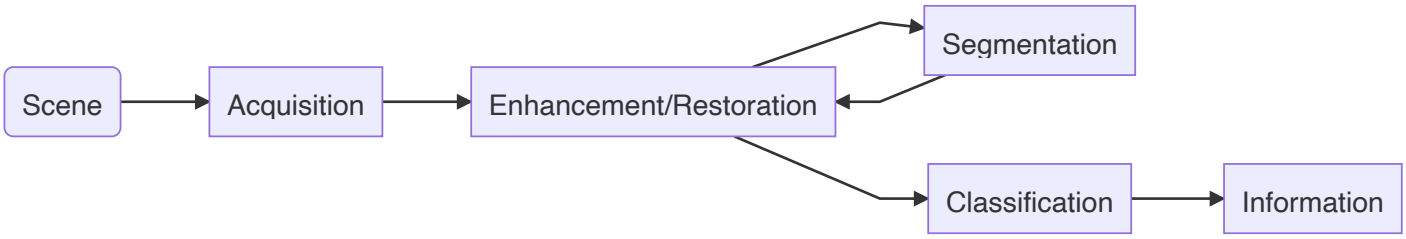
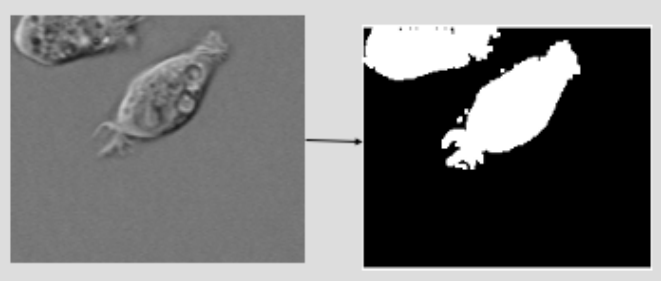
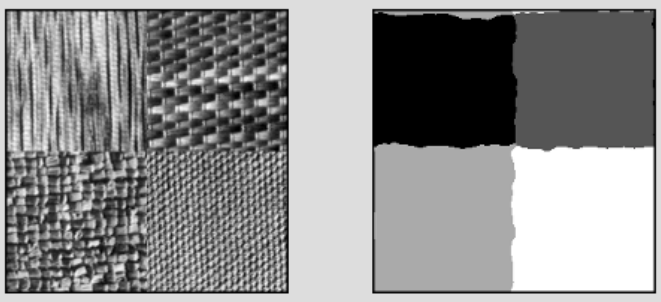
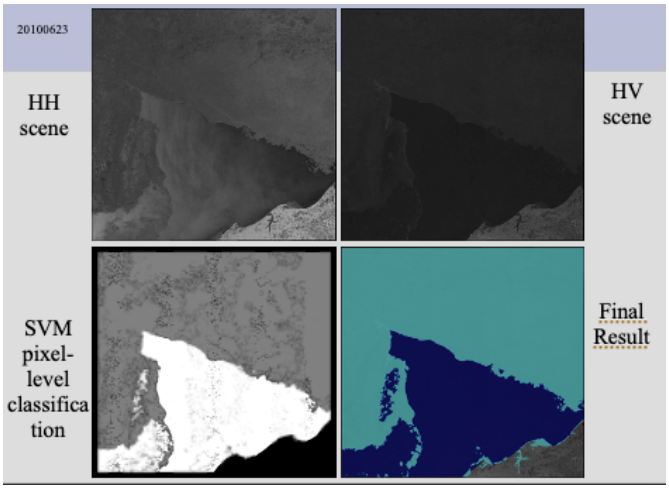
7. Segmentation

7.1 - Introduction

Flowchart



Examples:

Note	Diagrams
Underlying goal of image segmentation: is to **partition** an image into **homogeneous regions**	
Texture-based Segmentation: With proper features, scenes can be segmented based on texture appearance	
Segmentation of Radar-based Satellite Ocean Imagery	

Q&A:

Why Segment?

- Segmentation allows objects and regions to be analysed in a more meaningful manner
- Some applications of segmentation:
  - Object tracking
    - ex: people tracking for surveillance purpose
  - Medical image analysis
    - ex: tumor growth analysis
  - Remote sensing analysis

        - ex: determine the ratio of diff. types of sea-ice within a region
    - Face recognition
        - ex: partition face into individual parts for component recognition

Bottom-Up vs. Top-Down

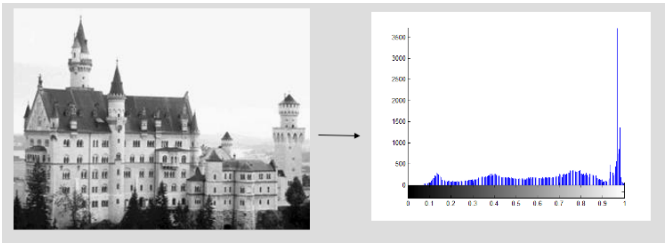
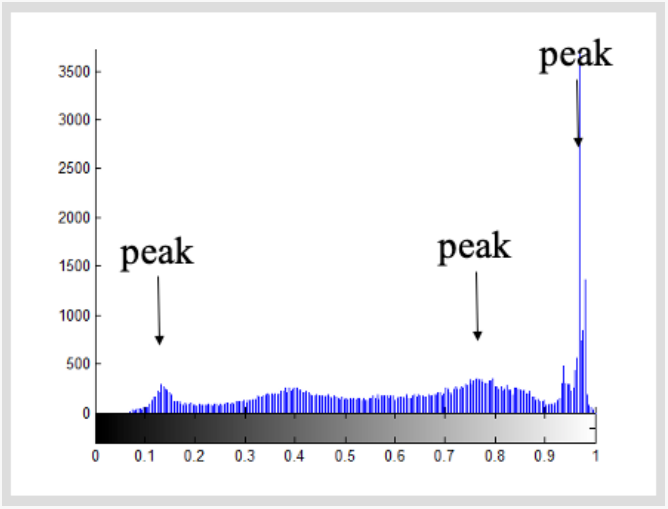
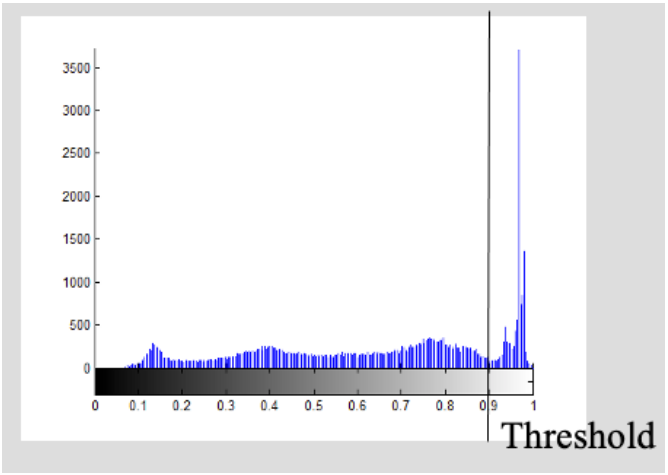
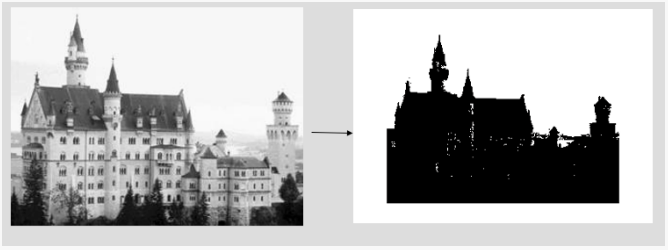
- B-U Segmentation:
    - Segment image into regions and then identify those regions that correspond to a single object
    - Uses features such as consistent grey level or tecture
        - And/or identification of boundaries to identify the object
- T-D Segmentation:
    - Use prior knowledge of the object (ex: shape, color, texture) to guide segmentation
- Example: identify all cows in an image database

7.2 - Histogram Methods

Intro - Histogram Methods:

- Simplest and computationally efficient form of segmentation

Concept:

Note	
- Consider Image Histogram	
- Analyze peaks of the histogram to determine the appropriate point for thresholding - **Challenge**: to determine appropriate peak (ex: to separate background and foreground)	
- **Goal**: Determine an appropriate point of thresholding based on what we want (ex: segment foreground and background)	
- Set all pixels to a set of fixed values based on threshold - Binary thresholding in example =>	

Problems with Finding a Threshold:

1. Histogram is **typically noisy** so difficult to isolate an appropriate T automatically
2. Peaks may not be noticeable due to **noisy appearance** and **large class variances** (may even appear unimodal)
    - Ex: sometimes, two peak may forma unimodal curve, and make it indistinguishable from the rest histogram
3. **Illumination differences** across image makes setting a **global threshold erroneous**
    - Ex: non-stationarity problem
4. Regions may be **discriminable** using other descriptors
    - Ex: texture (If its texture, it has pattern. Histogram is not good at capturing such)

Possible Solutions

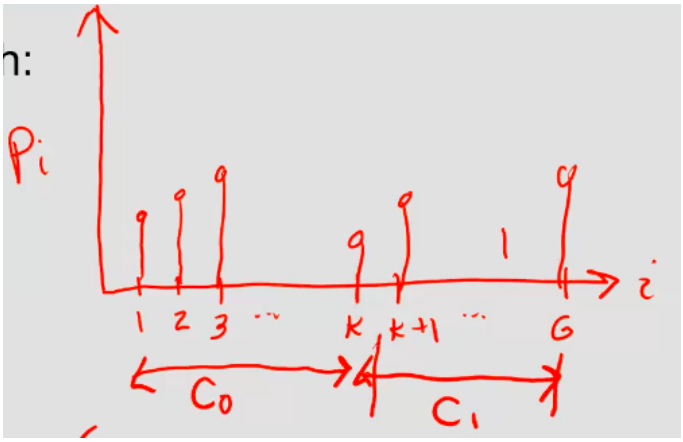
1. Otsu's Method: use statistics from histogram to find threshold
    - Anytime with a clear contrast between the background and foreground
2. Use **"clustering"**
    - ex: use criterion to group pixels into appropriate classes
        - Ex: Two Clustering-Based Methods (K-means & Finite Mixture Models)
            - => also see SD 372 - Pattern Recognition
3. Use method that accounts for spatial (within-image) relationships
    - Ex: Markov random fields [Advanced - we won't cover.]

Otsu's Method

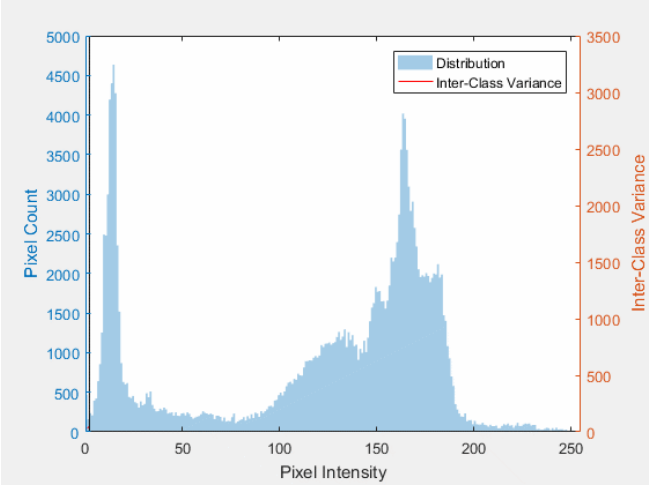
- **Standard method** for **blind**, **histogram**-based thresholding
- "A Threshold Selection Method from Grey Level histograms" (IEEE, 1979)
- Form a probability distribution p_i:
    - $p_i = \frac{n_i}{N}$
    - Where:
        - N represents total num of pixels
        - n_i represents num of pixels at each grey level

Otsu's Equation (Derivation & Sketch):

Sketch:



Wiki



Derivation - Otsu's Equation

Probabilities of class occurrence:

$$\omega_0 + \omega_1 = 1$$

$$\omega_0 = p_r(C_0) = \sum_{i=1}^{K} P_i = \omega(k)$$

$$\omega_1 = p_r(C_1) = \sum_{i=K+1}^{G} P_i = 1 - \omega(k)$$

Overall Probability:

$$\begin{aligned}\mu &= \sum_{i=1}^{G} i\, p_i \\ &= \omega_0 \mu_0 + \omega_1 \mu_1 \qquad \text{(weighted average)} \\ &= \sum_{i=1}^{K} i\, p_i + \sum_{i=K+1}^{G} i\, p_i\end{aligned}$$

Hence:

$$\mu_0 = \frac{\sum_{i=1}^{K} i\, p_i}{\omega_0}$$

$$\mu_0 = \frac{\sum_{i=K+1}^{G} i\, p_i}{\omega_1}$$

=> Now have μ_0 and μ_1 as functions of k.
=> need to optimize based on some criterion.

Here, use between-class variance:

$$\text{If } \mu(k) = \sum_{i=1}^{k} i\, p_i$$

$$\Rightarrow \mu_0 = \frac{\mu(k)}{\omega(k)} \qquad \mu_1 = \frac{\mu - \mu(k)}{1 - \omega(k)}$$

$$\Rightarrow \sigma_B^2 \overset{\Delta}{=} \omega_0 (\mu_0 - \mu)^2 + \omega_1 (\mu_1 - \mu)^2$$

The algorithm exhaustively searches for the threshold that minimizes the intra-class variance, defined as a weighted sum of variances of the two classes: σ_B^2
You want two cluster is far as possible in histogram.

$$\begin{aligned}
\sigma_B^2(k) &= \omega(k)\left[\frac{\mu(k)}{\omega(k)} - \mu\right]^2 + [1 - \omega(k)]\left[\frac{\mu - \mu(k)}{1 - \omega(k)} - \mu\right]^2 \\
&= \omega(k)\left[\frac{\mu(k) - \mu}{\omega(k)}\right]^2 + [1 - \omega(k)]\left[\frac{\mu(k)\mu - \mu(k)}{1 - \omega(k)}\right]^2 \\
&= \frac{[\mu\omega(k) - \mu(k)]^2}{\omega(k)} + \frac{[\mu\omega(k) - \mu(k)]^2}{1 - \omega(k)} \\
&= \frac{[1 - \omega(k)][\mu\omega(k) - \mu(k)]^2 + \omega(k)[\mu\omega(k) - \mu(k)]^2}{\omega(k)[1 - \omega(k)]} \\
&= \frac{[\mu\omega(k) - \mu(k)]^2}{\omega(k)[1 - \omega(k)]}
\end{aligned}$$

Hence:

$$\sigma_B^2(k) = \frac{[\mu\omega(k) - \mu(k)]^2}{\omega(k)[1 - \omega(k)]}$$

How to Implement?

Implementation

Matlab: just calculate $\sigma_B^2(k)$ as a vector

Programming: use update equations recursively for efficiency
EX:

$$\mu(k) = \sum_{i=1}^{k} i\, p_i$$

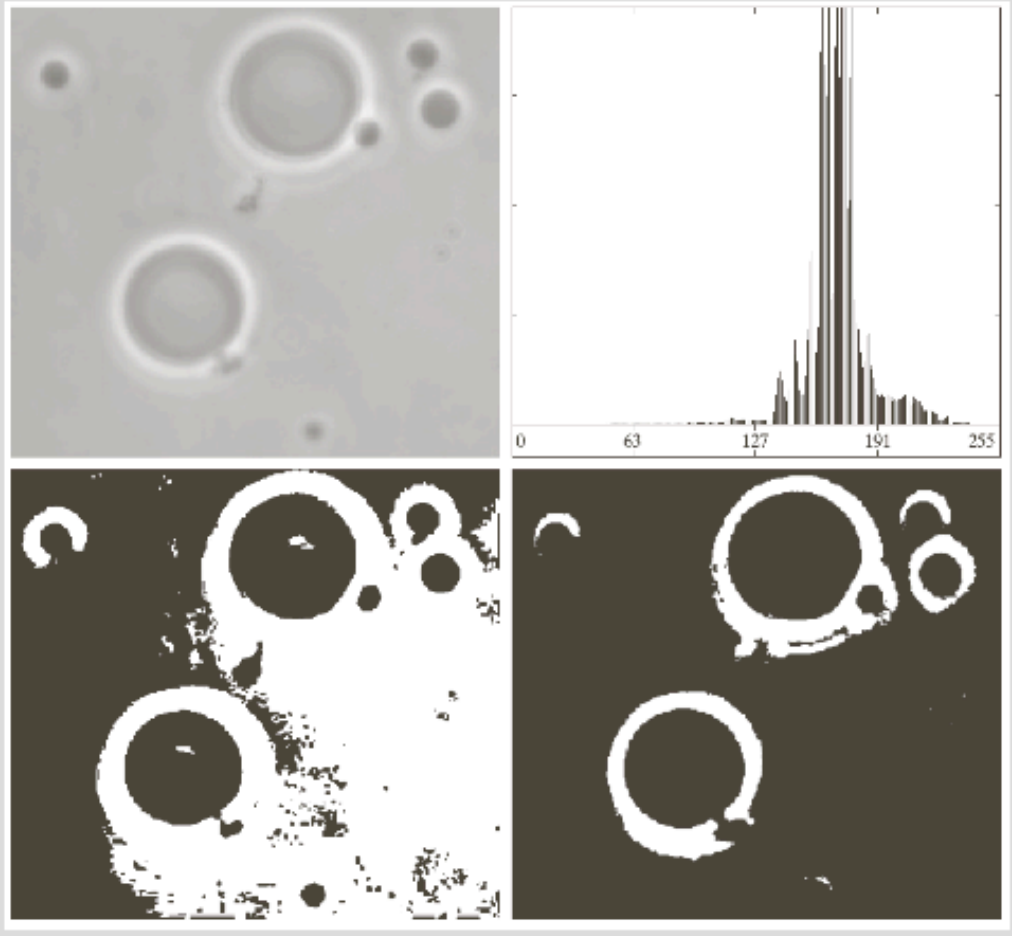
$$\mu(k+1) = \sum_{i=1}^{k+1} i\, p_i = \sum_{i=1}^{k} i\, p_i + (k+1)\, P_{k+1} = \mu(k) + (k+1)\, P_{k+1}$$

$$\omega(k) = \sum_{i=1}^{k} = P_i$$

$$\omega(k+1) = \omega(k) + P_{k+1}$$

$$\mu = \textbf{constant} = \sum_{i=1}^{G} i\, p_i$$

Example 1 - Otsu's Method:
- Anytime with a clear contrast between the background and foreground

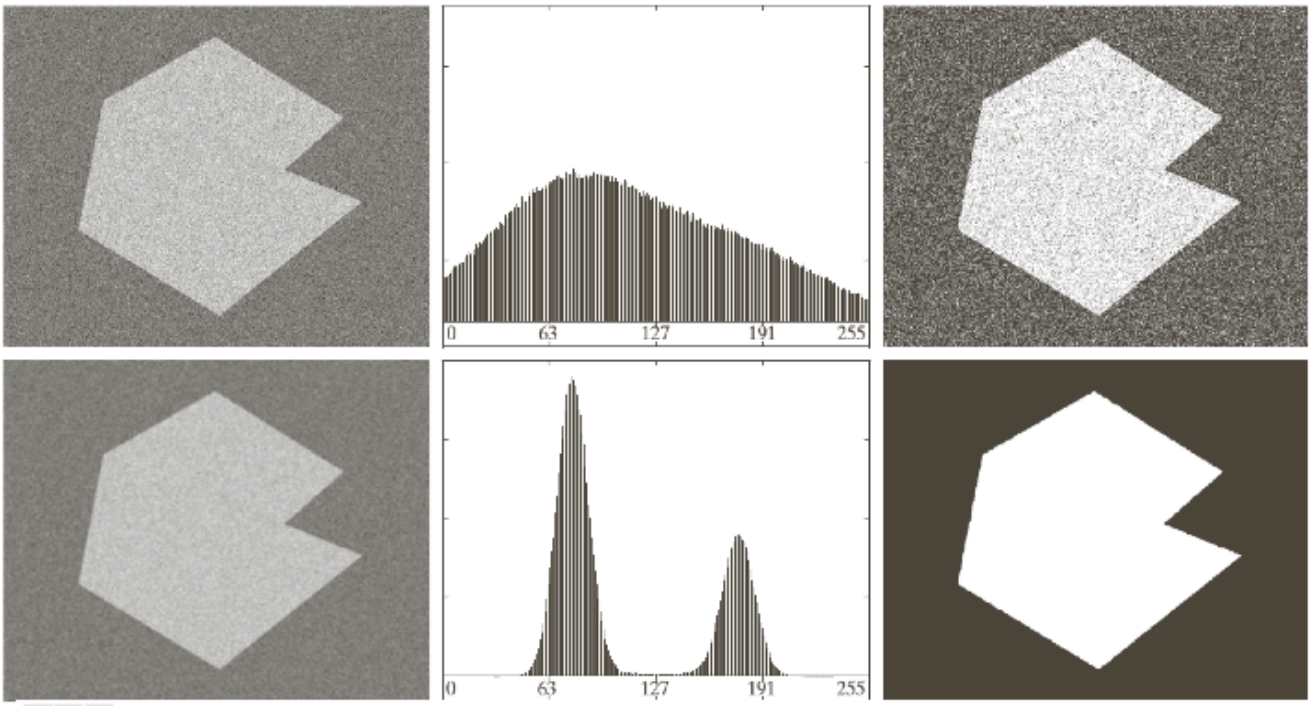


a b
c d

FIGURE 10.39
(a) Original image.
(b) Histogram (high peaks were clipped to highlight details in the lower values).
(c) Segmentation result using the basic global algorithm from Section 10.3.2.
(d) Result obtained using Otsu's method.
(Original image courtesy of Professor Daniel A. Hammer, the University of Pennsylvania.)

Example 2 - Smoothing + Otsu's Method

- after applying the smooth, u get bimodal instead of unimodal peaks, and run Otsu's method to separate foreground and background



a b c
d e f

FIGURE 10.40 (a) Noisy image from Fig. 10.36 and (b) its histogram. (c) Result obtained using Otsu's method. (d) Noisy image smoothed using a 5 × 5 averaging mask and (e) its histogram. (f) Result of thresholding using Otsu's method.

```
function level = otsu(histogramCounts)
  total = sum(histogramCounts); % total number of pixels in the image
  %% OTSU automatic thresholding
  top = 256;
  sumB = 0;
  wB = 0;
  maximum = 0.0;
  sum1 = dot(0:top-1, histogramCounts);
  for ii = 1:top
      wF = total - wB;
      if wB > 0 && wF > 0
          mF = (sum1 - sumB) / wF;
          val = wB * wF * ((sumB / wB) - mF) * ((sumB / wB) - mF);
          if ( val >= maximum )
              level = ii;
              maximum = val;
          end
      end
```

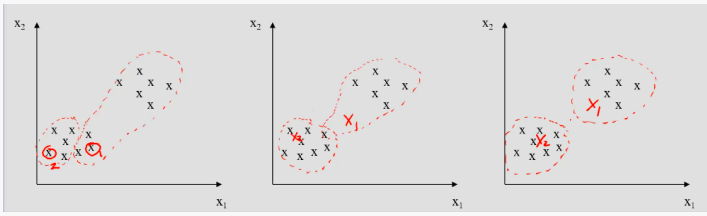
```
        wB = wB + histogramCounts(ii);
        sumB = sumB + (ii-1) * histogramCounts(ii);
    end
end
```

Two Clustering-Based Methods (K-means & Finite Mixture Models)

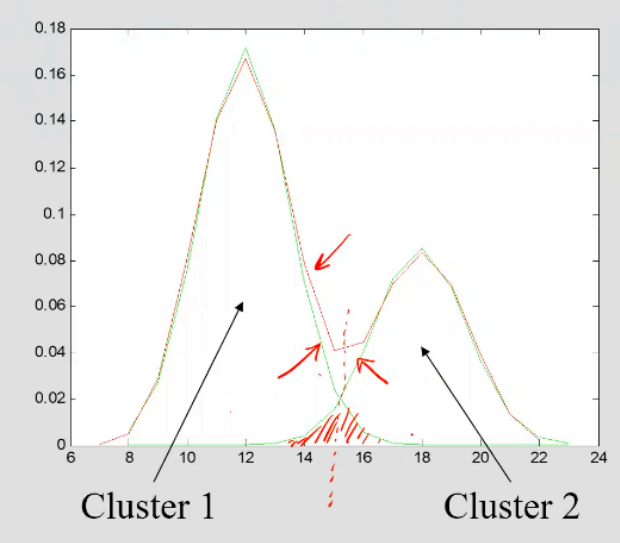
K-means [LAB + TUT]	**Finite Mixture Models**
- Create **partitions** by **grouping** pixels into **clusters**	- Suppose we know that there are m clustes/classes in the image - Suppose that the probability distribution of each cluster/ class can be modeled using a parametric model (ex: Gaussian, Gamma, Cauchy, etc.) => Turns into a Model based clusterization - **Idea**: We can model the probability distribution of the image as a mixture of m different probability ditributions, one for each cluster/class - **Formulation**: $f_X(x) = \sum_{i=1}^{m} a_i \, f_{Y_i}(x)$ - **Goal**: - Determine this set of m distributions - and determine which pixel values belong to each cluster/class based on which of these distributions give the highest probability

- **Steps for K-means** ("migrating means") clustering:
1. **Pick** (randomly) K pixels to act as the **initial centers** of the clusters (K: number of classes)
2. Assign each **pixel** to one **cluster** based on minimum Euclidean distance
3. If no pixels change clusters => STOP; otherwise, **recalculate cluster means** based on new pixel assignments
4. Return to Step 2.

