CS558: Computer Vision Notes

Steven DeFalco

January 7, 2024

Contents

1 Introduction (Lecture 1)

1.1 Why is vision hard?

- There is a loss of information due to projection from 3D to 2D
- Image colors depend on surface properties, illumination, camera response function, and interactions such as shadows
- Noise (e.g. sensor noise)
- Conflicts that exists among local and global clues (e.g. illusions)

1.2 Cameras

Cameras use a barrier to block of most of the light rays. This reduces blurring. The opening is known as the ***aperture***.

Lens and viewpoint determine perspective. Aperture and shutter speed determine exposure. Aperture and other effects determine depth of field. Film or sensor record image.

The ***pinhole model*** captures *pencil of rays* - all rays through a single point. The point is called the ***center of projection (COP)***. The image is formed on the ***image plane***. ***Effective focal length f*** is the distance from the COP to the image plane. The main idea with a painhold camera is to add a barrier to block off most of the rays: this reduces blurring.

Can't just make the aperture as small as possible because this causes less light to get through and the image will experience diffraction effects. When light changes medium, the directioon changes: diffraction.

A lens focuses light onto the film. There is a specific distance at which objects are *in focus*; other points project a *circle of confusion* in the image. Changing the shape of the lens changes this distance.

Thin lens optics is a simplifications of geometrical optics for well-behaved lenses. All parallel rays converge to one points on a plane located at the focal length f. All rays going through the center are not deviated; hence have the same perspective as the pinhole.

2 Image Formation (Lecture 2)

Field of view is how much volume is observed by a camera. As you increase the ***focal length***, we observe a narrower and narrower view of the scene; the field of view (FOV) decreases. Wheras at 17mm we may see 104°, at 1000mm

we may see 2.5°.

The field of view is govered by the size of the camera retina.

$$\phi = \tan^{-1}(\frac{d}{2f})$$

Telephoto makes it easier to select background (a small change in viewwpoint is a big change in background). Changing the focal length lets us move back from a subject, while mainting its size on the image, but moving back changes perspective relationships.

Definition 2.1 (Aperture) is the diamter of the lens opening (controlled by diaphragm). Expressed as a fraction of coal length, in ***f-number*** N. For example, $f/2.0$ on a 50mm lens means that the aperture is 25mm. $f/2.0$ on a 100mm lens means that the aperture is 50mm. A small f-number corresponds to a big aperture.

Remark Aperture controls depth of field. A smaller aperture increases the range in which the object is approximately in focus. But a small aperture reduces amount of light.

The ***Circle of Confusion*** (C) is an optical spot caused by a cone of light rays from a lens not coming to a perfect focus when imaging a point source and it depends on sensing medium, reproduction medium, viewing distance, human vision, etc.

Remark Telephoto makes it easier to select background (a small change in viewpoint is a big chagne in background). Changing the focal length lets us move back from a subject, while maintaining its size on the image, but moving back changes perspective relationships.

Shutter speed controls how long the film/sensor is exposed. It has a pretty much linear effect on exposure. It is usually in a fraction of a second: 1/30, 1/60, 1/125, 1/250, 1/500. On a normal lens, normal humans can hand-hold goes down to 1/60. The main effect of shutter speed is motion blur. Halving the shutter speed doubles the motion blur.
Exposure has two main parameters...

- aperture (in f number)
- shutter speed (in fractionm of a second)

Exposure = irradiance × time.

$$H = E \times T$$

Remark Iradiance (E) is controlled be aperture. Exposure time (T) is controlled by shutter.

Definition 2.2 (Reciprocity) The same exposure is obtained with an exposure twice as long and an aperture are half as big.
Assume we know how much light we need, then we have the choice of an infinite number of shutter speed/aperture pairs.

Remark What will guide our choice of a shutter speed?

- Freeze motion vs. motion blue, camera shake

What will guide our choice of an aperture?

- Depth of field, distortion reduction, diffracion limit

Often we must compromise and open more to enable faster speed (but shallow depth of field).

Definition 2.3 (Metering) Photsensitive sensor measure scene luminance (usually through the lens). A simple version of this would be a center-weighted average. It usually assumes that a scene is 18% gray which causes a probelm with light and dark scenes.

The camera metering system measures how bright the scene is. In ***aperture priority mode***, the photgrapher sets the aperture, the camera sets the shutter speed. In ***shutter-speed priority mode***, the photographer sets the shutter speed and the camer deduces the aperture. In both cases, reciprocity is exploited. In ***program mode*** the camera has almost complete control of the setttings. In ***manual mode***, the photographer has complete control of the settings.

Remark In ***aperture priority*** mode, there is direct depth of field control, but this can require impossible shutters speeds for a bright scene. In ***shutter speed priority*** mode, there is direct motion blur control, but this can require impossible aperture for a dark scene.

Sensitivity (ISO) is the gain applied to the sensor. It has a linear effect on the image.

A pixel in an image gets its value from some factors which include illumination strength and direction, surface geometry, surface material, nearby surfaces, and camera gain/exposure.

