Last time:
Feature Engineering 1: Points

- «Good» / Interest / Key **points** to find:
    - Flat, Edge and Corner
- **Harris** Corner Detection:
    - Taylor, Second moment matrix & Response function
- **SIFT**:
    - Scale invariant, dominant direction, descriptor
- **RANSAC**
    - Sample, model, inliers / outliers

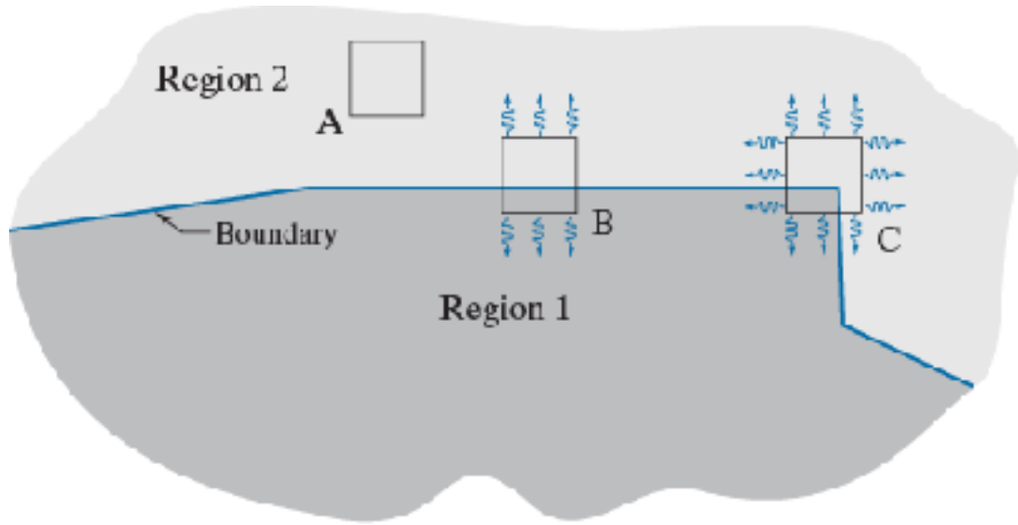
Summary

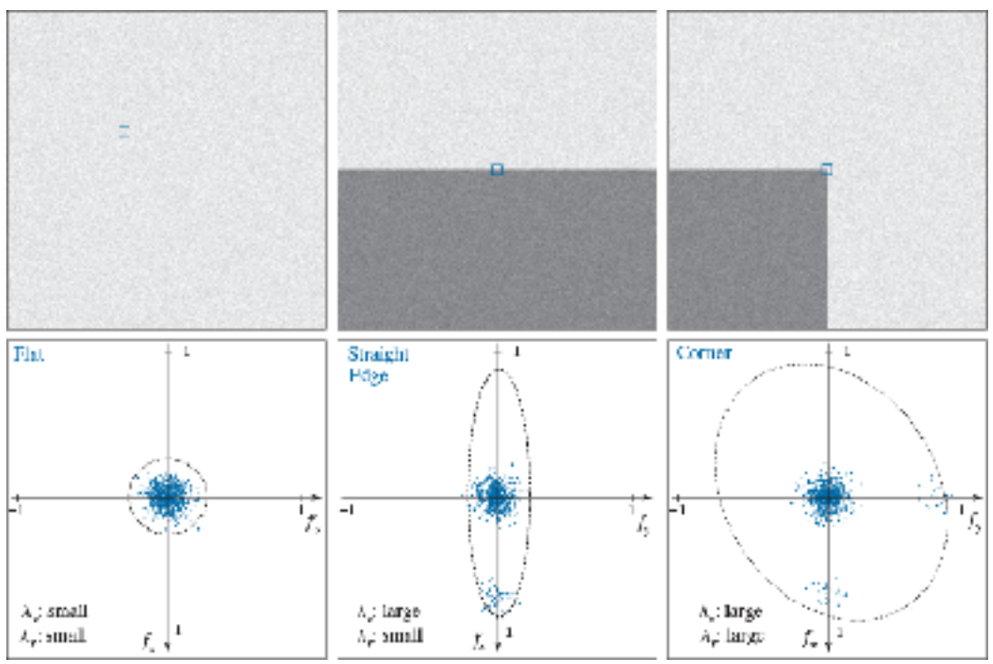
- **Harris**-Stephens Corner Detector (1988)

- **SIFT**: Scale Invariant Feature Transform (2004)

Harris-Stephens Corner Detector (1988)

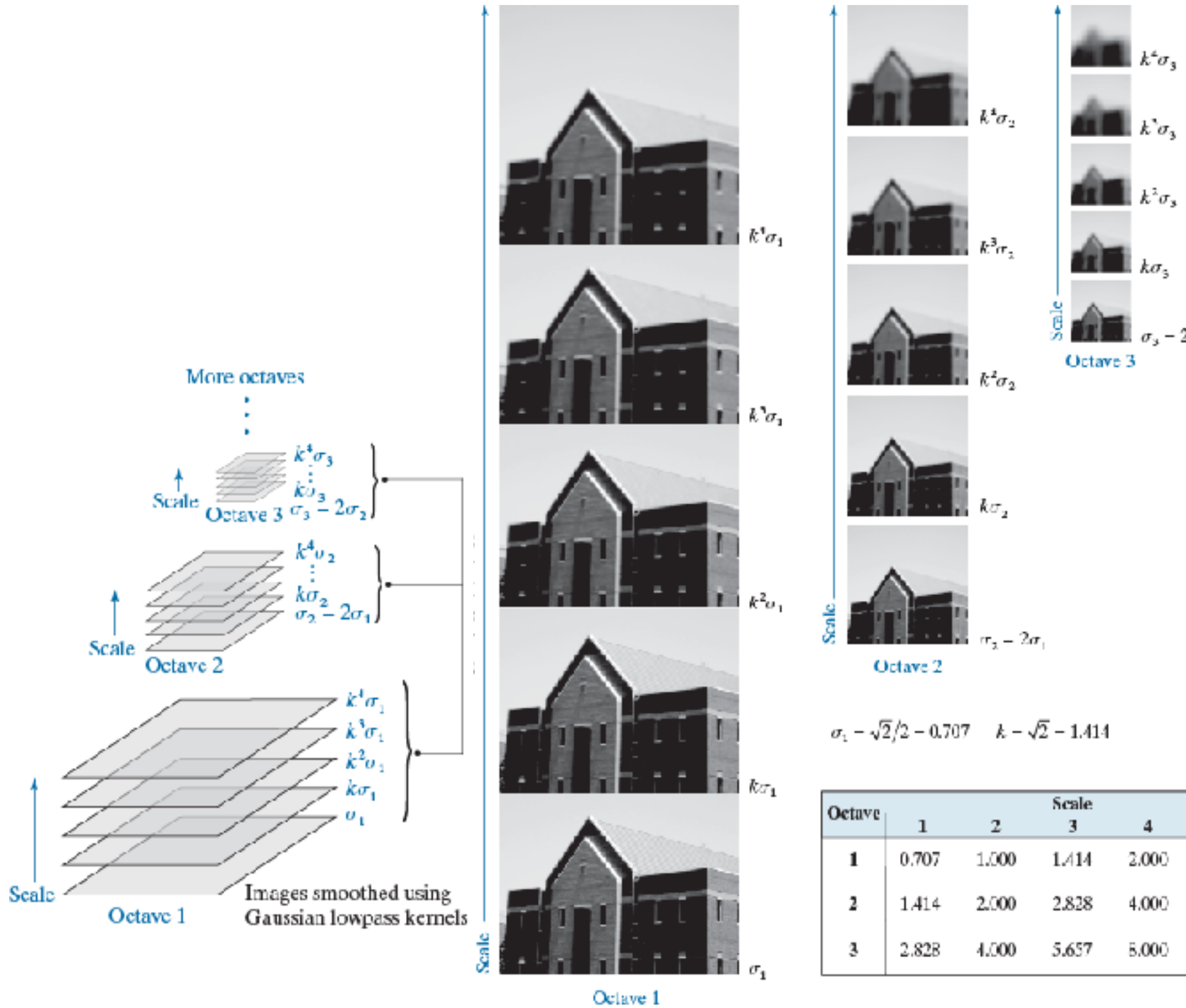
- Patch and shifted patch.
- Sum of weighted squared differences between the two patches.
- Taylor expansion: f-shifted ~ f + xf_x + yf_y
- Rewrite using f_x/f_y (derivatives) and in matrix form: C(x,y) = [x y] M [x y]^T, where M = sum*w*A, A = matrix of f_x/f_y
- R = I(anda)_x*I_y - k(I_x + I_y)^2 = det(M) - k*trace(M)^2
- Corner detected: R > T





Scale Invariant Feature Transform (SIFT, 2004)

- Construct the **scale space**:
- Procedure shown in figure
- Parameters specified: sigma, s, (k from s) and # octaves

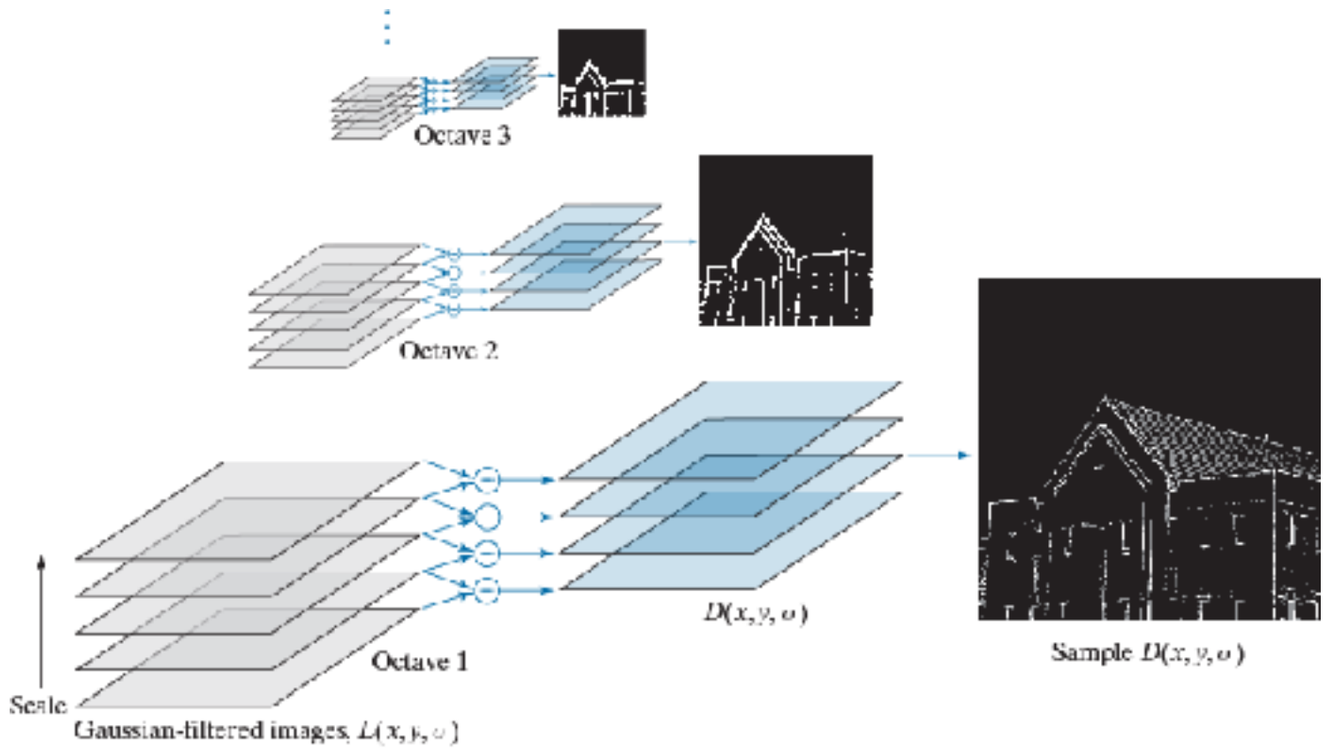


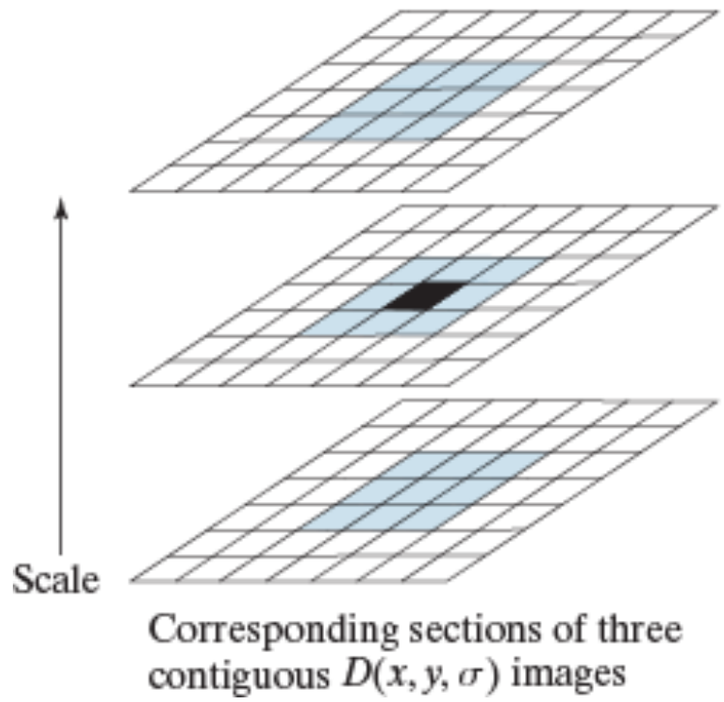
$\sigma_1 = \sqrt{2}/2 = 0.707 \quad k = \sqrt{2} = 1.414$

Octave	Scale 1	2	3	4	5
1	0.707	1.000	1.414	2.000	2.828
2	1.414	2.000	2.828	4.000	5.657
3	2.828	4.000	5.657	8.000	11.314

SIFT (2)

- Obtain the **initial** key-points:
  - Compute the difference of Gaussians D from the smoothed images in scale space.
  - Find the extrema in each D image





Corresponding sections of three contiguous $D(x, y, \sigma)$ images

SIFT (3)

- Improve the accuracy of the **location** of the key-points:
  - Interpolate the values of D via a Taylor extension.

SIFT (4)

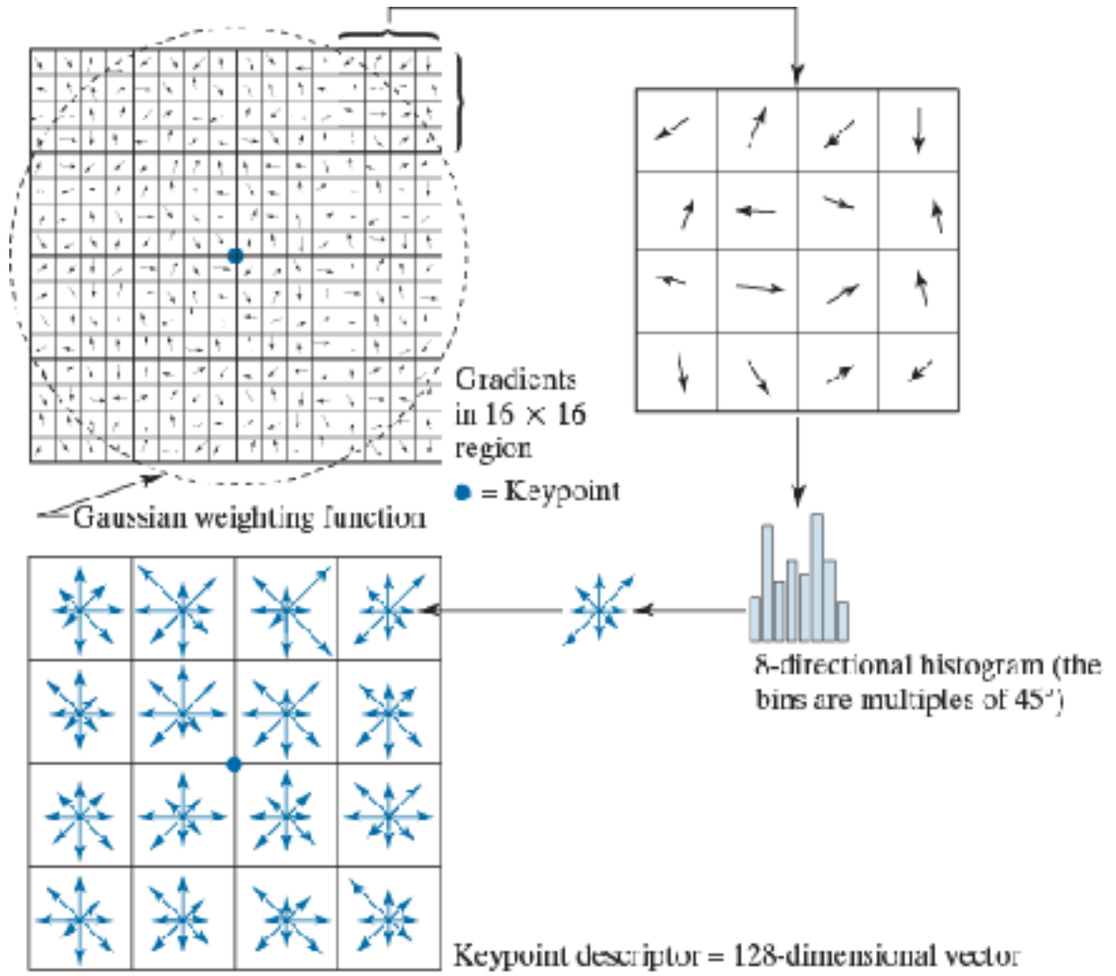
- Delete **unsuitable** key-points:
- Eliminate key-points that have low contrast and/or are poorly localized.
- All key-points whose values of D are lower than a threshold are deleted.
- Key-points associated with edges are also deleted

SIFT (5)

- Compute key-point **orientation**:
  - Compute the magnitude and orientation of each key-point using a histogram-based procedure.

