most relevant Haar features and combines them into a strong classifier. This ensures that each stage focuses on distinguishing between object and non-object regions effectively.
- Early stages are simple and reject most non-object regions quickly - early stages detect simple patterns.
- Later stages are more complex and focus only on regions that pass earlier stages, reducing computational cost.
- The cascade structure ensures that most irrelevant regions are discarded early, making it computationally efficient.

Face Detection

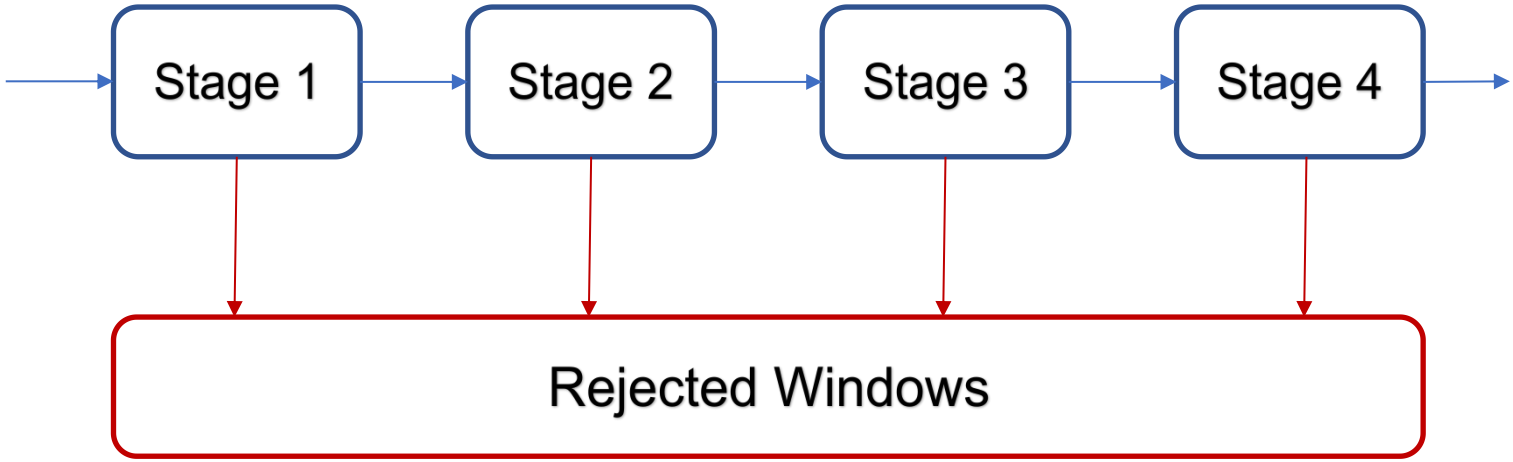
Cascade of Classifier



https://docs.opencv.org/4.5.5/db/d28/tutorial_cascade_classifier.html

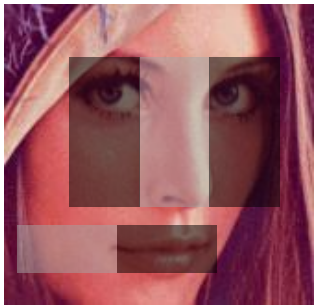
Face Detection

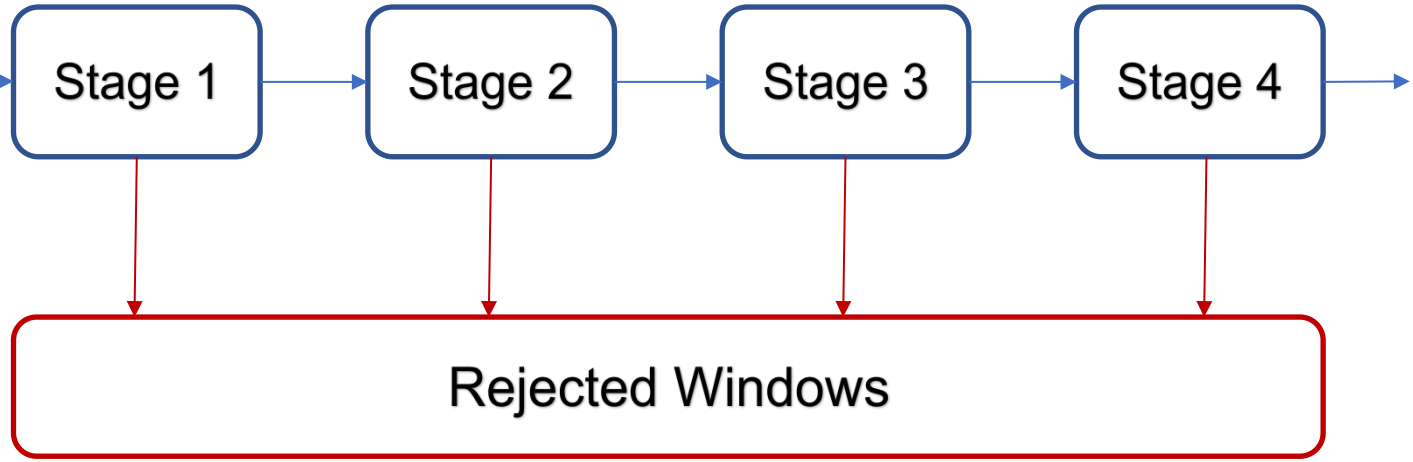
Cascade of Classifier