There are two basic models of reflection:

- ***Specular***: light bounces off at the incident angle (e.g. mirror)
- ***Diffuse***: light scatters in all directions (e.g. brick, cloth, rough wood)

In the ***Lambertian reflectance model***, some light is absorbed and the remaining light is scattered. Examples include soft cloth, concrete, matte paints. Most surfaces have both specular and diffuse components. ***Specularity*** is the spot where specular reflection dominates (typically reflects light source).

Definition 2.4 (Lambert's Cosine Law) Intensity *does not* depend on viewer angle.

- Amount of reflected light proportional to $\cos(\theta)$
- Visible solid angle also proportional to $\text{cost}(\theta)$

Most surfaces have both specular and diffuse components. ***Specularity*** is a spot where specular reflection dominates (typically reflects light source). ***Intensity*** depends on illumination of angle because less light comes in at oblique angles.
ρ = albedo, S = directional source, N = surface normality, I = reflected intensity

$$I(X) = \rho(x)(S\dot{N}(x))$$

Remark When light hits a typical surface...

- Some light is absorbed $(1-p)$: more absorbed for low albedos
- Some light is reflected diffusely: independent of viewing direction
- Some light is refelcted specularly: light bounces off (like a mirror), depends on viewing direction

Bidirectional reflectance distribution function is a model of local reflection that tells how bright a surface appears when viewed from one direction when light falls on it from another. It is a ration of measured outgoing radiance in direction $\theta_e, \Phi_e)$ to irradiance from direction $\theta_i, \Phi_i)$.

$$\rho(\theta_i, \phi_i, \theta_e; \lambda) = \frac{L_e(\theta_e, \phi_e)}{E_i(\theta_i, \phi_i)} = \frac{L_e(\theta_e, \phi_e)}{L_i(\theta_i, \phi_i)\text{cos}\theta_i \text{d}\omega}$$

Light is composed of a spectrum of wavelengths. Humans see in RGB. There are red, green, and blue cones in the eyes. RGB is represented a 3D value represented in each direction between 0 and 255. For example, red is R=255, G=0, and B=0.
Observed intensity depends on light sources, geometry/material of reflecting surface, surrounding objects, camera settings, etc. Objects cast light and shadows on each other. Differences in intensity are primary cues for shape.

2.1 Pixels and Linear Filters

Definition 2.5 (Image filtering) for each pixel, compute function of local neighborhood and output a new value. Same function is applied at each position. The output and input images are typically the same size. In linear filtering the function is a weighted sum/difference of pixel values. This idea can be used to enhance images, extract information from images, and detect patterns.

$$h[m,n] = \sum_{k,l} g[k,l] f[m+k, n+l]$$

A ***box filter*** replaces each pixel with an average of its neighborhood. This achieves a smooth effect (removes sharp features).

Remark The sobel filters are $[[1,0,-1],[2,0,2],[1,0,-1]]$ and $[[1,2,1],[0,0,0],[-1,-2,-1]]$.

Convolution is like filtering, but the function definition varies slightly.

$$h[m,n] = \sum_{k,l} g[k,l] f[m-k, n-l]$$

Some key properties of linear filters include the following:

- **Linearity**: filter$(f_1 + f_2)$ = filter(f_1) + filter(f_2)
- **Shift invariance** (same behavior regardless of pixel location): filter(shift(f)) = shift(filter(f))

Additionally, linear filters are commutative, associative, distribute over addition, scalars factor out, and possess hthe identity property.

Definition 2.6 (Gaussian Filter) ***Gaussian filters*** remove "high-frequency" components from the image (images become more smooth). Convolution with self is another gaussian, so can smooth with small-width kernel, repeat, and get same result as larger-width kernel would have. Convolving two times with Gaussian kernel of width σ is same as convolving once with kernel of width $\sigma\sqrt{2}$

$$G_\sigma(x,y) = \frac{1}{2\pi\sigma^2} \exp^{-\frac{x^2+y^2}{2\sigma^2}}$$

By separability of the Gaussian filter, we can also say that

$$G_\sigma(x,y) = (\frac{1}{2\pi\sigma^2} \exp^{-\frac{x^2}{2\sigma^2}})(\frac{1}{2\pi\sigma^2} \exp^{-\frac{y^2}{2\sigma^2}})$$

The 2D Gaussian can be expressed as the product of two functions, one a function of x and the other a function of y. In this case, the two functions are (identical) 1D Gaussian. To choose a filter size, ensure that the values at the edges are near zero.

3 Filters and Edge Detection (Lecture 3)

3.1 Pixels and Linear Filters

An image is a ***matrix*** of values. There are low-values for dark pixels and high-values for light pixels.

Remark A basic ***horizontal gradient*** filter is $[[0,0,0],[-1,0,1],[0,0,0]]$ or simply $[-1,0,1]$. A basic ***vertical gradient*** filter is $[[0,-1,0],[0,0,0],[0,1,0]]$ or simply $[[-1],[0],[1]]$.

To remove noise in an image, we can replace each pixel with a *weighted* average of its neighborhood. The weigths are called the *filter kernel*. The types of ***noise*** are...