SIFT (6)

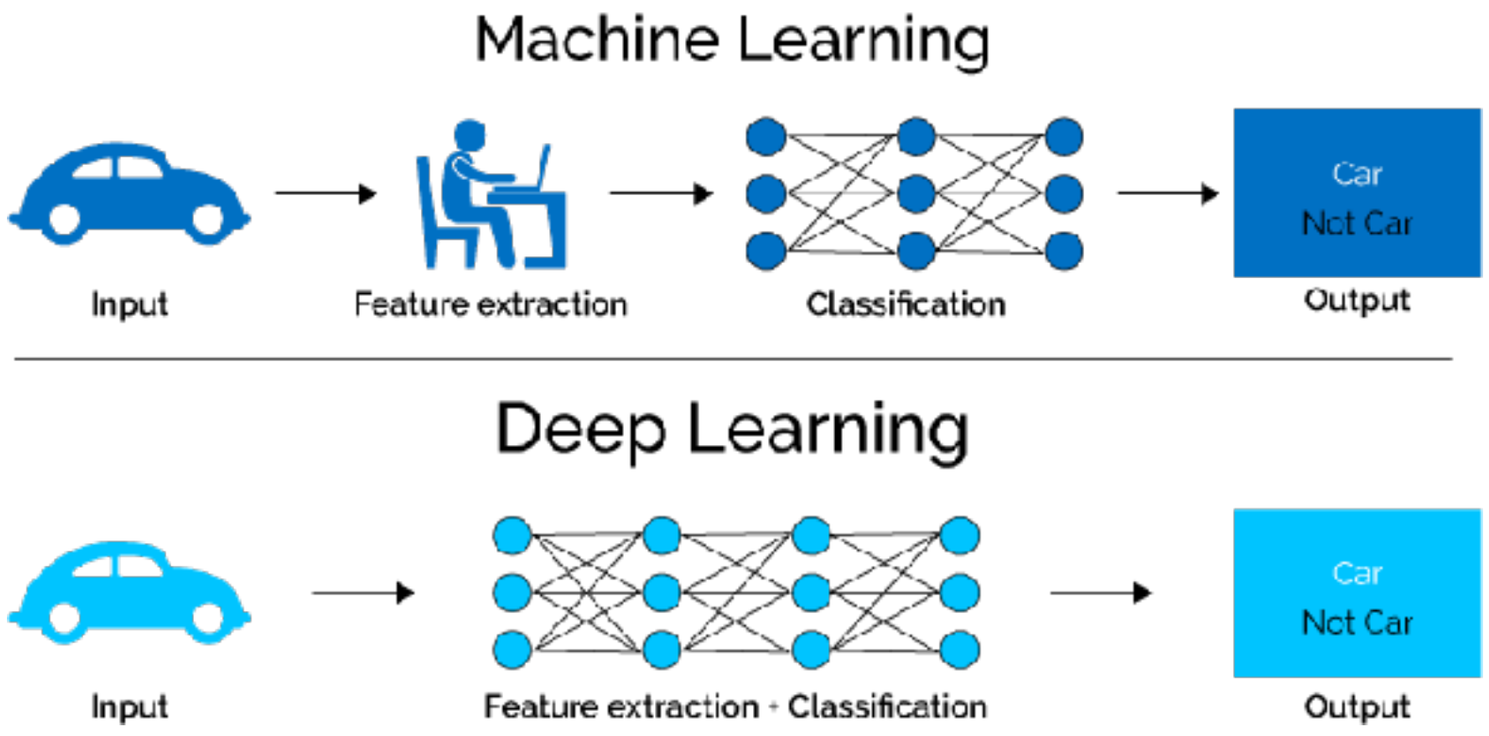
- Compute key-point **descriptors**:
  - Use the method shown in the figure to compute a feature descriptor vector for each key-point.
  - If a region of size 16x16 around each key-point is used, the result will be a 128-dimensional feature vector for each key-point.



This time..

Traditional CV (ML-based):

Feature Engineering 2: Regions



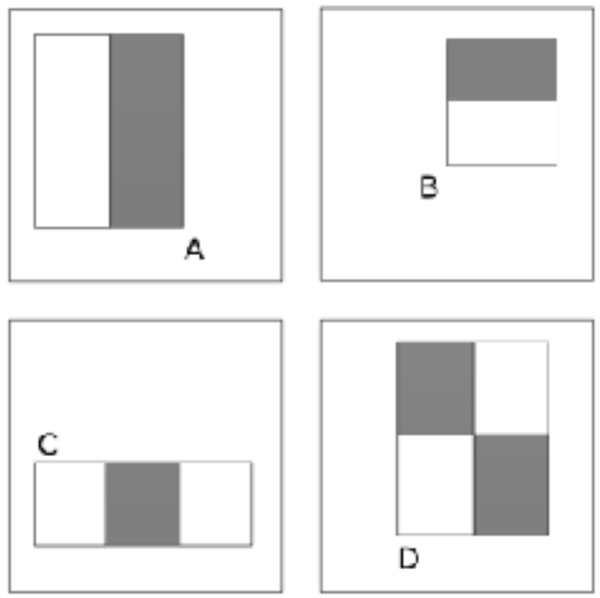
Agenda

- Viola Jones (2001) Orientation, not very relevant at the exam
- HOG (2005)
- DPM (2010) Orientation, not very relevant at the exam
- Traditional CV (ML) example: HOG (FE) + SVM (Class)

Orientation, not very relevant at the exam

Viola Jones (face detection)

- Characteristics:
  - Robust
  - Real-time
  - Face detection (other uses)

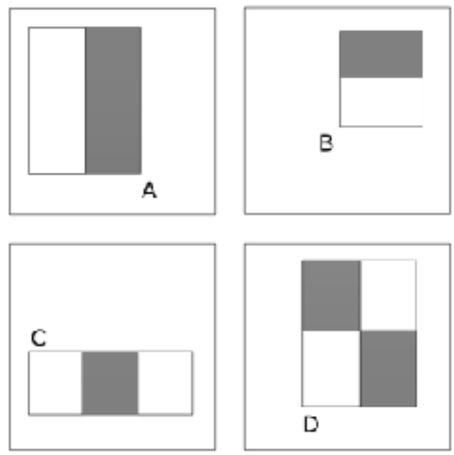


Example rectangle features shown relative to the enclosing detection window

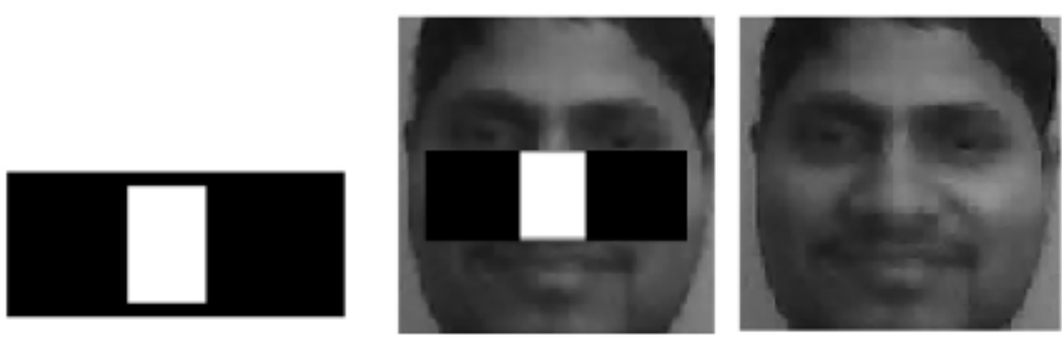
- Algorithm (four stages):
  - Haar Feature Selection
  - Creating an Integral Image
  - Adaboost Training
  - Cascading Classifiers

Haar Feature that looks similar to the eye region which is darker than the cheeks

VJ-1: Haar Features



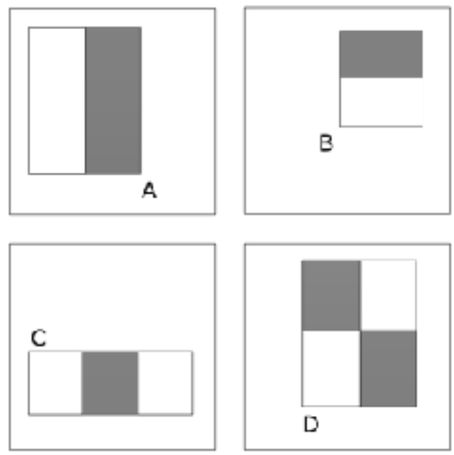
- Four different types of features used in the framework (top right)



- Sums of image pixels within (many) rectangular areas

- Value of any given feature is the sum of the pixels within white rectangles subtracted from the sum of the pixels within black rectangles

VJ-1: Haar Features (2)



- All human faces share some similar properties.
  - The nose bridge region is brighter than the eyes
  - The eye region is darker than the cheeks.

Haar Feature that looks similar to the bridge of the nose is applied onto the face

- **Value** = difference in brightness between the white & black rectangles over a specific area.
- Three **types**: two-, three-, four-rectangles.

Haar Feature that looks similar to the eye region which is darker than the cheeks

- Each feature is related to a special **location** in the sub-window

3rd and 4th kind of Haar Feature

VJ-2: Integral Image

- A image representation called the integral image (summed area table) evaluates rectangular features in constant time.

- A description of computing a sum in the summed-area table data structure/ algorithm: **four** array references

- Because each feature's rectangular area is always adjacent to at least one other rectangle, it follows that any **two-rectangle** feature can be computed in **six** array references, any **three**-rectangle feature in **eight**, and any **four**-rectangle feature in **nine**.

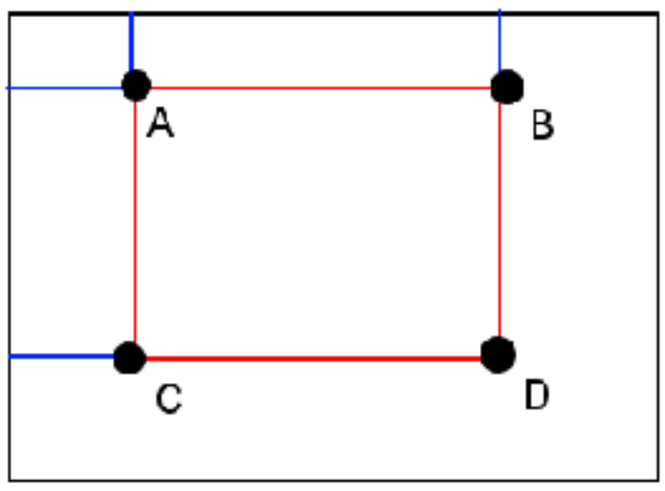
1.

31	2	4	33	5	36
12	26	9	10	29	25
13	17	21	22	20	18
24	23	15	16	14	19
30	8	28	27	11	7
1	35	34	3	32	6

2.

31	33	37	70	75	111
43	71	84	127	161	222
56	101	135	200	254	333
80	148	197	278	346	444
110	186	263	371	450	555
111	222	333	444	555	666

15 + 16 + 14 + 28 + 27 + 11 =
101 + 450 – 254 – 186 = 111



Sum = D - B - C + A

VJ-3: Adaboost Training

- In a standard 24x24 pixel sub-window, there are a total of M = 162,336 possible features.

- Expensive to evaluate them all when testing an image.

- VJ employs a variant of the learning algorithm AdaBoost to **both** select the best features and to train classifiers that use them.

- The algorithm constructs a "**strong**" classifier as a linear combination of weighted simple "weak" classifiers.

- AdaBoost variant not part of the syllabus.

VJ-4: Cascade architecture

- On average only 0.01% of all sub-windows are positive (i.e. faces). Equal computation time is spent on all sub-windows.

- Should spend most time only on potentially positive sub-windows. A simple **2-feature classifier** can achieve almost 100% detection rate with 50% FP rate. That classifier can act as a 1st layer of a series to filter out most **negative** windows

- 2nd layer with 10 features can tackle “harder” negative-windows which survived the 1st layer, and so on… The strong classifiers are arranged in a cascade in order of complexity (achieves gradually better detection rates), where each successive classifier is trained only on those selected samples which pass through the preceding classifiers (to be real-time).

- In cascading, each stage consists of a strong classifier. So all the features are grouped into several stages where each stage has certain number of features.

- The job of each stage is to determine whether a given sub-window is **definitely not a face or may be a face**. A given sub-window is immediately discarded as not a face if it fails in any of the stages.

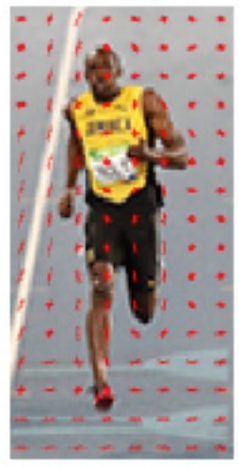
HOG (pedestrian detection)

- Paper: Dalal and Triggs (2005)

- Feature descriptor: **representation** of an image or an image **patch**. **Simplifies** the image by extracting **useful** information.

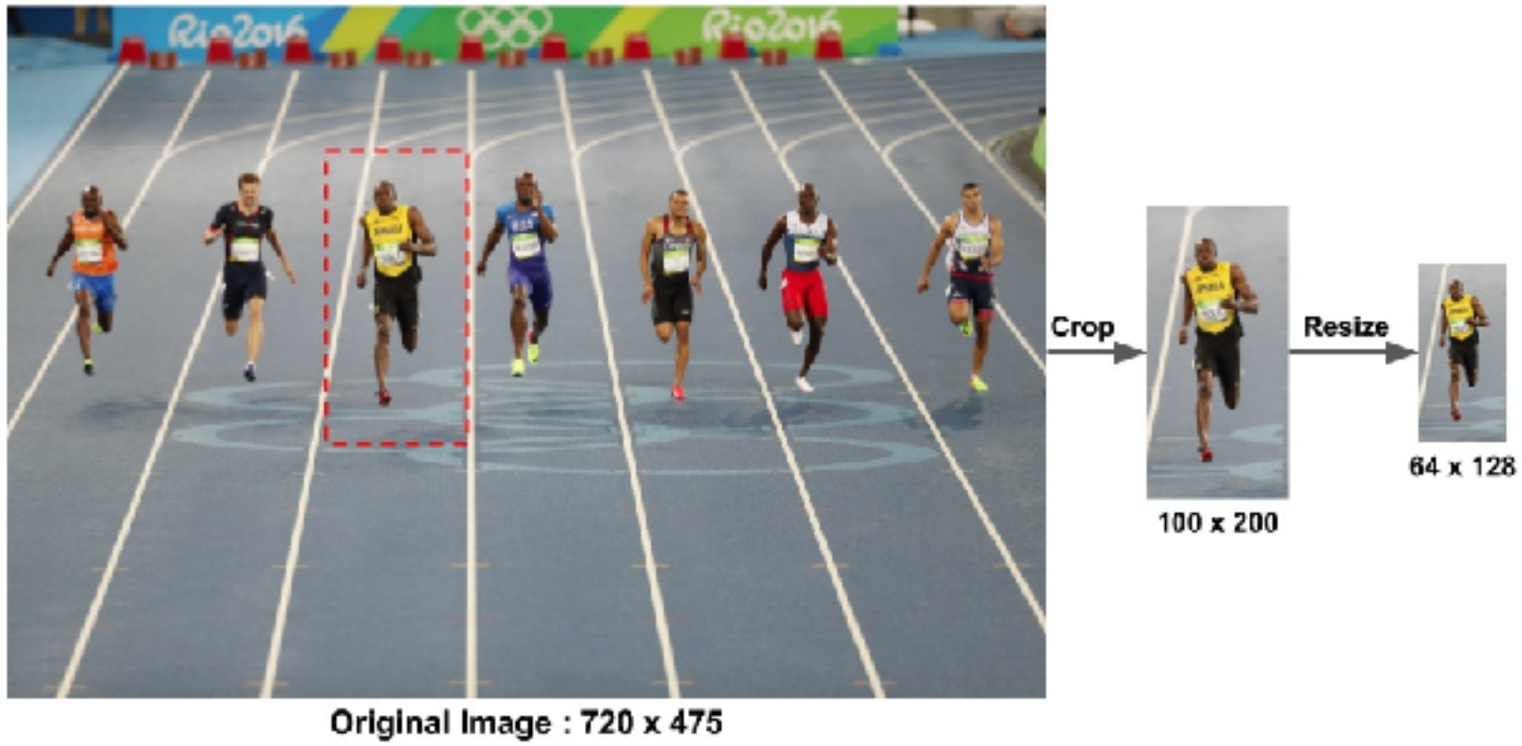
- Converts width x height x 3 (channels) to n. Original HOG: input image is of size **64 x 128 x 3** and the output feature vector is of length 3780. Good results when fed into an image classification algorithm like SVM.