Face Detection

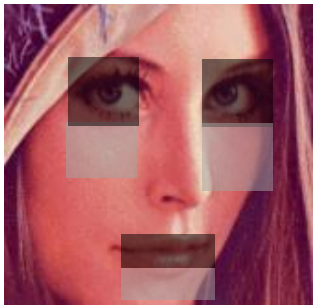
Cascade of Classifier

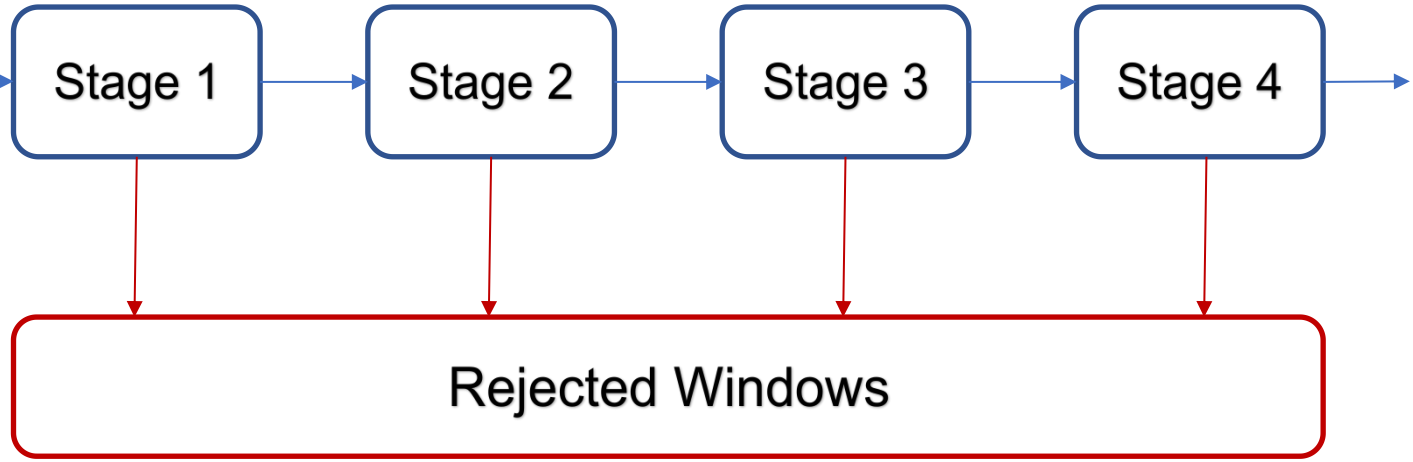




Face Detection

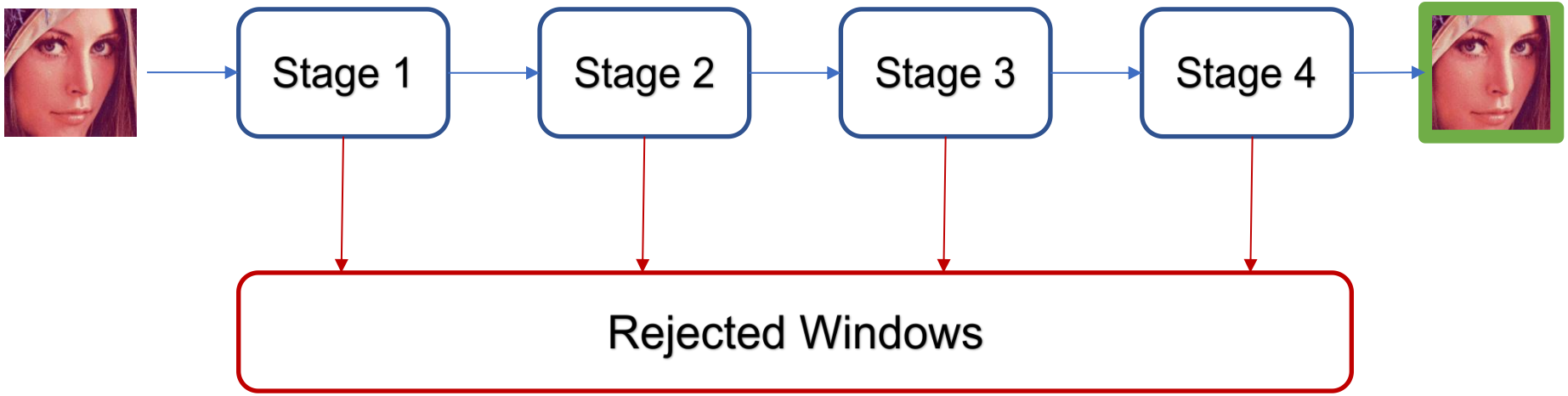
Cascade of Classifier





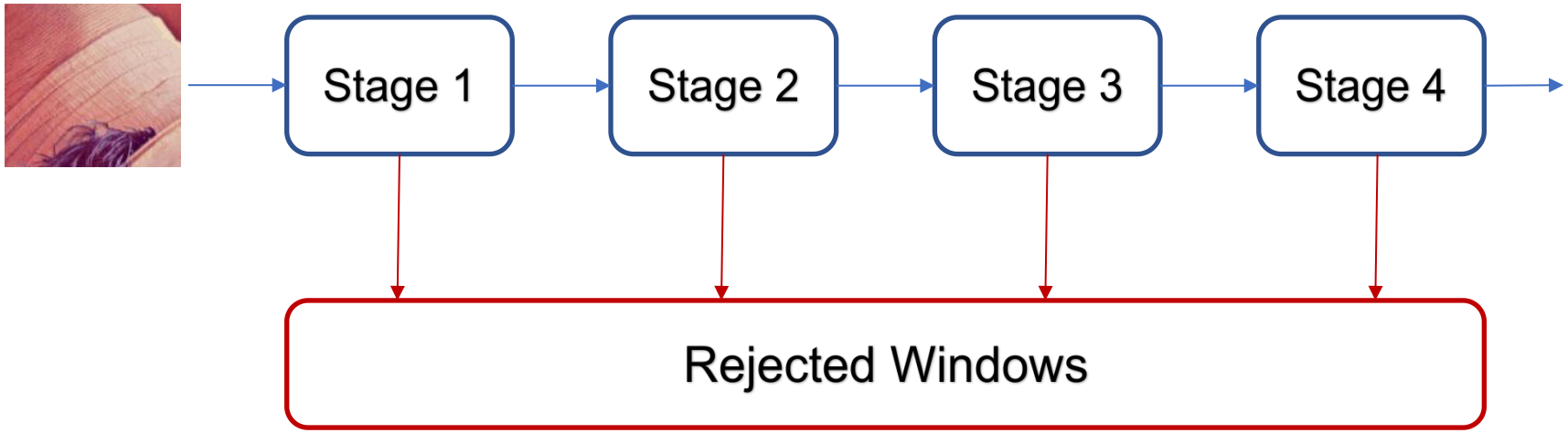
Face Detection

Cascade of Classifier



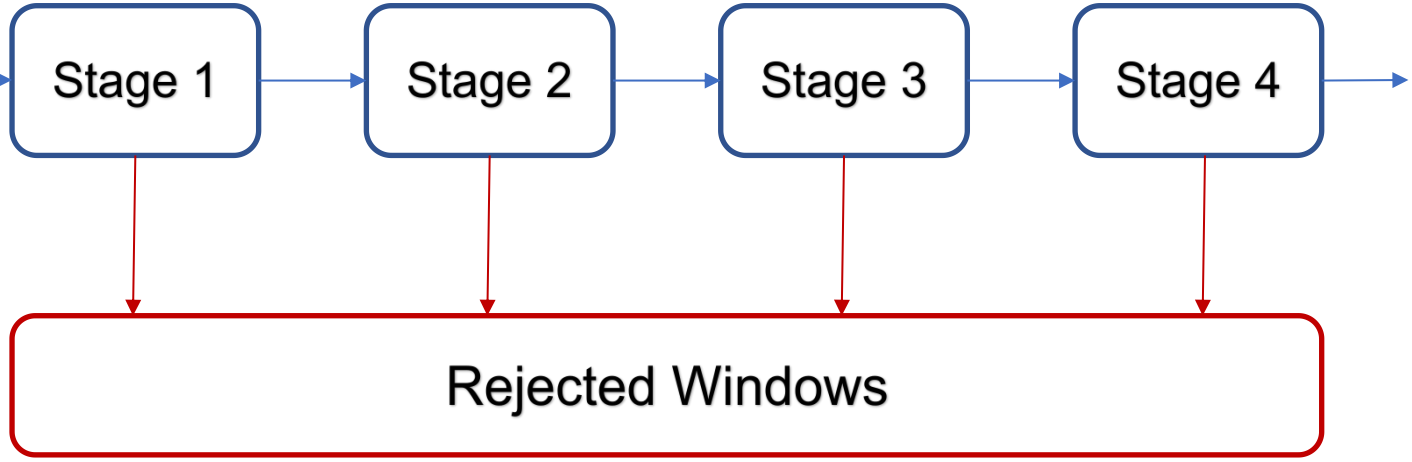
Face Detection

Cascade of Classifier