- ***Salt and pepper nosie***: contains random occurrences of black and white pixels
- ***Impulse noise***: contains random occurrences of white pixels
- ***Gaussian noise***: variations in intensity drawn from a Gaussian normal distribution. It is the sum of many independent factors. This is good for small standard deviations but assumes independence and zero-mean noise.

Definition 3.1 (Median Filtering) A ***median filter*** operates over a window by selecting the median intensity in the window. This is an *excellent* method for combating salt-and-pepper noise (i.e. random black and white pixels). This method also does a good job **retaining edges**.

Remark A basic ***sharpening filter*** is $\frac{1}{9}[[1,1,1],[1,1,1],[1,1,1]]$. This filter accentuates differences with local average.

The *goal* of ***edge detection*** is to identify sudden changes in an image. Intuitively, most semantic and shape information from the image can be encoded in the edges. Edges are caused by a variety of factors:

- surface normal discontitnuity
- depth discontinuity
- surface color discontinuity
- illumination discontinuity

An ***edge*** is a place of rapid change in the image intensity function.

Remark For a 2D function $f(x,y)$, the partial derivative is:

$$\frac{\partial f(x,y)}{\partial x} \approx \frac{f(x+1,y)-f(x,y)}{1}$$

Definition 3.2 (Image Gradient) The gradient of an image $f = [\frac{\partial f}{partialx}, \frac{\partial f}{\partial y}]$. The gradient points in the direction of most rapid increase in intensity. The gradient direction is given by $\theta = \tan^{-1} = (\frac{\partial f}{\partial y}/\frac{\partial f}{\partial x})$. The edge strength is given by $\sqrt{(\frac{\partial f}{\partial x})^2 + (\frac{\partial f}{\partial y})^2}$

Definition 3.3 (Derivative theorem of Convolution) Differentiation is convolution, and convolution is associative:

$$\frac{d}{dx}(f * g) = f * \frac{d}{dx}g$$

Definition 3.4 (The Canny edge detector)

1. Filter image with derivative of Gaussian
2. Find magnitude and orientation of gradient
3. Non-maximum suppression: thin wide "ridges" down to single pixel width
4. Linking and thresholding (**hysteresis**)
   - define two thresholds: low and high
   - use the high threshold to start edge curves and low threshold to continue them

Definition 3.5 (Hysteresis thresholding) use a high threshold to start edge curves, use a low threshold to continue them. The threshold at low/high levels is to get weak/strong edge pixels. Trace connected components, starting from strong edge pixels.

The choice of ***Gaussian kernel spread/size*** depends on desired behavior. A large σ detects large scale edges. A small σ detects fine features.

The ***characteristics of good features*** are repeatability, saliency (each feature is distinct), compactness/efficiency, and locality

The basic idea of ***corner detection***... We can easily recognize the point by looking through a small windows. Shifting a window in *any direction* should give a *large* change in intensity. In a "flat" region, there is no intensity change in any direction. On an "edge", there is no change in intensity along the edge direction. In a "corner", there is significant change in all directions. The change in appearance of window W for the shift $[u, v]$:

$$E(u, v) = \sum_{(x,y)\in W} [I + (x + u, y + v) - I(x, y)]^2$$

We also have the first-order Taylor approximation for small motions $[u, v]$:

$$I(x + u, y + v) \approx I(x, y) + I_x u + I_y v$$

The quadratic approximation can be written as

$$E(u, v) = [uv]M[\frac{u}{v}]$$

where M is a *second moment matrix* computed from the image derivatives.

Definition 3.6 (The Harris Corner Detector)

1. Compute partial derivatives at each pixel
2. Compute second moment matrix M in a Gaussian window around each pixel

3. Compute corner response function R
4. Threshold R
5. Find local maxima of response function (non-maximum suppression)

When interpreting the second moment matrix; first, consider the axis-aligned case (gradients are either horizontal or vertical)

$$M = [[a, 0,], [0, b]]$$

If either a or b is close to 0, then this is **not** a corner, so look for locations where both are large.
We want corner locations to be *invariant* photometric transformations and *covariant* to geometric transformations

- ***Invariance***: image is transofrmed and corner locations do not change
- ***Covariance***: if we have two transformed versions of the same image, features should be detected in corresponding locations

Theorem 3.1 (Nyquist-Shannon Sampling Theorem) When sampling a signal at discrete intervals, the sampling frequency must be $\geq 2 \times f_{\max}$. $f_{\max}$ is the maximum frequency of the input signal. This will allow us to reconstruct the original perfectly from the sampled version.

The solutions to ***anti-aliasing*** are to sample more often. Get rid of all frequencies that are greater than half the new sampling frequency; you will lose information, but it's still better than aliasing. A simpel algorithm for downsampling by a factor of 2 is as follows:

1. Start with image(h, w)
2. Apply low-pass filter
3. Sample every other pixel

4 Corners and Matching (Lecture 4)

For the classification of image points using eigenvalues of $M \ldots$

- **edge**: $\lambda_2 >> \lambda_1$ or $\lambda_1 >> \lambda_2$
- **corner**: λ_1 and λ_2 are large, $\lambda_1 \approx \lambda_2$, E increases in all directions
- **flat region**: λ_2 and λ_1 are small and E is almost constant in all directions

Corner detection is partially *invariant* to **affine intensity change**. **Corner location** is *covariant* with respect to **translation**. **Corner location** is *covariant* with respect to **rotation**. **Corner location** is *not covariant* to scaling.