- HOG feature descriptor: **distribution** (=histograms) of **directions** of gradients (=oriented gradients). **Gradients** (x and y **derivatives**) of an image are useful because the **magnitude** of gradients is large around edges and corners (regions of abrupt intensity changes) and we know that edges and corners contain a lot more information about object **shape** than flat regions.



Histogram of Oriented Gradients

HOG: **Step 1**: Preprocessing



- Pedestrian detection, 64×128 patch, multiple locations and scales, fixed aspect ratio (1:2, e.g. 100x200).

- Image (arbitrary size), patch (100×200) is cropped out of the image and resized to 64×128.

HOG: **Step 2**: Calculate the Gradient Images

- Calculate the horizontal and vertical first **derivatives**. **Filter** the image with the kernels to the right (or use Sobel).

-1	0	1

-1
0
1

- The x-gradient **fires** on vertical lines and the y-gradient fires on horizontal lines. The magnitude of gradient fires where ever there is a sharp **change** in intensity.

$$g = \sqrt{g_x^2 + g_y^2}$$

$$\theta = \arctan \frac{g_y}{g_x}$$

- The gradient image removed a lot of non-essential information (e.g. constant colored background), but highlighted outlines, i.e. you can look at the **gradient image** and still easily say there is a **person** in the picture.

- At every pixel, the gradient has a **magnitude and a direction**. The magnitude of gradient at a pixel is the maximum of the magnitude of gradients of the three channels, and the angle is the angle corresponding to the maximum gradient.

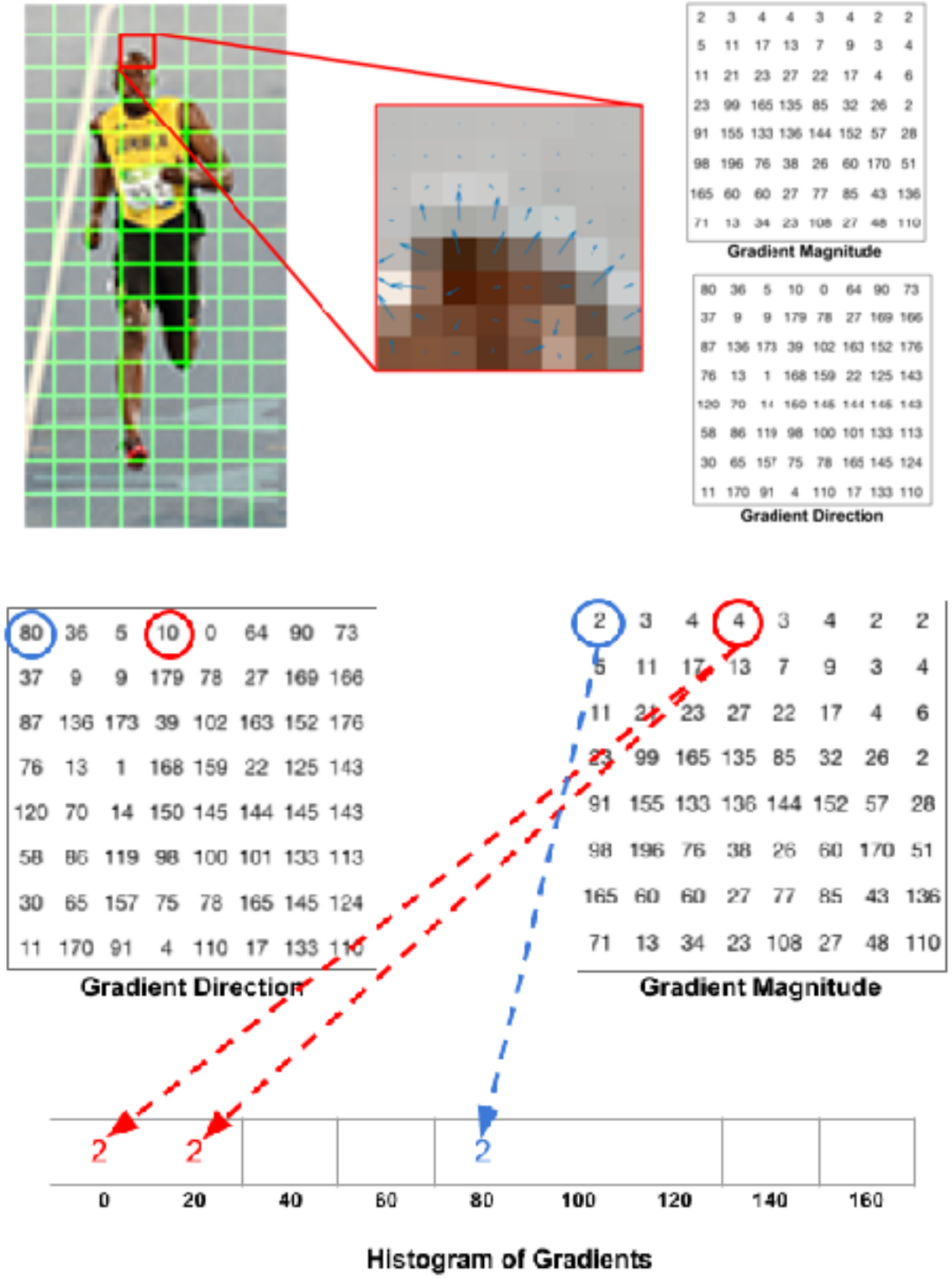
HOG: **Step 3:** Calculate Histogram of Gradients in 8×8 cells

- Divided image patch into 8×8 cells and calculate a **histogram** of gradients for each cell.

- **Compact** representation: An 8×8 image patch contains 8x8x3 = **192** pixel values. The gradient of this patch contains 2 values (magnitude and direction) per pixel which adds up to 8x8x2 = **128** numbers. these 128 numbers are represented using a 9-bin histogram which can be stored as an array of **9** numbers. Compact and more robust to noise.

- Histogram: 9 bins, corresponding to angles 0, 20, 40, 60 … 160, "**unsigned**" gradients.

- A **bin** is selected based on the **direction**, and the **vote** (the value that goes into the bin) is selected based on the **magnitude**. **Blue** angle (direction) of 80 degrees and magnitude of 2. **Red** has an angle of 10 degrees and magnitude of 4. Since 10 degrees is half way between 0 and 20, the vote by the pixel splits evenly into the two bins.

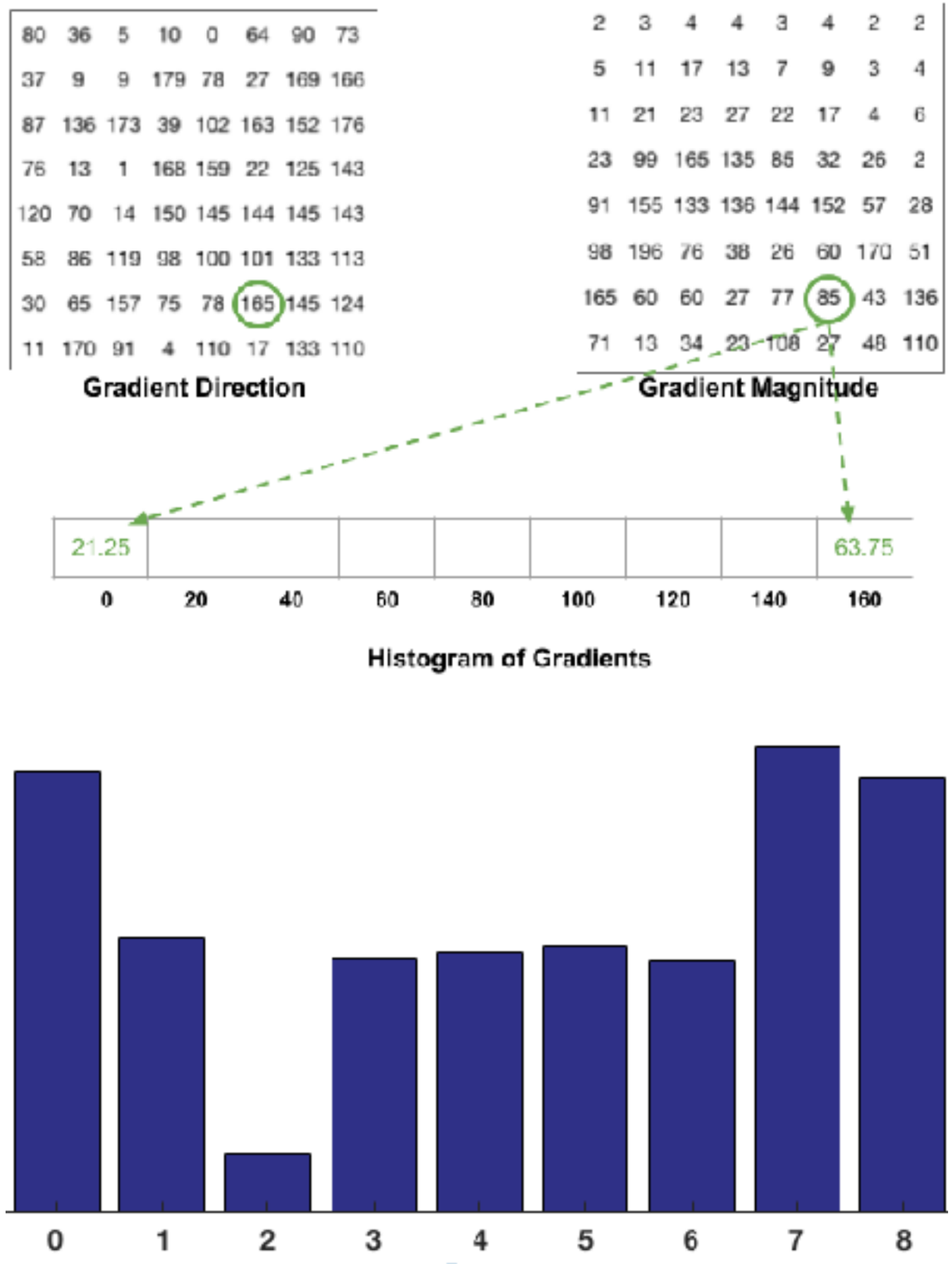


HOG: **Step 3:** Calculate Histogram of Gradients in 8×8 cells (2)

- If the angle is greater than 160 degrees, it is between 160 and 180 and we know the angle wraps around making 0 and 180 equivalent.

- Pixel with angle 165 degrees contributes proportionally to the 0 (180) degree bin and the 160 degree bin.

- The contributions of all the pixels in the 8×8 cells are added up to create the 9-bin histogram.

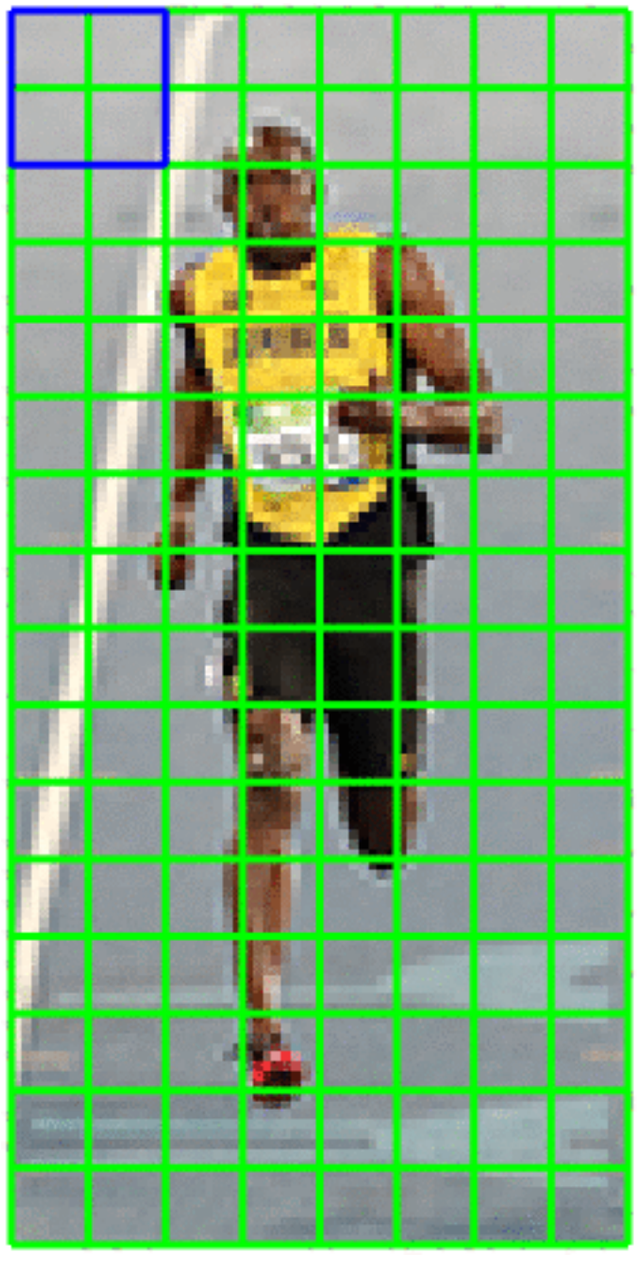


HOG: **Step 4:** 16×16 Block Normalization

- If you make the image **darker** by dividing all pixel values by 2, the gradient **magnitude** will change by half, and therefore the histogram values will change by half.

- We would like to “normalize” the histogram so they are not affected by lighting variations.

- A 16×16 block has 4 histograms which can be concatenated to form a 36 x 1 element vector (4x9) and it can be normalized just the way a 3×1 vector is normalized (dividing each element of this vector by the L2 norm of the vector).



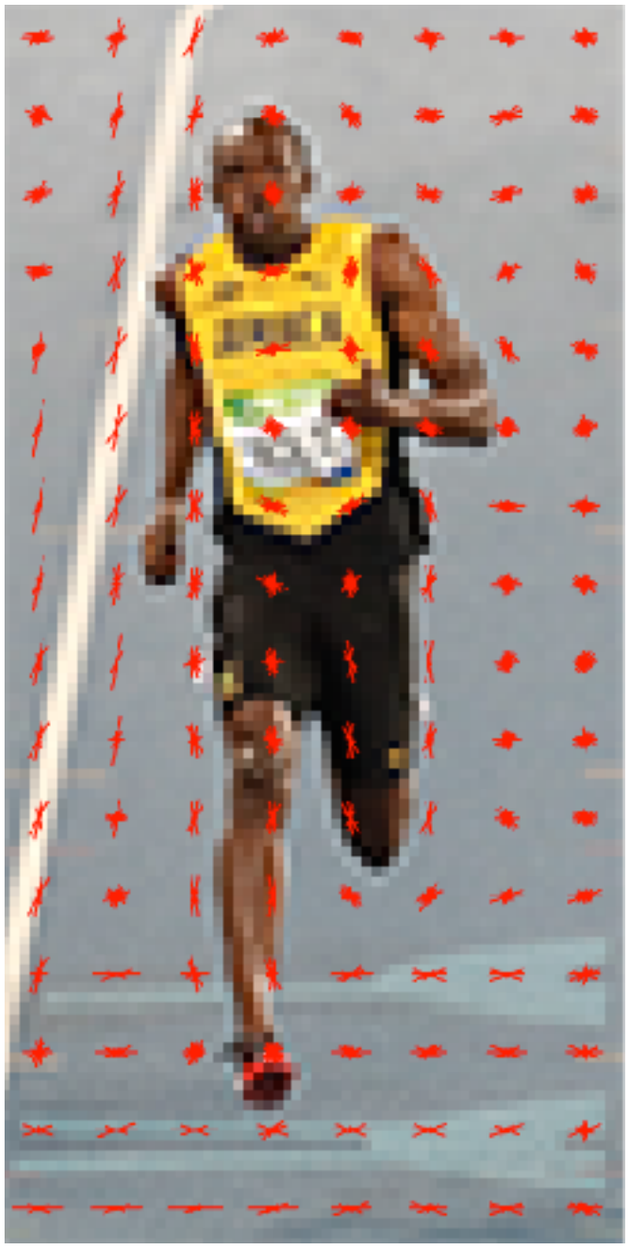
HOG: **Step 5:** Calculate the HOG feature vector & visualizing HOG

- Each 16×16 block is represented by a **36**×1 vector.

- There are 7 horizontal and 15 vertical positions of the 16×16 block making a total of 7 x 15 = **105** positions.

- When we concatenate the 36-vectors for all 105-positions into one giant vector we obtain a 36×105 = **3780** dimensional vector.