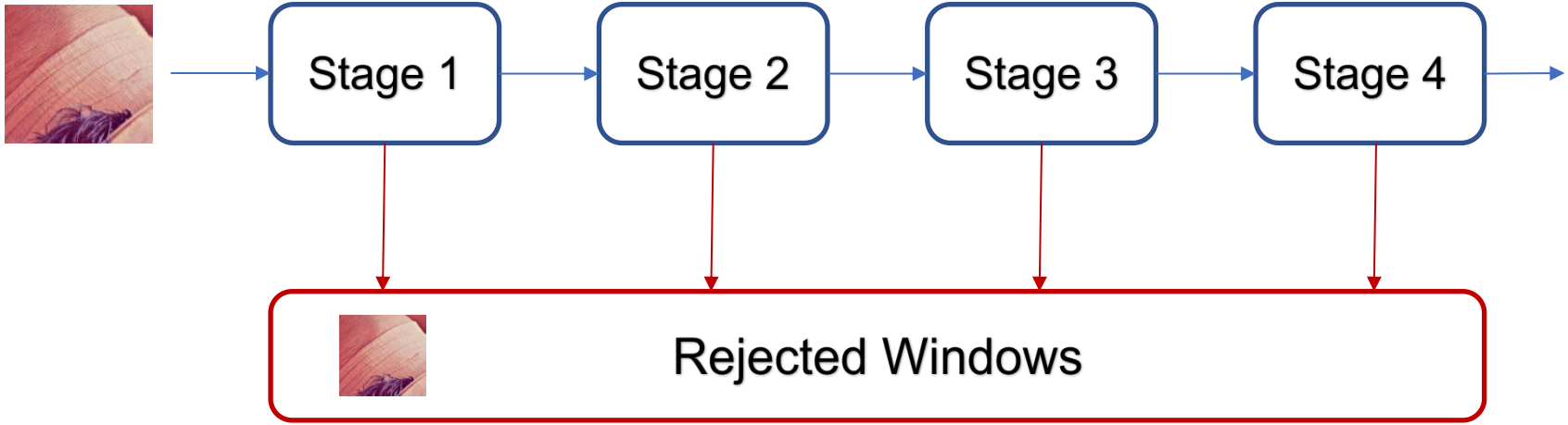
Face Detection

Cascade of Classifier

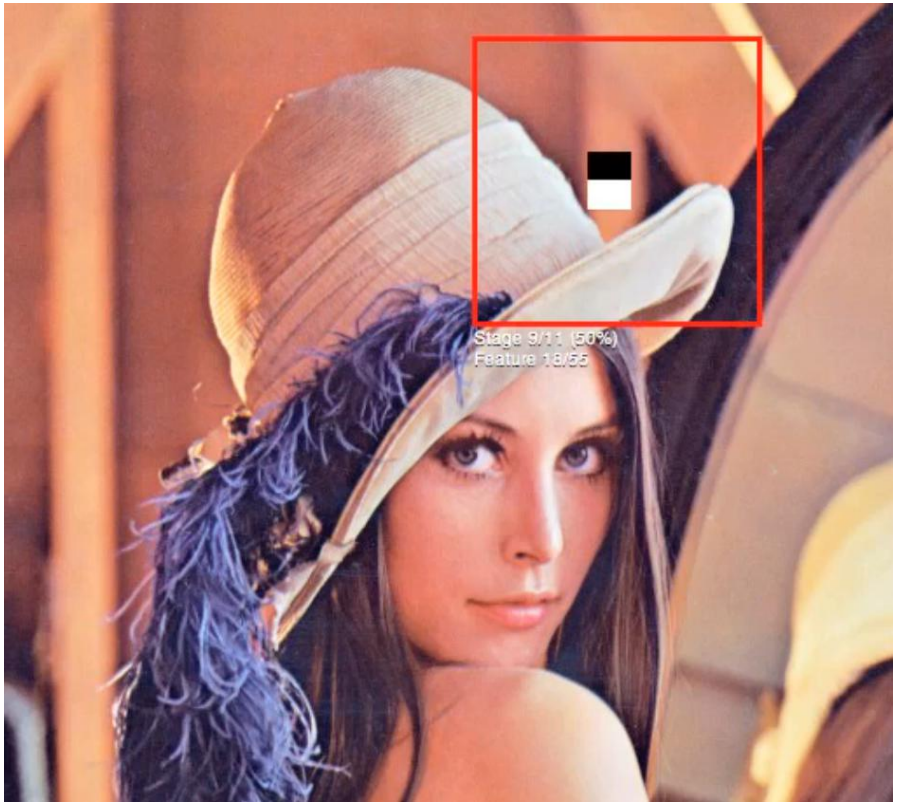


Face Detection

Cascade of Classifier

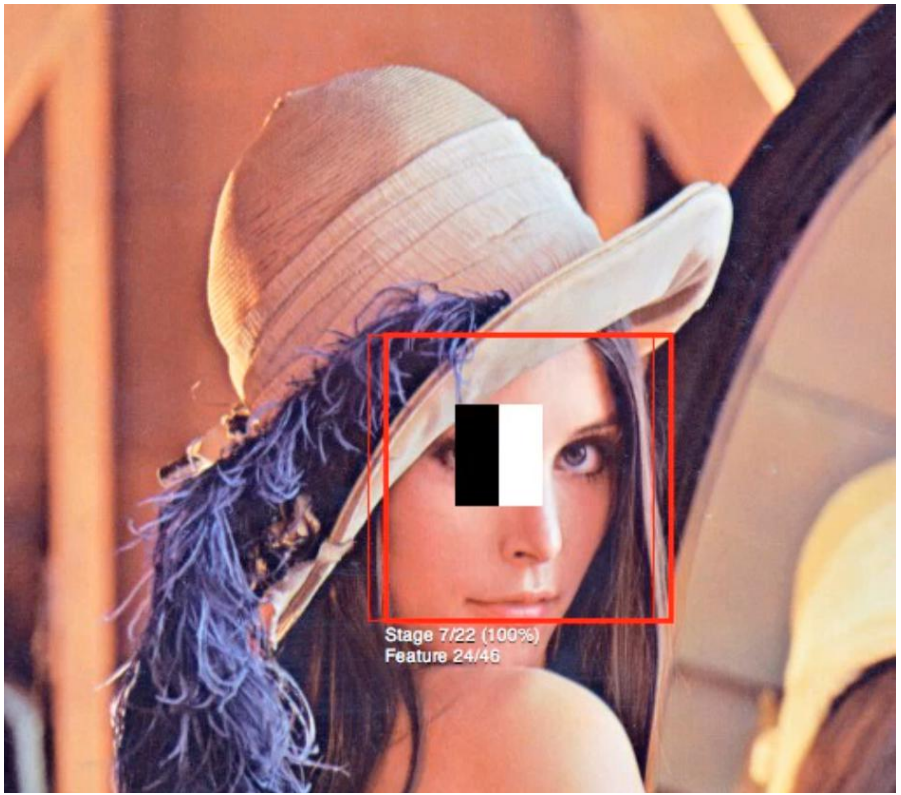


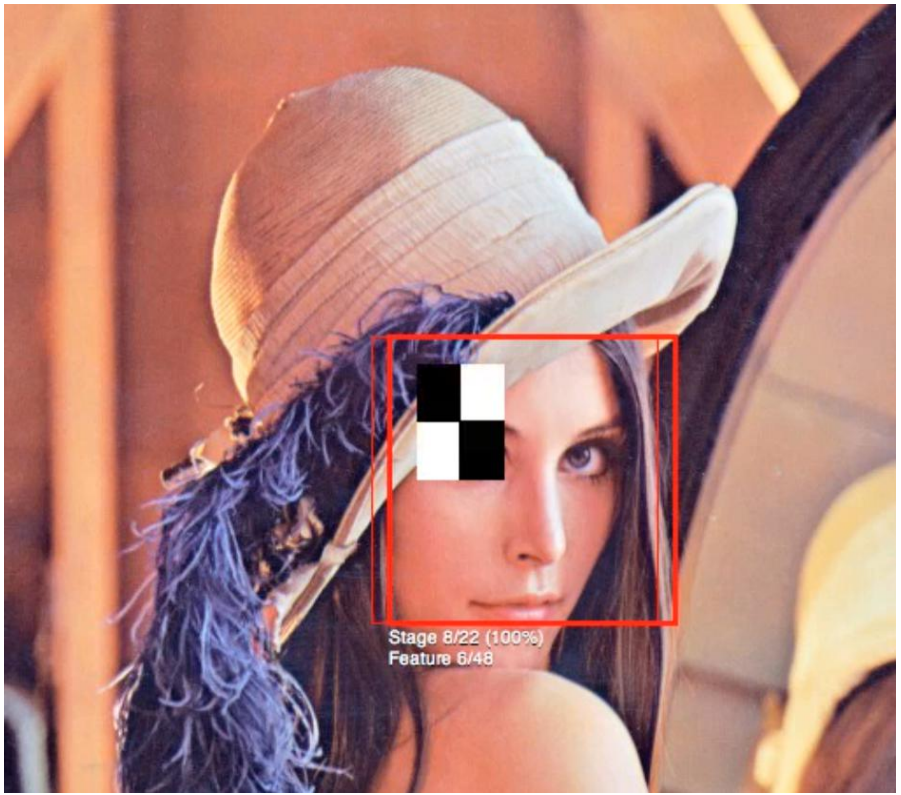
Face Detection



https://vimeo.com/12774628

Face Detection





https://vimeo.com/12774628

Face Detection - OpenCV

Python:

```
cv.CascadeClassifier.detectMultiScale(  image[, scaleFactor[, minNeighbors[, flags[, minSize[, maxSize]]]]] ) -> objects
cv.CascadeClassifier.detectMultiScale2( image[, scaleFactor[, minNeighbors[, flags[, minSize[, maxSize]]]]] ) -> objects, numDetections
cv.CascadeClassifier.detectMultiScale3( image[, scaleFactor[, minNeighbors[, flags[, minSize[, maxSize[, outputRejectLevels]]]]]] ) -> objects, rejectLevels, levelWeights
```