Sketch of the steps:



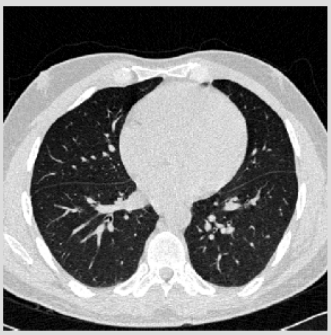
- Hopefully we reach the global munimum in 3rd plot, (not always happen)



- Steps:
. 1. Learn the parameters of the distribution models for the m clusters
. - ex: for Gaussian, learn the mean and std. deviation
. 2. For each possible pixel:
. 2-1) Determine the probability that its pixel intensity belongs to each of the m clustes based on the distribution model
. 2-2) Assign the pixel's cluster label to the cluster that provides the highest probability
. 3. Therefore, thresholds between clustes coincide at points of **equal probabilities**

K-means Example - Three class segmentation:

Example: Suppose we have two classes modeled by 2 Gaussians
$(\mu_1 = 3, \sigma_1 = 1, \mu_2 = 5, \sigma_2 = 1)$
- What is the threshold between these two classes?
. => Set up distribution equations for each class



$$f_{Y_1}(x) = \exp[-((x-3)^2)/(2(1)^2)]$$

$$f_{Y_2}(x) = \exp[-((x-5)^2)/(2(1)^2)]$$

=> Since threshold is at point of equal probabilities:

$$f_{Y_1}(x) = f_{Y_2}(x)$$

$$\exp[-((x-3)^2)/(2(1)^2)] = \exp[-((x-5)^2)/(2(1)^2)]$$

$$-((x-3)^2) = -((x-5)^2)$$

$$x^2 - 6x + 9 = x^2 - 10x + 25$$

$$4x = 16$$

$$x = 4$$

Histogram Segmentation - Pros and Cons

Advantages:

- Efficient (usually requires only one pass for simple segmentation cases)
- Good for multiple ditinct partitions in histogram

Disadvantages (Limitations):

- Often difficult to determine proper peaks in the histogram
- Difficult in situations where intensity is not sufficient to distinguish between two partitions
    - Ex: textured regions containing different mixes of intensities
- No spatial context

7.3 - Edge-Based Methods

Intro.

- **Edges**: defined as any pixels that belong to a line defining a high rate of change
- **Boundaries**: **defined as edges** that **separate regions** that are homogeneous in same feature
- Example: a region might contain a textured pattern that contains many edges

Segmentation Using Edge Strength

- For Segmentation based on edge strength:
    1. Perform edge detection
    2. Convert **edges** to **closed contours** (boundaries),
        - since edges detection:
            - a) misses boundaries
            - b) detects edges that are not boundaries

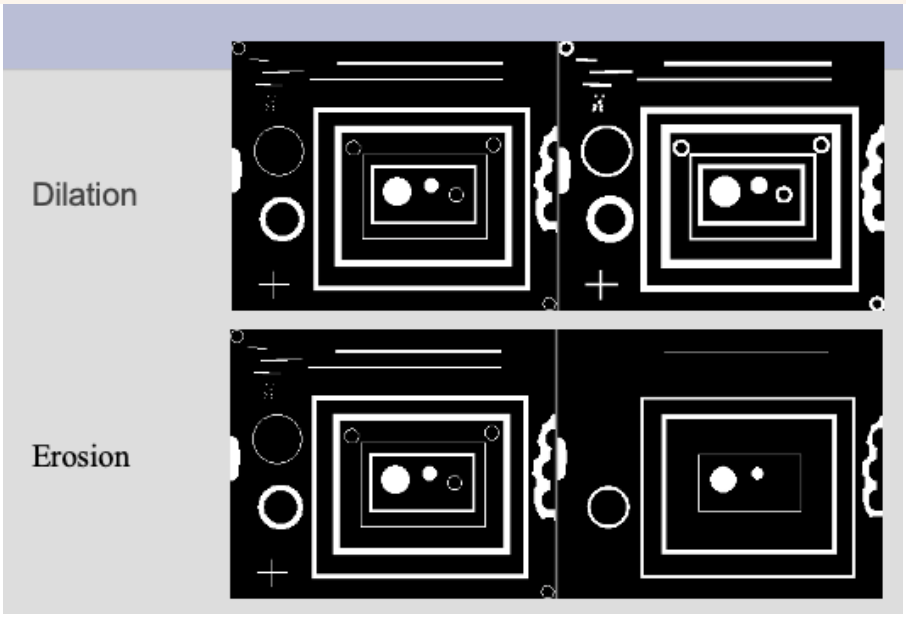
Morphological Operators

Image Morphology

- Image morphology provides a means to **convert** detected **edges** into **closed contours** (**boundaries**) that are more meaningful
- Morphology provides a means to perform edge linking (to complete boundaries) and removing serious edges that do not represent boundaries
- **Morphology**: form and structure of an object
- Note: focus here will be on binary images, but concepts can be extended to n grey levels

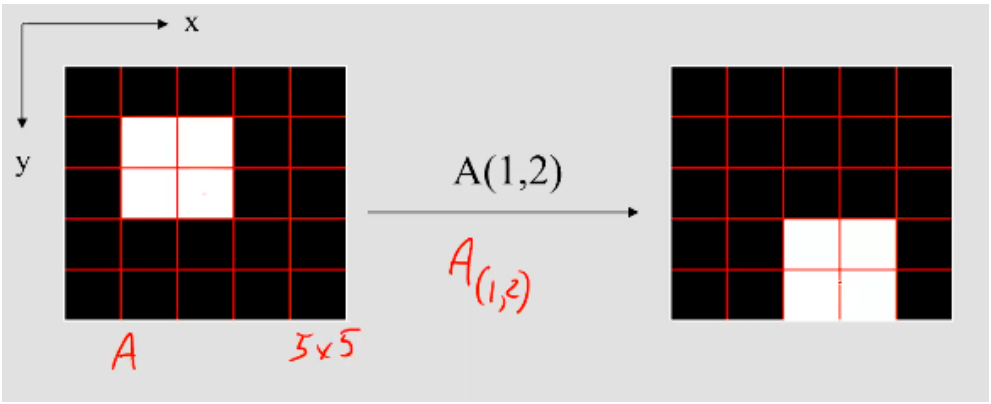
General Ideas:

- Two Basic Morphological operations:
    - **Dilation**: Set pixel to 1 if any neighbouring pixels are the value 1
    - **Erosion**: Set pixel to 0 if any neighbouring pixels are the value 0



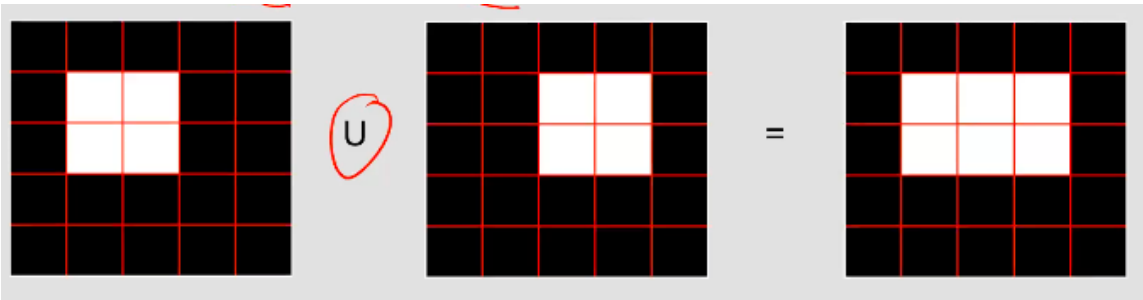
Translation:

- Mathematically, morphology can be implemented using set theory.
  - Set A: set of all white pixels (ex: detected edges)
- Translation: $(A)\gamma = \{c | c = a + \gamma, a \in A\}$
  - Where a, c, γ are in (x,y) coordinates
  - Basic building block for **Dilation and Erosion**



Dilation OR Gate:

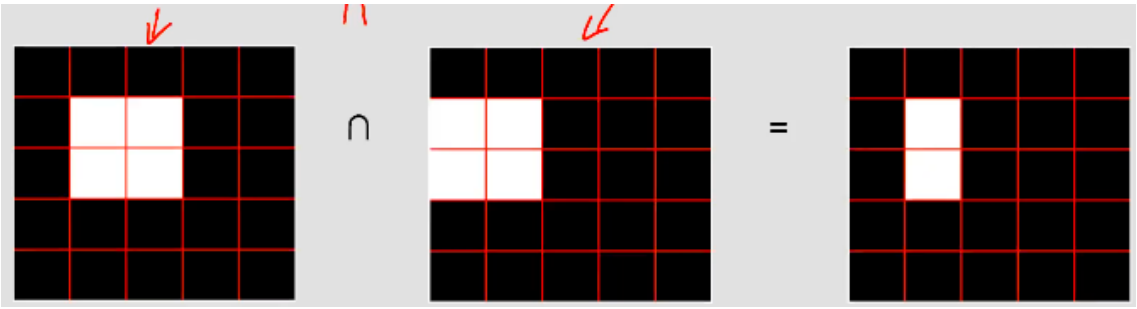
- Dilation: $C = A \oplus B = \{c \,|\, c = a + b, a \in A, b \in B\} = \bigcup_{b \in B} (A)_b = \bigcup_{a \in A} (B)_a$ (commutative)
  - A: image
  - B: "structuring element"
    - set of pixels referred to as the "structuring element" which defines the nature of the dilation
    - can take any shape
- Example:
  - if $B = \{(0,0), (1,0)\}$ = [x |]
  - $C = A \oplus B = (A + \{0,0\}) \bigcup (A + \{1,0\})$



  - C is a dilation of A using structuring element B

Erosion AND Gate:

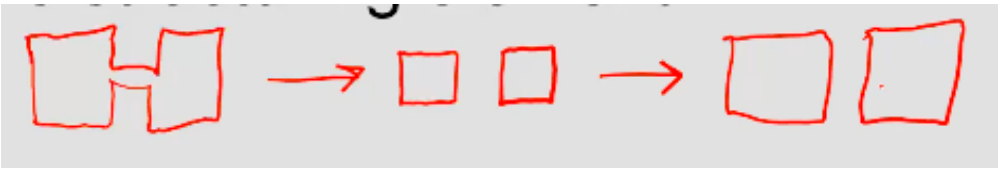
- Erosion: $C = A \ominus B = \bigcap_{b \in B} (A)_{-b}$ (not commutative)
  - A: image
  - B: "structuring element"
- Example:
  - $B = \{(0,0), (1,0)\}$ = [x |]
  - $C = A \ominus B = (A + \{0,0\}) \bigcap (A + \{-1,0\})$



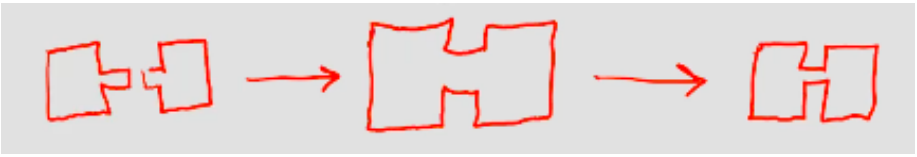
- C is an erosion of A using structuring element B

Openings (ED) and Closings (DE)

- Normally, single dilation/erosions not used
  - Often, these are iterated
- O/C: Combination of D/E in different order
- **Opening**: erosion followed by dilation using same structuring element

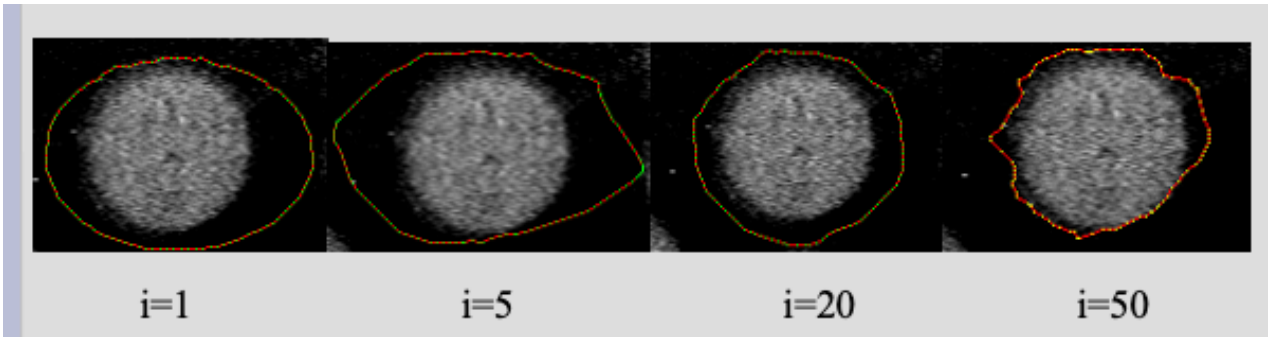


- **Closing**: dilation followed by erosion using the same structuring element



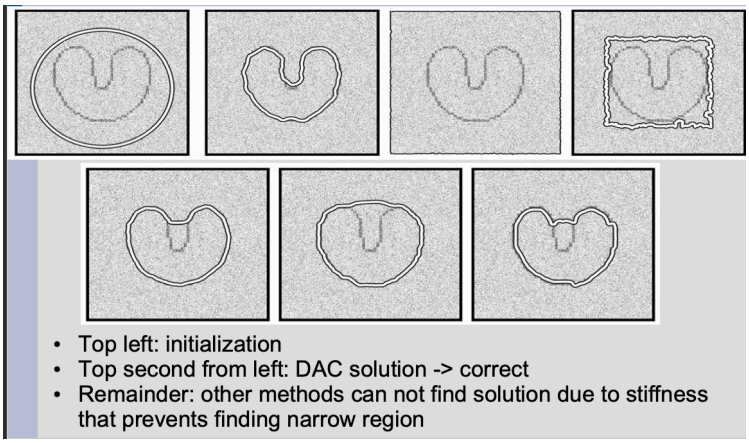
Active Contours ("Snakes") - Advanced Method

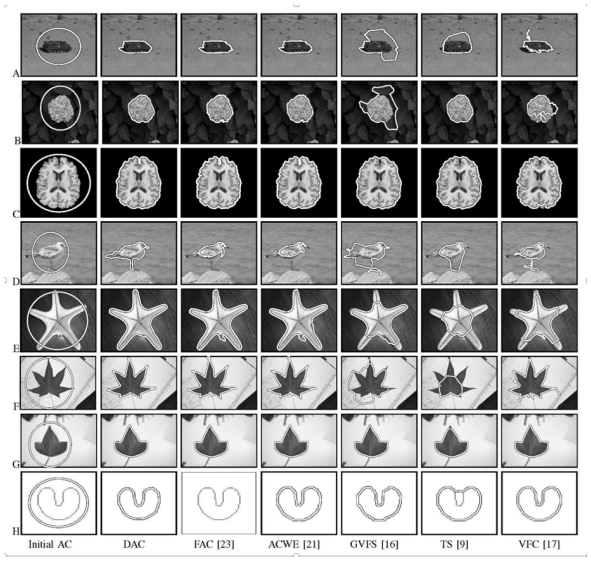
- Create partitions by forming rough boundaries around regions/objects of interest
  - and refining the boundaries until it matches the actual boundaries of the objects
- **Steps:**
  - Create initial, approximate boundary around the object
  - Calculate energy gradient between the current location of the boundary and its neighboring pixels
  - Expand or contract the boundary based on the gradient
- **Ex**:



- **Optimization function** based on two components that are summed:
    1. External force (normally an edge gradient)
    2. Internal stiffness (resistance to bending to offer some form to the system)

Decoupled Active Contour





Pros and Cons:

Advantages:

- Good for segmentating and tracking objects with deformable motion and clear boundaries
- Motion information can be extracted from the contour

Disadvantages:

- Relatively slow
- Ill-suited for situations where your regions do not have clearly defined boundaries
- Sensitive to noise
- Multiple objects need separate initializations which requires user interaction

8. Image Compression

8.1 Concepts

Why Compress Image?

- Popylarity of digital imaging systems
  - (EX: digital still-image/video cameras, digital x-ray/CT/MR/ultrasound systems, scanners)
  - has lead to a large volume of imaging data being created each day
- Uncompressed digital images have considerable storage and transmission **bandwidth requirements**
- To help with these problems, image and video **compression** is a necessity

Concept:

- Goal:
  - reduce amount of data that needs to be stored to represent an image
- Idea:
  - reduce the amount of redundant data that needs to be stored to reconstruct image
- Transform image data into a form that reduces statistical correlation and reduces information redundancy

Graph of Various Compression Techniques

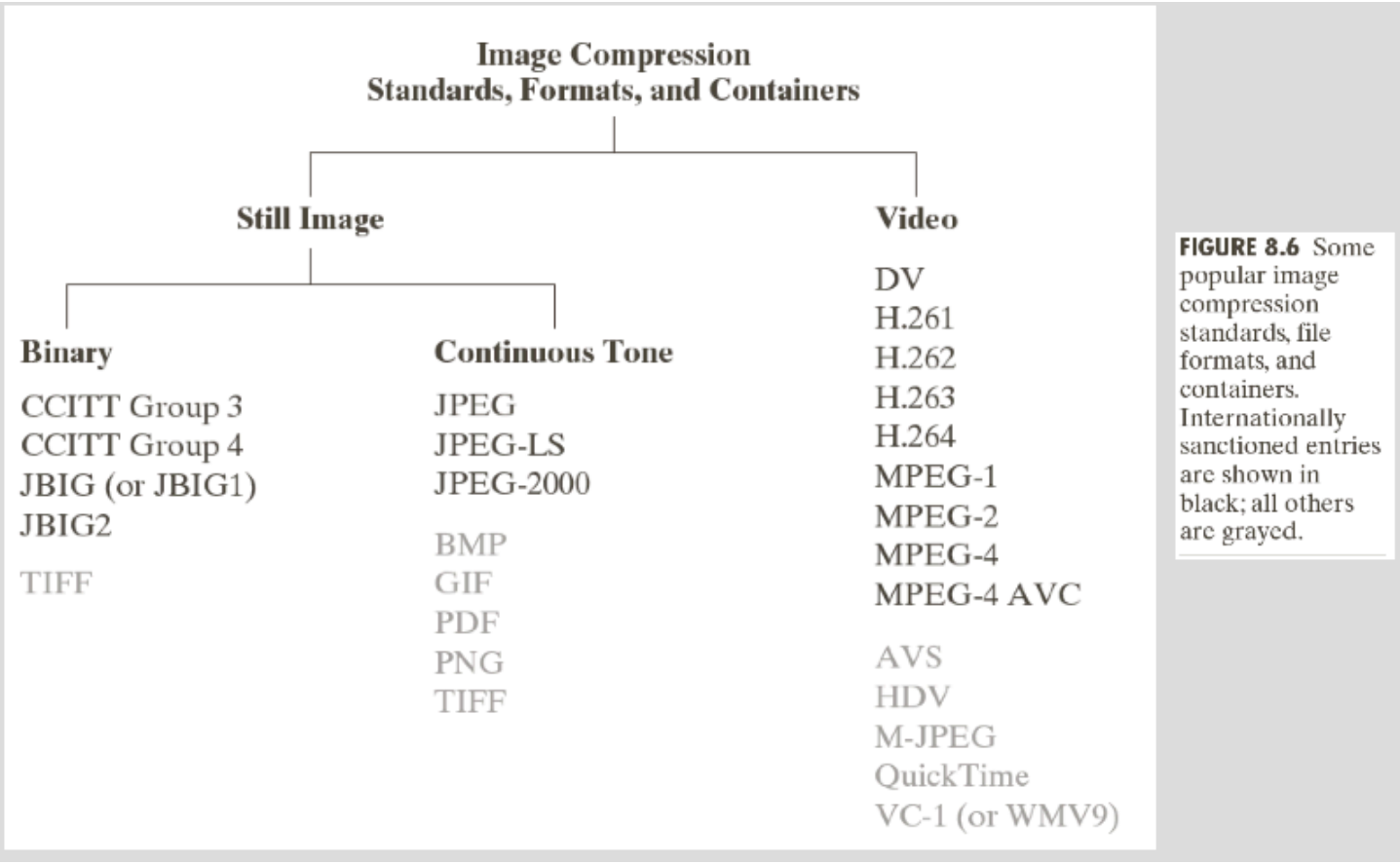


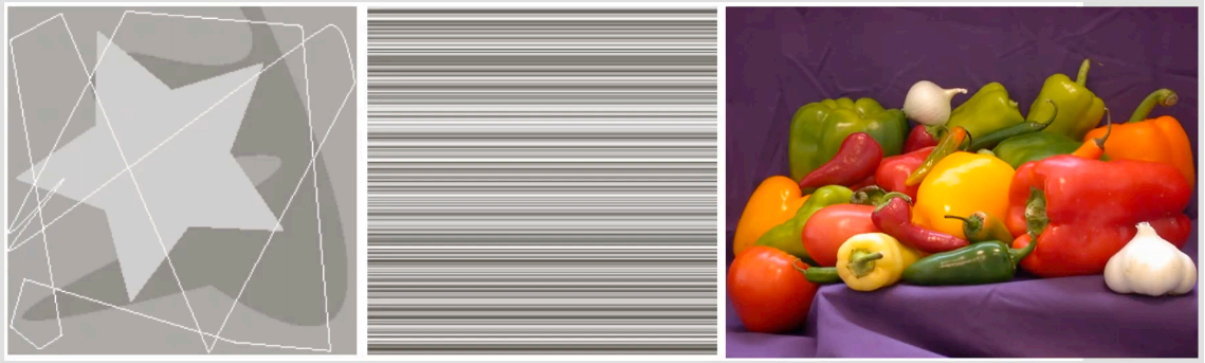
FIGURE 8.6 Some popular image compression standards, file formats, and containers. Internationally sanctioned entries are shown in black; all others are grayed.