- **Viz**: image patch is usually visualized by plotting the 9×1 normalized histograms in the 8×8 cells. Dominant direction of the histogram captures the shape of the person



Orientation, not very relevant at the exam

DPM

Deformable Part Model

The Origins of DPM

Dalal & Triggs '05

- Histrogram of Oriented Gradients (HOG)
- SVM training
- Sliding window detection

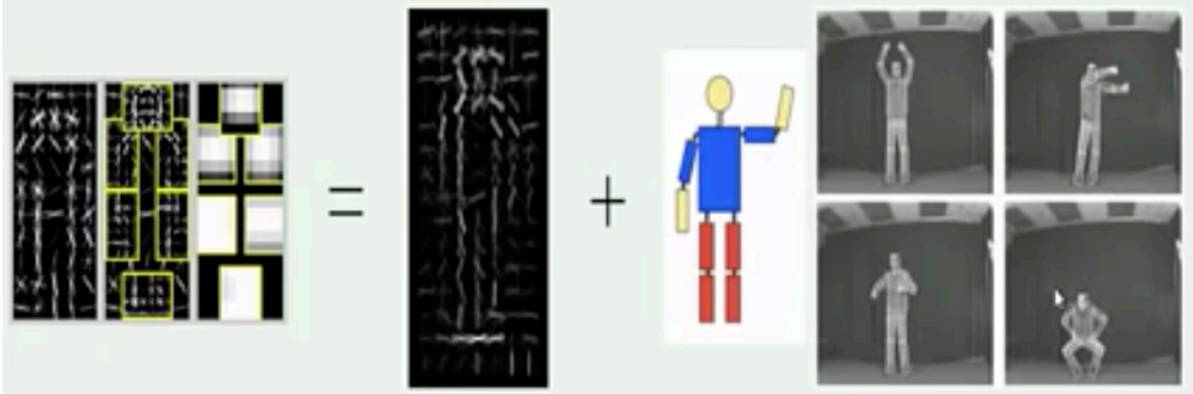
Fischler & Elschlager '73

Felzenszwalb & Huttenlocher '00

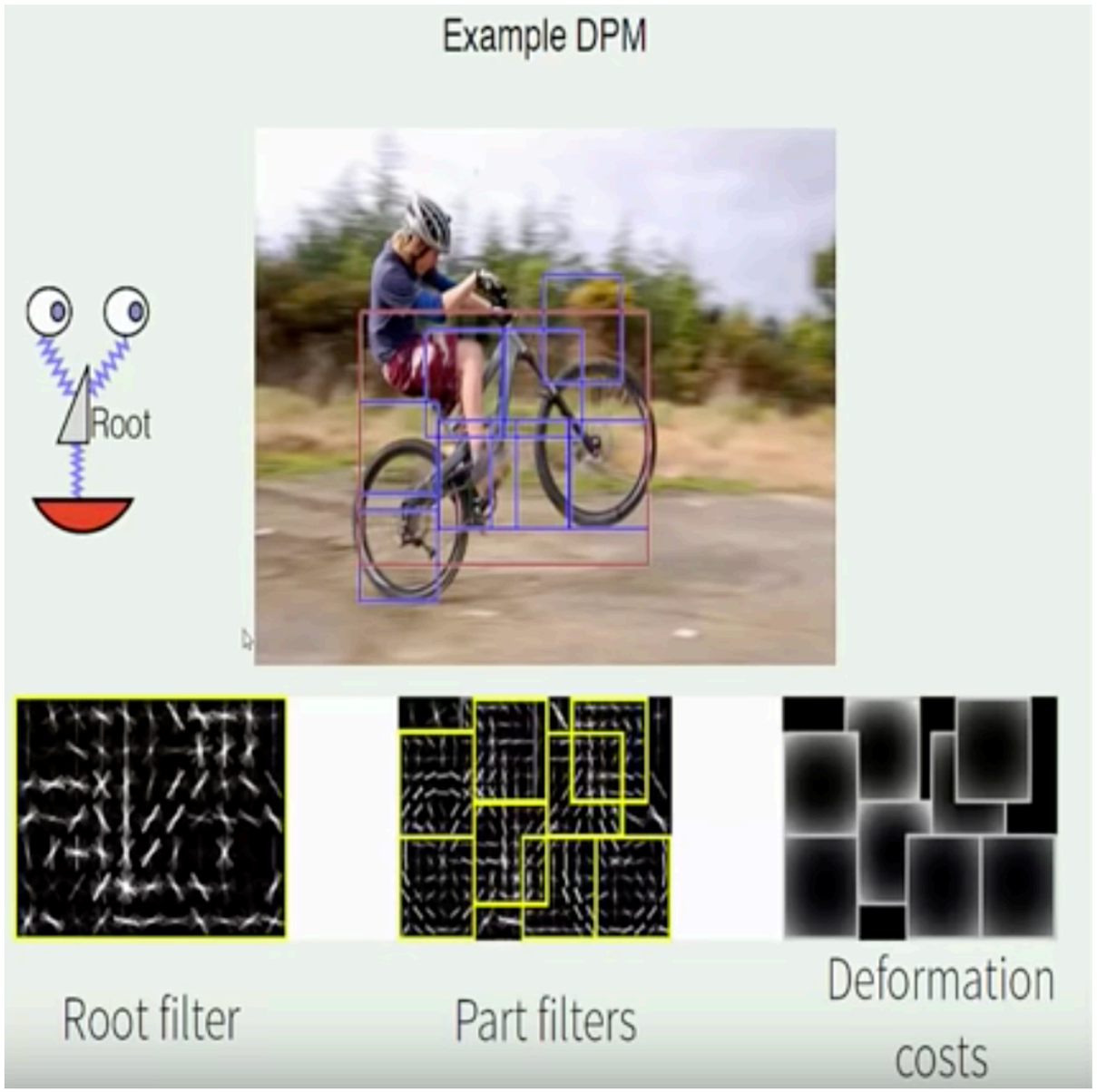
- Pictorial structures
- Weak appearance models
- Non-discriminative training

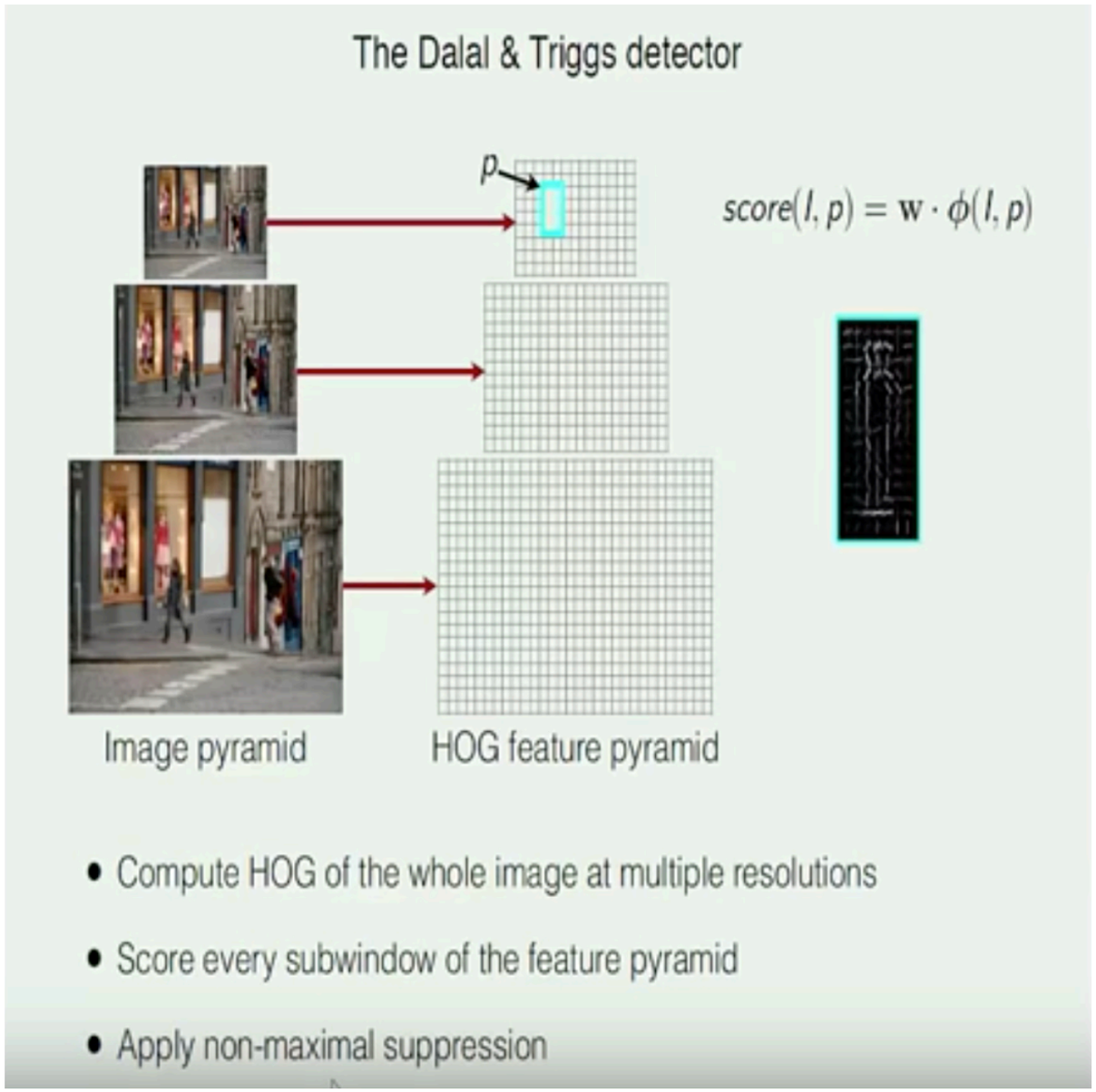
DPM key idea

Port the success of Dalal & Triggs into a part-based model



DPM (2)


Example DPM
Root
Root filter
Part filters
Deformation costs

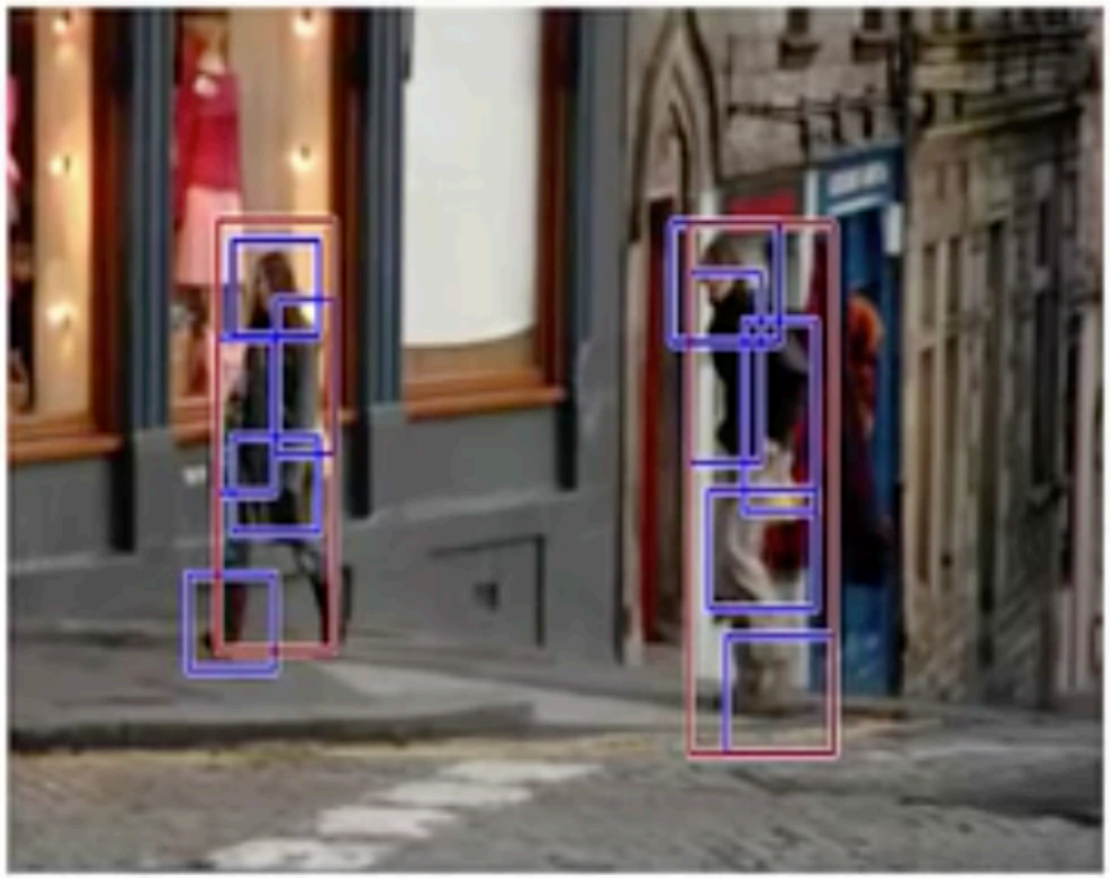

The Dalal & Triggs detector
p
score(l, p) = w · φ(l, p)
Image pyramid
HOG feature pyramid
• Compute HOG of the whole image at multiple resolutions
• Score every subwindow of the feature pyramid
• Apply non-maximal suppression

DPM (3)

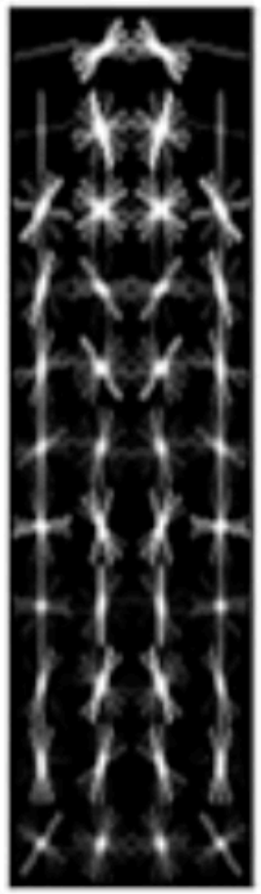
Why DPM?

- Great variations in objects
  - Non-rigid deformations, e.g. human in different poses
  - Intra-class variability, e.g. cars in various shape
  - Variations caused by different viewpoints and illumination
- A better representation of objects – deformable part model (DPM)
  - The Dalal & Triggs detector acts as the root filter in DPM

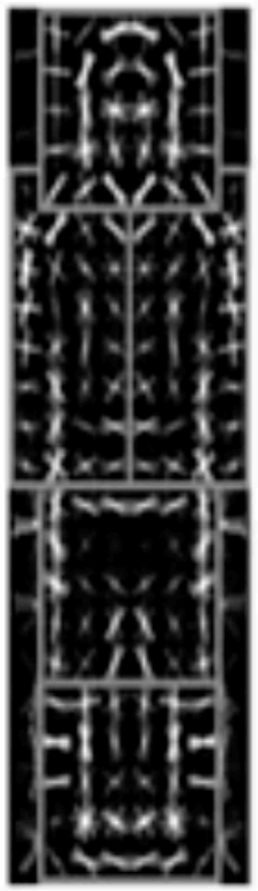
DPM (4)



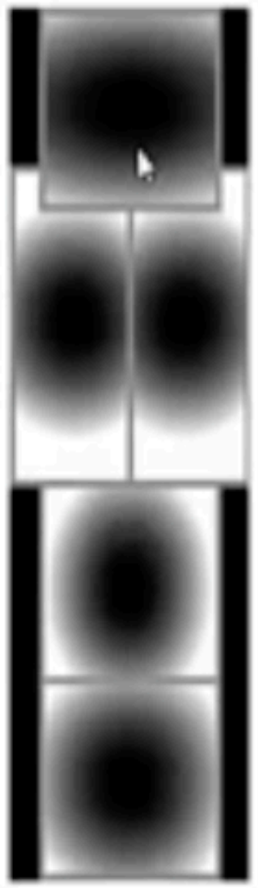
Example detection result



Coarse root filter



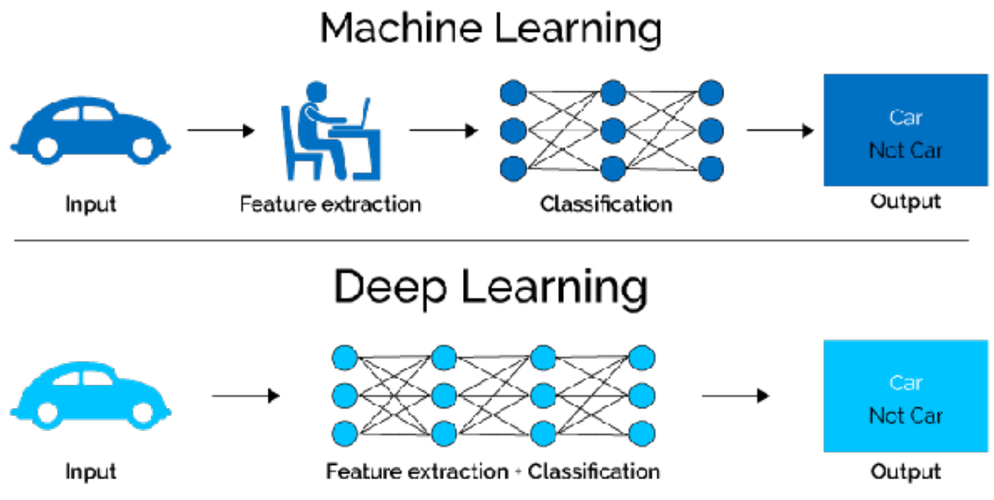
Higher resolution part filters



Deformation models

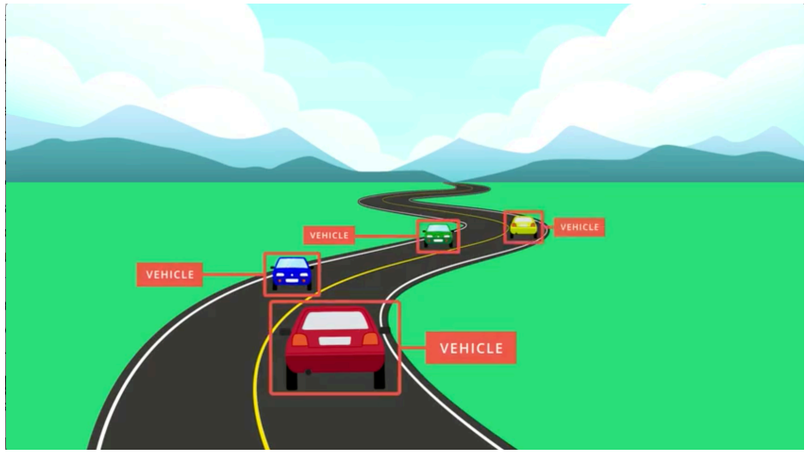
Traditional CV (ML) example: **HOG** (FE) + SVM (Class)