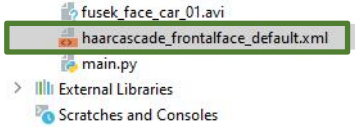
Detects objects of different sizes in the input image. The detected objects are returned as a list of rectangles.

Parameters

image	Matrix of the type CV_8U containing an image where objects are detected.
objects	Vector of rectangles where each rectangle contains the detected object, the rectangles may be partially outside the original image.
scaleFactor	Parameter specifying how much the image size is reduced at each image scale.
minNeighbors	Parameter specifying how many neighbors each candidate rectangle should have to retain it.
flags	Parameter with the same meaning for an old cascade as in the function cvHaarDetectObjects. It is not used for a new cascade.
minSize	Minimum possible object size. Objects smaller than that are ignored.
maxSize	Maximum possible object size. Objects larger than that are ignored. If `maxSize == minSize` model is evaluated on single scale.

https://docs.opencv.org/4.5.5/d1/de5/classcv_1_1CascadeClassifier.html#aaf8181cb63968136476ec4204ffca498

Face Detection - OpenCV



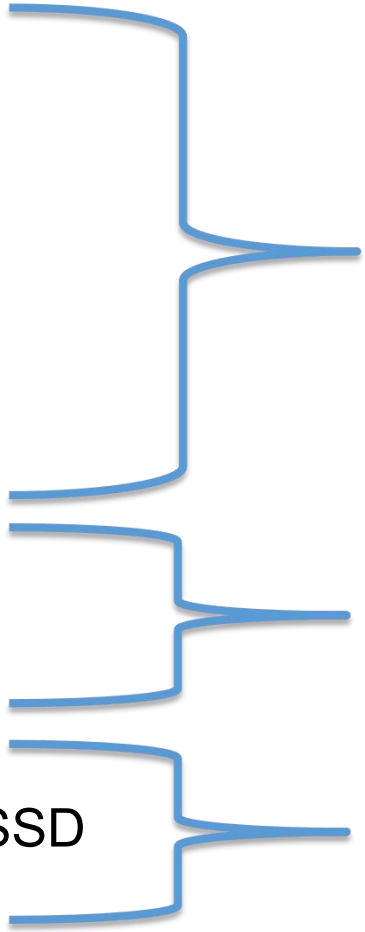

```python
def face_detect():
    cv2.namedWindow("face_detect", 0)
    video_cap = cv2.VideoCapture("fusek_face_car_01.avi")
    face_cascade = cv2.CascadeClassifier("haarcascade_frontalface_default.xml")

    while True:
        ret, frame = video_cap.read()
        paint_frame = frame.copy()
        if ret is True:
            faces = face_cascade.detectMultiScale(frame,
                                                  scaleFactor=1.2,
                                                  minNeighbors=3,
                                                  minSize=(100, 100),
                                                  maxSize=(500, 500))
            for one_face in faces:
                cv2.rectangle(paint_frame, one_face, (0, 0, 255), 12)
                cv2.rectangle(paint_frame, one_face, (255, 255, 255), 4)

        cv2.imshow("opencv_frame", paint_frame)
        if cv2.waitKey(2) == ord("q"):
            break
```

Object Detection

- Haar
- HOG
- LBP
- SIFT, SURF
- CNNs
- R-CNNs/YOLO/SSD



Traditional Approaches (sliding window)