Three Types of Redundancy

Types:

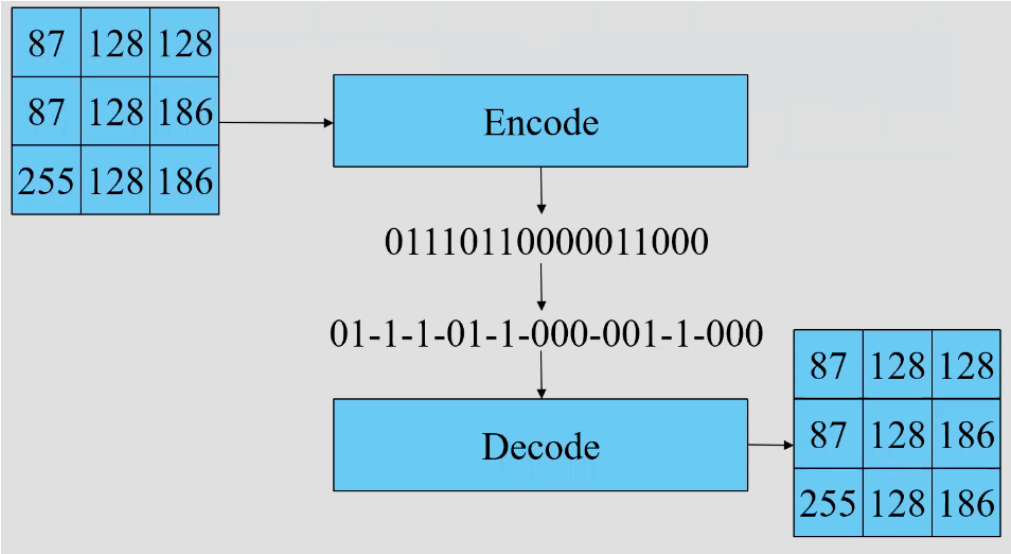
- Images can be generally described by three types of redundancies
  1. **Coding Redundancy**
  2. **Spatial and temporal redundancy**
  3. **Psycho-visual redundancy/ Irrelevant Information**

- Color is sth. we can squeeze down a lot, since HVS is not chromatic senstive relative to illumination changes



1. Coding Redundancy:

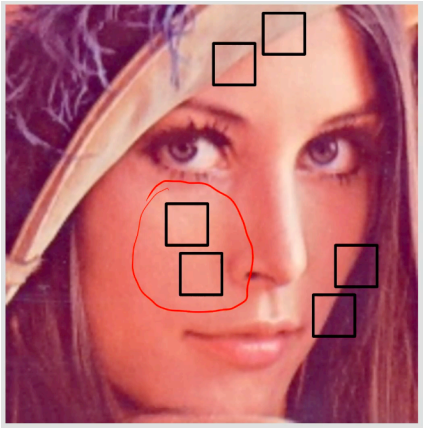
- Gray levels in an image can be viewed as random variables
- **Histogram** shows the **probability** that **each gray** level **appears** in an image
- In most images, certain gray levels are more probable than others
- By adjusting code length based on probability of gray levels, one can reduce coding redundancy
- Example:



2-a. Spatial Redundancy [Images]

- Most **images** possess a lot of structural and intensity self-similarity
- Therefore, value of a given pixel or region can be reasonably predicted by **neighboring pixels** or regions
    - neighboring pixels has high correlations

- One can take advantage of this **self-similar nature of images** to **reduce** the amount of information that needs to be stored.
- Example:



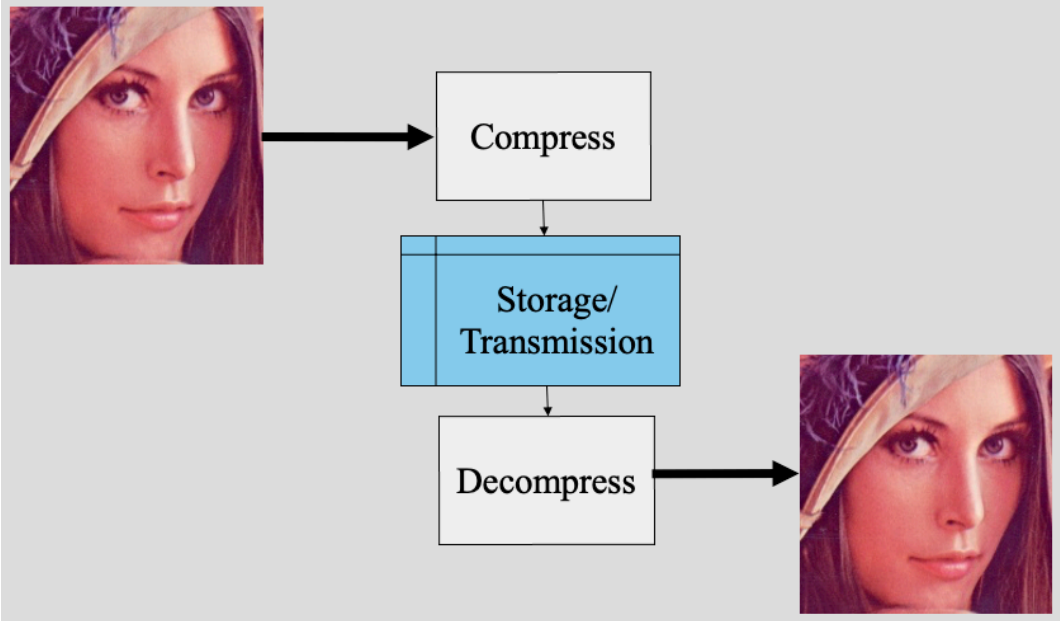
2-b. Temporal Redundancy [Video]

- In a **video sequence**, **pixels and/or regions** within each frame are similar to or dependent on pixels and/or regions from **adjacent frames**.
- Therefore, they can be reasonably predicted by temporally neighboring pixels or regions to reduce the amount of information that needs to be stored.

3. Psycho-visual Redundancy

- Eye doesn't respond equally to all visual information
- Certain information less important than others to the human vision system
    - Examples:
        - **More** sensitive to **illumination** changes than color changes
        - **Less** sensitive to **distortions** in **textures** than uniform regions
        - **Less** sensitive to **distortions** at **high spatial frequencies**
- Can reduce the amount of information needed to represent an image by removing information that's non-essential for visual processing
    - Examples:
        - Store **less color** information
        - Store **less information for uniform regions** than structures such as edges
        - Store **less information about high frequency** image characteristics
- Example:

Image Compression Framework



Types of Image Compression

Lossless Compression

- Exploits **coding** redundancy
- **Advantage**:
  - all of the image information is retained (allows for perfect image reconstruction)
- **Disadvantage**:
  - poor compression performance
- **Examples**:

    - Run-length encoding
    - Dictionary algorithms (e.g., LZW)
    - Entropy encoding (e.g. Huffman, arithmetic)
- Note: sacrifice compression for retaining data fedality

Lossy Compression

- Decompressed image is **different** but **close enough** to original image
- Exploits **spatial** and **psycho-visual** redundancies
- **Advantages**:
    - High compression performance
- **Disadvantages**:
    - Loss in original image information
- **Examples**:
    - JPEG and MPEG

Lossy - Fixed-rate Compression

- The compression rate achieved is the same for all images
- **Advantages**:
    - Simple and fast decoding
    - Allows for random access of information
- **Disadvantages**:
    - Low compression performance
- **Popular application**:
    - texture compression for real-time 3D applications (e.g., DXTC)
    - computer graphics

Lossy - Variable-rate Compression

- The compression rate achieved varies depending the underlying image characteristics
- **Advantages**:
    - High compression performance
- **Disadvantages**:
    - More complex to encode and decode
    - Difficult to randomly access information
- Popular **application**:

- **still image** and video compression (e.g., JPEG/MPEG)

8.2/3 Coding Methods

Coding Methods:

- **Huffman Coding**
  - **Lossless**
  - **Variable Rate** (data dependent)
  - Used as **final step** in **JPEG**, MP3, etc.
- **Block Transform Coding (BTC)**
  - **Lossy**
  - **Fixed Rate (deterministic)**
  - Used in computer graphics

1. Variable Length Coding (**Huffman Coding** : Tree Graph Based : Unicoding : Lossless)

- Decrease code length as probability of occurrence increases
- Can achieve **much lower coding redundancy** by reducing **the number of bits** needed to **store** data in an image
- Problem: how do we actually determine what code to use?
  - One set of codes typically not well-suited for all images
- **Decrease code length as probability of occurrence increases**
- **Example:**

r_k	$p_r(r_k)$	Code 1	$l_1(r_k)$	Code 2	$l_2(r_k)$
$r_{87} = 87$	0.25	01010111	8	01	2
$r_{128} = 128$	0.47	10000000	8	1	1
$r_{186} = 186$	0.25	11000100	8	000	3
$r_{255} = 255$	0.03	11111111	8	001	3
r_k for $k \neq 87, 128, 186, 255$	0	—	8	—	0

  - For ‘Code 1’:
    - average number of bits required to code image is 8 bits per pixel (bpp)
  - For ‘Code 2’:

$$L_{avg} = \sum_{k=0}^{L-1} l(r_k) p_r(r_k)$$
$$= l(r_{87}) p_r(r_{87}) + l(r_{128}) p_r(r_{128}) + l(r_{186}) p_r(r_{186}) + l(r_{255}) p_r(r_{255})$$
$$= (2)(0.25) + (1)(0.47) + (3)(0.25) + (3)(0.03)$$
$$= 1.81$$

- Code 1 has **high coding redundancy** as it requires more bits than necessary to code

Approach: Huffman

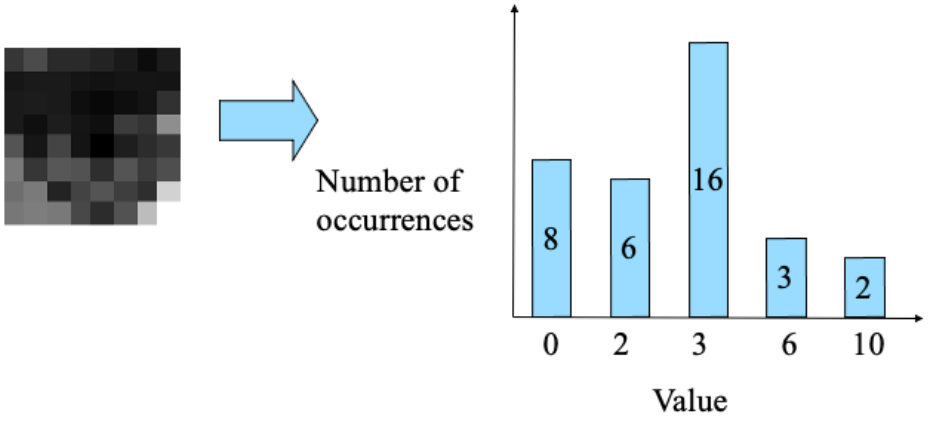
- Popular method for removing coding redundancies is *Huffman coding*
- Used extensively within common standards for compression (JPEG, H.264, MPEG-1,2,4, etc.)
- Based on probabilities
  - **Most** commonly **occurring** symbol receives **shortest** code
  - **Least** common symbols receive **longest** codes
- Source symbols can be intensities, pixel differences, run lengths, etc.

Data-Adaptive Variable Length Coding

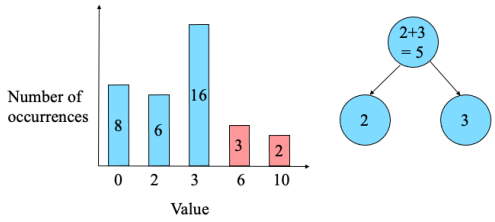
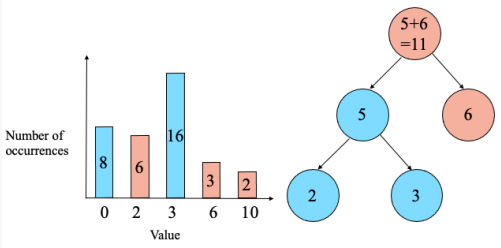
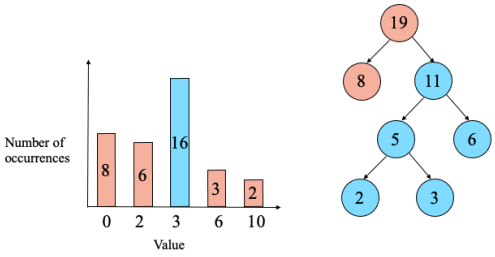
- Idea of Huffman coding: change the set of codes used to compress achieve better compression **specifically for the image**
- Note: Huffman will be used to **code coefficients** produced by a **filtering** process, not the image data directly, using JPEG

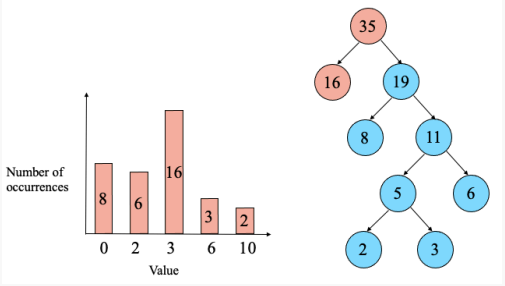
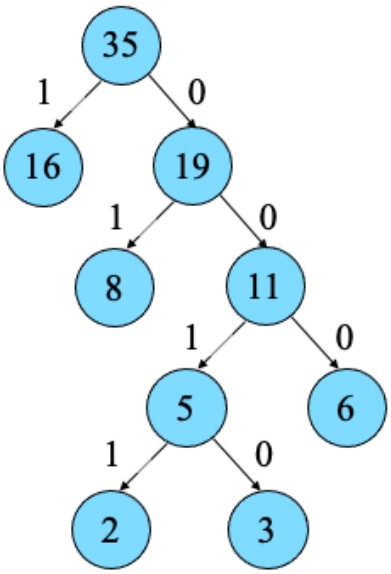
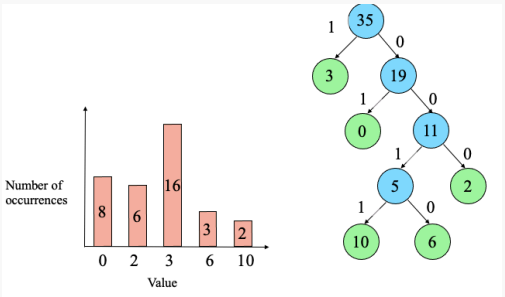
Huffman Coding : Lossless

- **Goal:** Build minimal length encodings based on frequency of occurrences in the image
- **Steps:**
  1. Determine frequency of occurrences for each possible value in the image

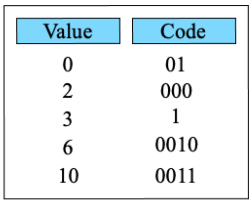
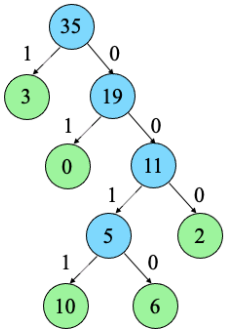


2. Construct **Huffman tree**
    - Huffman Trees are **binary trees** where:
        - Root node has highest probability of occurrence
        - Lowest leaf nodes have the lowest probability of occurrence
        - Probability of occurrence decreases as we traverse down the tree
    - Detailed Steps:

Action	Graph Example
2-a). Take the two lowest occurrences as the leaf nodes and the sum of the occurrences as their parent node	
2-b). Compare value of the parent node with next lowest occurrence -Lower of the two becomes left child node -Higher of the two becomes right child node -Sum of the two becomes parent node	
2-c). Repeat until all values have been used	

...	
2-d). Assign '1' to the left child node and '0' to right child node	
2-e). Replace leaf nodes with their corresponding values	

3. **Encode image** based on codes generated from **Huffman tree**:
    -

Action (cont.)	Graph Example
3-a). Compute set of codes from Huffman tree by traversing tree	 

Value	Code
0	01
2	000
3	1
6	0010
10	0011

Decoding

- Q: [cont. from Huffman tree example]
  - So, if you had a sequence:
    - 101101110011011001010000
  - How would you decode it?
    - 1-01-1-01-1-1-0011-01-1-0010-1-000
      - **Unicoding** => Lossless
  - Or
    - 3,0,3,0,3,3,10,0,3,6,3,2
- A:

Compression Efficiency

- A: [cont. from Huffman tree example]

$$L_{avg} = \sum_{k=0}^{L-1} l(r_k) pr(r_k)$$
$$= (0.2286)(2)+(0.1714)(3)+(0.4571)(1)+(0.0857)(4)+(0.0571)(4)$$
$$=2$$

  - This gives us a compression ratio of:
    - 8 bits per coefficient/2 bits per coefficient = 4:1

Another Huffman Coding Example

Original source		Source reduction			
Symbol	Probability	1	2	3	4
a_2	0.4	0.4	0.4	0.4	0.6
a_6	0.3	0.3	0.3	0.3	0.4
a_1	0.1	0.1	0.2	0.3	
a_4	0.1	0.1	0.1		
a_3	0.06	0.1			
a_5	0.04				

FIGURE 8.7
Huffman source reductions.

Original source			Source reduction							
Symbol	Probability	Code	1		2		3		4	
a_2	0.4	1	0.4	1	0.4	1	0.4	1	0.6	0
a_6	0.3	00	0.3	00	0.3	00	0.3	00	0.4	1
a_1	0.1	011	0.1	011	0.2	010	0.3	01		
a_4	0.1	0100	0.1	0100	0.1	011				
a_3	0.06	01010	0.1	0101						
a_5	0.04	01011								

FIGURE 8.8
Huffman code assignment procedure.

- Compatibility K-Lite Codec pack
  - “internationally sanctioned” = approved by ISO – International Standards Organization or IEC or ITU-T
  - HEVC (high ef)
  - successor to HEVC is under active development. Known as VVC (Versatile Video Coding), the standard is due for completion in October 2020 and is likely be approved as H.266 (VVC). Working Draft 3 of the VVC specification was released in October 2018, so the work towards standardisation is broadly on track. However, the licensing model for VVC remains unclear until the standard is complete and essential features of the codec are finalised. The Media Coding Industry Forum – formed in September 2018 – is working to avoid the problematic patent situation that has hitherto thwarted HEVC’s widespread adoption across the industry.ficiency video coding) to be released in 2020; one-quarter size of MPEG-2 to achieve 4K resolution

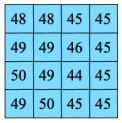
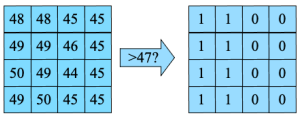
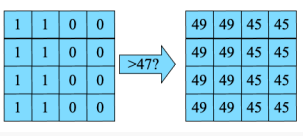
2. Fixed-rate Compression (Palette-based Compression)

- Idea:
  - As mentioned earlier, values of pixels can be well predicted by neighboring pixels
  - Instead of storing all pixel values, store a few pixel values that are representative of the neighborhood

- Encode other pixels in the neighborhood as the closest value in the palette

Block Truncation Coding (BTC) : Lossy Compression

- Blocked-Based Coding
  - Block-based algorithm that preserves local mean and standard deviation of image
- NOTE:
  - Not the same as Block Transform Coding; perhaps a subset or rudimentary form of Block Transform Coding.
- **Steps:**

Instructions	**Examples**
1.Divide image into 4x4 pixel blocks 2. For each block, compute the mean and std. deviation of pixel intensities	• e.g.,  $\mu \approx 47$ $\sigma \approx 2$
3. Classify each pixel in the block as follows $g(x,y) = \begin{cases} 1, & f(x,y) > \mu \\ 0, & f(x,y) \le \mu \end{cases}$	• e.g., 
4. Store binary block along with mean and standard deviation 5. To decode, compute decoding values l (low) and h (high) based on mean and standard deviation $l = \mu - \sigma\sqrt{\frac{n_{x>\mu}}{n_{total}-n_{x>\mu}}}$ $h = \mu + \sigma\sqrt{\frac{n_{total}-n_{x>\mu}}{n_{x>\mu}}}$ NOTE: Once a threshold, xth, is selected the output levels of the *quantizer* (a and b) are found such that the first (mean) and second moments are preserved in the output.	• e.g., $l = 47 - 2\sqrt{\frac{8}{16-8}} = 45$ $h = 47 + 2\sqrt{\frac{16-8}{8}} = 49$
6. Decode binary block as follows $\hat{f} = \begin{cases} h, & g(x,y) = 1 \\ l, & g(x,y) = 0 \end{cases}$	• e.g., 

BTC Compression Rate

- Suppose we are given a 4x4 grayscale image, with each pixel represented by a value from 0 to 255.
- Number of bits required to store this image in an uncompressed format is 4x4x8bits = 128 bits
- Bit rate of image in an uncompressed format is 8 bpp (bits per pixel)
- **EX:**
  - Supposed we compress the image using BTC
  - Mean and standard deviation each require **8 bits**
  - Binary block requires **4x4x1bit=16 bits**
  - Number of bits required to store this image in BTC compressed format is
    - **16bits**+2x**8bits**=**32 bits**
  - Bit rate of image in BTC compressed format is **32bits/16pixels** = **2 bpp** (bits per pixel)
    - [Compression rate = 8 bpp:2 bpp or **4:1**]

3. Transform-Based Compression : Lossy Compression : Freq. Based

- As mentioned earlier, we wish to transform image data into a form that **reduces statistical correlation**
- We also saw that when images are transformed into the **frequency domain** using the **Fourier Transform**
  - Most of the energy **resides** in **low frequency** components
  - A good approximation of the original image can be **reconstructed** using a **few components**

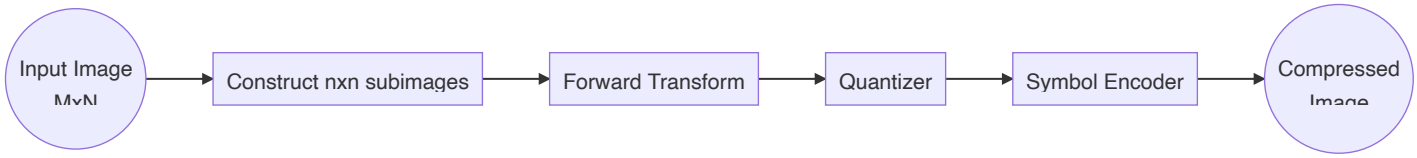
Block Transform Coding

- Idea: What if we apply an image transform like the Fourier Transform to an image and **encode the transform coefficients** in a **reduced, lossy** form instead?
- Many of the transform **coefficients** have **low** associated **energies** and **can be discarded or coarsely quantized** with little image distortion
- **Issue**: Image **transforms** such as Fourier Transform are have relatively expensive from both computational and storage perspective
  - Must **hold all data in memory at once** to perform transform
  - Must **account for all pixels** in image when **computing** transform
- Difficult to implement in consumer-level devices such as DVD players and digital cameras

- **Solution**:
  - Break images into a set of smaller sub-images ("**blocks**")
  - Apply image transform on the sub-images independently
- **Advantage:**
  - The amount of information that needs to be **stored** to **transform** a sub-image is **small**
  - Since all operations are **independent**, they can be **performed in parallel** to improve computational efficiency

Framework

Compression Flowchart



Decompression Flowchart

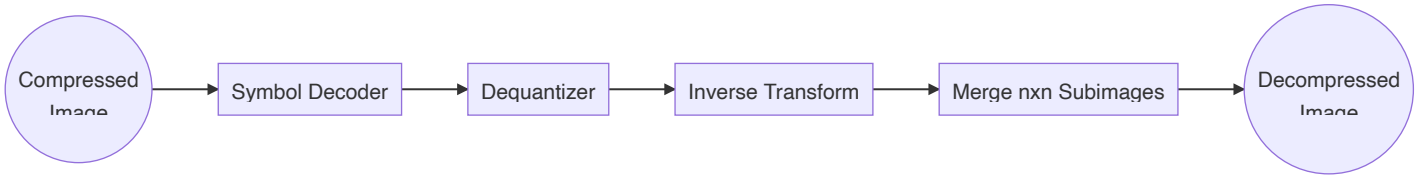


Image Transforms

- Converts images from one representation (e.g., spatial) to another (e.g., frequency)
- **Forward transform**
  - $T(u,v) = \sum_{x=0}^{n-1} \sum_{y=0}^{n-1} g(x,y)\, r(x,y,u,v)$

$$T(u,v) = \sum_{x=0}^{n-1}\sum_{y=0}^{n-1} g(x,y)\, r(x,y,u,v)$$

Transform coefficients — image — Forward transform kernel

- **Inverse transform**
  - $g(x,y) = \sum_{u=0}^{n-1}\sum_{v=0}^{n-1} T(u,v)\, s(x,y,u,v)$

$$g(x,y) = \sum_{u=0}^{n-1}\sum_{v=0}^{n-1} T(u,v)\, s(x,y,u,v)$$

Inverse transform kernel

  - Essentially representing an image using a set of basis functions

Selection of Image Transforms

- Selection of image transform is very important to compression performance as well as computational performance
- Some well-known image transforms:
  1. Karhunen-Loeve (KL) transform
  2. Fourier transform (FT)
  3. Walsh-Hadamard transform (WHT)
  4. Discrete cosine transform (DCT)

Table of Four Image Transforms

Transforms	**Pros & Cons**
Karhunen-Loeve (KL) transform - Optimal transform in terms of compression . - Minimizes mean square error . - Statistically decorrelated (off-diagonal elements of covariance matrix are zero)	

$$G = \Phi^T F$$

$$F = \Phi G \text{ (since } \Phi^{-1} = \Phi^T)$$

$$\text{where } \Phi^T \Sigma \Phi = \Lambda$$

Columns are eigenvectors | Covariance | Eigenvalues

NOTE: Also known as **principal components analysis (PCA)**

- Basis functions essentially based on the **eigenvector**
- **Eigenvectors** associated with **highest eigenvalues** will be associated with **most variance**

Advantages:
- Provides optimal compression performance from an energy compaction perspective

Disadvantages:
- Computationally expensive, since the transform is data-dependent and deriving basis functions is non-trivial

=> not good for operational compression.