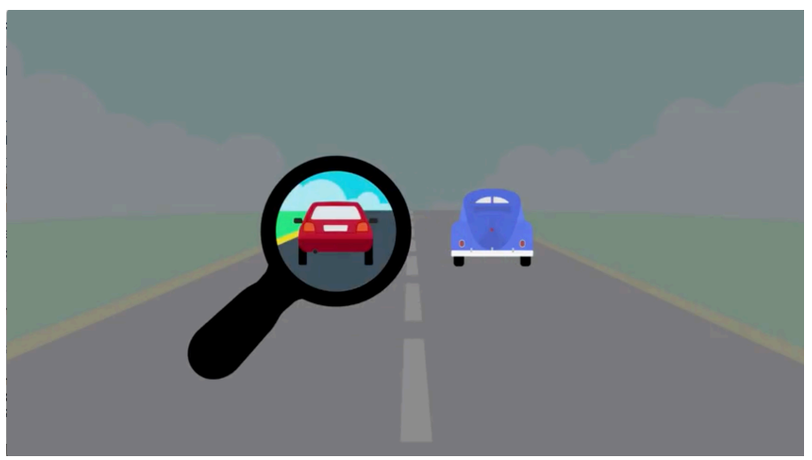
- «Traditional» CV = Feature extraction + Classification (ML)
  - Application example: vehicle detection (& tracking)
  - Labeled data
  - Feature extraction, e.g. HOG
  - Classification, e.g. SVM
  - Sliding window approach
  - Tracking



Object Detection

- How to **find** what your looking for.
- **Difficult**: where, size, how many?
- Focus: **vehicles** (pedestrians, traffic signs, etc.). Knowing where they are and anticipate where they will go.
- Visual **features** -> classify as vehicle, search for detections, track from frame to frame in video
- **Essence** of CV: recognizing what's in an image (letting the car see the world like we do)





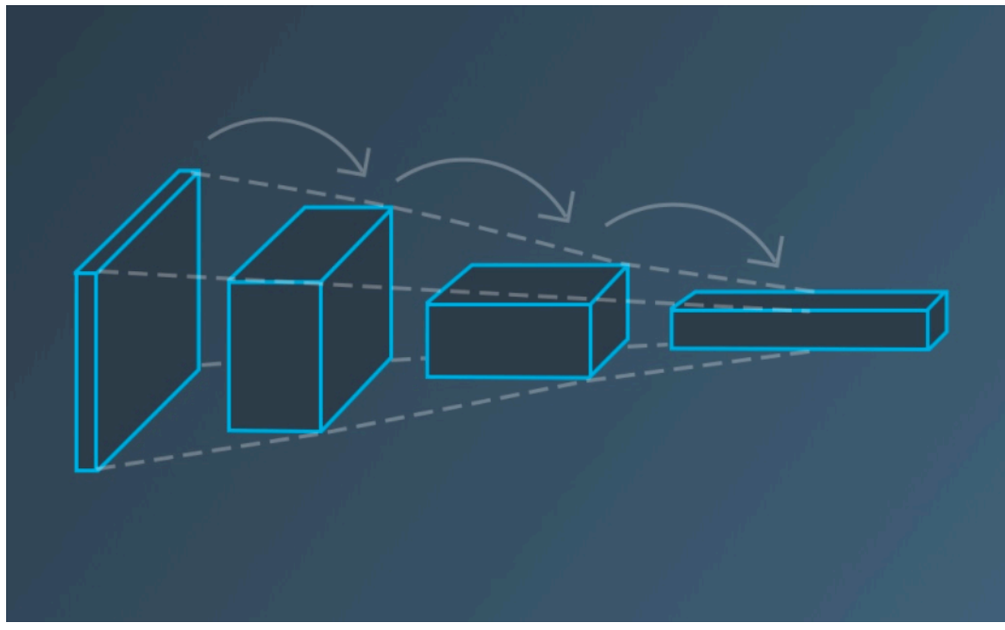
Both approaches..

- Traditional techniques for CV and DL approaches.

- Traditional CV: tune parameters by hand, **intuition** about what works and why.

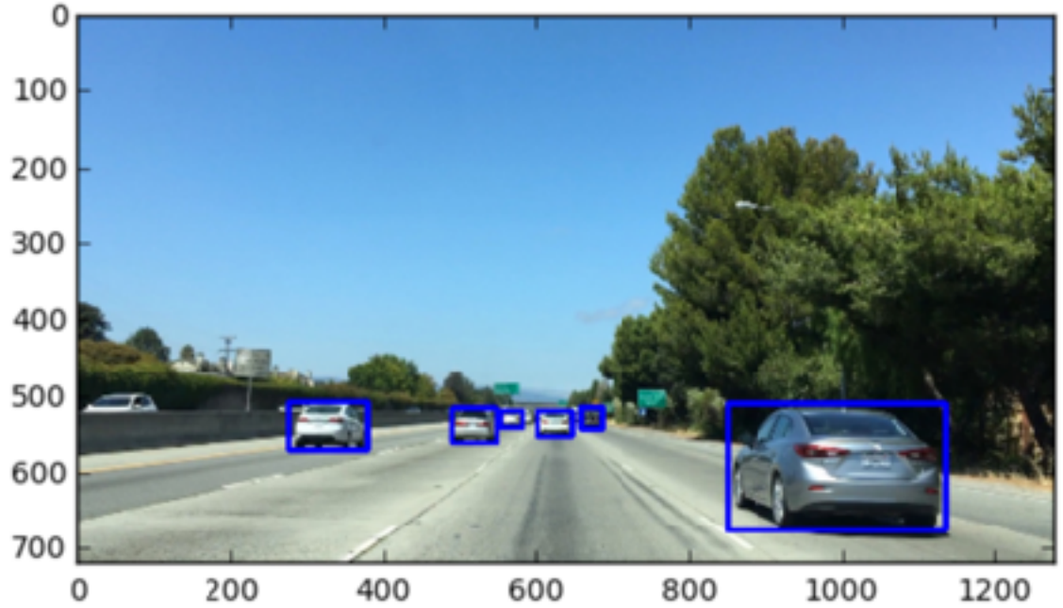
- DL for CV: often works **better** (but not everything, yet), magic / black box.

- Learning both: maximum **insight** and best **performance**.



«Human vehicle detector»

- Easy but **time** consuming
- **Need** labeled training data
- cv2.rectangle(*)

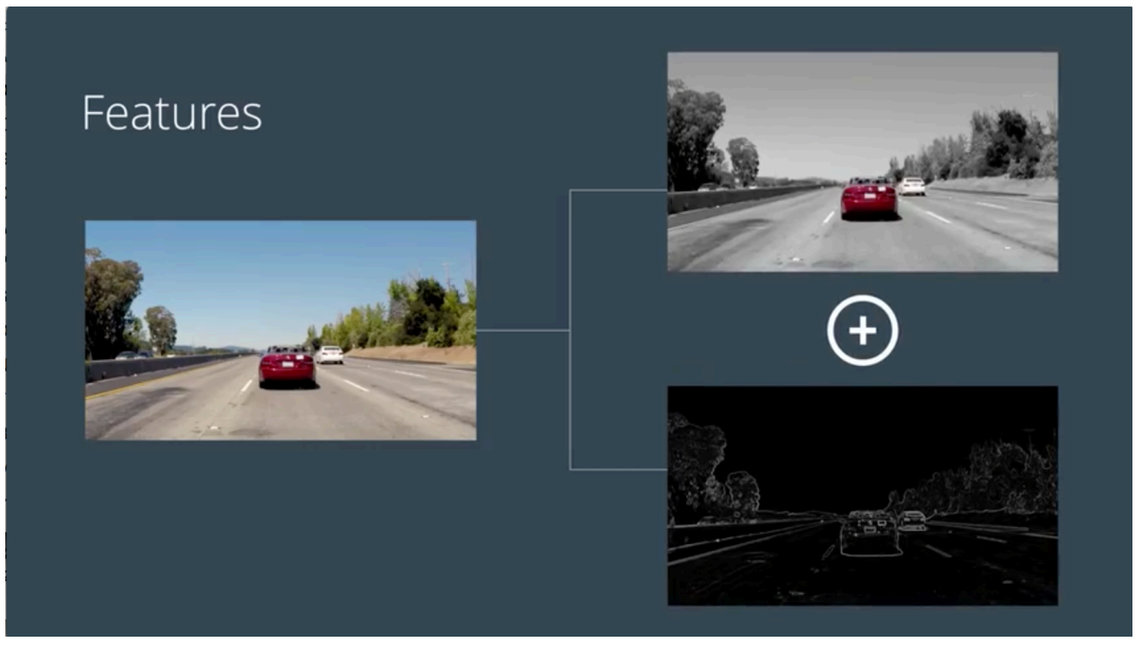


- What **characteristics** are useful to identify a car?

*) http://docs.opencv.org/2.4/modules/core/doc/drawing_functions.html

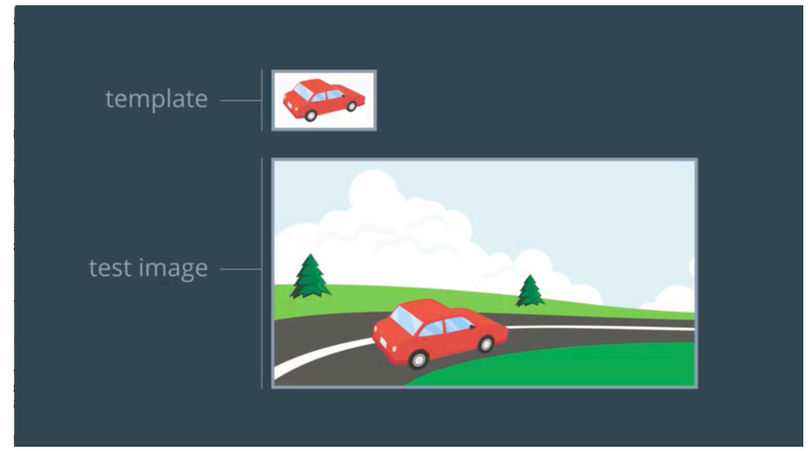
Feature intuition

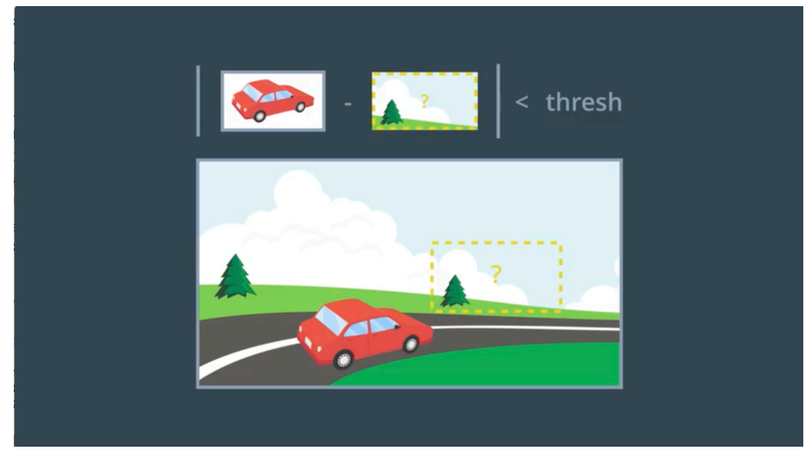
- What **differentiate** a car from the rest in an image?
  - Color
  - Shape
- Features:
  - Pixel **intensity** (raw)
  - **Histogram** of pixel intensity
  - **Gradients** of pixel intensity

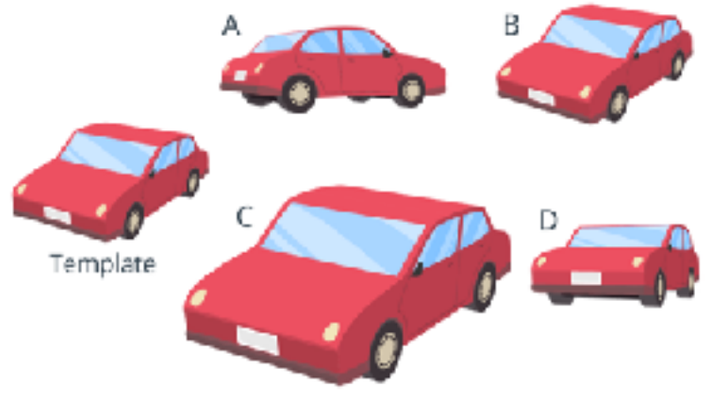


Feature extraction (FE): Template

- Simplest: intensities alone
- Given a **template** of what we want to find (e.g. car).
- Does a **new region** contain what we want to find (car)?
- Threshold diff template and various regions.
- Challenges?

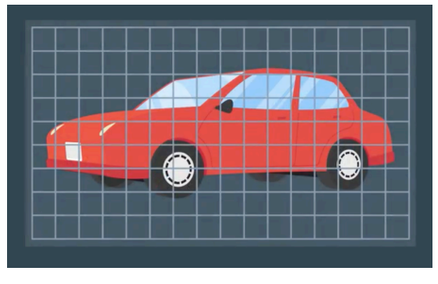






FE: Template

- **Machine** detector (vs. human detector):
  - Given 6 (wanted) car templates
    - **Search** (surveillance) images for matches
- Challenge: Usually we don't know **exactly** what you are looking for.
- cv2.matchTemplate(*)

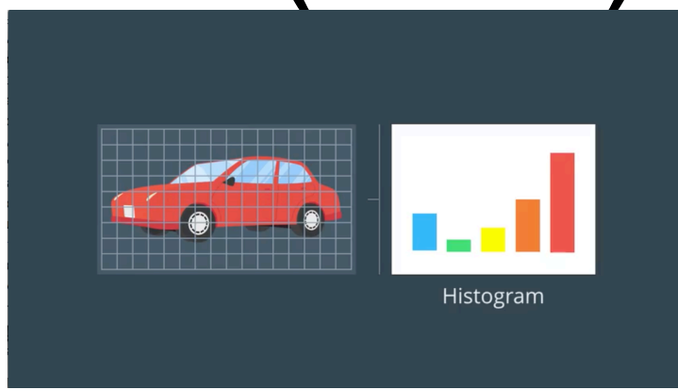


*) http://docs.opencv.org/2.4/modules/imgproc/doc/object_detection.html

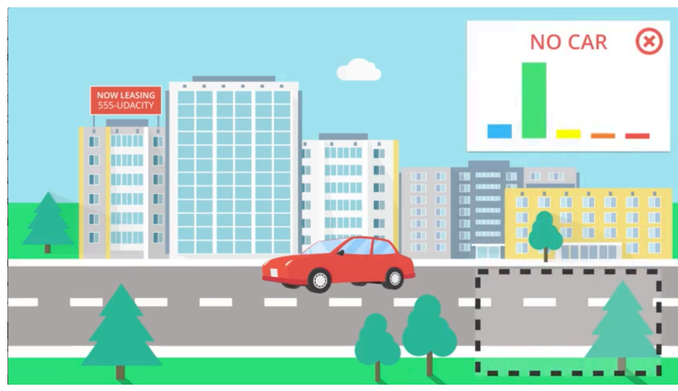
**) http://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_imgproc/py_template_matching/py_template_matching.html

FE: Histogram of Intensities (HoI)

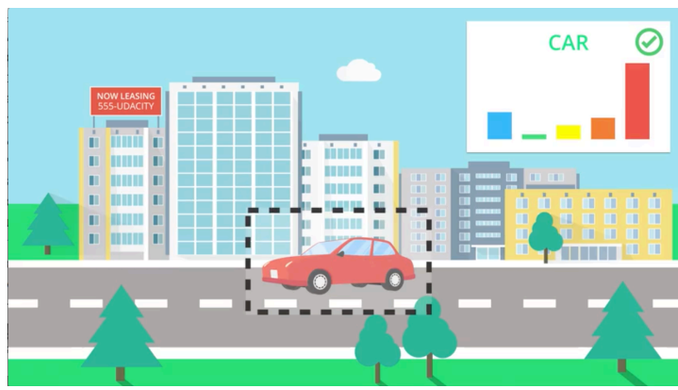
- Need a transform that is more **robust** in terms of **appearance**, e.g. HoI (normalized).