4.1 Keypoint Matching

The **goal** of **keypoints** is to detect points that are *repeatable* and *distinctive*.

1. Find a set of distinctive key-points
2. Define a region around each keypoint
3. Extract and normalize the region content
4. Computate a local descriptor from the normalized region
5. Match local descriptors

Automatic scale selection functions will find the ideal scale to include in the regions around each keypoint by testing different scales and measuring the responses/.

We want to extract features with characteristic scale that is *covariant* with the image transformation. To detect blobs, convolve the iamge with a "blob filter" at multiple scales and look for extrema of filter response in the resulting *scale space*. Find maxima and minima of blob filter response in space and scale. We want to find the characteristic scale of thfe bob by convolving it with Laplacians at several scales and looking for the maximum response; however, Laplacian response decays as scale increases.

The response of a derivative of Gaussian filter to a perfect step edge decreases as σ increases. To keep the response the same (scale-invariant), must multiply Gaussian derivative by σ. Laplacian is the second Gaussian derivative, so it must be multiplied by σ^2.

We define the ***characteristic scale*** of a blob as the scale that produces peak of Laplacian response in the blob center. To perform a ***scale-space blob detector***...

1. Convolve image with scale-normalized Laplacian at several scales
2. Find maxima of squared Laplacian response on scale-space

Remark We can approximate the Laplacian with a difference of Gaussians:

- Laplacian: $L = \sigma^2(G_{xx}(x, y, \sigma) + G_{yy}(x, y, \sigma)$
- Difference of Gaussians: $DoG = G(x, y, k\sigma) - G(x, y, \sigma)$

Local descriptors should be robust, distinctive, compact, efficient. Most available descriptors focus on edge/gradient information; they often capture texture information and color is rarely used. Scaled and rotated versions of the same neighborhood will give rise to blobs that are related by the same transformation.

To eliminate rotation ambiguity we must assign a unique orientation to circular image windows. To do this we create a histogram of local gradient directions in the path and assign canonical orientation at peak of smoothed histogram.

Definition 4.1 (SIFT) detects features with characteristic scales and orientations. Very robust detection and description technique that can handle changes in viewpoint, significant changes in illumination. SIFT is fast and efficient and can run in real time.

1. Run DoG detector. Find maxima in location/scale space and remove edge points
2. Find all major orientations. Bin orientations into 36 bin histogram (weight by gradient magnitude, weight by distance to center). Return orientations within 0.8 of peak of histogram; use parabola for better orientation fit.
3. For each (x,y,scale,orientation), create a descriptor:
   - Sample 16×16 gradient mag. and re. orientation
   - Bin 4×4 samples into 4×4 histograms
   - Threshold values to max of 0.2, divide by L2 norm
   - Final descriptor: $4 \times 4 \times 8$ normalized histograms

4.2 Fitting

We would like to form a higher-level, more compact representation of the features in the image by grouping multiple features according to a simple model. In ***fitting***, we will choose a parametric model to represent a set of features. If we, for example, know which points belong to the line, we can find the "optimal" line parameters using something such as least squares line fitting or total least squares.

Definition 4.2 (Least Squares Line Fitting) Let's say the data is $(x_1, y_1), \ldots, (x_n, y_n)$, the line equation is $y_i = mx_i + b$, and we want to find (m, b) to minimize.

$$E = \sum_{i=1}^{n} (y_i - mx_i - b)^2$$

$$B = (X^T X)^{-1} X^T Y$$

Definition 4.3 The distance between point (x_i, y_i) and line $ax + by = d$ $(a^2 + b^2 = 1)$ is $|ax_i + by_i - d|$. Find (a, b, d) to minimize the sum of squared perpendicular distances.

$$E = \sum_{i=1}^{n} (ax_i + by_i - d)^2$$

RANSAC (random sample consensus) is a very general framework for model fitting in the presence of outliers. Generally, it will...

1. Randomly select minimal subset of points
2. Hypothesize a model
3. Compute error function
4. Select points consistent with model
5. Repeat *hypothesize-and-verify* loop

5 RANSAC and Hough Transform (Lecture 5)

5.1 RANSAC

Remark Least squared line fitting works well for well-behaved similar data. However, a single outlier can signifcantly reduce the accuracy of least squared line fitting.

An alternative to this is **RANSAC** (as defined above).

Definition 5.1 (RANSAC for line fitting) Repeat N times...

1. Draw s points uniformly at random
2. Fit line to these s points
3. Find *inliers* to thsi line among the remaining points (i.e. points whose distance from the line is less than t)
4. If there are d or more inliers, accept the line and refit using all inliers.