Fourier transform
- Transformation kernels

$$r(x,y,u,v) = e^{-j2\pi(ux+vy)/n}$$

$$s(x,y,u,v) = \frac{1}{n^2} e^{j2\pi(ux+vy)/n}$$

Advantages:
- Hardware acceleration available on CPUs

Disadvantages:
- -Relatively poor compression performance

Walsh-Hadamard Transform (WHT)
- Transformation kernels

$$r(x,y,u,v) = s(x,y,u,v) = \frac{1}{n}(-1)^{\sum_{i=0}^{m-1}[b_i(x)p_i(x)+b_i(y)p_i(v)]}$$

$$\text{where } n = 2^m$$

- NOTE: $b_i(x)$ is the ith bit (from right to left) in the binary representation of (x), $p_i(x) = b$
- WH kernels consist of **alternating** plus (white) and minus (black) 1s in **checkerboard** pattern

Advantages:
- Computationally simple

Disadvantages:
- Relatively poor compression performance (worse than Fourier transform)

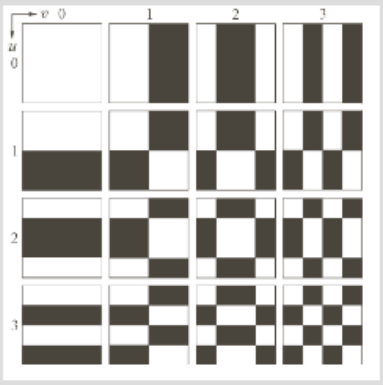


FIGURE 8.22 Walsh-Hadamard basis functions for $n = 4$. The origin of each block is at its top left.

Source: Gon

Discrete Cosine Transform (DCT)

- Transformation kernels

$$r(x,y,u,v) = s(x,y,u,v)$$
$$= \alpha(u)\alpha(v)\cos\left[\frac{(2x+1)u\pi}{2n}\right]\cos\left[\frac{(2y+1)v\pi}{2n}\right]$$

where

$$\alpha(u) = \begin{cases} \sqrt{\frac{1}{n}} & for\ u = 0 \\ \sqrt{\frac{2}{n}} & for\ u = 1,2,\ldots,n-1 \end{cases}$$

-NOTE: Forward and inverse transforms are the same, therefore, easier to implement in hardware.

- Kernel Ex:

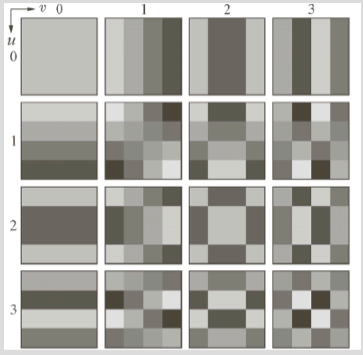


FIGURE 8.23 Discrete-cosine basis functions for $n = 4$. The origin of each block is at its top left.

Source: Gonzalez and Woods

Advantages:

- **Computational efficient** (easy to implement in hardware)
- **High compression performance** (closely approximates performance of KLT for many images)

=> Given these benefits, DCT has become an international **standard** for transform coding

How do we deal with Color?

- As mentioned before, RGB color space is highly redundant and correlated
  - Reduces compression performance
- **Solution**: Use a color space that decorrelates information such as luminance and color
  - e.g., Before image transform, convert image from RGB color space to **YCbCr**
  - Allows luma and **chroma** channels to be processed independently

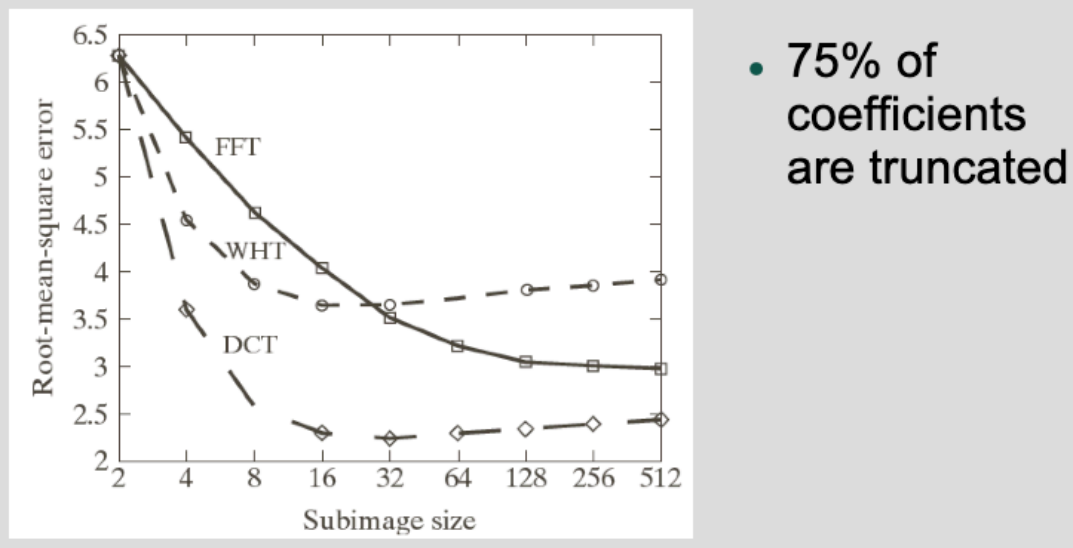
Chroma Subsampling

- The human vision system is significantly more sensitive to variations in brightness (luma) than color (chroma)
- **Idea**:
  - **reducing** the amount of chroma information stored compared to the amount of luma information should **have little impact** on perceived image quality
- **Example: JPEG**
  - In JPEG, image is converted from RGB to YCbCr
  - The resolution of the Cb and Cr channels are reduced
  - Commonly, the Cb and Cr channels are sub-sampled by at factor of 2 both horizontally and vertically

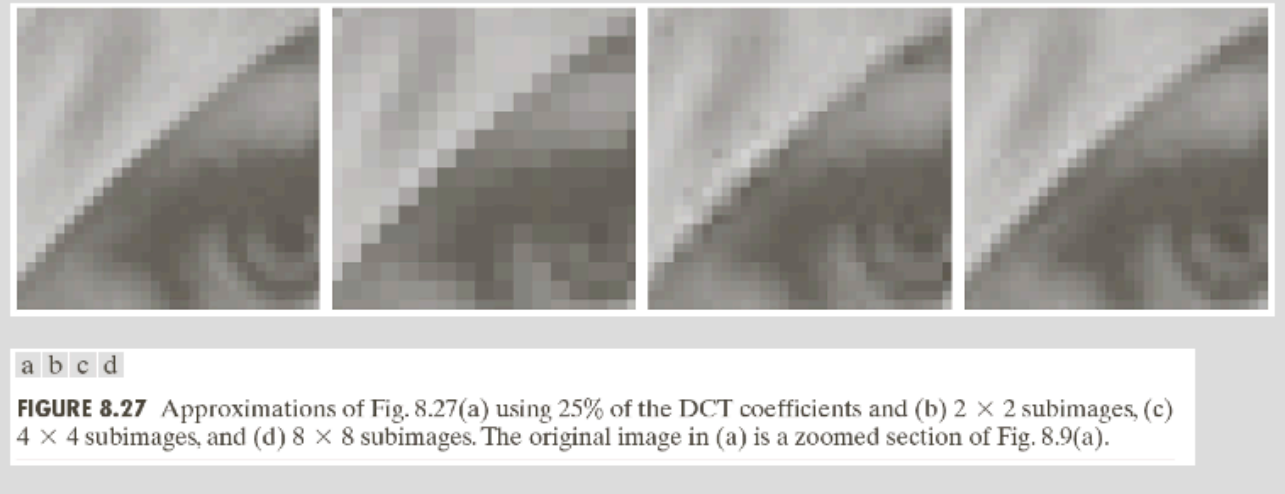
Sub-image Construction

- After chroma **subsampling**, the individual channels are divided into a set of **n *x* n sub-images**
- Generally, compression **performance increases** as sub-image size **increases**
- However, computational **complexity increases** as sub-image size **increases**
- Drawing a balance between compression performance and compression efficiency, 8x8 and 16x16 sub-image sizes are used

- **Reconstruction Error vs. Sub-image Size:**



- **Reconstruction Quality vs. Sub-image Size:**



a b c d

FIGURE 8.27 Approximations of Fig. 8.27(a) using 25% of the DCT coefficients and (b) 2 × 2 subimages, (c) 4 × 4 subimages, and (d) 8 × 8 subimages. The original image in (a) is a zoomed section of Fig. 8.9(a).

Quantization

- As mentioned earlier, the human vision system is much more sensitive to variations in low frequency components than high frequency components
- Also, much of the energy is packed in the low frequency components
- **Idea**:
  - **high frequency** components can be represented **coarsely ("quantized")** without perceptually noticeable degradation in image quality
- **Steps:**
  1. Transform image *f* from the spatial representation to *T* in the transform domain
  2. Quantize *T* based on a quantization matrix *Z* designed for the human vision system based on perceptual importance

$$\hat{T}(u,v) = round\left[\frac{T(u,v)}{Z(u,v)}\right]$$

- **Quantization Matrix (JPEG)**

16	11	10	16	24	40	51	61
12	12	14	19	26	58	60	55
14	13	16	24	40	57	69	56
14	17	22	29	51	87	80	62
18	22	37	56	68	109	103	77
24	35	55	64	81	104	113	92
49	64	78	87	103	121	120	101
72	92	95	98	112	100	103	99

  Big deonimators -> Big quantization in high frequency (Bottom-Right corner)

- **Effect of Quantization Level on Image Quality**

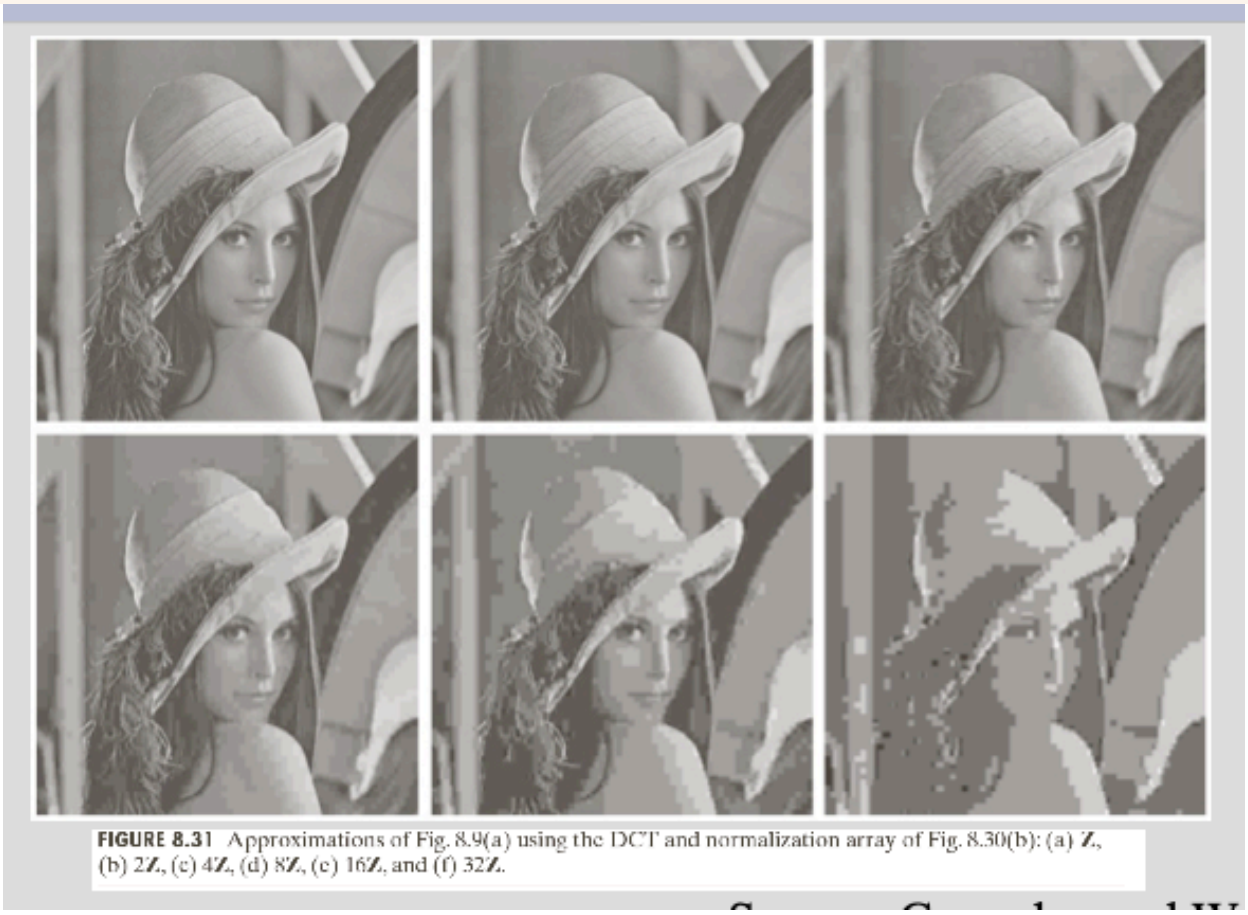
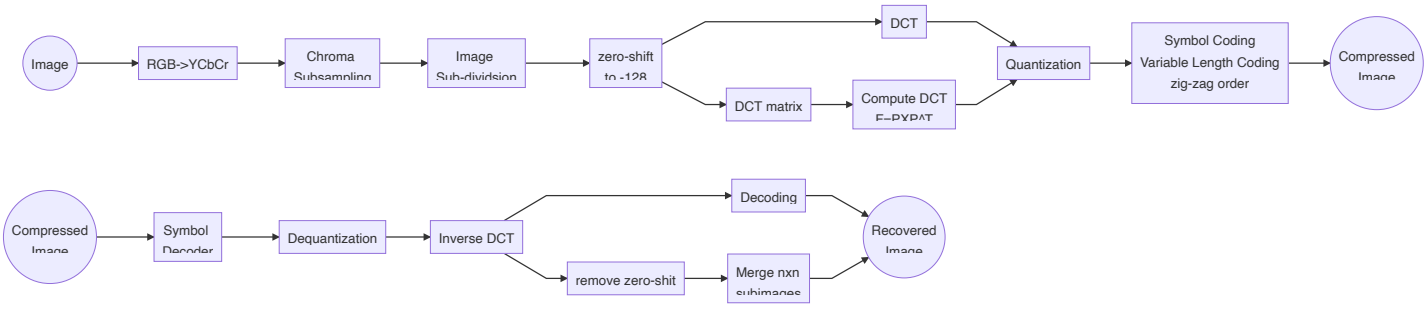


FIGURE 8.31 Approximations of Fig. 8.9(a) using the DCT and normalization array of Fig. 8.30(b): (a) Z, (b) $2Z$, (c) $4Z$, (d) $8Z$, (e) $16Z$, and (f) $32Z$.

- Observations:
  - As quantization increases
    - **fine** image **detail** starts **to be lost** (e.g., mouth and feathers start to degrade until completely disappearing)
    - **Blocking artifacts** (i.e., visible boundaries between sub-images) becomes increasingly prominent
  - However, uniform regions with little detail are significantly less affected by quantization

- **Adaptive Quantization**
  - Fact:
    - **High** quantization is **perceptually acceptable** in **uniform** regions
    - **Low** quantization is **needed** in regions with **structural detail**
  - **Idea:**
    - Adjust degree of quantization based on amount of image detail within a sub-image
      - **Measure level of image detail** (e.g., variance) of the sub-image
      - **Decrease quantization** for **sub-images with high image detail**

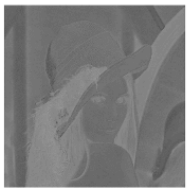
8.4 JPEG Detailed Process (Table):



Steps

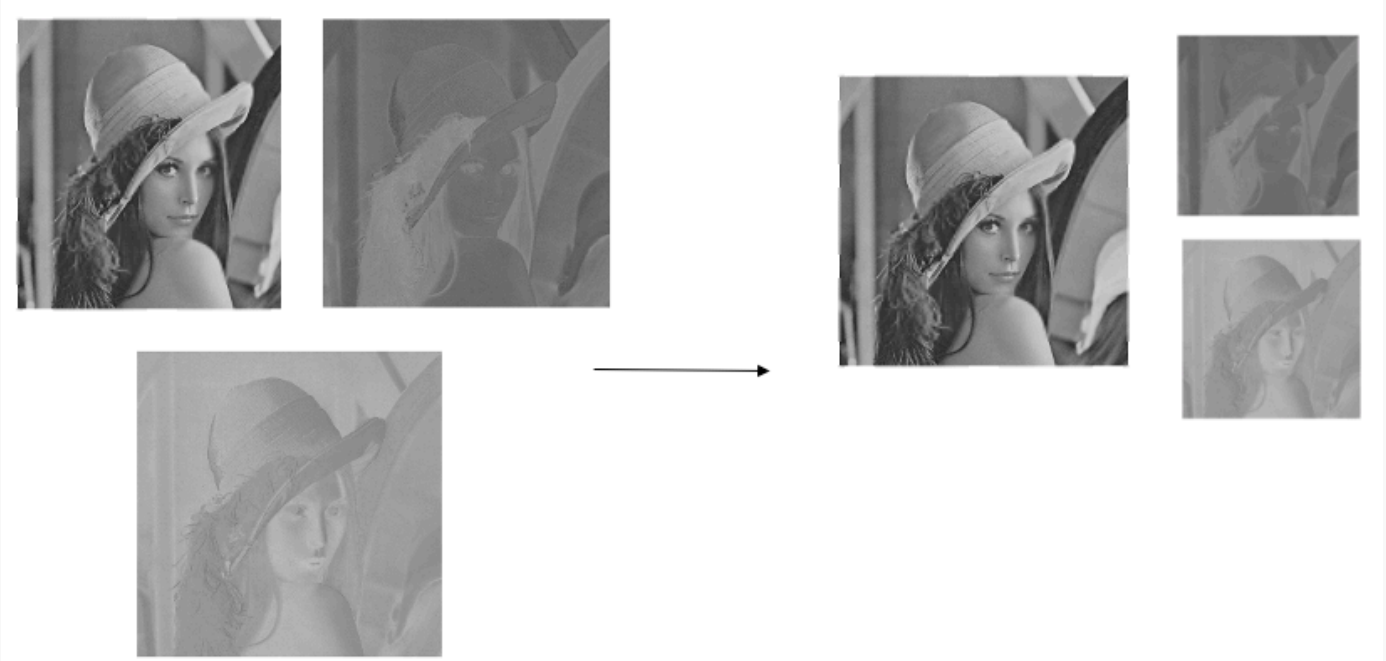
1. **Color Space Transform**

- Convert image from RGB Color space to YCbCr color Space



2. **Chroma Subsampling**

- Reduce the resolution of Cb and Cr channels by a factor of two in each dimension.



3. **Image Sub-division**

- For each channel, divide the image into 8x8 blocks

4. **Zero-shift**

- For each block, shift values by -128 to center around zero

. “Zero shift reduces the internal precision requirements in the DCT calculations.”

70	84	63	62	59	52	44	53
50	52	52	49	47	49	49	48
52	53	52	45	42	45	48	67
52	46	54	49	46	75	69	128
90	50	80	48	35	55	64	72
114	70	92	89	66	90	74	87
107	115	59	79	90	76	66	173
113	117	114	82	64	89	158	202

→

-58	-44	-65	-66	-69	-76	-84	-75
-78	-76	-76	-79	-81	-79	-79	-80
-76	-75	-76	-83	-86	-83	-80	-61
-76	-82	-74	-79	-82	-53	-59	0
-38	-78	-48	-80	-93	-73	-64	-56
-14	-58	-36	-39	-62	-38	-54	-41
-21	-13	-69	-49	-38	-52	-62	45
-15	-11	-14	-46	-64	-39	30	74

5. **DCT (Discrete Cosine Transform) Kernel**

- Note: Figure helps to visualize the filters, not how implemented in practice!!!!

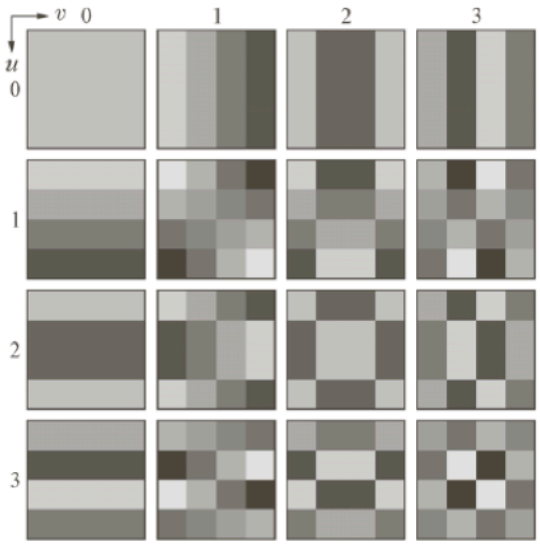


FIGURE 8.23
Discrete-cosine basis functions for $n = 4$. The origin of each block is at its top left.

- Figure helps to visualize the filters
- Note: not how implemented in practice!!!!

Source: Gonzalez and Woods

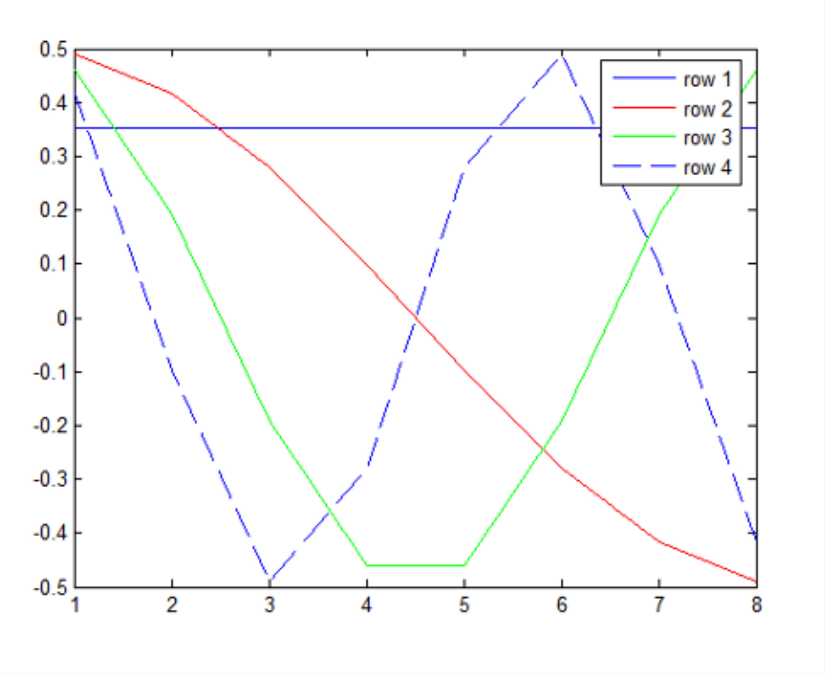
- (0,0): 1st 4x4 filter : averaging filter => Sub Img * Each Filter (Convolution) => 4x4 block

=> this is a visualization but not actual implementation, proceed for **Step 6,7 with DCT Matrix**

- Top Left: low frequency, Bottom Right: high frequency

6. **DCT Matrix**

- Composed of 8x 1-D cosines of varying frequencies (arranged in rows)



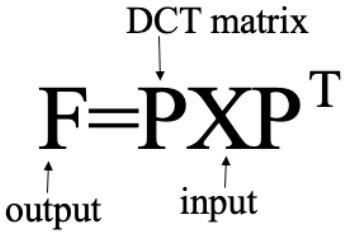
- Calculated using 1-d version of DCT formula introduced in Section 8-3

. - Each row is calculated using a 1-d version of the 2-d version of the DCT in Section 8.3.

```
0.3536    0.3536    0.3536    0.3536    0.3536    0.3536    0.3536    0.3536
0.4904    0.4157    0.2778    0.0975   -0.0975   -0.2778   -0.4157   -0.4904
0.4619    0.1913   -0.1913   -0.4619   -0.4619   -0.1913    0.1913    0.4619
0.4157   -0.0975   -0.4904   -0.2778    0.2778    0.4904    0.0975   -0.4157
0.3536   -0.3536   -0.3536    0.3536    0.3536   -0.3536   -0.3536    0.3536
0.2778   -0.4904    0.0975    0.4157   -0.4157   -0.0975    0.4904   -0.2778
0.1913   -0.4619    0.4619   -0.1913   -0.1913    0.4619   -0.4619    0.1913
0.0975   -0.2778    0.4157   -0.4904    0.4904   -0.4157    0.2778   -0.0975
```

T = dctmtx(SIZE);