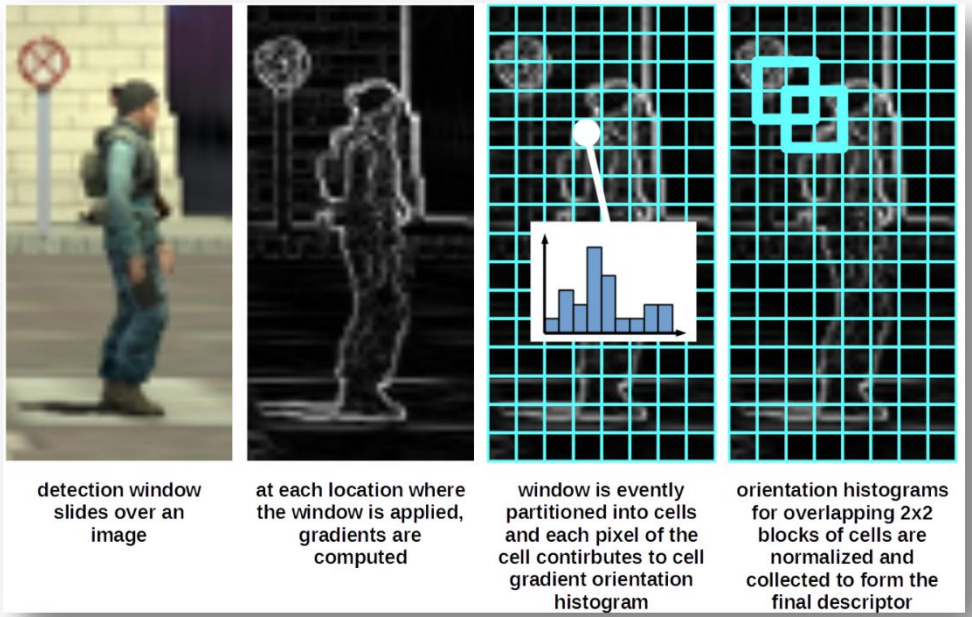
KeyPoints Approaches

Deep Learning Approaches

Histograms of Oriented Gradients (HOG)

Basic Steps:

- In HOG, a sliding window is used for detection.
- The window is divided into small connected cells.
- The histograms of gradient orientations are calculated in each cell.
- For each window position, compute its HOG descriptor and pass it to a classifier (e.g., Support Vector Machine) trained to detect objects like pedestrians.



Olejniczak, Michał & Kraft, Marek. (2017). Taming the HoG: The Influence of Classifier Choice on Histogram of Oriented Gradients Person Detector Performance. 552-560. 10.1007/978-3-319-59063-9_49.

Histograms of Oriented Gradients (HOG)

https://scikit-image.org/docs/stable/auto_examples/features_detection/plot_hog.html

Basic Idea of SVM

SVM aims to find the optimal decision boundary, called a *hyperplane*. The hyperplane maximizes the margin, which is the distance between the hyperplane and the nearest data points from each class, known as *support vectors*

The algorithm chooses the hyperplane with the largest possible margin to improve classification accuracy and generalization for new data points.

Linear and Nonlinear Classification:

- **Linear SVM**: If the data is linearly separable, SVM uses a straight-line hyperplane.
- **Nonlinear SVM**: For complex datasets where linear separation isn't possible, SVM employs the *kernel trick*. Kernels map input data into a higher-dimensional feature space, enabling linear separation in that transformed space. Common kernels include polynomial, radial basis function (RBF), and sigmoid

Support Vectors: These are the critical data points closest to the hyperplane that influence its orientation and position. Removing or altering these points can change the hyperplane, making them essential for building the classifier

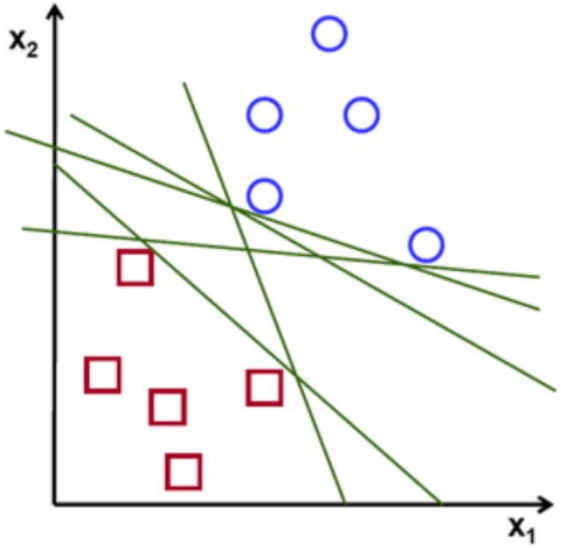
https://docs.opencv.org/4.9.0/d1/d73/tutorial_introduction_to_svm.html

SVM

A Support Vector Machine (SVM) is a discriminative classifier formally defined by a separating hyperplane. In other words, given labeled training data (*supervised learning*), the algorithm outputs an optimal hyperplane which categorizes new examples.

In which sense is the hyperplane obtained optimal? Let's consider the following simple problem:

For a linearly separable set of 2D-points which belong to one of two classes, find a separating straight line.