As far as choosing the parameters for **RANSAC line fitting**:

1. The initial number of points s is typically the minimum needed to fit the model
2. Choose the distance threshold t so probability for inliers is p
3. Choose the number of samples N so that, with probability p, at least one random sample is free from outliers (outlier ratio: e)

The parameters can all be defined by the following equivalence

$$(1-(1-e)^s)^N = 1-p$$

$$N = \frac{\log(1-p)}{\log(1-(1-e)^s))}$$

The outlier ratio e is often unknown *a priori*, so pick worst case (e.g. 50%), and adapt if more inliers are found (e.g. 80% would yield $e = 0.2$).

Some **Pros of RANSAC**:

- simple and general
- applicable to many different problems
- often works well in practice

Some **Cons of RANSAC**:

- Computational time grows quickly with fraction of outliers and number of parameters
- Not as good for getting multiple fits (though one solution is to remove inliers after each fit and repeat)
- sensitivity to threshold t

5.2 Hough Transform

Hough transform relies on letting each feature vote for all the models that are compatibly with it, and hopefully the noise features will not vote consistently for any single model. The general outline of the ***Hough Transform*** is to:

- discretize *parameters space* into bins
- for each feature point in the image, put a vote in every bin in the parameter space that could have generated this points
- find bins that have the most votes

Remark A line in the image corresponds to a point in Hough space. For example, the point (x_0, y_0) in the image space maps to $b = -x_0 m + y_0$ in the Hough space.

Remark The probelms with the (m, b) parameter spcae representation are that there are unbounded parameter domains and vertical lines require infinite m. An alternative representation, is the ***polar coordinate system***: $x\cos\theta + y\sin\theta = \rho$. Thus, each point (x, y) will add a sinusoid in the (θ, ρ) parameter space.

The general outline of the algorithm is as follows:

- Initialize accumulator H to all zeros
- For each feature point (x, y) in the image:
  For $\theta = 0$ to 180
  - $\rho = x\cos\theta + y\sin\theta$
  - $H(\theta, \rho) = H(\theta, \rho) + 1$
- Find the value(s) of (θ, ρ) where $H(\theta, \rho)$ is a local maximum. The detected line in the image is given by $\rho = x\cos\theta + y\sin\theta$

As the level of uniform noise increases, the maximum number of votes increases too. To **deal with noise**...

- Choose a good grid/discretization
  - **Too coarse**: large vote counts obtained when too many different lines correspond to a single bucket
  - **Too fine**: miss lines because some points that are not exactly colinear cast votes for different buckets
- Increment neighboring bins (smoothing in accumulator array)
- Try to get rid of irrelevant features
  - E.g. take only edge points with signficant gradient magnitude

Definition 5.2 (Generalized Hough Transform) We want to find a template defined by its reference point (center) and several distinct types of landmark points in stable spatial configuration. For each type of landmark point, store all possible displacement vectors towards the center. For each feature in the new image, look up that feature type in the model and vote for the possible center locations associated with that type in the model.

Remark (Practical tips for Voting) When voting, minimize irrelevant tokens first. Choose a good grid / discretization. Vote for neighbors also (smoothing in accumulator array). Use direction of edge to reduce parameters by 1. To read back which points voted for *winning* peaks, keep tags on the votes.

Some **Pros of Hough Transform**:

- All points processed independently
- Can deal with occlusion and gaps
- Can detect multiple instances of a model
- Some robustness to noise: noise points unlikely to contribute consistently to any single bin

Some **Cons of Hough Transform**:

- Complexity of search time increases exponentially with the number of model parameters
- Non-target shapes can produce spurious peaks in parameter space
- It's hard to pick a good grid size

5.3 Alignment

With ***alignment***, we want to find parameters of model that maps one set of points to another. We are correcting for change in the observer's position relative to the observed. Typically, we want to solve for a global transformation that accounts for most true correspondences. Some difficulties include noise, outliers, and many-to-one matces/multiple objects.

Definition 5.3 (Parametric Warping) Transformation T is a coordinate change.

$$p' = T(p)$$

T is the same for any point p and can be described by just a few numbers (parameters).

Scaling a coordinate means multiplying each of its components by a scalar. ***Uniform scaling*** means that this scalar is the same for all components. ***Non-uniform scaling*** uses different scalars per component. The scaling operations are

$$x' = ax$$

$$y' = by$$

The scalars for a 2-D rotation can be defined as

$$x' = x\cos(\theta) - y\sin(\theta)$$

$$y' = x\sin(\theta) + y\cos(\theta)$$

Even through $\sin(\theta)$ and $\cos(\theta)$ are nonlinea functions of θ,

- x' is a linear combination of x and y
- y' is a linear combination of x and y

And it naturally follows that the inverse transformation is rotation by $-\theta$. For rotation matrices, we can say that $R^{-1} = R^T$.

Affine transformations are combinations of linear transformations and translations. Properties of affine transformations include the following:

- Lines map to lines
- Parallel lines remain parallel
- Ratios are preserved
- Closed under composition

Projective transformations are combinations of affine transformations and projective warps. Properties of projective transformations include the following:

- Lines map to lines

- Parallel lines do not necessarily remain parallel
- Ratios are not preserved
- Closed under composition
- Models change of basis
- Projective matrix is defined up to a scale (8 DOF)

Affine transformations follow a simple fitting procedure: linear least squares. They are used to approixmate viewpoint changes for roughly panar objects. We can use to initialize fitting for more complex models. Affine transofrmations are linear systems with six unknowns. Each match gives us two linearly independent equations: need at least three to solve for the transformation parameters.