7. **Compute DCT:** $F = PXP^T$




```
-58   -44   -65   -66   -69   -76   -84   -75
-78   -76   -76   -79   -81   -79   -79   -80
-76   -75   -76   -83   -86   -83   -80   -61
-76   -82   -74   -79   -82   -53   -59     0
-38   -78   -48   -80   -93   -73   -64   -56
-14   -58   -36   -39   -62   -38   -54   -41
-21   -13   -69   -49   -38   -52   -62    45
-15   -11   -14   -46   -64   -39    30    74
```

X

```
-438   -24    90   -39    22   -11    25   -12
-159    37   -67    26   -19     2    -8     3
  73     3    18   -15    -1   -20   -31    -3
  -8    58   -33    29     8     8     9    -3
  24   -22    26   -18   -19    13     4     9
  14   -16     6    -3    31   -44    -5   -22
  16    15   -10     2   -17    26     4     6
 -25    26     6    -1     5   -10    -9     5
```

F

DCT Memory Requirements

- Notice that the DC coefficient (i.e., F(0,0)) requires more than 8 bits to represent
- Since much of the energy is stored in the DC coefficient
- Therefore, internally, more bits are required for storage during DCT computation
- This does not increase memory requirements that much as we are only operating on small 8x8 blocks at a time.

floor(blkproc(f - 128, [8, 8], 'P1 * x * P2', T, T'));

8. **Quantization**

- Divide DCT coefficients by quantization matrix

$$R(u,v) = \text{round}\left(\frac{F(u,v)}{Q(u,v)}\right)$$

Input coefficients

Quantized coefficients

Quantization matrix

$$Q = \begin{bmatrix} 16 & 11 & 10 & 16 & 24 & 40 & 51 & 61 \\ 12 & 12 & 14 & 19 & 26 & 58 & 60 & 55 \\ 14 & 13 & 16 & 24 & 40 & 57 & 69 & 56 \\ 14 & 17 & 22 & 29 & 51 & 87 & 80 & 62 \\ 18 & 22 & 37 & 56 & 68 & 109 & 103 & 77 \\ 24 & 35 & 55 & 64 & 81 & 104 & 113 & 92 \\ 49 & 64 & 78 & 87 & 103 & 121 & 120 & 101 \\ 72 & 92 & 95 & 98 & 112 & 100 & 103 & 99 \end{bmatrix}$$

- Quantized coefficients composed of mostly zeros and small values
- 0s are those not recoverable, lost by quantization

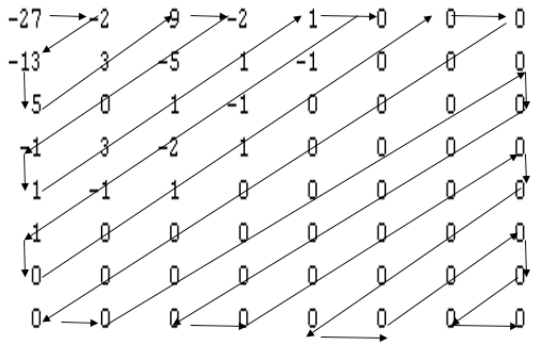
$$F = \begin{bmatrix} -438 & -24 & 90 & -39 & 22 & -11 & 25 & -12 \\ -159 & 37 & -67 & 26 & -19 & 2 & -8 & 3 \\ 73 & 3 & 18 & -15 & -1 & -20 & -31 & -3 \\ -8 & 58 & -33 & 29 & 8 & 8 & 9 & -3 \\ 24 & -22 & 26 & -18 & -19 & 13 & 4 & 9 \\ 14 & -16 & 6 & -3 & 31 & -44 & -5 & -22 \\ 16 & 15 & -10 & 2 & -17 & 26 & 4 & 6 \\ -25 & 26 & 6 & -1 & 5 & -10 & -9 & 5 \end{bmatrix} \rightarrow R = \begin{bmatrix} -27 & -2 & 9 & -2 & 1 & 0 & 0 & 0 \\ -13 & 3 & -5 & 1 & -1 & 0 & 0 & 0 \\ 5 & 0 & 1 & -1 & 0 & 0 & 0 & 0 \\ -1 & 3 & -2 & 1 & 0 & 0 & 0 & 0 \\ 1 & -1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

round(blkproc(F_trans, [8 8], 'x ./ P1', k*Z));

9. **Variable-Length Coding**

1. Arrange coefficients in zig-zag order
2. Apply a run-length sequential encoding scheme

- Perform variable-length coding (e.g., Huffman)



OTHER NOTE:

This encoding mode is called baseline *sequential* encoding.

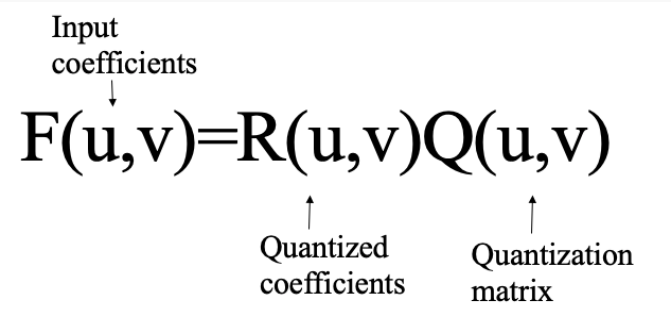
Baseline JPEG also supports *progressive* encoding. While sequential encoding encodes coefficients of a single block at a time (in a zigzag manner), progressive encoding encodes similar-positioned coefficients of all blocks in one go, followed by the next positioned coefficients of all blocks, and so on. So, if the image is divided into N 8×8 blocks {B0,B1,B2, ..., Bn-1}, then progressive encoding encodes Bi(0,0) for all blocks, i.e., for all i = 0, 1, 2, ..., N-1. This is followed by encoding Bi(0,1) coefficient of all blocks, followed by Bi(1,0)-th coefficient of all blocks, then Bi(2,0)-th coefficient of all blocks, and so on. It should be noted here that once all similar-positioned coefficients have been encoded, the next position to be encoded is the one occurring next in the zigzag traversal as indicated in the figure above. It has been found that Baseline Progressive JPEG encoding usually gives better compression as compared to Baseline Sequential JPEG due to the ability to use different Huffman tables (see below) tailored for different frequencies on each "scan" or "pass" (which includes similar-positioned coefficients), though the difference is not too large.

In order to encode the above generated coefficient pattern, JPEG uses Huffman encoding. JPEG has a special Huffman code word for ending the sequence prematurely when the remaining coefficients are zero.

[source Wikipedia]

10. **Dequantizing- Recovering the Block**

- Multiply quantized DCT coefficients by quantization matrix



-27	-2	9	-2	1	0	0	0
-13	3	-5	1	-1	0	0	0
5	0	1	-1	0	0	0	0
-1	3	-2	1	0	0	0	0
1	-1	1	0	0	0	0	0
1	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

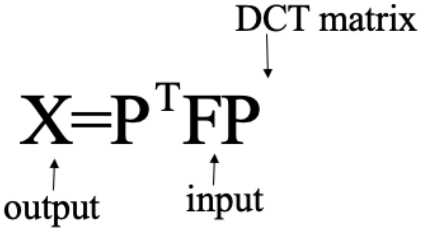
R

-432	-22	90	-32	24	0	0	0
-156	36	-70	19	-26	0	0	0
70	0	16	-24	0	0	0	0
-14	51	-44	29	0	0	0	0
18	-22	37	0	0	0	0	0
24	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

F

11. **Apply Inverse of DCT**

- Compute Inverse DCT



-432	-22	90	-32	24	0	0	0
-156	36	-70	19	-26	0	0	0
70	0	16	-24	0	0	0	0
-14	51	-44	29	0	0	0	0
18	-22	37	0	0	0	0	0
24	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

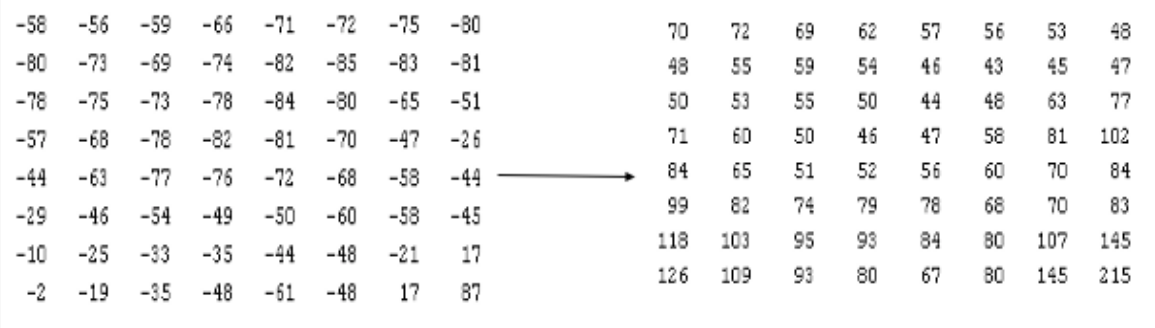
F

-58	-56	-59	-66	-71	-72	-75	-80
-80	-73	-69	-74	-82	-85	-83	-81
-78	-75	-73	-78	-84	-80	-65	-51
-57	-68	-78	-82	-81	-70	-47	-26
-44	-63	-77	-76	-72	-68	-58	-44
-29	-46	-54	-49	-50	-60	-58	-45
-10	-25	-33	-35	-44	-48	-21	17
-2	-19	-35	-48	-61	-48	17	87

X

12. **Decoding**

- For each block, shift values by 128 to remove the effects of zero-shift during encoding
- How do you calculate the MSE between the original block and the decoded block?



Extra Note on: **KLT Block Transform Compression**

- Recalculate every time => so its not efficient comparing to DCT
- For each **8x8 block**, create a **64-d vector**
- Using all 64-d vectors from the image, calculate a 64x64 covariance matrix
- From the covariance matrix, determine the K largest **eigenvalues** that retain a certain % of the energy
- The transformation matrix is then the first K eigenvectors associated with the largest K eigenvalues

8.5 Advanced Concepts

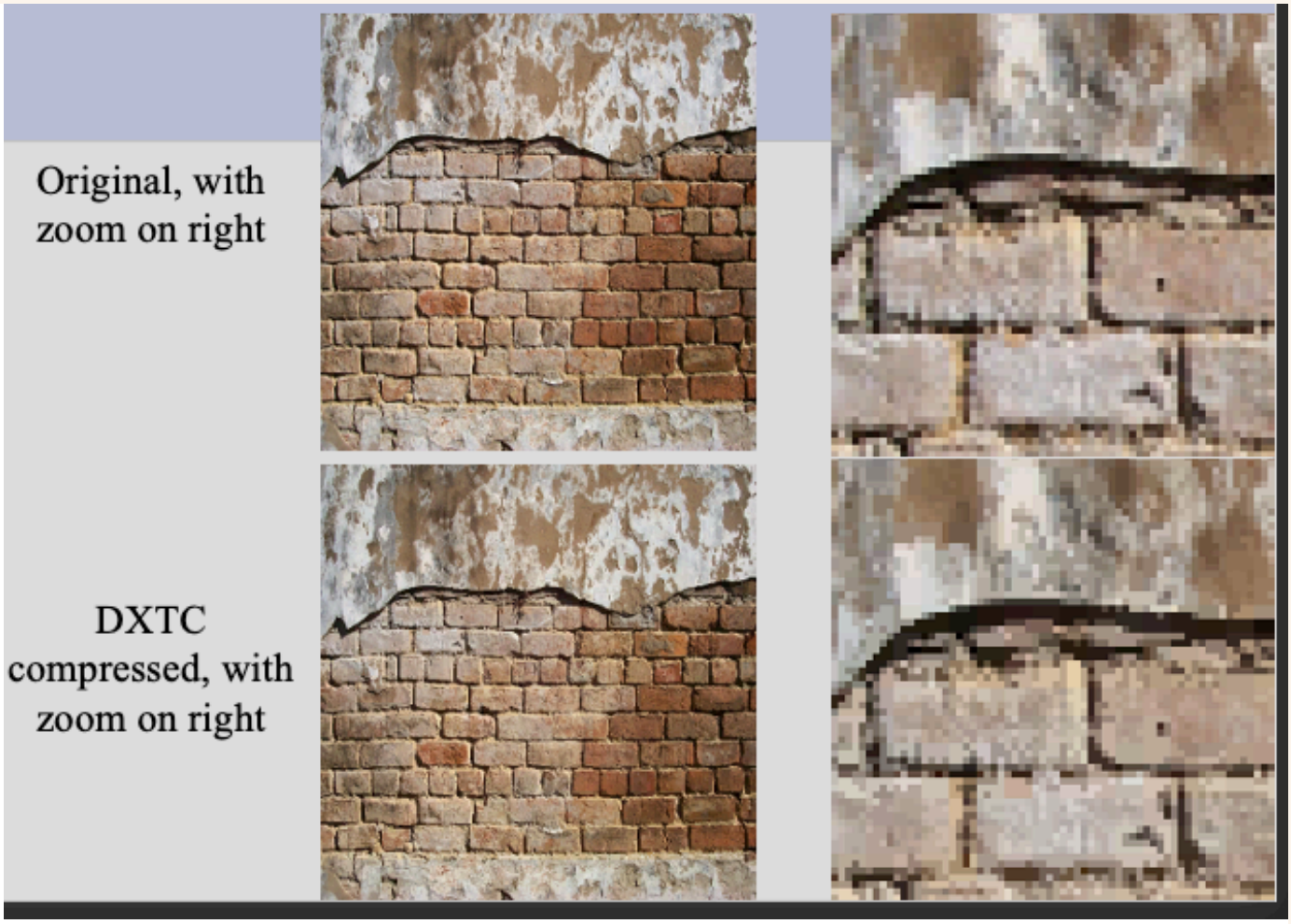
DXTC: Fixed Rate: Extends BTC

- Used for texture compression in the Direct3D standard
- Well suited for 3D real-time applications as it allows for random texel access
- Very fast due to hardware acceleration on all current video cards
- **Extends BTC (block transform coding) for color images**
- In addition to **spatial redundancy**, also takes advantage of **psycho-visual redundancy** (through **quantization**)
- Also known as S3 Texture Compression (S3TC)
- **STEPS:**
  1. Divide image into **4x4 blocks**
  2. For each block, store two **16-bit representative color** values **C0 (high) and C1 (low),** where
     - 5 bits allocated for red
     - 6 bits allocated for green

        - 5 bits allocated for blue
    3. Compute two additional color values
        - $c_2 = \frac{2}{3}c_0 + \frac{1}{3}c_1, c_3 = \frac{1}{3}c_0 + \frac{2}{3}c_1$
    4. Assign a value from **0 to 3 (2 bits)** to each pixel based on which of the **four color values** they are **closest**
        - Requires storage for 4x4 two-bit lookup table for storage
    5. To decode, replace values from lookup table with one of the four color values
- NOTE:
    - **Texture compression** is a specialized form of image compression designed for storing texture maps in 3D computer graphics rendering systems. Unlike conventional image compression algorithms, texture compression algorithms are optimized for random access. Four characteristics that differentiate texture compression from image compression: decoding speed (must be fast!), random access (ordering unknown), compression rate & visual quality (lossy compression tolerable), encoding speed (not critical)

DXTC Compression Rate

- Suppose we are given a **4x4 color image**, with each pixel represented by R, G, and B values ranging from **0 to 255** each
- Number of bits required to store this image in an uncompressed format is:
    - **4x4x(3x8bits)=384 bits**
- Bit rate of image in **uncompressed** format is **384 bits/16 pixels = 24 bpp**
- Supposed we compress the color image using DXTC
- The **high and low representative color** values **C0 and C1** each require **16 bits**
- Each value in the **4x4 lookup table** represents **4 possible values**, thus requiring 4x4x**2bit**=**32 bits**
- Number of bits required to store in DXTC compressed format is **2x16bits + 32bits = 64 bits**
- Bit rate of color image in a DXTC format is 64bits/16pixels=4 bpp
- The compression rate of DXTC for the color image can then be computed as
    - $BPP_{uncompressed} : BPP_{DXTC} = 24 : 4 = 6 : 1$
- Image Example of DXTC

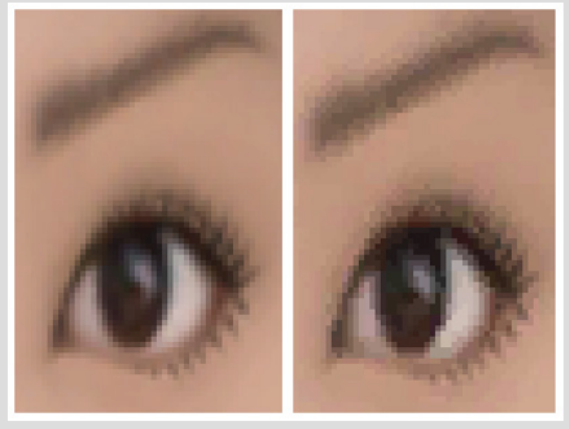
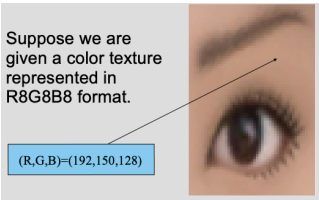
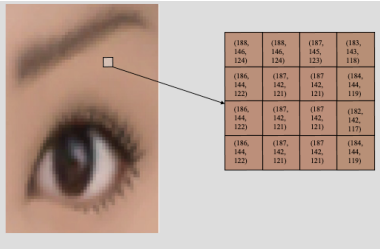
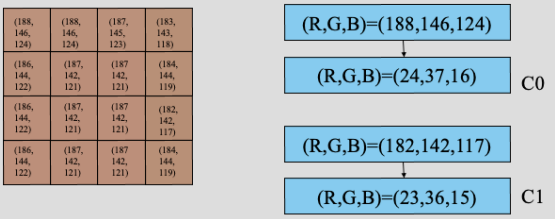
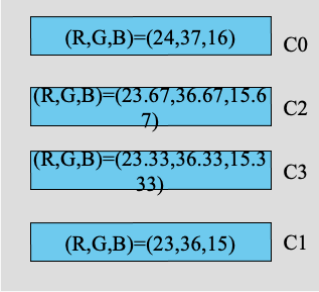
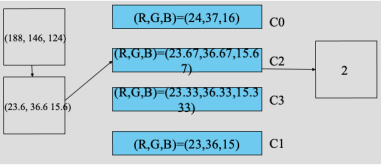
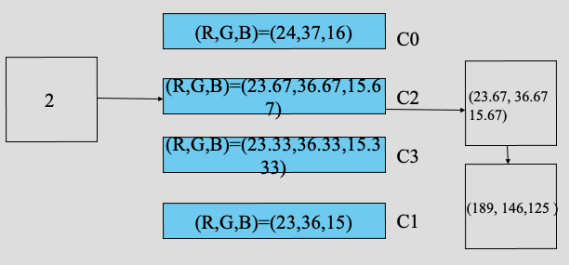


- Artifact at gradual transition

Observation

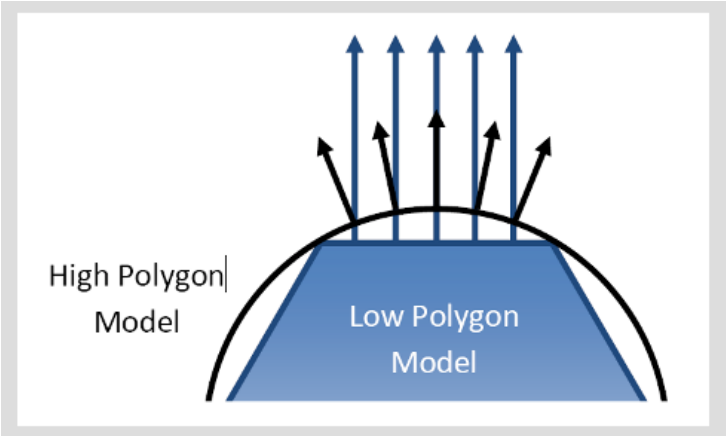
- Image remains **very sharp and clear**
- Solid, uniform regions are **well represented**
  - **Quantization does not perceptually affect image quality in this case**
- Blocking artifacts can be seen at smooth transitions
- Reason: using a total of 4 colors does not sufficiently represent such regions, which require more color values to represent the smooth transition

Example Tbale

->	->	->
6:1 compression using DXTC  - Blocking artifacts at smooth transitions		Divide image into 4x4 blocks 
Store two 16-bit representative color values C0 (high) and C1 (low) in R5G6B5 format - Largest value in Euclidean distance 	Compute two additional color values (e.g., using simple interpolation) 	Assign a value from 0 to 3 to each pixel based on closest color value 
To decode, replace values from lookup table with one of the four color values 		

Normal Mapping

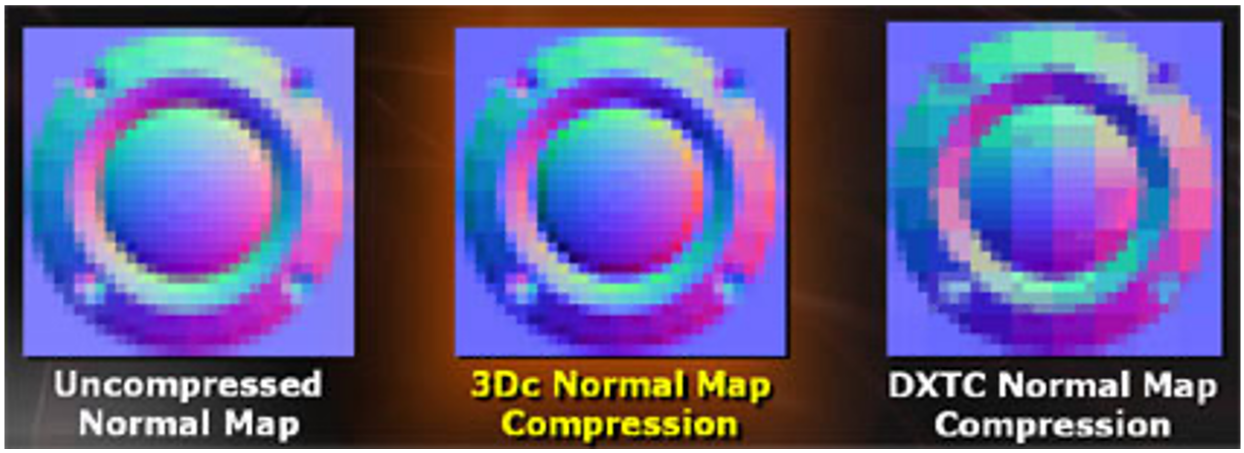
- Complex 3D models in a scene provide a greater sense of realism within a 3D environment
- However, its **expensive** from both a **computational and memory** perspective to process such c**omplex 3D models with high geometric detail**
- Solution: use **Normal Mapping** to give the **sense** that there is more geometric detail by changing **lighting** based on supposed **geometry**



- Create high resolution model and a corresponding low resolution model you want to use
  - Cast ray from each **texel** on **low-res model**
  - Find **intersection** of ray with **high-res model**
- Save the normal from high-res model where the ray intersects
- NOTE:
  - - Faking the lighting of details in image e.g., “bump mapping”; add details without adding polygons

3Dc

- Each pixel in a normal map has three values (x,y,z), which represent a normal vector
- The x, y, and z coordinates of a normal vector are independent from each other
- This makes **DXTC poorly** suited for compressing **normal** maps since it **relies on inter-channel correlations**
- Solution: **3Dc**, an extension of BTC for normal maps
  - 3Dc is an open standard
- 3Dc vs. DXTC Normal Map Compression:



- NOTE:
  - Open standard
  - First developed by ATI (Markham) and now used by Nvidia

How does 3Dc work?