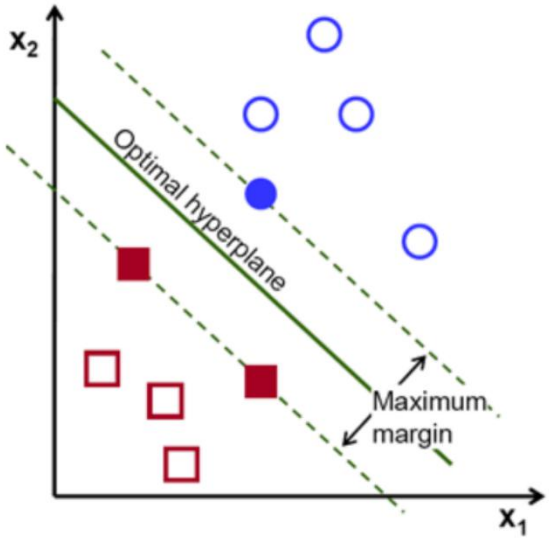


https://docs.opencv.org/4.9.0/d1/d73/tutorial_introduction_to_svm.html

SVM

In the above picture you can see that there exists multiple lines that offer a solution to the problem. Is any of them better than the others? We can intuitively define a criterion to estimate the worth of the lines: *A line is bad if it passes too close to the points because it will be noise sensitive and it will not generalize correctly.* Therefore, our goal should be to find the line passing as far as possible from all points.

Then, the operation of the SVM algorithm is based on finding the hyperplane that gives the largest minimum distance to the training examples. Twice, this distance receives the important name of **margin** within SVM's theory. Therefore, the optimal separating hyperplane *maximizes* the margin of the training data.



https://docs.opencv.org/4.9.0/d1/d73/tutorial_introduction_to_svm.html

AdaBoost

Basic Idea of AdaBoost

AdaBoost (short for Adaptive Boosting) is a powerful ensemble learning algorithm that combines multiple weak classifiers to create a strong classifier. It is widely used for classification tasks and works particularly well with decision trees as its base learners.

AdaBoost builds a series of weak classifiers (often decision stumps, which are single-split decision trees). A weak learner is a model that performs slightly better than random guessing. The algorithm trains weak classifiers sequentially, where each subsequent classifier focuses more on the errors (misclassified samples) made by the previous ones. This allows the ensemble to progressively improve its performance.

Initially, all data points are assigned equal weights. After each weak classifier is trained, AdaBoost increases the weights of misclassified data points so that the next classifier pays more attention to these "hard" examples. Correctly classified points receive lower weights. Each weak learner is assigned a weight based on its accuracy (or error rate). Better-performing classifiers are given higher weights. The final prediction is made by combining the weighted votes of all weak classifiers in the ensemble.

AdaBoost

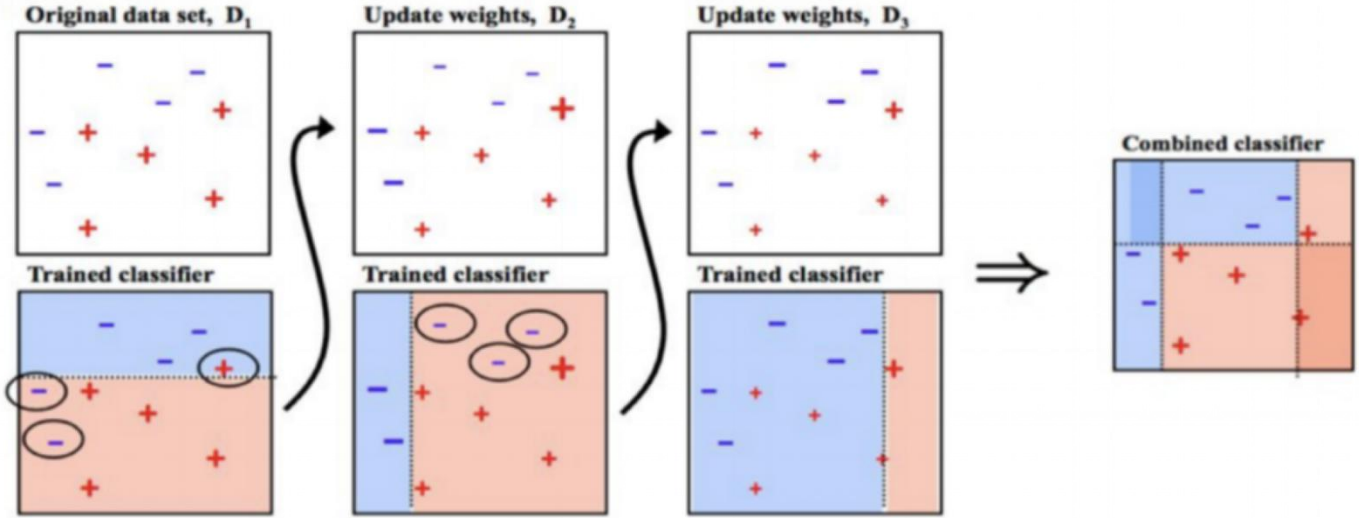


FIGURE 9: A description of how the AdaBoost algorithm works. The learner is incrementally *boosted* at each iteration, where the wrongly classified points from the last iteration are prioritized and the weights assigned to them are adjusted [41].

Iddianozie, Chidubem & Bertolotto, Michela & McArdle, Gavin. (2020). Exploring Budgeted Learning for Data-Driven Semantic Inference via Urban Functions. IEEE Access. PP. 1-1. 10.1109/ACCESS.2020.2973885.