Definition 5.4 (Homography) a plane projective transformation (transformation taking a quad to another arbitrary quad). This can be used to stitch together a panorama.

In ***robust feature-based alignment***, we extract features, compute *putative* matches, and then enter a loop where we

- hypothesize transformation T
- verify transformation (serach for other matches consistent with T)

When generating ***putative correspondences***, we need to compare feature descriptors of local patches surrounding interest points.

5.4 Template Matching

How can we use filtering to find a correspondence between some sample patch and an image in which we can presumably find that patch (or a similar one)?

What is a good measure of similarity between two patches?

- Filtering
- Zero-mean filtering
- Sum of squares difference
- Normalized cross correlation

In template matcing with image pyramids we match the template at current scale and then downsample the image. These steps are repeated until the image is very small and we take responses above some threshold.

6 Feature Tracking

Given a feature in one image, how can we find the best match in another image (assuming that the images are consecutive frames in video)?

Given two subsequent frames, estimate the point translation. To achieve this we can use the Lucas-Kanade Tracker which makes assumptions such as:

- **brightness constancy**: projection of the same point looks the same in every frame
- **small motion**: points do not move very far
- **spatioal coherence**: points move like their neighbors

The ***brightness constancy*** equation says $I(x, y, t) = I(x + u, y + v, t + 1)$ for some displacement (u, v).

To deal with larger motion we can perform ***iterative refinement***.

1. Initialize $(x', y') = (x, y)$
2. Compute (u, v)
3. Shift window by $(u, v) : x' = x' + u; y' = y' + v$
4. Recalculate I_t
5. Repeat steps 2-4 until change is small (use interpolation for subpixel values)

Definition 6.1 (Shi-Tomasi Feature Tracker) Find good features using eigenvalues of the second-moment matrix; good features to track are the ones whose motion can be eastimated reliably. Track from frame to frame with the Lucas-Kanade tracker; this amounts to assuming a translation model for frame-to-frame movement.

In general with KLT tracking... Find a good point to track (Harris corners). Use intensity second moment matrix and difference across frames to find displacement. Iterate and use coarse-to-fine search to deal with larger movements. When creating long tracks, check appearance of registered patch against appearance of initial patch to find points that have drifted. Some issues can include the window size: small windows are more sensitive to noies and may miss larger motins, while large windows are more liekly to cross an occlusion boundary.

7 Grouping and Segmentation

The major processes for segmentation are bottom-up and top-down. In ***bottom-up***, we group tokens with similar features (pixels belong together because they

look similar). In ***top-down***, group tokens that likely belong to the same object (pixels belong together because they are from the same object).

The goal of ***clustering*** is to choose three *centers* as the representative identities, and label every pixel according to which of these centers it is nearest to. Best cluster centers are those that minimize SSD between all points and their nearest cluster center. If we knew the ***cluster centers***, we could allocate points to groups by assigning each to its closest center. If we knew the ***group memberships***, we could get the centers by computing the mean per group.

7.1 K-Means Clustering

Definition 7.1 (K-means Clustering) Randomly initialize the k cluster centers, and iterate between the two steps we just saw.

1. Randomly initialize the cluster centers, $c_1, \ldots, c_k$
2. Given cluster centers, determine points in each cluster. For each point p, find the closest c_i. Put p into cluster i.
3. Given points in each cluster, solve for c_i. Set c_i to be the mean of points in cluster i
4. If c_i have changed, repeat step 2

This will always converge to *some* solution. This can be a *local minimum*; does not always find the global minimum of objective function.

Pros of K-means:

- Simple, fast to compute
- Convergest to local minium of within-cluster squared error

Cons/Issues of/with K-means:

- Setting K
- Sensitive to initial centers
- Sensitive to outliers
- Detects spherical clusters
- ASsumes means can be computed

Depending on what we choose as the *feature space*, we can group pixels in different ways. We can group pixels based on intensity similarity, color similarity, intensity+position similarity, texture similarity, etc.

7.2 Mean Shift Algorithm

The ***mean shift algorithm*** seeks *modes* or local maxima of density in the feature space. The ***cluster*** is all data points in the attraction basin of a mode. The ***attraction basin*** is the region for which all trajectories lead to the same mode.
Pros of mean shift:

- Does not assume shape on clusters
- One parameter choice (bandwidth/window size)
- Generic technique
- Finds multiple modes

Cons of mean shift:

- Selection of bandwidth
- Does not scale well with dimension of feature space

7.3 Superpixel Algorithm

In ***superpixel algorithms*** the goal is to divide the image into a large number of regions, such that each regions lies within object boundaries.

Definition 7.2 (Meyer's Watershed Segmentation) .