- Instead of operating on all channels together, treat x, y, and z coordinate channels separate from each other
- In most systems, all **normal vectors** are unit vectors with a length of 1
- Also, z component assumed to be positive since it should point out of the surface
- lIdea: Instead of storing z, **compute z based on x and y**
  - $z = \sqrt{1 - (x^2 + y^2)}$
- Since z is not stored, storage requirements have effectively been **reduced by 1/3**

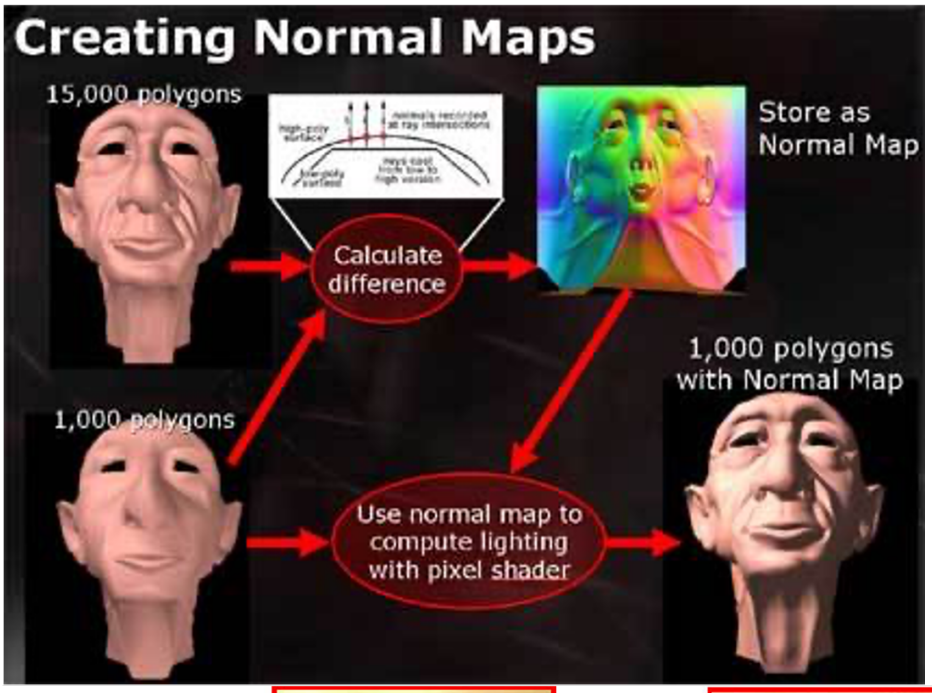
How does 3Dc encoding work?

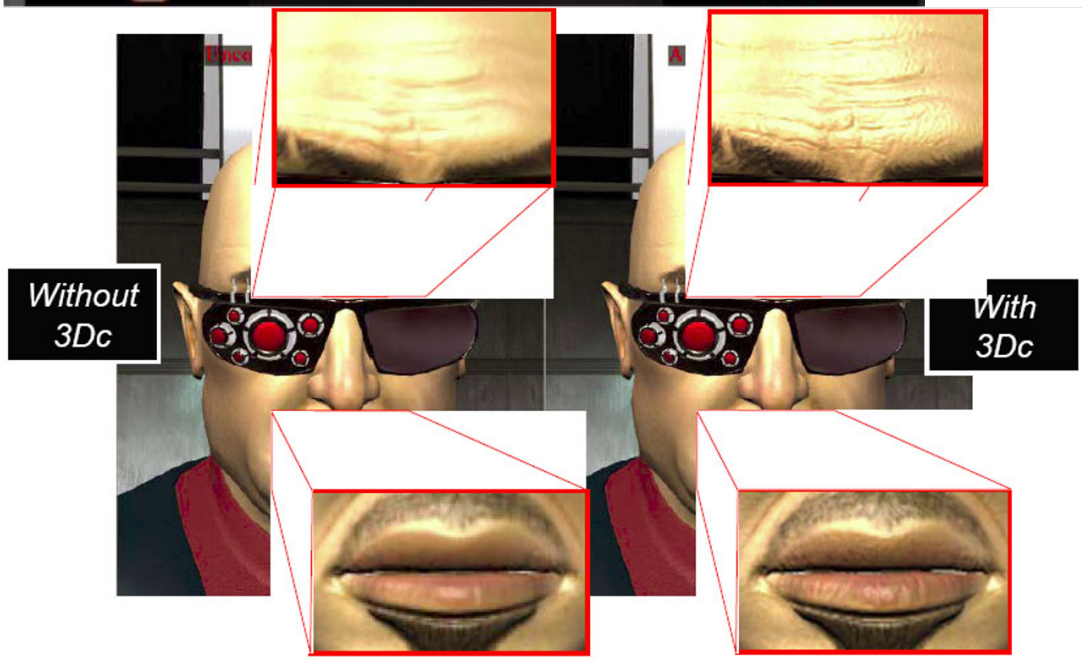
- Steps:
  1. Discard z channel
  2. For the x and y channels, divide **normal map into 4x4 blocks**
  3. For each block, store **two 8-bit** representative coordinate values (V0 and V1)
  4. Compute **6 intermediate coordinate** (vs 2 in DXTC) values by using simple **linear interpolation** between V0 and V1
  5. Assign a value **from 0 to 7 to each pixel** based on the closest of the **8 coordinate** (vs 4 in DXTC) values V0,V1,...,V7
     - Creates a **4x4 3-bit lookup** table for storage

3Dc Compression Rate:

- Suppose
  - we are given a **4x4 normal map**, with each pixel represented by **x, y, and z** values ranging from **0 to 2^(16-1)** each.
  - Number of bits required to store this image in an uncompressed format is **4x4x(3x16bits)=768 bits**
  - The bit rate of the **normal map** in an uncompressed format is **48 bpp (bits per pixel)**
- Suppose
  - we compress the normal map using 3Dc
  - The high and low representative coordinate values **V0 and V1 each require 8 bits**
  - Each value in the **4x4 lookup table** represents **8 possible values**, thus requiring **4x4x3bit=48 bits**
- Soln:
  - $\frac{2}{3}$ channels must be stored (i.e., **2 lookup tables, 2 sets of V0 and V1**, etc.)
  - Number of bits required to store this color image in 3Dc compressed format is **(2x8bits+48bits)x2=128 bits**
  - The bit rate of the normal map in a 3Dc compressed format is **128 bits/16 pixels = 8bpp**
  - Effective compression rate for 3Dc in this case is:
    - **48/8=6:1 compression**

How does 3Dc decoding work?

- Steps:
  - For each block in the x and y channels, replace values from lookup table with one of the 8 coordinate values (2 stored values and 6 interpolated values)
  - Compute z based on x and y to get all three coordinates for each normal vector
- Ex:
  - 

  - 

Predictive Coding

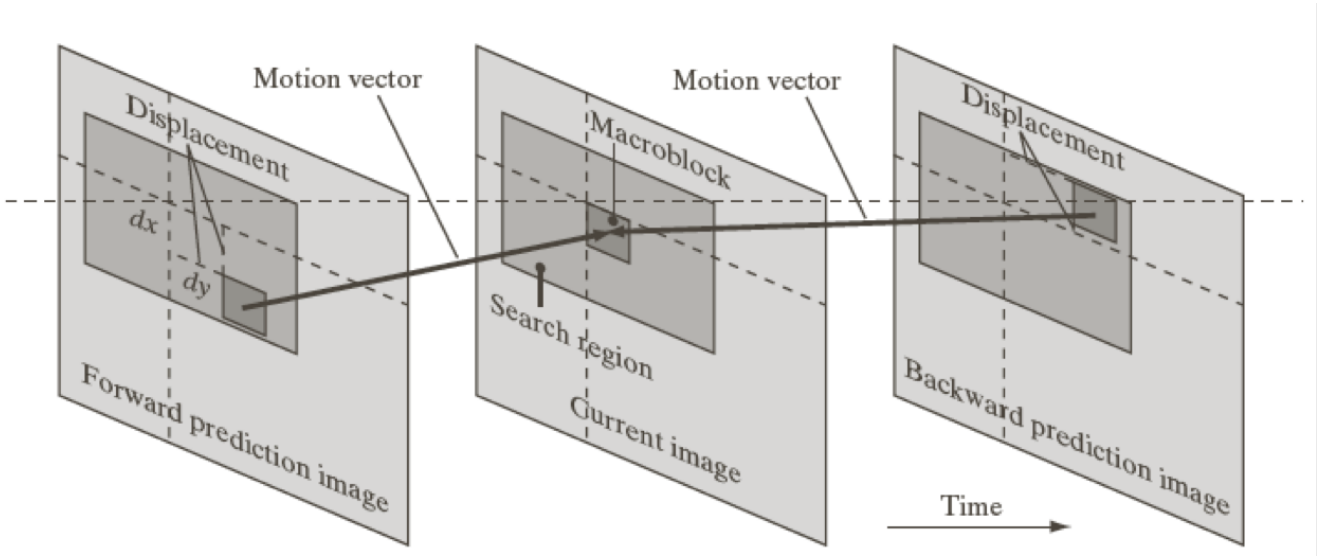
- Images and videos contain a large amount of spatial and temporal redundancy
- Pixels in an image or video frame should be reasonably predicted by other pixels in
  - The same image (**intra-frame prediction**)
  - Adjacent frames (**inter-frame prediction**)

Intra-frame Predictive Coding

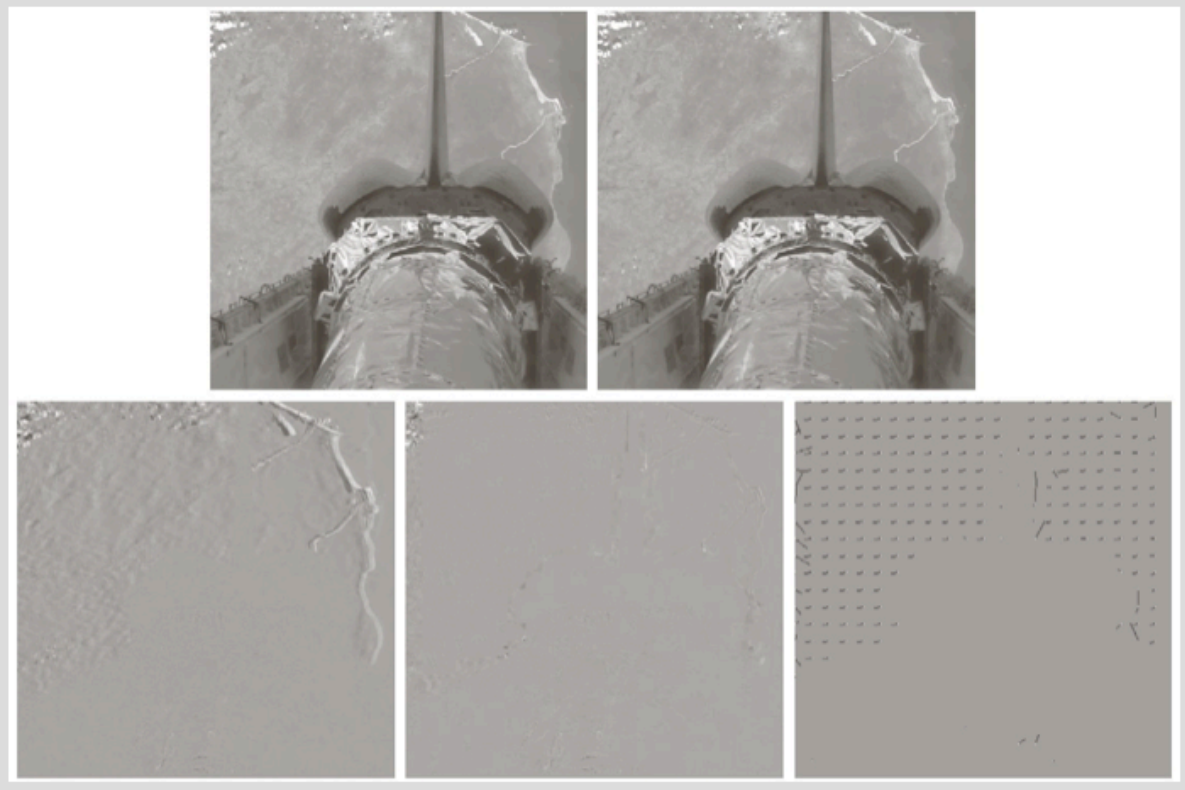
- For a sub-image *f*, find the sub-image *p* that is most similar to *f* (**block matching**)
- One approach is to find the sub-image that minimizes the mean absolute distortion (MAD)
  - $MAD(x,y) = \frac{1}{mn}\sum_{i=1}^{m}\sum_{j=1}^{n}|f(x+i,y+j) - f(x+i+dx,x+j+dy)|$
- Usually performed on the luminance channel
- Encode and store vector (dx,dy)
- Calculate the error residual between the two sub-images
  - $e(x,y) = f(x+i,y+j) - f(x+i+dx,x+j+dy)$
    - where i,j spans the dimension of the sub-image
- Transform prediction error residual with image transform and quantization

Inter-frame Prediction Coding

- Similar to intra-frame coding, but instead of within the same image, the prediction coding is performed between frames



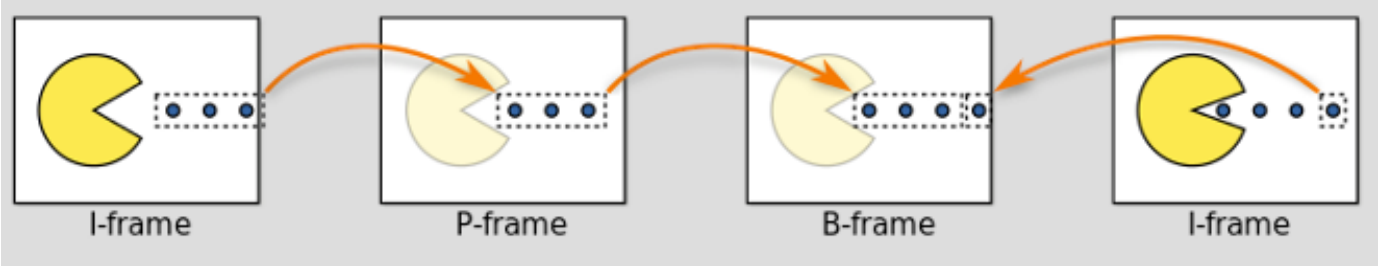
- Result:



- (a) And (b) Two views of Earth that are thirteen frames apart in an orbiting space shuttle video. (c) A prediction error image without motion compensation. (d) the prediction residual with motion compensation. (e) The motion vectors associated with (d).

Types of "Frames"

- I-frame “Intra-coded”: a compressed image
  - => Poorest compression
- P-frame “Predicted”: compresses using previous frame
  - => Good compression
- B-frame “Bi-predictive”: compresses using previous frame and next frame
  - => Best compression



  - Note: “picture” is a general term which can be either a frame (complete image) or field (odd or even scanned).
  - Interlaced video has each from sent as odd-field followed by even-field.

9. Deep Learning and Image Processing

9.1 Supervised ML

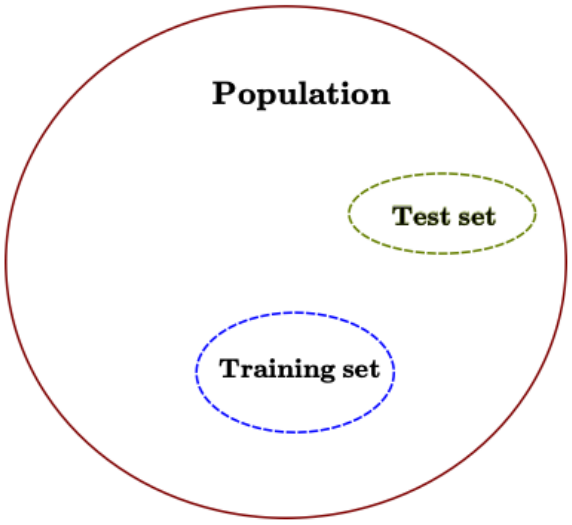
- Motivation:
  - Images are projections of the real world
  - We want to extract **semantic** information from these images that tell use something about its setting
  - Analytically coming up with mathematical model that describe real world elements is difficult.
  - **Machine learning** presents flexible models with parameters that can be shaped to recognize different tasks.

Supervised machine learning

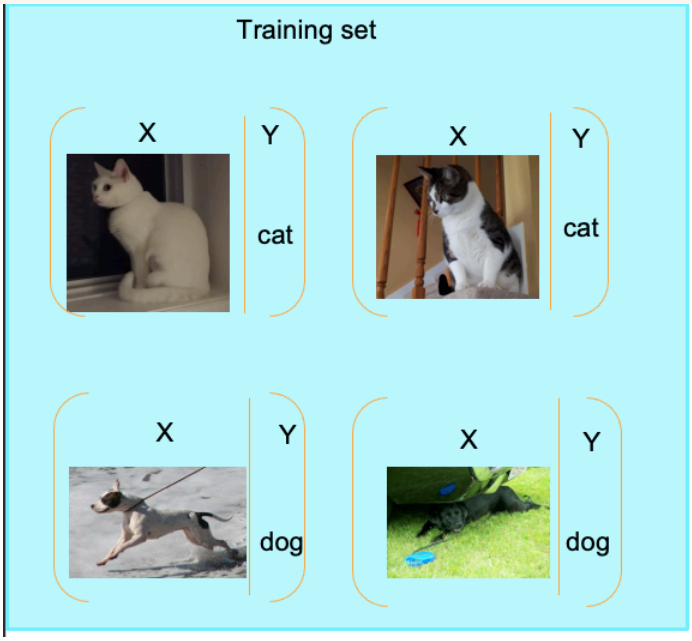
- We want to solve some problem that involves transforming
  - (**X** [input data] → **Y** [labels])
  - Ex X: Images, Videos, any real data
  - Ex Y: Discrete classes - {'cat', 'dog'}, Regression - $\mathbb{R}$
- **Idea:** Implement an algorithm called a **Model.**
  - **Model** transforms **X** → **P** [prediction] ≈ **Y** based on a sample of data.
-

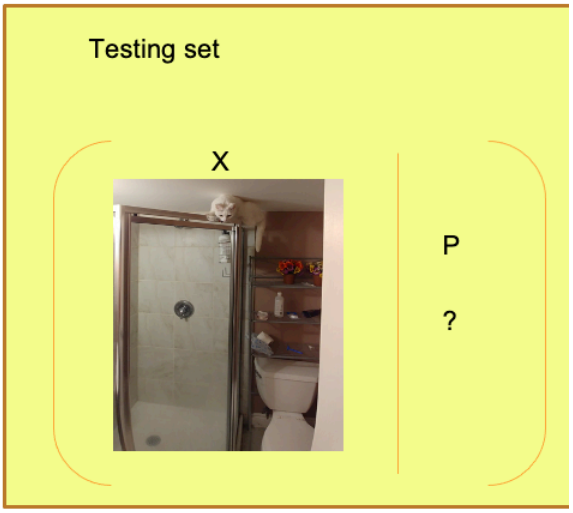
Statistics of Supervised learning

- For any problem, we can consider all possible pairs of **X** and **Y** as instances of a **population.**
- In supervised learning, we are given a **sample** of **(X,Y)** called a **training set**.
- Goal: Tune a **model** based on the **training set** to make predictions on unseen examples from the **population (**called the **testing set).**

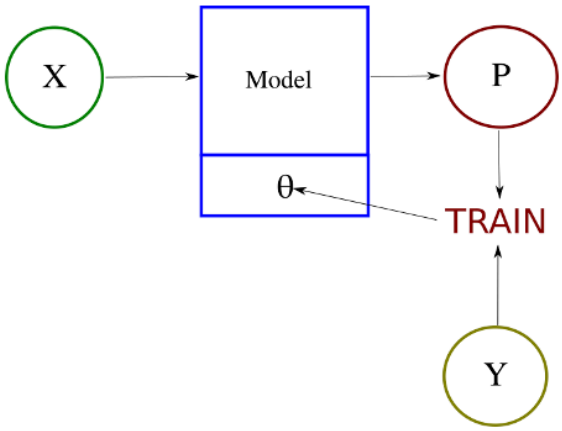


General Process



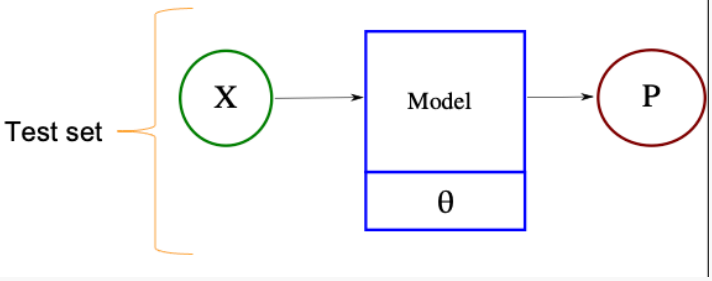


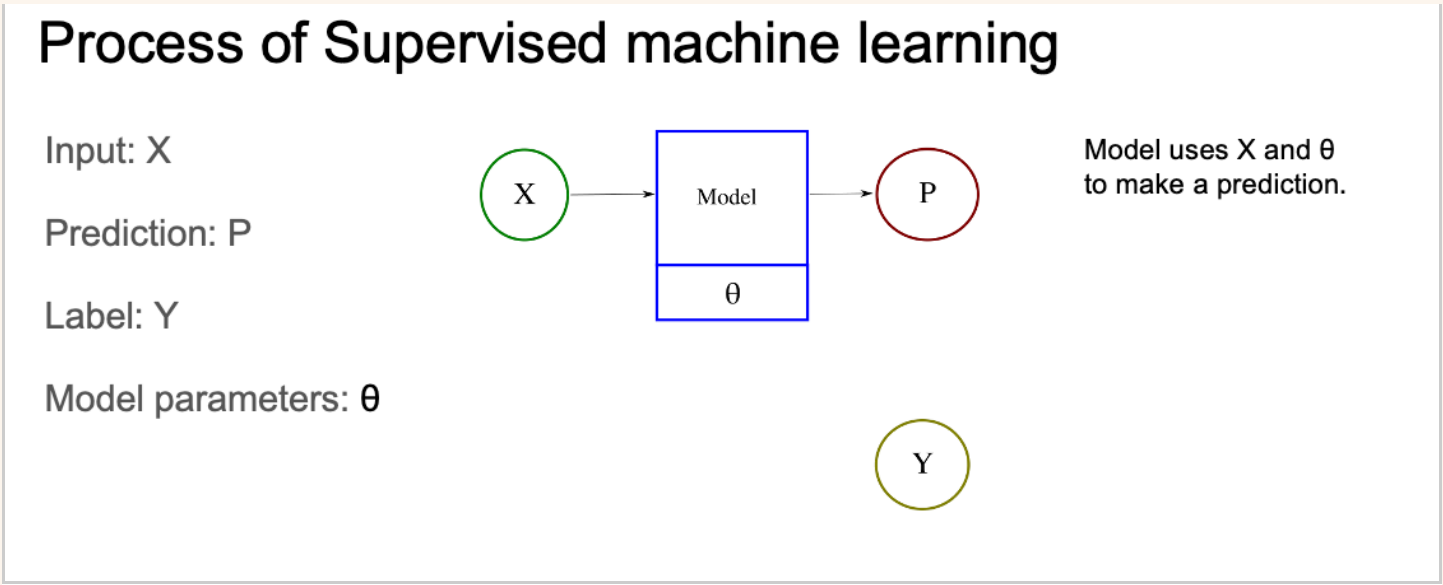
Process of Supervised machine learning



Training algorithm adjusts θ so that P is closer to Y.

Goal: Get good values of θ so that model will make accurate predictions on population.