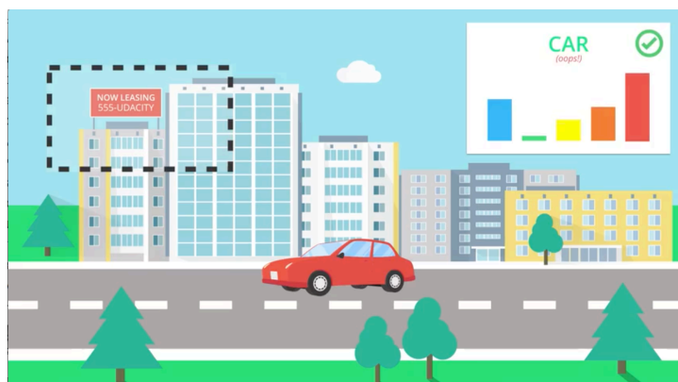
- **Compare** HoI for target object with regions from a test image.



- Similar color distributions will result in a close **match** (not **sensitive** to a perfect arrangement of pixels any more, e.g. size and orientation).

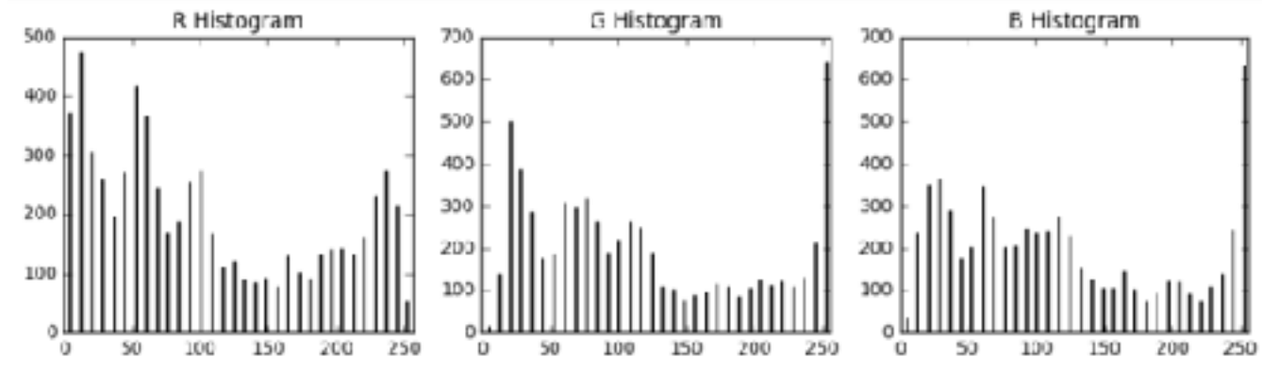


- But distribution might match some **unwanted** region as well.



FE: HoI (2)

- Template region shown earlier.
- Histograms of the R, G, and B **channels**
- Specify the number of **bins** (e.g. 32)
- Concatenate into a **feature vector**



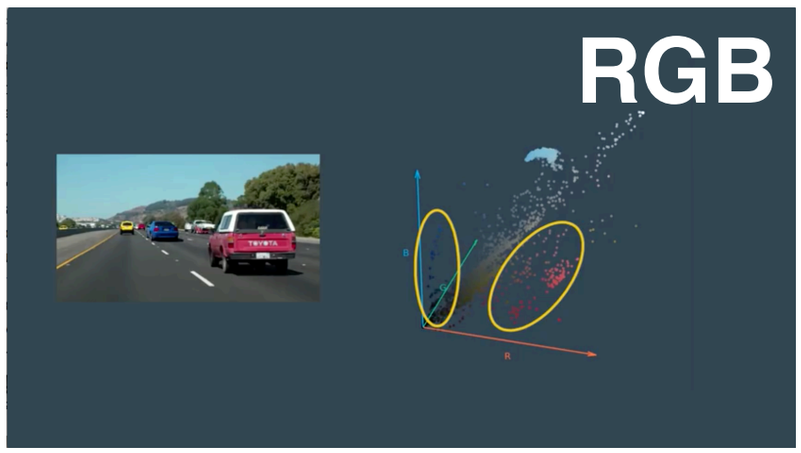
Color spaces and Invariance

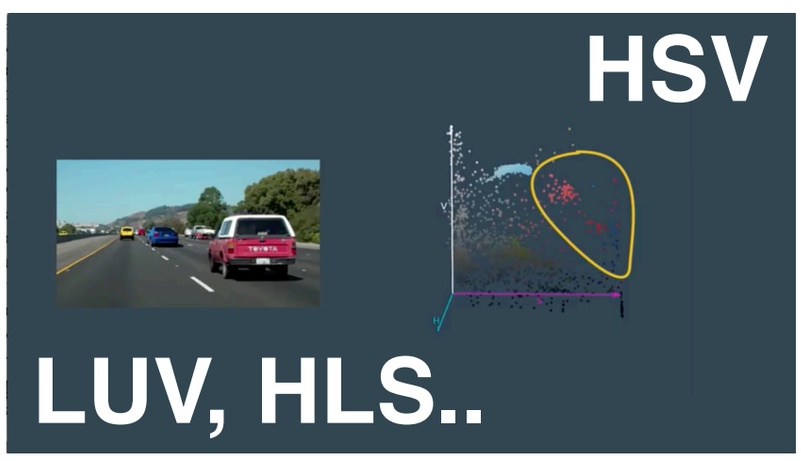
- Challenge (raw intensities / colors or histograms of those): Same class but **different colors**.

- Looking for features that make your object of interest stand out: maybe **other color spaces** can help out (dim **reduction**)

- Common to look at many **car / no-car** images in various color spaces (hand crafted features)

- cv2.cvtColor()





FE: raw pixel intensities

Original | 64 x 64 pix | 32 x 32 pix

- **Template** matching **not robust**, raw intensities **still useful** but would generate a **long** feature vector.
- **Downsampling** (spatial binning)
- How long will a feature vector based on the raw intensities be?
- cv2.resize()

FE: Histogram of Oriented Gradients (HOG)

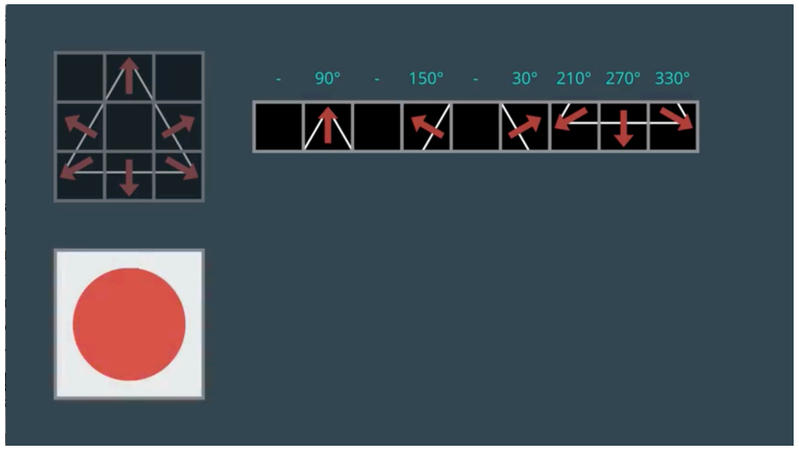
-1	0	1

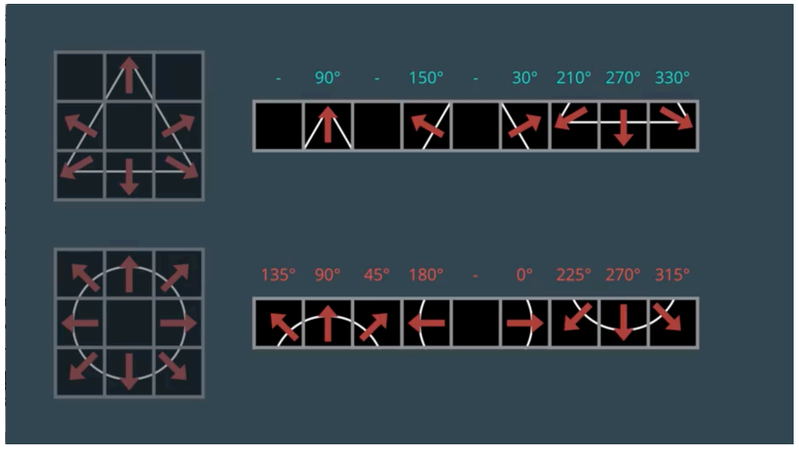
-1
0
1

$$g = \sqrt{g_x^2 + g_y^2}$$

$$\theta = \arctan \frac{g_y}{g_x}$$

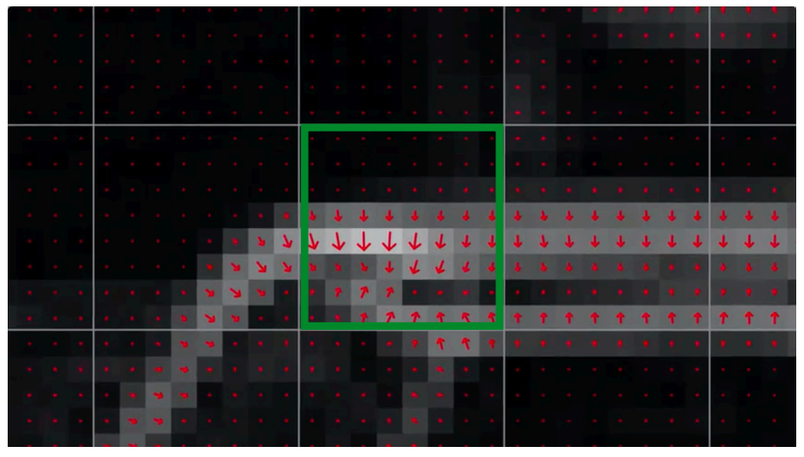
- So far: Manipulating intensities and generating **Features** like:
  - raw intensities (down-sampled, proper color space).
  - Histogram of intensities (distribution)
- What about **Shape** and gradients.
- To understand the **method**: simple shapes, gradient, grid cells, flat array, signature for triangle or circle.
- To **sensitive**. Need to be distinct and **flexible** (to capture variability)

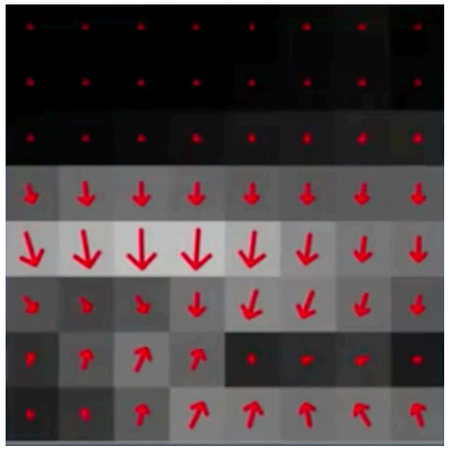




FE: HOG (2)

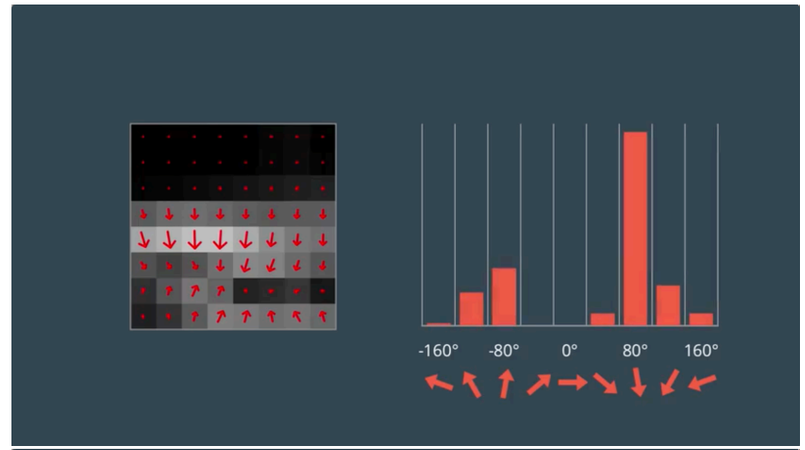
- Realistic image (**region**): 64*64 pixels, blown up
- Compute **gradient** magnitude and direction
- **Group** into cells of 8 times 8 pixels
- Inside each **cell** we now have 64 pixels each containing a gradient magnitude and direction or **orientation**.



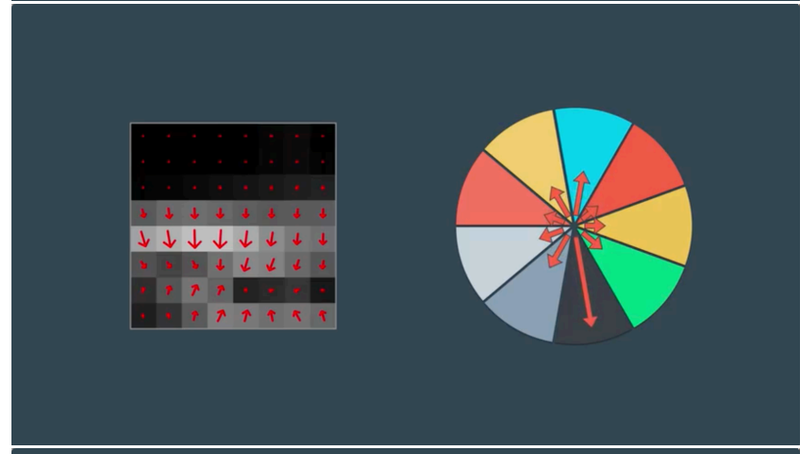


FE: HOG (3)

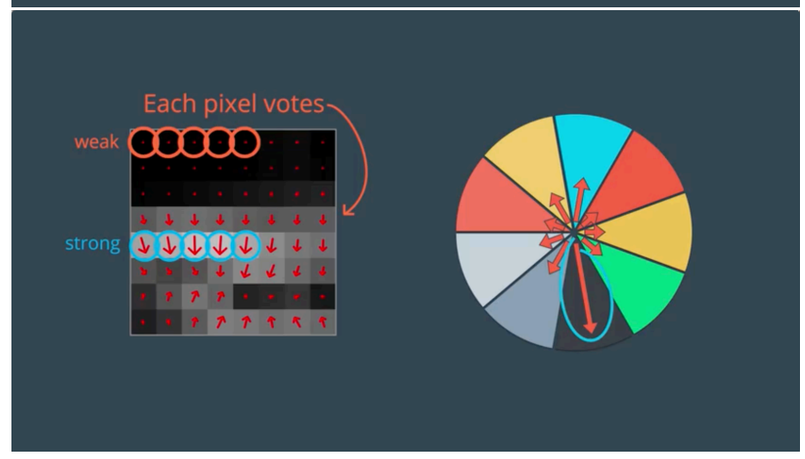
- From this we compute a **histogram** of oriented gradients (HOG). We use **9 bins** here. «Weighted» votes



- Alternatively this can be visualized like a **wheel** with a dominant vector for the cell.

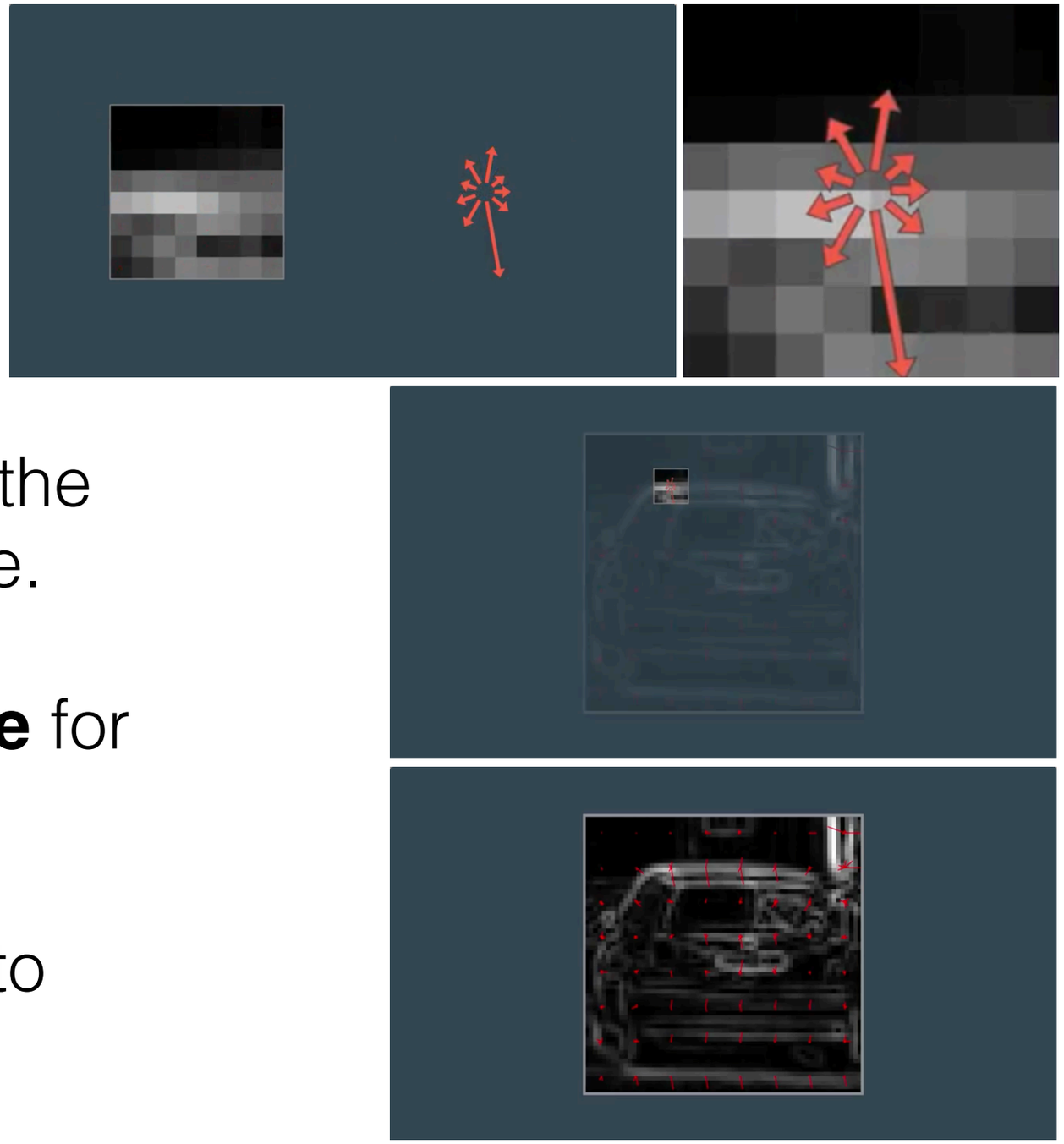


- Each pixel (64 here) in the cell **vote** on an **orientation** and the contribution is determined by the gradient **magnitude**.