1. Choose local minima as region seeds
2. Add neighbors to priority queue, sorted by value
3. Take top priority pixel from queue
   (a) If all labeled neighbors have same label, assign that label to pixel
   (b) Add all non-marked neighbors to queue
4. Repeat step 3 until all finished (meaning pixels in queue are on the boundary)

Remark You can use Gaussian or median filter to reduce the number of regions.

Pros of watershed:

- Fast (¡ 1 second for 512x512 images)
- Preserves boundaries

Cons of watershed:

- Only as good as the soft boundaries (which may be slow to compute)

- Not easy to get variety of regions for multiple segmentations

This is a good algorithm for superpixels and hierarchical segmentation.

Definition 7.3 (SLIC) .

1. Initialize cluster centers on pixel grid in steps S
2. Move centers to position on 3x3 window with smallest gradient
3. Compare each pixel to cluster center within $2S$ pixel distance and assign to nearest
4. Recompute cluster centers as mean color/position of pixels belonging to each cluster
5. Stop when residual error is small

7.4 Bags of Features/Visual Words

Origin 1 is texture. ***Texture*** is characterized by the repetition of basic elements or *textons*. For stochastic textures, it is the identity of the textons, not their spatial arrangement, that matters.

Origin 2 is ***bag-of-words models*** which are the frequencies of words from a dictionary. The steps for bag-of-features are

1. Extract local features
2. Learn *visual vocabulary*
3. Quantize local features using visual vocabulary
4. Represent images by frequencies or *visual words*

If the vocabulary size is too small, visual words are not representative of all patches. If it is too large, then quantization artifacts and there is overfitting. The correct size is application-dependent.

The **pros of bags of words** are

- flexible to geometry/deformations/viewpoint
- compact summary of image content
- provides vector representation for sets
- very good results in practice

The **cons of bags of words** are

- basic model ignores geometry—must verify afterwards, or encode via features
- background and foreground mixed when bag covers whole image
- optimal vocabulary formation remains unclear

With ***spatial pyramid***, we split the image into quadrants recursively and compute the bag of words for each feature within each of those quadrants. When we perform a level 1 spatial pyramid separation, we have k bins for the full image (from level 0) and 4 sets of k bins (one for each quadrant) which come from the level 1 representation.

Remark ***Bag of visual words*** is only about counting the number of local descriptors assigned to each cluster

Soft assignment called ***kernel codebook encoding*** involves casting a weighted vote into the most similar clusters (essentially distance-weighted visual words). A more complex option is to perform **VLAD**.

Definition 7.4 (Vector of Locally Aggregated Descriptors (VLAD)) Given a codebook $\{\mu_i, i = 1 \dots N\}$, e.g. learned with K-means, and a set of local descriptors $X = \{x_t, t = 1 \dots T\}$:

- Assign $NN(x_t) = \arg\min_{\mu_i} ||x_t - \mu_i||$
- Compute $v_i = \sum_{x_t : NN(x_t) = \mu_i} x_t - \mu_i$
- Concatenate v_i's + l_2 normalize

The basic framework for ***generic category regonition*** is

- Build/train an object model: choose representation then learn or fit parameters
- Generates candidates in new image
- Score the candidates

With ***supervised classification***, we are given a collection of *labeled* examples and want to come up with a function that will predict the labels of new examples. The ***risk*** of a classifier is the probability of making some mistake multiplied by the cost of making that cost. We want to choose a classifier so as to minimize this total risk. The optimal classifier will minimize total risk. At the decision boundary, either choice of label yields the same expected loss. There are two general strategies to minimize the expected mislcassification

- Use the training data to build representative probability model; separately model class-conditional densities and priors (generative)
- Directly construct a good decision boundary, model the posterior (discriminiative)

For ***window-based object detection*** *training*: obtain training data, define features, and define the classifier. Then when *given a new image*: slide the window and score by classifier. Recall, that a classifier maps from a sample to a label. Different types of classifiers include exemplar-based, linear classifier, non-linear classifier, and generative classifiers.

Remark Training labels dictate that two examples are the same or different, in some sense. **Features and distance measures** define visual similarity. **Goal of training** is to learn feature weights or distance measures so that visual similarity predicts label similarity. *We want the simplest function that is confidently correct.*

Using K-NN (K Nearest Neighbors) is simple and thus a good classifier to try first. There is no training time (unless you want to learn a distance function).

Definition 7.5 (Boosting) Initially, weight each training example equally. In each boosting round:

- Find the weak learner that acheives the lowest *weighted* training error
- Raise weights of training examples misclassified by current weak learner

Compute final classifier as linear combination of all weak learners. Boosting combines *weak learners* into a more accurate *ensemble classifier*.

Definition 7.6 (Dalal-Triggs Pedestrian Detector) Runs as follows:

1. Extract fixed-size window at each position and scale
2. Compute HOG (histogram of gradient) features within each window
3. Score the window with a linear SVM classifier
4. Perform non-maxima suppression to remove overlapping detections with lower scores

Definition 7.7 (Part-based Models) Define object by collection of parts modeled by appearance and spacial configuration.

8 Deep Learning for Computer Vision

Binary perceptrons take multiple binary inputs and use thresholds and weights (parameters) to output a threshold weighted linear combination. Layering these ***neurons*** (perceptrons) allows for more meaningful, complex models.