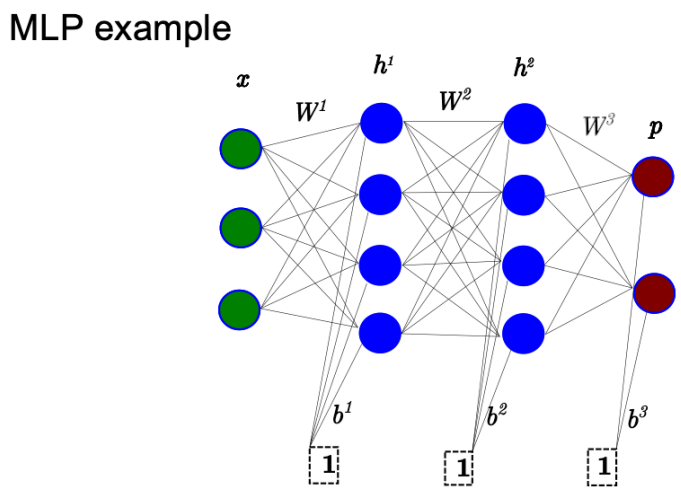
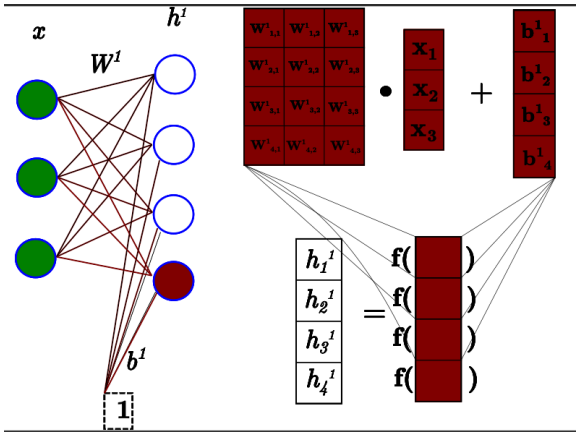
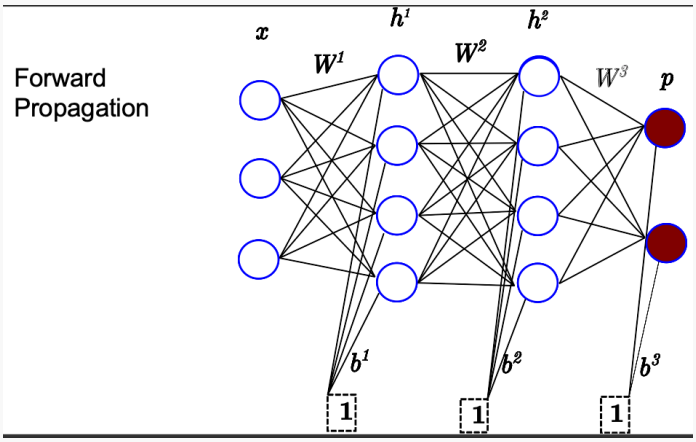




9.2 Neural Networks - Feedforward

- Idea:
  - What we want from a supervised learning model:
    - Can fit to any unknown relationships
    - Behaviour controlled by parameters
    - Parameters that can be tuned from the Training Set.
  - Neural networks fit this criteria
- **Neural Networks:**
  - Composed of **hidden layers**
  - Non-linear functions applied to linear combinations of previous layer(s)
  - The **network** connects several hidden layers together to produce a prediction.
- **Multilayer Perceptron** Formulation:
  - For **Layer** t:
    - **Parameters:**
      - $W^t \in \mathbb{R}^{m \times n}$, $b^t \in \mathbb{R}^m$
    - **Hidden Layers:**
      - $h^t \in \mathbb{R}^m$, $h^{t-1} \in \mathbb{R}^n$
    - **Hidden Layer Formula:**
      - $h_i^t = f(\sum_j W_{i,j}^t h_j^{t-1} + b_i^t)$
      - $h^t = f(W^t h^{t-1} + b^t)$

- **MLP Example:**

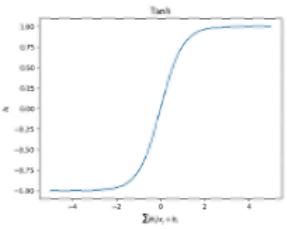
MLP Example =>	
 - 2 hidden layer - W: weight matrix, b: bias vector	
	

- Non-linear Function
  - classical hidden layer function: **tanh** function [-1,1]
  - binary classification: **sigmoid** function [0,1]
  - popular in CV: **relu** function [0, ∞]
  - **Softmax function**: normalize to 1 => good for probability
  - IMPOTRANT: they must be **differentiable**

Non-linear function

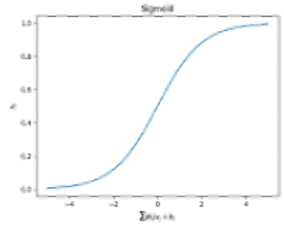
- Many different possibilities.

$$\tanh(z) = \frac{\exp(2z) - 1}{\exp(2z) + 1}$$



$$softmax(\mathbf{z}) = \frac{\exp(z_i)}{\sum_j \exp(z_j)}$$

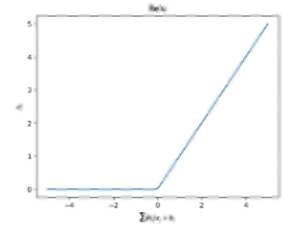
$$sigmoid(z) = \frac{1}{1 + \exp(-z)}$$



- Normalizes vector to sum to 1

- Must be differentiable

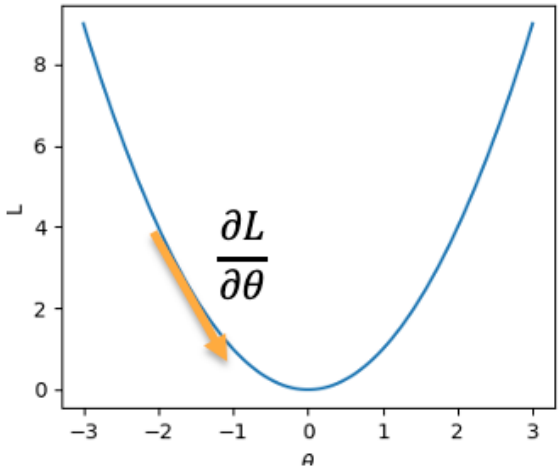
$$relu(z) = \begin{cases} z \; if \; z > 0 \\ 0 \; else \end{cases}$$



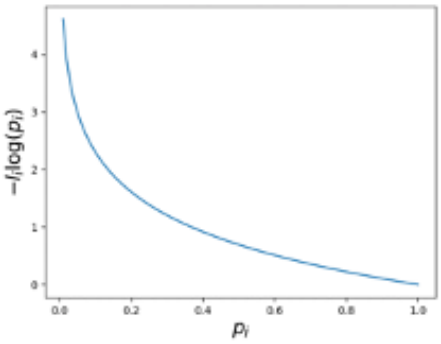
9.3 Neural Networks - Training

Training

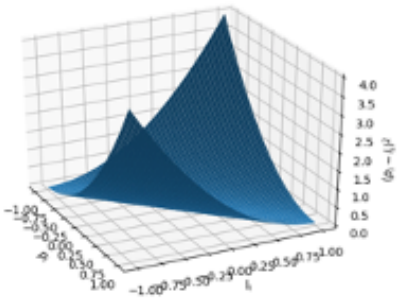
- Training typically done with **stochastic gradient descent**.
  - Loss function: L(p, y) →discrepancy between **p** and **y**.
  - Adjust parameters using the examples in training set
    - $\theta \leftarrow \theta - \alpha \frac{\partial L}{\partial \theta}$



- **Discrete Classification:**
  - $L = \sum -l_i \ \log p_i$
  - **Cross Entropy**



- **Regression Tasks:**
  - $L = \frac{1}{N}(l_i - p_i)^2$
  - **Mean Squared Error:**

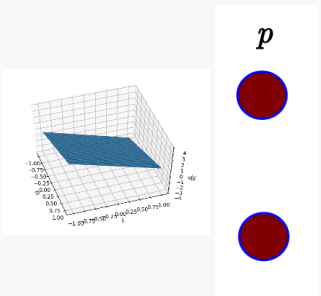
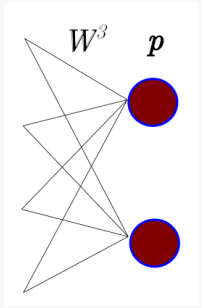
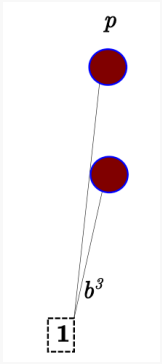
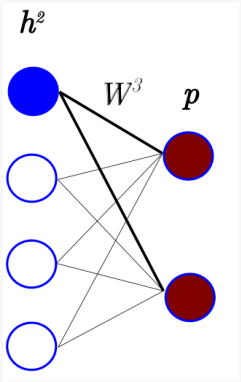


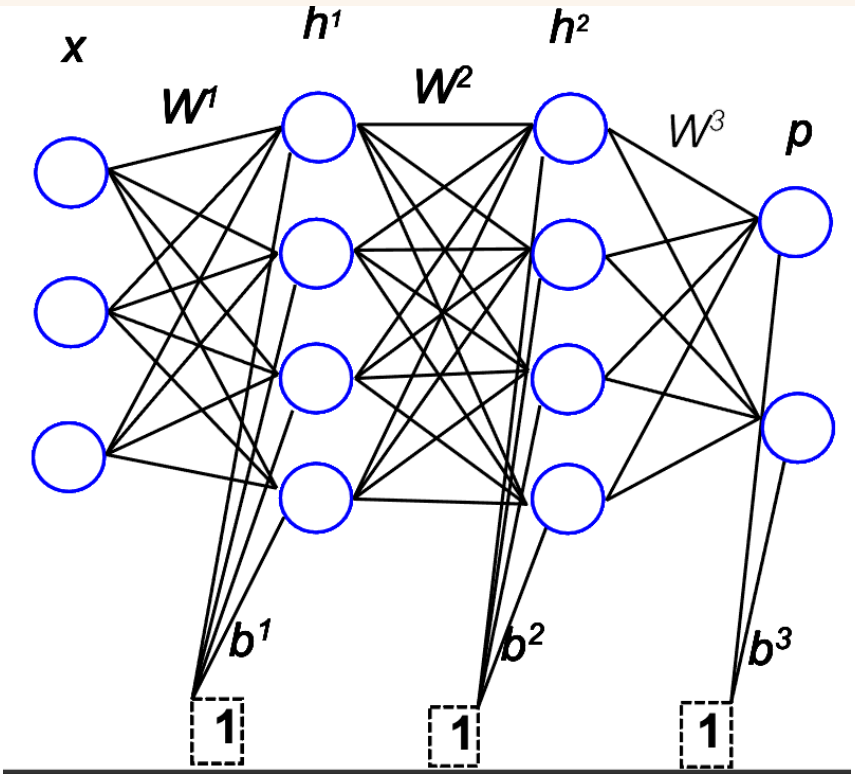
Backpropagation

- Evaluation – forward propagation.
- Gradient descent: update parameters by steps to reduce loss. $\theta \leftarrow \theta - \alpha \frac{\partial L}{\partial \theta}$
- How do we compute derivatives for each parameter?
  - Applying **chain rule** in a smart manner.

    $$\frac{\partial L(f(t))}{\partial t} = \frac{\partial L}{\partial f}\frac{\partial f}{\partial t}$$
  - Start at end, move to start.
- Training – **back propagation**:

Layers to x Properties Table

Loss to prediction	**Layer to weight**	**Layer to Bias**	**Layer to Layer**
Compute $\frac{\partial L}{\partial p_i}$ Ex: $L = \sum -l_j \log p_j$ $\frac{\partial L}{\partial p_i} = \frac{-l_i}{p_i}$ $L = \frac{1}{N}\sum (l_i - p_i)^2$ $\frac{\partial L}{\partial p_i} = \frac{2}{n}(p_i - l_i)$	$p_i = f(\sum_j W^3_{i,j} h^2_j + b^3_i)$ $\frac{\partial L}{\partial W^3_{i,j}} = \frac{\partial L}{\partial p_i}\frac{\partial p_i}{\partial W^3_{i,j}}$ Note: assumes that p_i, p_j are independent. Slightly different for softmax: $\frac{\partial L}{\partial W^3_{i,j}} = \sum_k \frac{\partial L}{\partial p_k}\frac{\partial p_k}{\partial W^3_{i,j}}$	$p_i = f(\sum_j W^3_{i,j} h^2_j + b^3_i)$ $\frac{\partial L}{\partial b^3_i} = \frac{\partial L}{\partial p_i}\frac{\partial p_i}{\partial b^3_i}$ If not dependent: $\frac{\partial L}{\partial b^3_i} = \sum_k \frac{\partial L}{\partial p_k}\frac{\partial p_k}{\partial b^3_i}$	$p_i = f(\sum_j W^3_{i,j} h^2_j + b^3_i)$ $\frac{\partial L}{\partial h^2_i} = \sum_i \frac{\partial L}{\partial p_i}\frac{\partial p_i}{\partial h^2_i}$
			

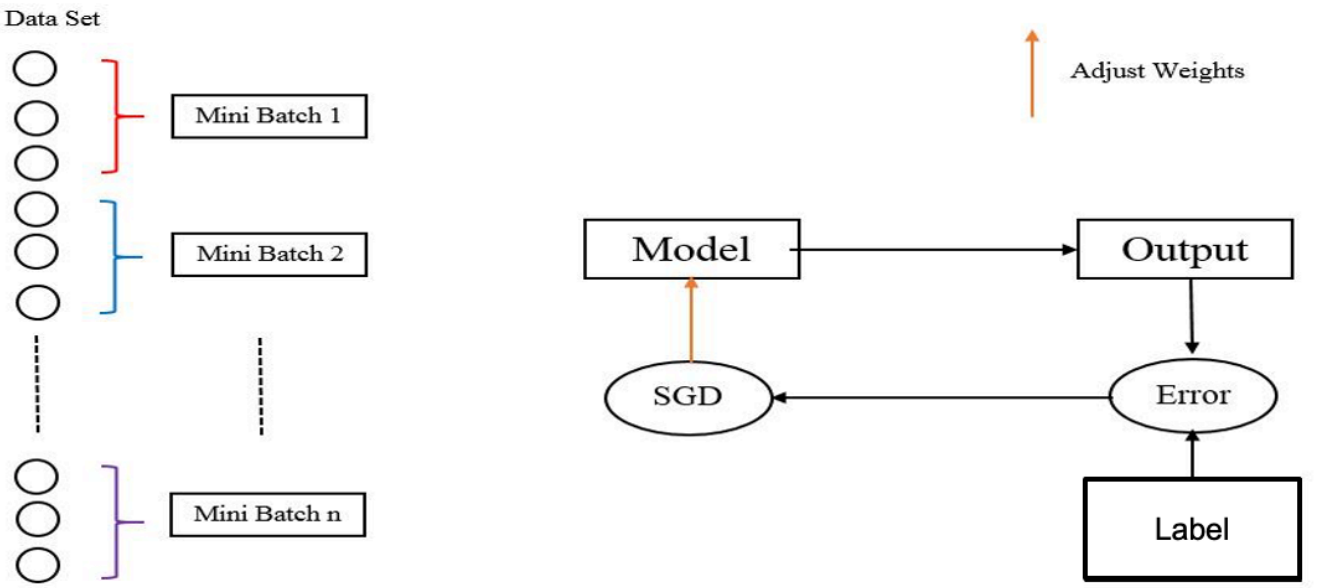


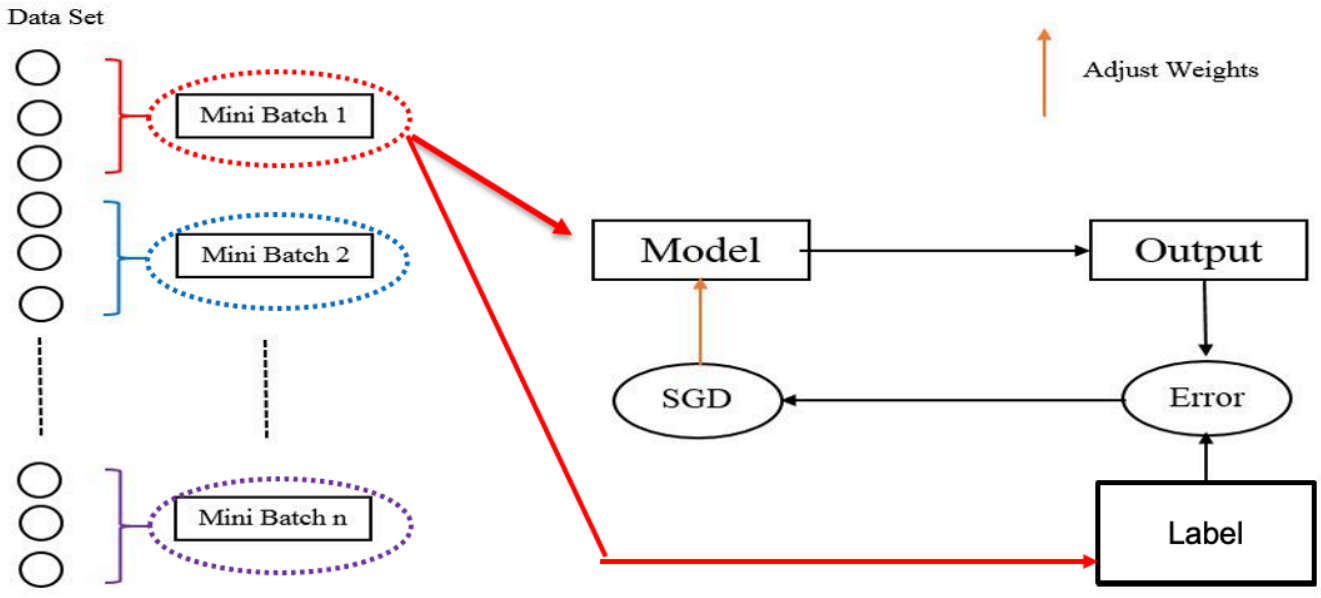
Stochastic Gradient Descent

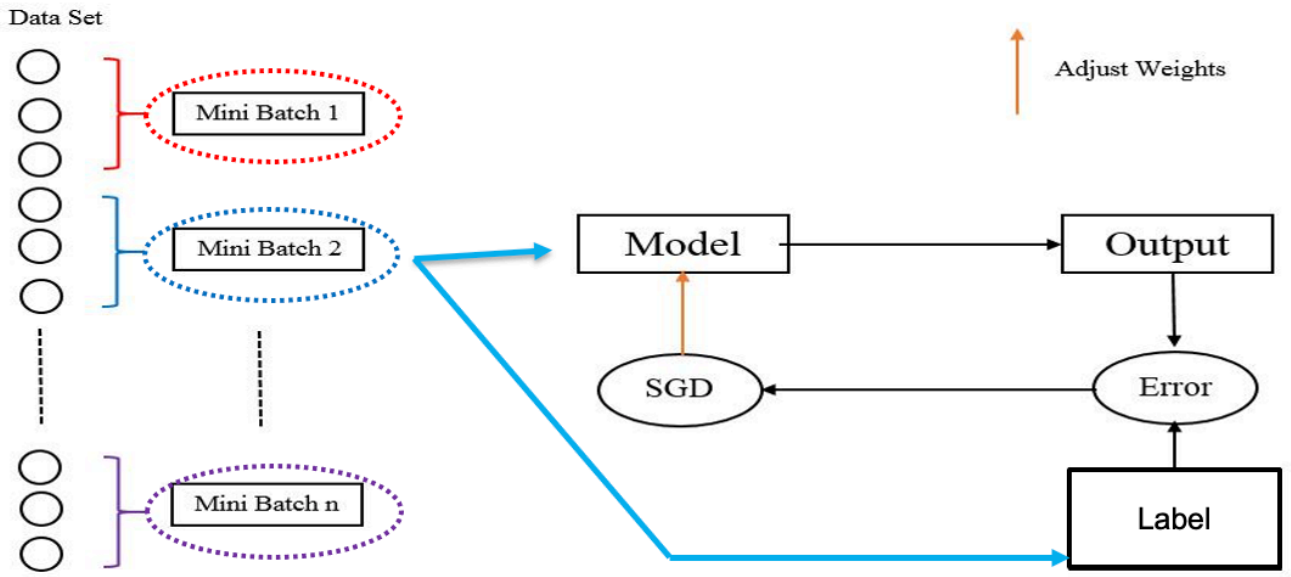
- Divides training set into “mini-batches”
  - Computes average update $\theta \leftarrow \theta - \alpha \frac{\partial L}{\partial \theta}$ for each mini-batch
  - More robust than updating for each sample, or over all samples.
- An “epoch” is when all mini-batches are processed in the training set.
- Networks will train over many epochs until error reaches an appropriate level.

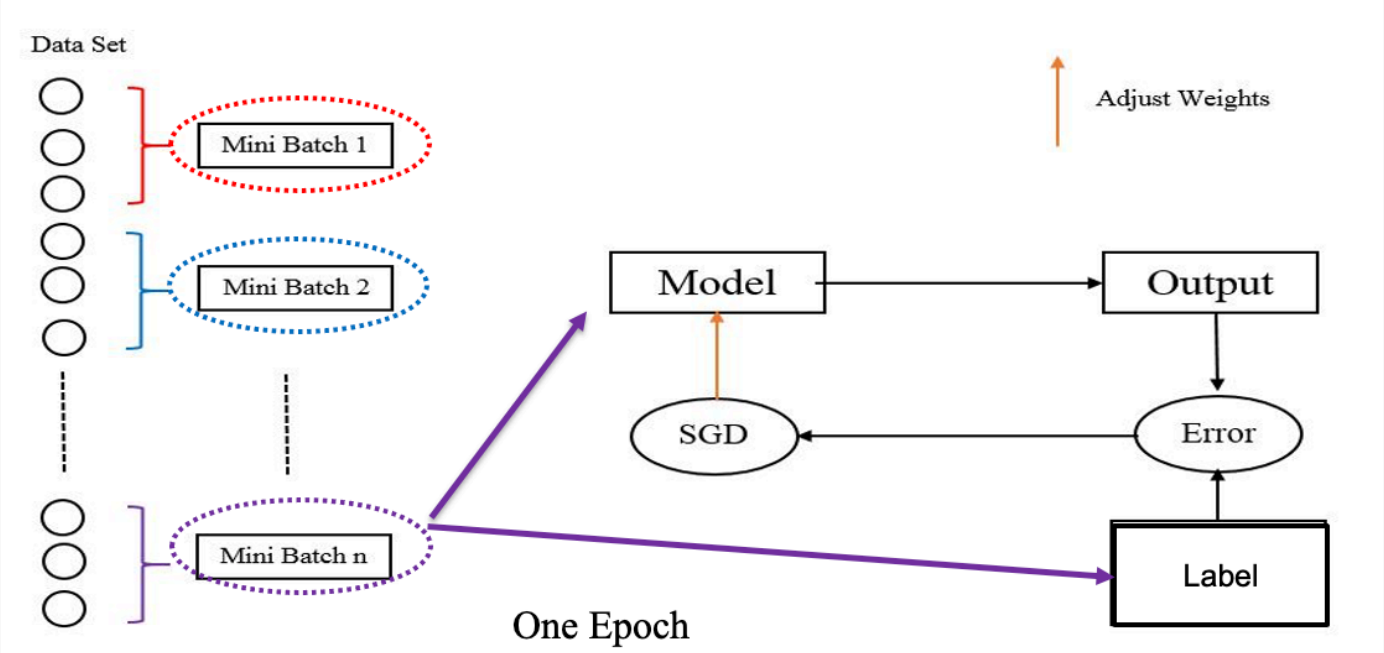
Training Cycle Table

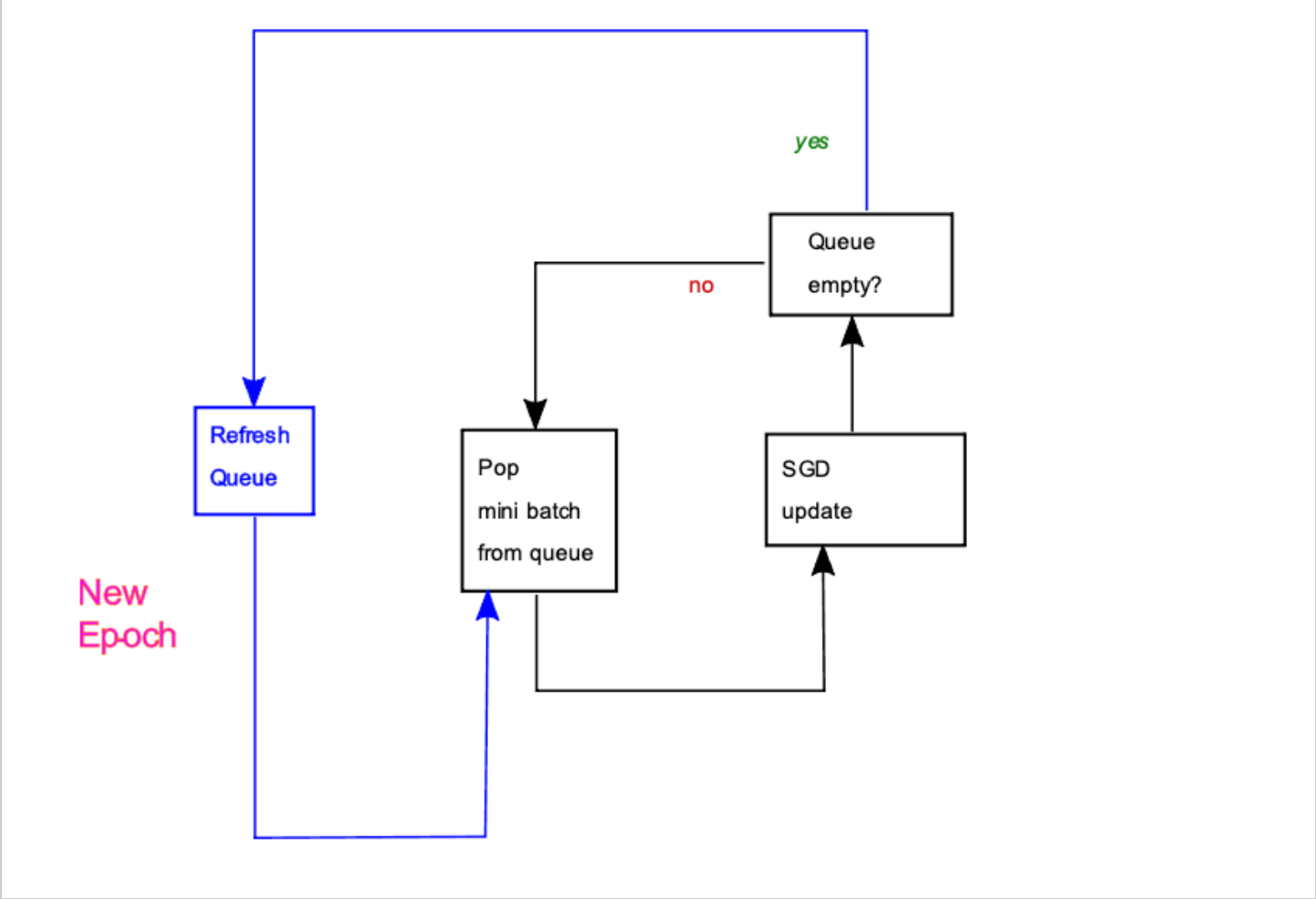
Training Cycle


Data Set
Mini Batch 1
Mini Batch 2
Mini Batch n
Adjust Weights
Model
Output
SGD
Error
Label