FE: HOG (4)

- We do this for **all cells**.

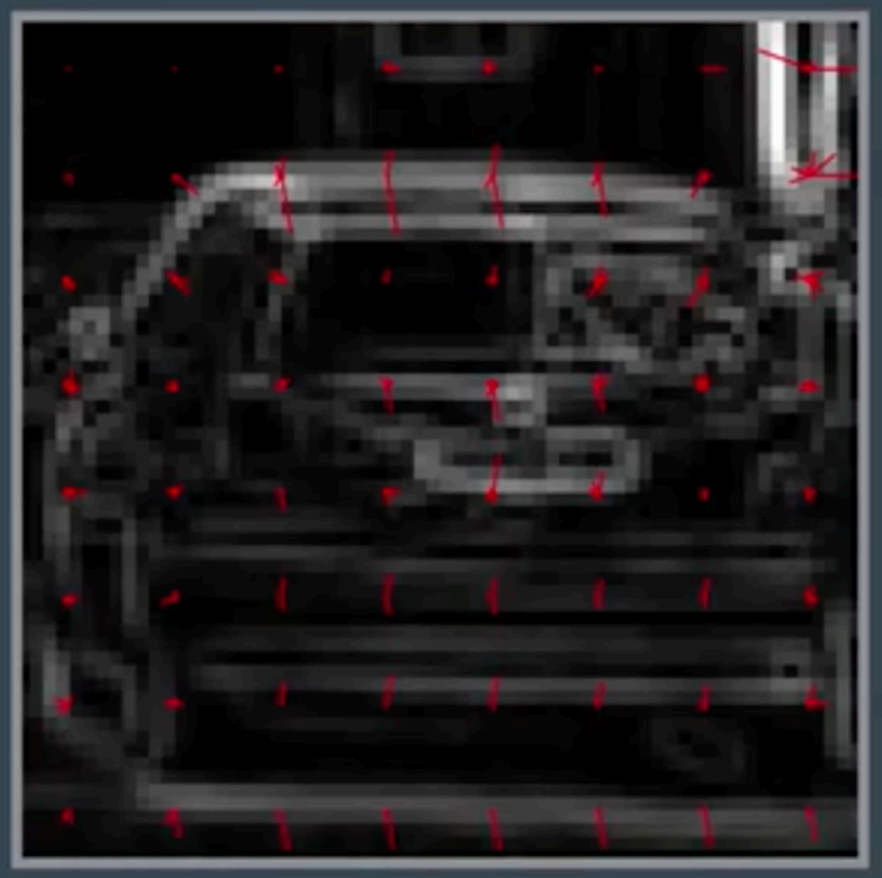


- Finding and displaying the **dominant** gradient in each cell the original structure start to emerge.

- This can be used as a **signature** for a given **shape** (HOG feature)

- Advantages: build in the ability to **accept small variations** in the shape

FE: HOG (5)



HOG Features

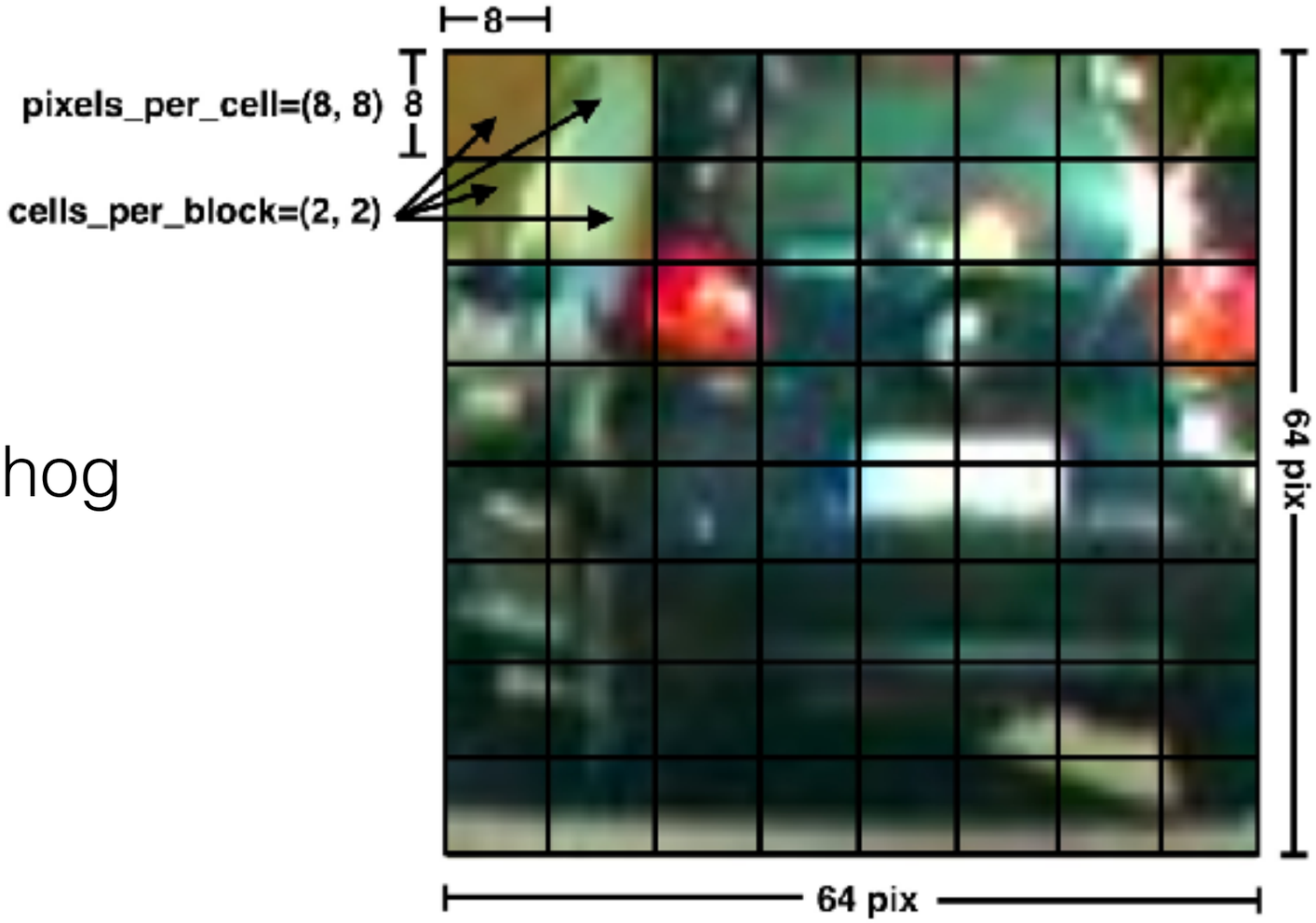
Robust to variations in shape

Parameters to tweak:

- # of orientation bins
- grid of cells
- cell sizes
- adding overlap between cells
- block normalization

FE: HOG (6)

- http://scikit-image.org/



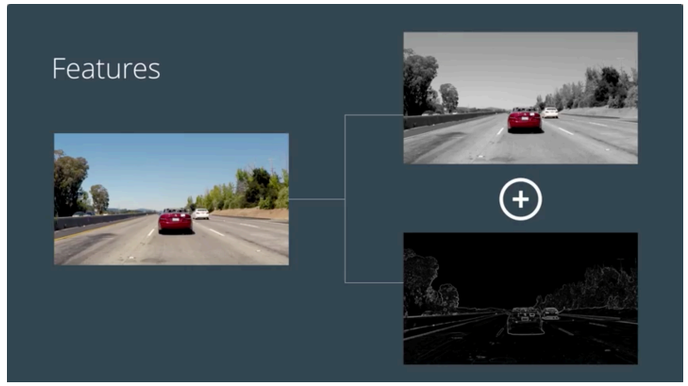
- **from** skimage.feature **import** hog
- hog()
  - orientations=9
  - pixels_per_cell=(8, 8)
  - cells_per_block=(2, 2)
- size(HOG feature vector) ?

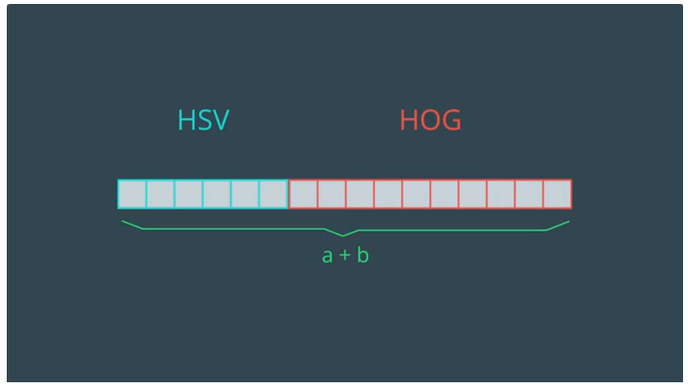
http://scikit-image.org/docs/dev/api/skimage.feature.html?highlight=feature%20hog#skimage.feature.hog

http://scikit-image.org/docs/dev/auto_examples/features_detection/plot_hog.html

http://scikit-image.org/docs/dev/auto_examples/features_detection/plot_hog.html

Combine & Normalize Features

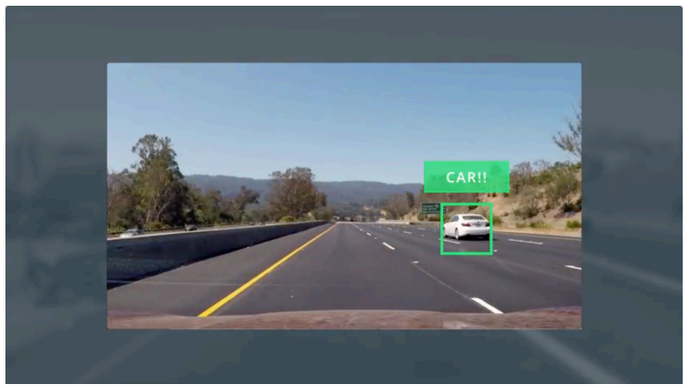


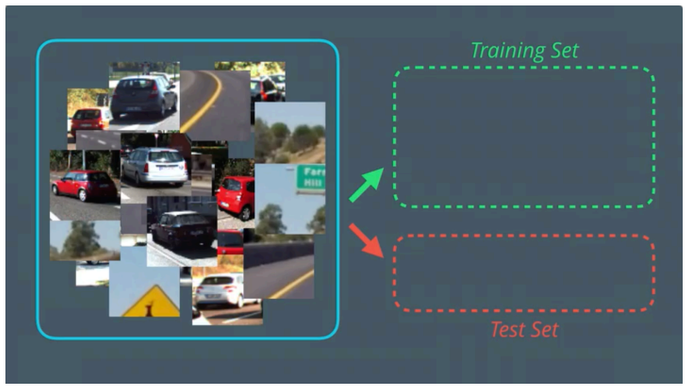


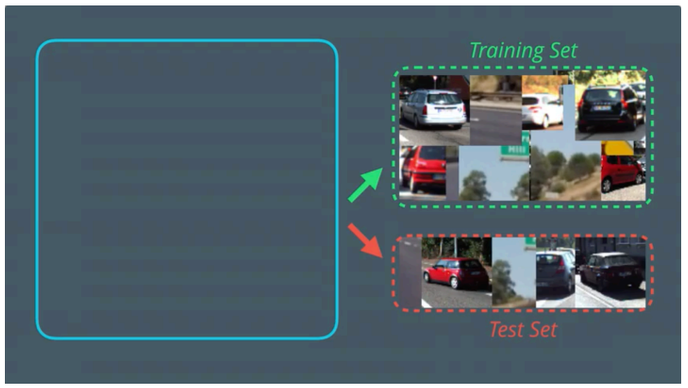
- **Combine** color / intensity and shape based features (complement, more robust).
- **Normalize** the combined feature vector
- **Balance**? (approx. same # of el. in a & b)
- Remove **redundant** elements?

Classification: (Labeled) Data

- **Know** how to **find features** now.
- Want to use this to **train** a classifier to distinguish between cars and no car (background)
- For this we need (a lot of) **data** (example from both classes)
  - Crop out, same size etc.
  - Balanced
  - Training / Test (random, also balanced)
- We also need a **sliding window imp.**







Classification: Training & Testing

- Training phase: 1) **extract** features from each sample in the training set. 2) **Supply** these feature vectors together with the labels to the training alg. of the chosen classifier.

- Initialize a **model** and tweak it's parameters using features and labels (iterative, calc. error and update par.)

- **Verify** how the model performs on unseen data from the test set (larger error, drops, generalize)

- But **what classifier** should we use. Experimentation. HOG+SVM ok

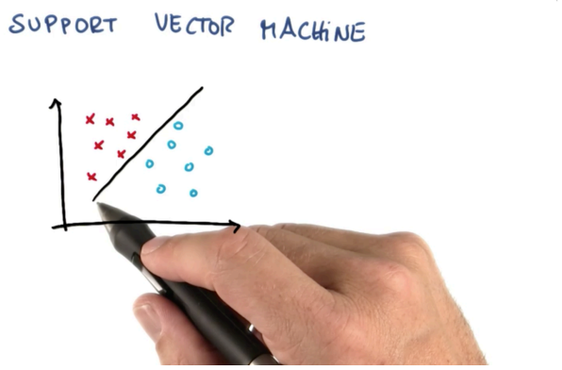
Classification: SVM

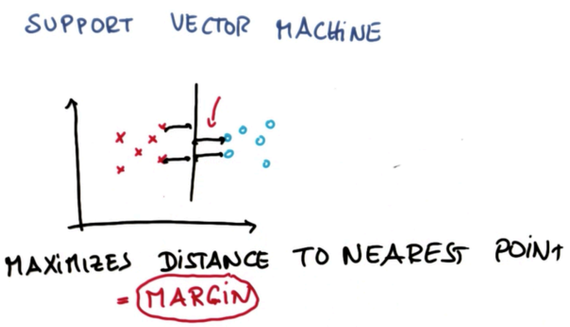
- **Given**: many feature vectors, **Find**: decision boundary (line or hyperplane). **Margin** (robustness).