Definition 8.1 (Sigmoid Neurons) neurons that offer stability by ensuring that a small perturbation in the input leads to a small change in the ouput. This function takes continuous inputs and outputs continuous values instead of a step function with hard cutoffs. The thresholds are no longer rigid, but instead soft.

$$\sigma(z) = \frac{1}{1_e^{-x}}$$

A ***cost function*** encodes the difference between generated/predicted output and expected output. We attempt to find the combination of parameters/weights that minimizes the cost function. Gradient descent is this process of minimizing the cost function and follows this general form

$$\delta C \approx \frac{\partial C}{\partial v_1}\delta v_1 + \frac{\partial C}{\partial v_2} v_2$$

Realistically, the cost function and optimization must be generalized to many variables. Note that a small change in a variable will then have a small change in cost.

8.1 Neural Nets for Vision

To build an object recognition system, we will use data to optimize features for the given task. We want to use a parameterized function such that

- features are computed efficiently
- features can be trained efficiently

The goal is an ***end-to-end recognition system*** in which everything becomes adaptive and there is no distinction between feature extraction and classification. A big non-linear system will be trained from raw pixels to labels.

Function composition is at the core of deep learning methods. Each *simple function* will have parameters that are subject to training. Each black box (layer) can have trainable parameters; their composition makes a highly non-linear system.

The **key ideas** of ***Neural Networks*** are...

- Learn features from data
- Use differentiable functions that produce features efficiently
- End-to-end learning: no distinction between feature extractor and classifier
- *Deep* architectures: cascade of simpler non-linear modules.

8.2 Supervised Deep Learning

The goal is to predict the target label of unseen inputs.

Different types of classification include:

- Classification: image $\rightarrow$ label
- Regressio: image $\rightarrow$ image

- Structured prediction: image → some structure

Forward propagation is the process of computing the output of the network given its input. The non-linearity $u = max(0, v)$ is called ***ReLU*** activation function. Each output hidden unit takes as input all the units at the previous layer: each such layer is called ***fully connected***.

Remark The mapping between layers cannot be linear because composition of linear functions is a linear function. In this case, the neural network would reduce to (1 layer) logisistic regression.

When input has hierarchical structure, the use of a hierarchical architecture is potentiall more efficient because intermediate computations can be re-used. Deep learning architectures are efficient also because they use *distributed representations* which are shared across classes.

Learning consists of minimizing the loss (plus some regularization term) with respect to parameters over the whole training set.

$$\theta^* = argmin_\theta \sum_{n=1}^{P} L(x^n, y^n; \theta)$$

Backpropagation is the procedure to compute graidents of the loss with respect to parameters in a multi-layer neural network. Let's say we want to decrease the loss by adjusting $W^1_{i,j}$. We could consider a very small $\epsilon = 1 \times 10^{-6}$ and compute, then update $W^1_{i,j}$, and check the effect on the cost function.

The ***stochastic gradient descent*** optimization function works as follows

$$\theta \leftarrow \theta - \eta \frac{\partial L}{\partial \theta}, \eta \in (0, 1)$$

8.3 Loss Functions

- ***Regression***: If our target values are continuous, a common approach is to use mean squared error (MSE)
- ***Two-class Classification***: Binary cross-entropy
- ***Multi-class Classification***: Categorical cross-entropy

8.4 Convolutional Neural Networks

In a ***convolutional layer*** the same parameters are shared across different locations (assuming the input is stationary), and then we perform convolutions with learned kernels. We will learn the weights for multiple filters throughout this process.

$$h^n_j = max(0, \sum_{k=1} K h^{n-1}_k \times w^n_{kj}$$

This represents the output feature map at position (k, j) using the input feature map and the learned kernel w as inputs.

Remark If the input to a convolutional layer has the size $M \times D \times D$, then the output has size $N \times (D-K+1) \times (D-K+1)$. The kernels have $M \times N \times K \times K$ coefficients (which have to be learned). The cose is $M \times K \times K \times N \times (D - K + 1) \times (D - K + 1)$

Remark Usually, there are mo output feature maps than input feature maps. Convolutional layers can increase the number of hidden untits by big factors (and are expensive to compute). The size of the filters has to match the size/scale of the patterns we want to detect (task dependent)

A standard neural net applied to images scales quadratically with the size of the input and does not leverage stationarity. The solution is to connect each hidden unit to a small patch of the input and share the weight across space: this is called convolutional layer. A network with convolutional layers is called ***convolutional network***.

By ***pooling*** (e.g. taking max) filter responses at different locations we, gain robustness to the exact spatial location of features. This is done in a ***pooling layer***.

Remark Convolutional filters are trained in a supervised manner by back-propagating classification error.

In ***stochastic gradient descent***, only a single data-point is evaluated at a time. The training data is randomly ordered such that index $i(t)$ is given at descent iteration t. The goal of ***dropout*** is to increase sparsity. By randomly removing activations, neurons can then more quickly build up strong relationships with certain inputs. This results in increased neuron specialization and simulates averaging over a variety of models.