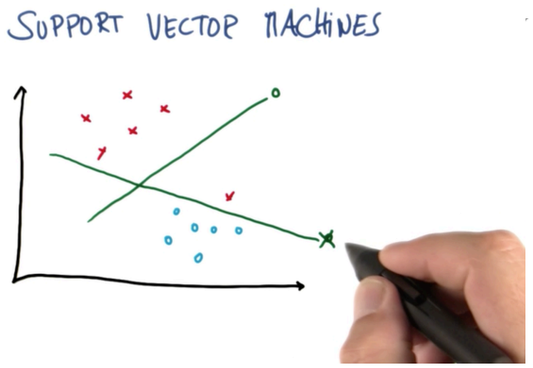
- **Difficult** data (correct first). **Outliers** (robust)

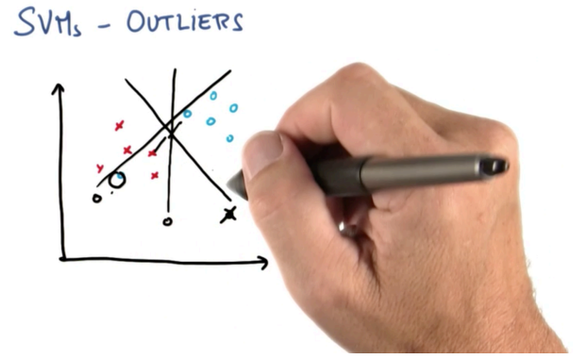
- **Non-Linearly separable**, new variable

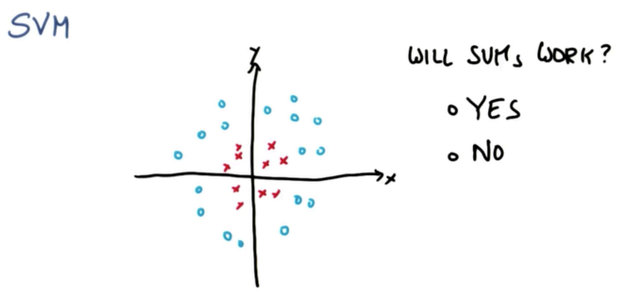
- **Kernel-trick** and non-linear SVMs, parameters and overfitting

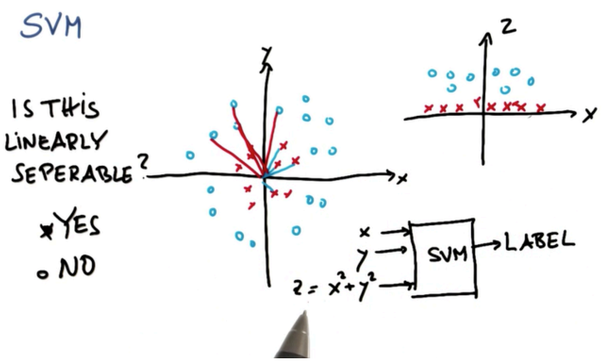


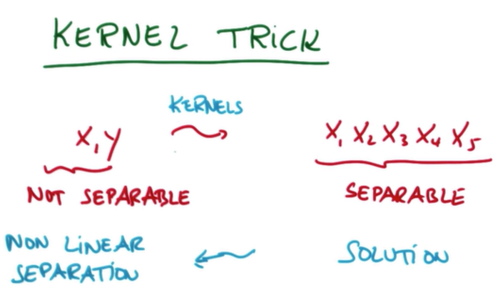


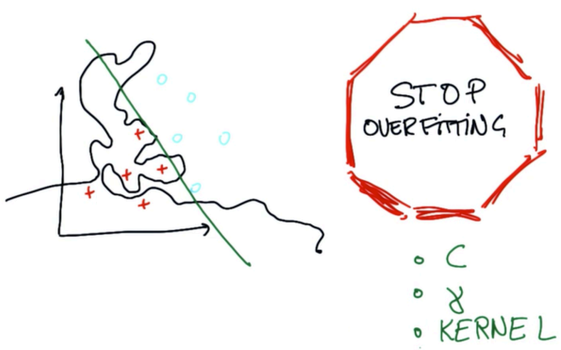






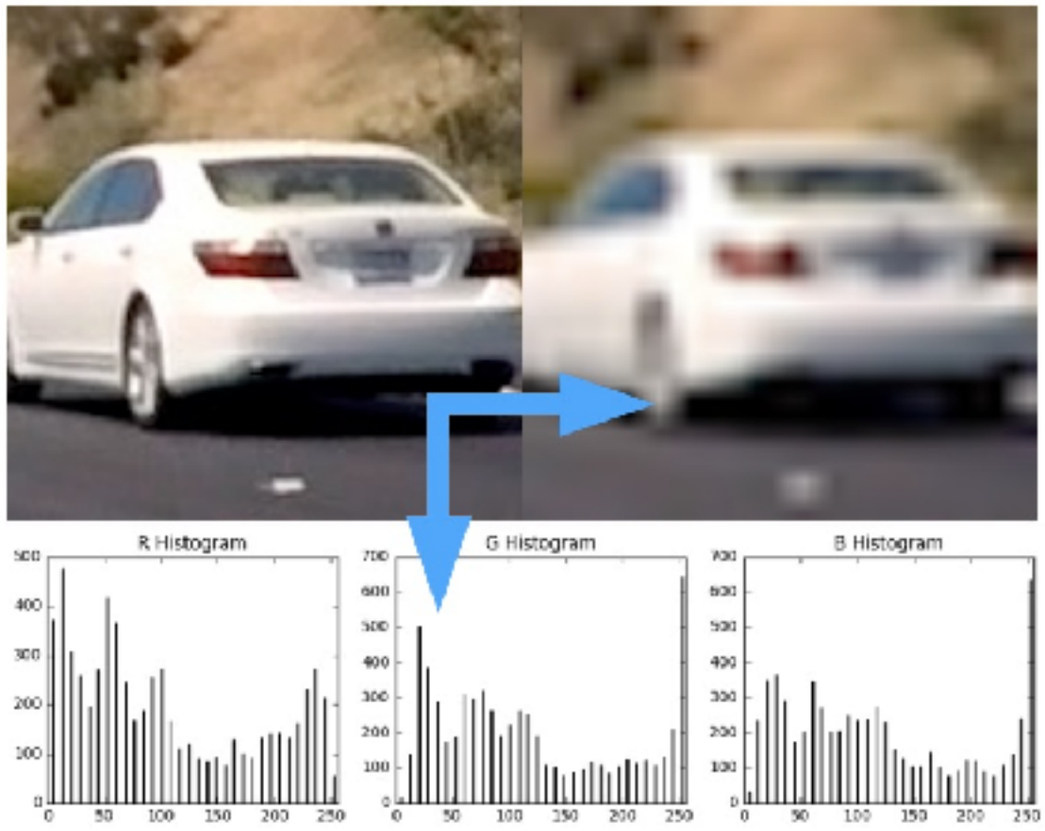






Classification: Example

- Procedure:
  - Read all car / non-car images (**data**).
  - **Extract** feature vectors: Intensities + Histogram of Intensities (in optimal color space).
  - **Combine**, normalize and scale (zero mean and unit variance) all feature vectors
  - Defina a **label** vector (binary here)
  - **Split** data into training and test sets
  - **Train** your favorite classifier (e.g. a lin SVM)
- **Results**: ok. Alternatives: just HOG or this + HOG



- **from** sklearn.svm **import** LinearSVC
- svc = LinearSVC()
- svc.fit(X_train, y_train)
- svc.score()
- svc.predict()

Putting it all to use: Sliding window

- **Search** for the target object using a **sliding** window approach

- How **many** windows at what **size** and the amount of **overlapp**.

- Where in the image would you **start and stop** the search.

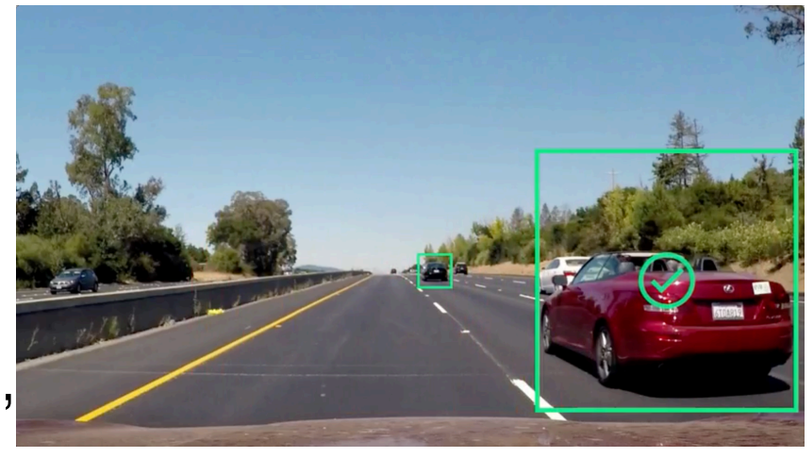
- **#** regions/windows scale with image size, window size and overlap.

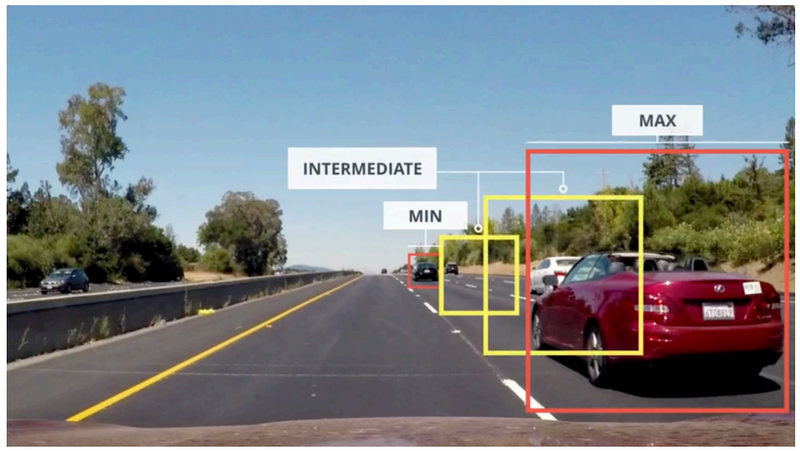
Multi-scale windows

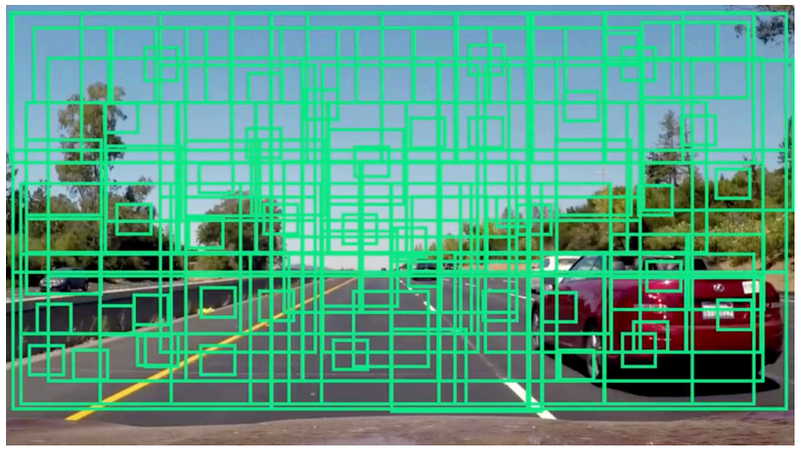
- Dont know the **size** of the object you search for. Search on multiple **scales**, min and max scales.

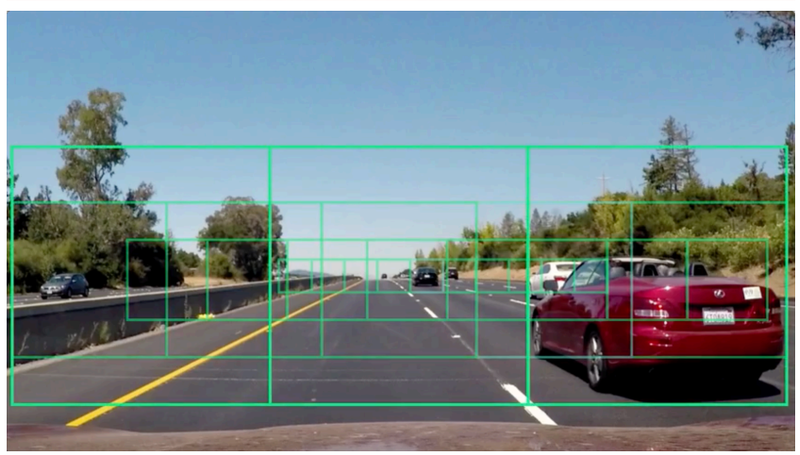
- Total search windows can become huge, decreasing **performance**. Only search **part** of the window.

- Small objects will appear near the horizon and big objects will appear close.







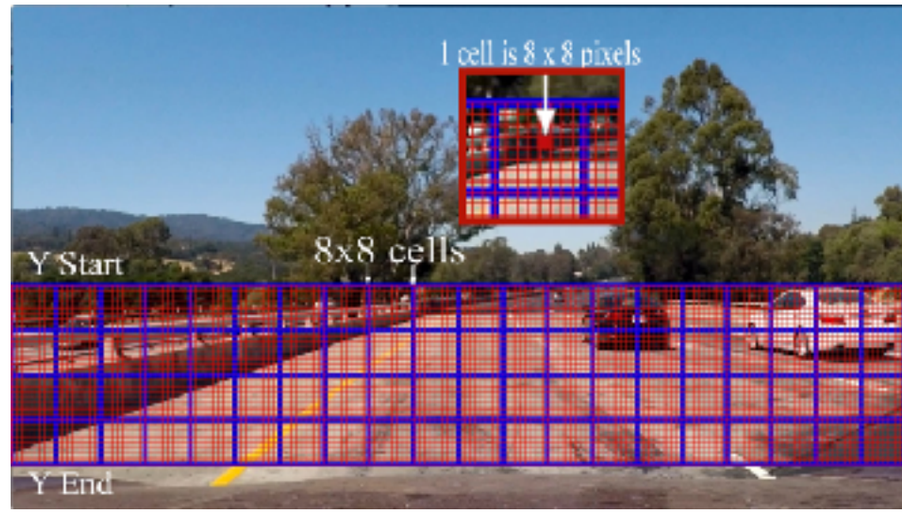


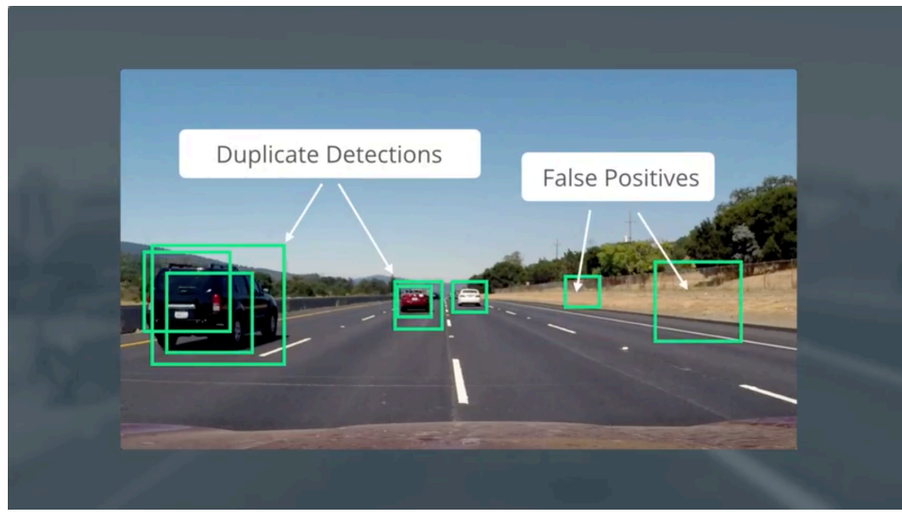
Challenges

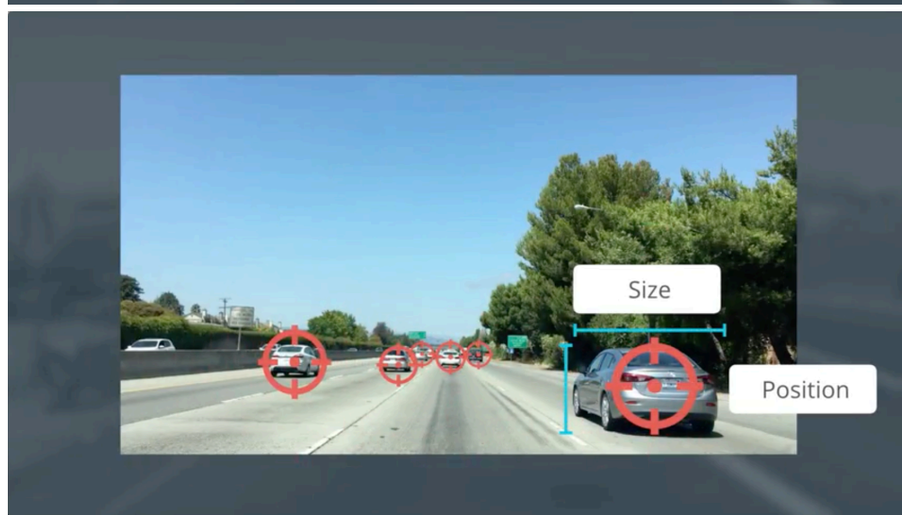
- **Performance**, i.e. only extract HOG features once, for relevant parts of the image. Sub-sample for detection on multiple scales.

- Multiple overlapping **duplicate** detections & **False Positives** (brakes)

- Robust and accurate estimates / **tight** bounding box (not running into other cars). **Important** for path planning and motion control.







Tracking

- Track detected objects trough a **video** stream.
- Alt1: Run detection on **each frame**
- Alt2: Utilizing what you **know** about one frame when processing the next frame.
- **Observation**: the target object moves just a little bit from frame to frame (search area, window size, scale, etc.)

Detection & Tracking: Overall pipeline

- In each frame: **search** for vehicles using a sliding window technique and **record** the bbox position of positive detection.

- When you get **multiple** overlapping detections of the same object use the **centroid** of these windows.

- **False positives** can be filtered out by looking at what detections **appear** in one frame but not in the next.

- Once you have a good detection you can record how the **centroid moves from frame to frame** and eventually **estimate** where it will appear in each subsequent frame.

Summary

- «Traditional» CV = Feature extraction + Classification (ML approach)
  - Need to decide which **features** to use, intensities / color and gradient based, experimentations usually needed.
  - Need to choose and train a **classifier**, linear SVM is a good tradeoff between speed and accuracy, experimentations with other classifiers can be useful.
  - Use a **sliding window** approach to search for your object of interest, multi-scale but minimize the amount of search and handle duplicate detections and false positives.
  - Having a robust detection you can now **track** this detection in a video stream minimizing the amount of search needed.