Data Set
Mini Batch 1
Mini Batch 2
Mini Batch n
Adjust Weights
Model
Output
SGD
Error
Label


Data Set
Mini Batch 1
Mini Batch 2
Mini Batch n
Adjust Weights
Model
Output
SGD
Error
Label


Data Set
Mini Batch 1
Mini Batch 2
Mini Batch n
Adjust Weights
Model
Output
SGD
Error
Label
One Epoch

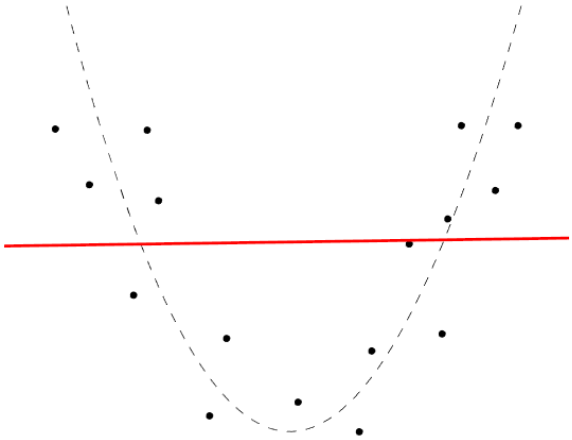


Universal Approximation Theorem

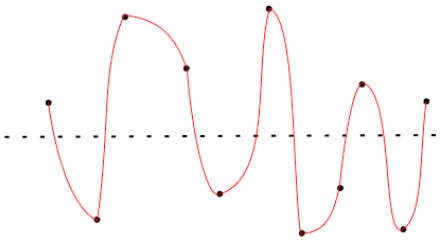
- A neural network with the right number of nodes and proper parameters can approximate (almost) any real function.
- Other types of models rely on feature engineering to perform complex tasks
  - Support vector machines, Random forest, and Logistic regression.
  - Neural networks relax this requirement

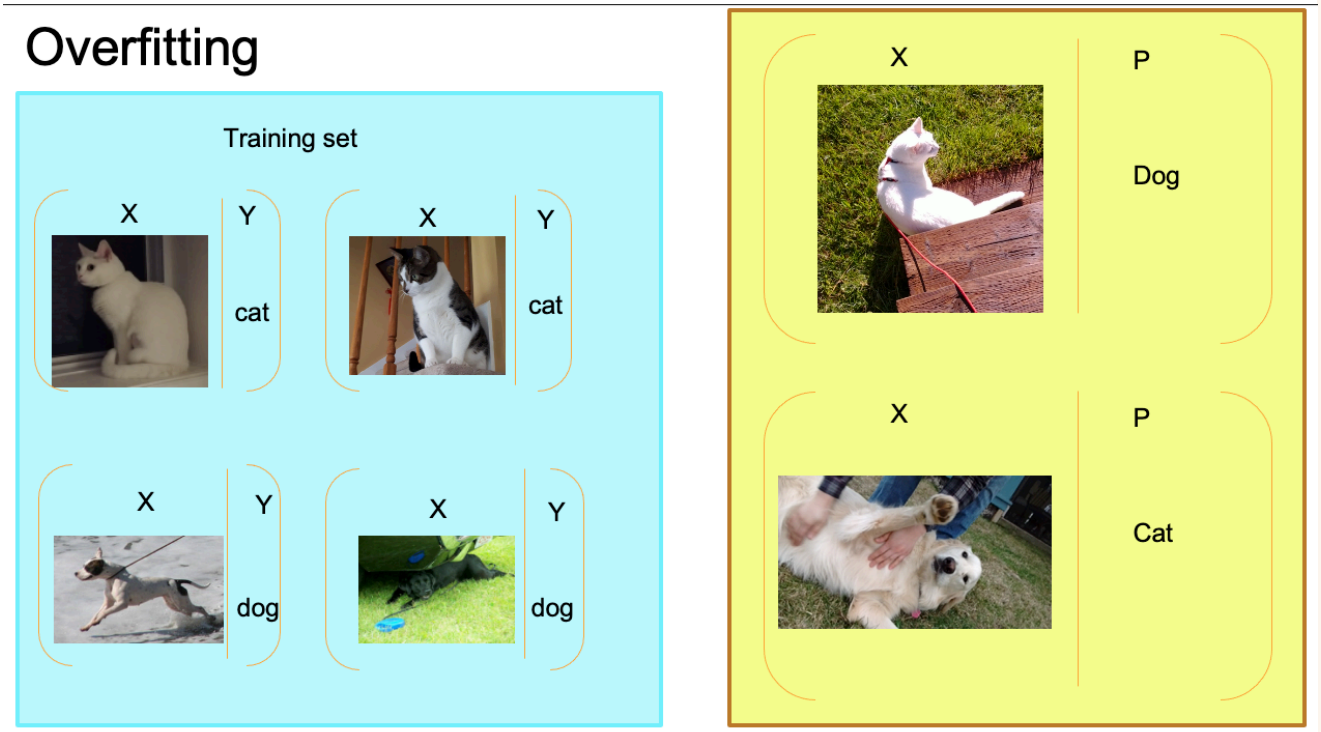
Challenges

1. Underfitting
   - Model doesn't have enough capacity to learn X → Y relationship.



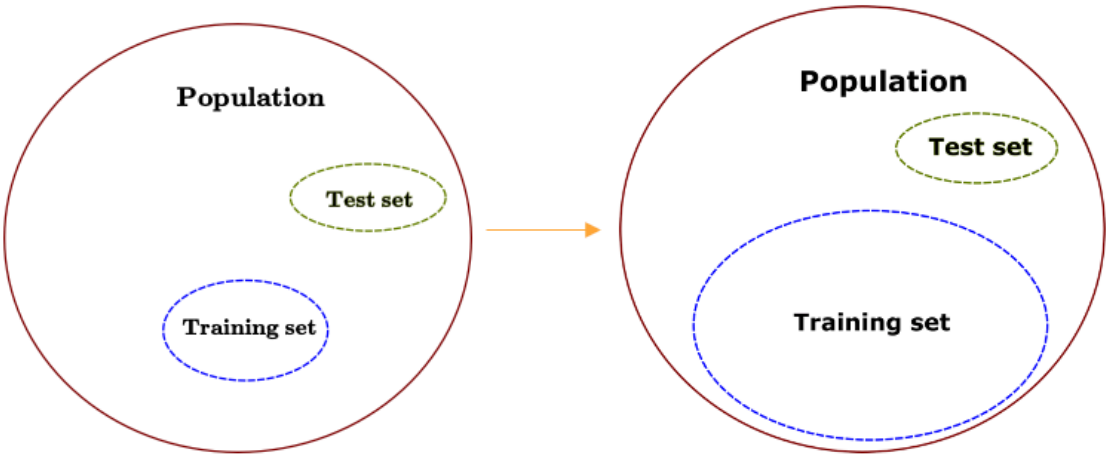
2. Overfitting
    - Model has fit too rigidly to the biases in the training set.
    - Indicator: Prediction accuracy on training set is **very** different than accuracy on testing set.





- **Strategies to combat Overfitting**:

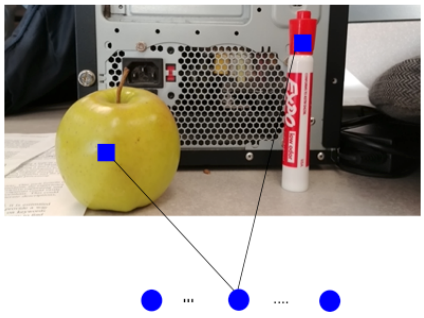
- Less Parameters
- Get more data
- Other Techniques:
    - Regularization
    - Dropout – randomly disable nodes during training
    - Data augmentation



9.4 Convolutional Neural Networks

Applying MLP to images:

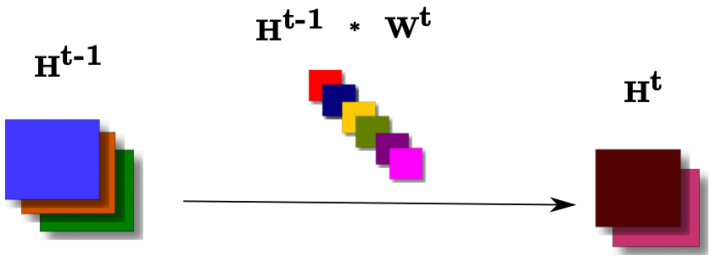
- High parameterization
- In natural images, adjacent pixels are spatially correlated.
    - Fully connected hidden layers do not exploit this property.



Convolutional neural network

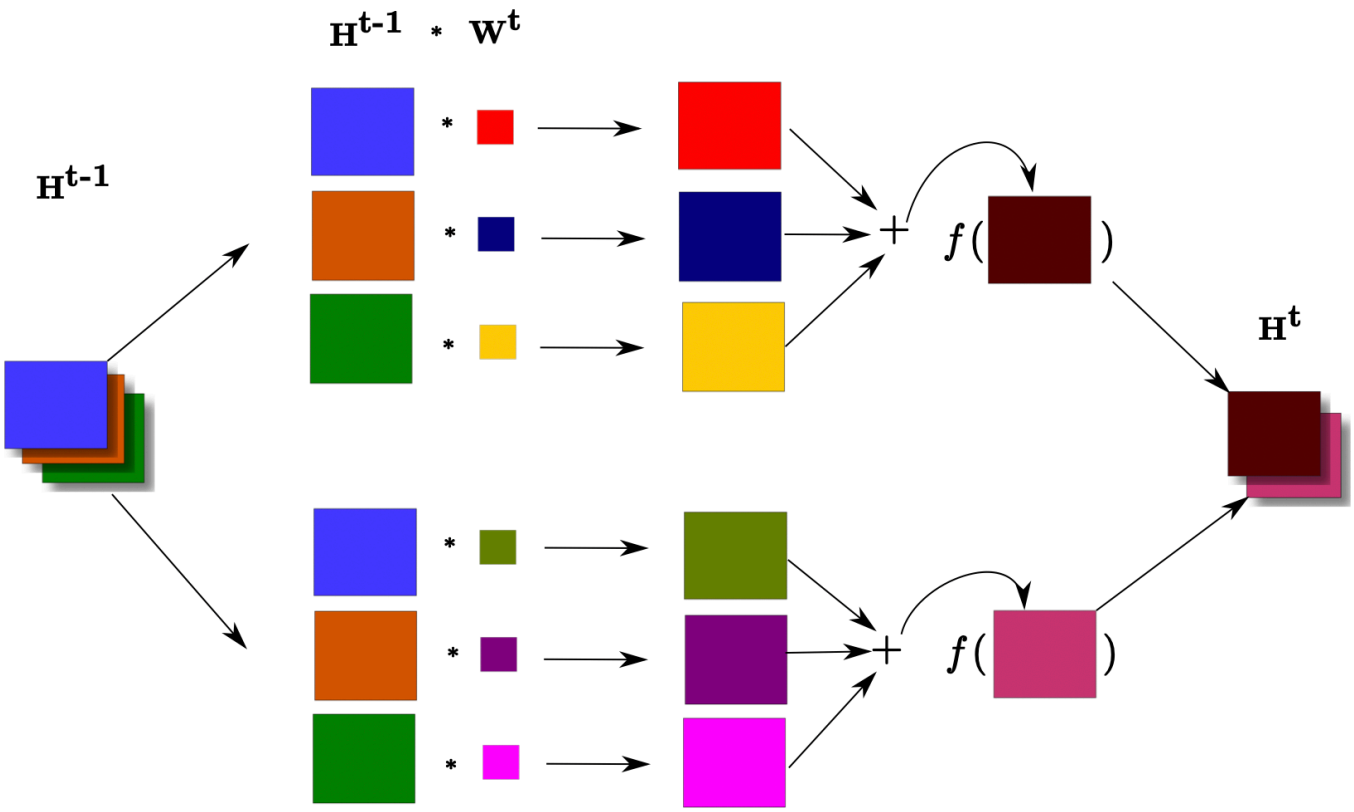
- https://medium.com/datadriveninvestor/convolution-neural-networks-vs-fully-connected-neural-networks-8171a6e86f15

- Composed of **convolutional layers**
  - Replaces fully connected matrix multiplication with convolution (actually correlation).

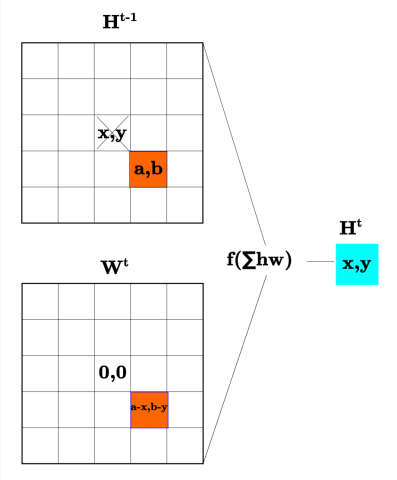
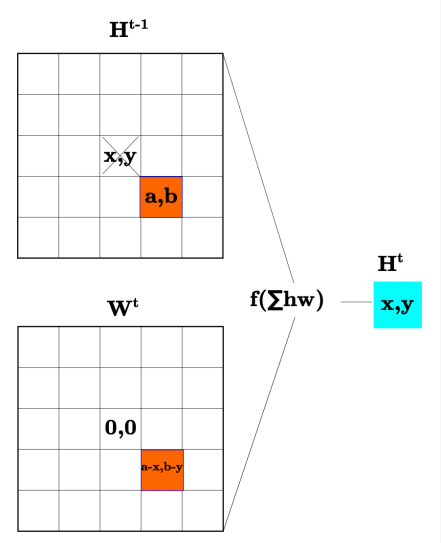
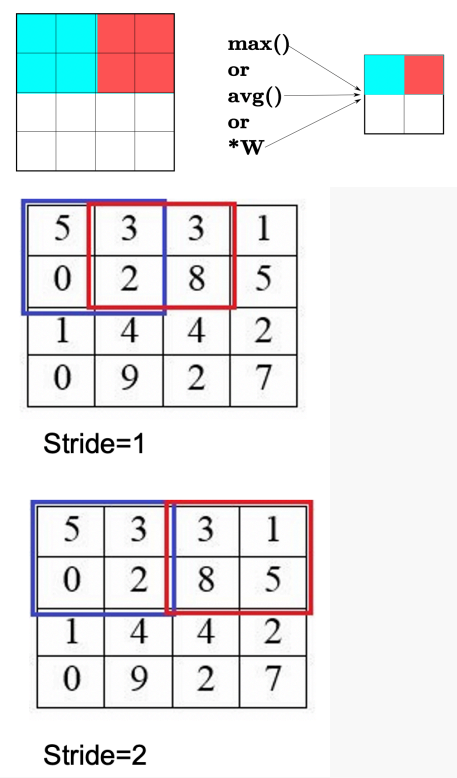


- **Layers**
  - $H^t \in \mathbb{R}^{X \times Y \times c}$, $H^{t-1} \in \mathbb{R}^{A \times B \times d}$
- **Parameters**
  - $W^t \in \mathbb{R}^{m \times n \times c \times d}$, $b^t \in \mathbb{R}^c$
  - $m, n \ll X, Y$
  - A total of $c \times d$ conv. filters that to interact between the channels of each layer
- **Formula:**
  - $H^t_{:,:,i} = f(\sum_j (H^t_{:,:,i} * W^t_{:,:,i,j}) + b^t_i)$

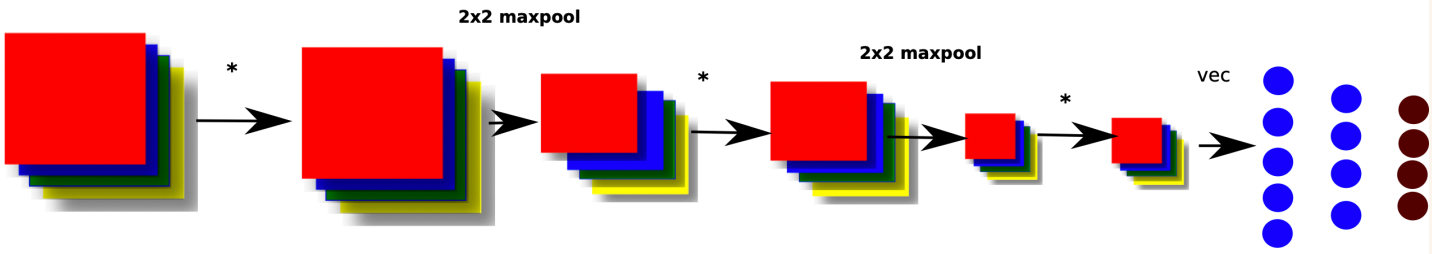
Convolutional Layer



Training CNN - Layer Table

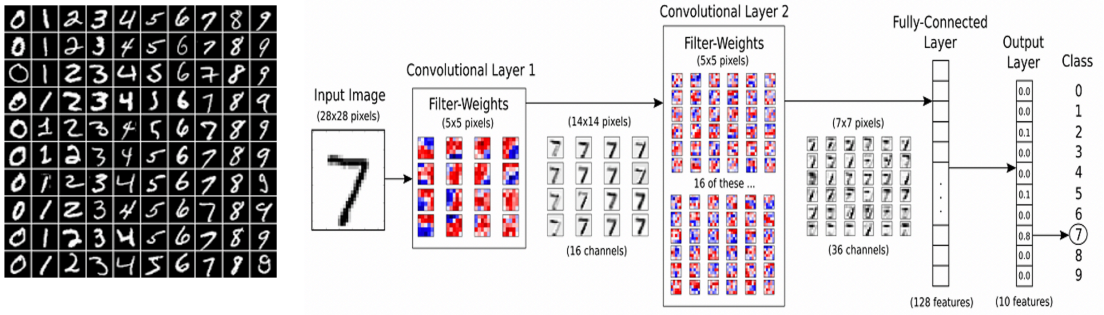
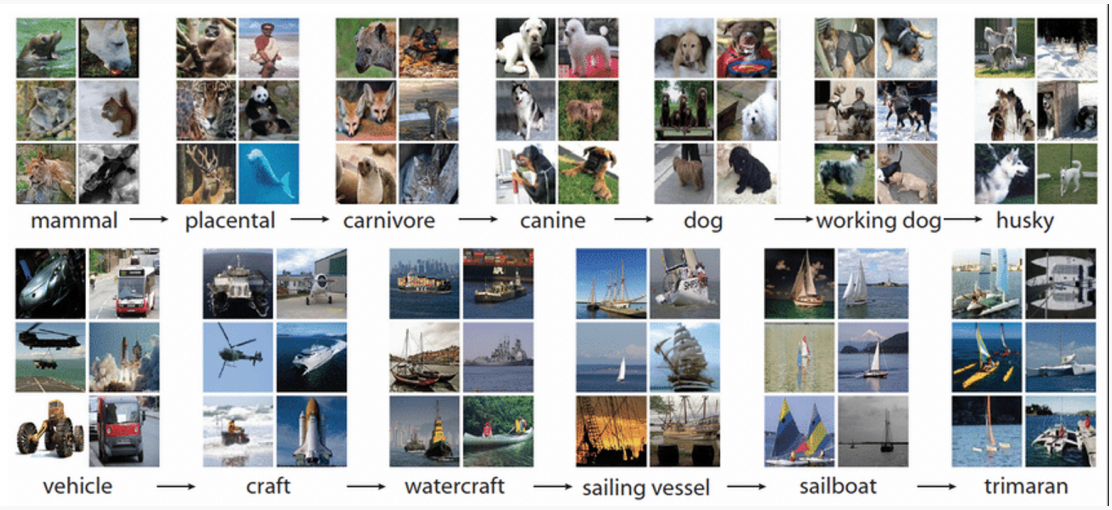
Layer to Layer	Layer to Weight	Other Layers
$\frac{\partial L}{\partial H^{t-1}_{a,b,j}} = \sum_x \sum_y \frac{\partial L}{\partial H^t_{x,y,i}} \frac{\partial H^t_{x,y,i}}{\partial H^{t-1}_{a,b,j}}$	$\frac{\partial L}{\partial W^t_{a-x,b-y,i,j}} = \sum_x \sum_y \frac{\partial L}{\partial H^t_{x,y,i}} \frac{\partial H^t_{x,y,i}}{\partial W^t_{a-x,b-y,i,j}}$	- It is useful to reduce spatial dimension. - Analyze image features at different scales - Can be accomplished through **striding**
		

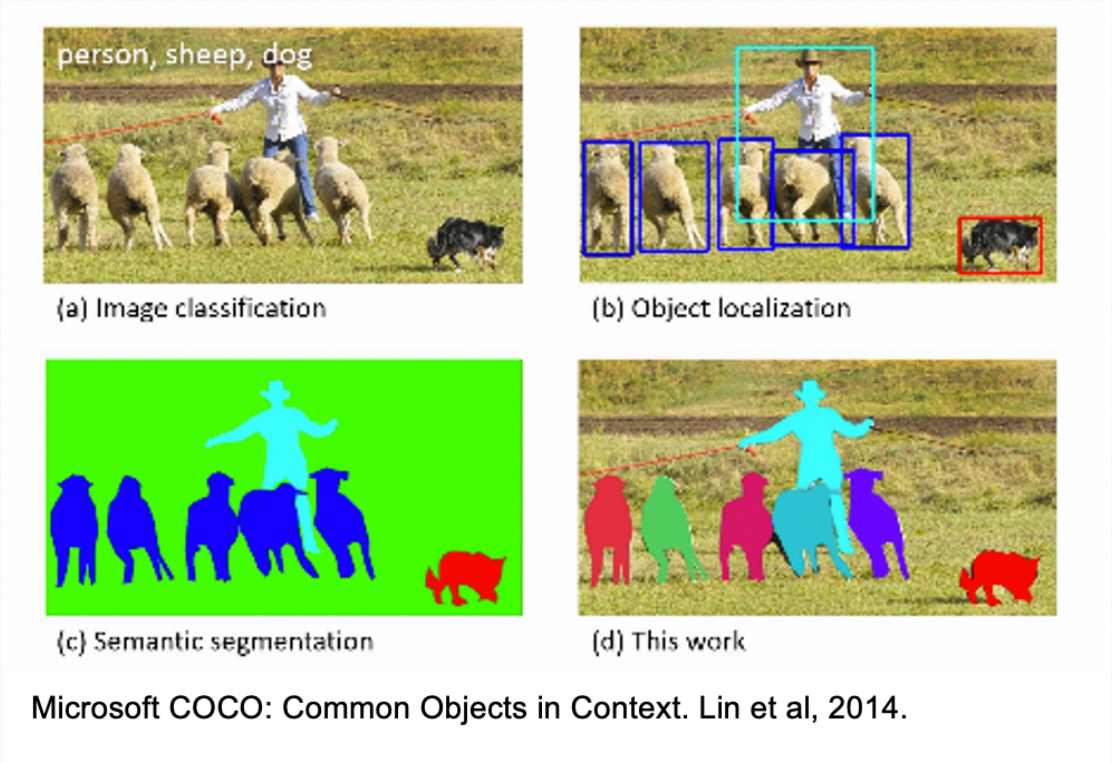
Example CNN Architecture



Intermediate layers contain “deep semantic features” of the input

Applications Table

Application & Details	Image
MNIST - A large database of handwritten digits - Used for training and testing different Machine Learning models	
ImageNet - a large dataset of annotated photographs intended for computer vision research - around 14 million images - 21 thousand groups - 1 million images that have bounding box annotations	
ImageNet - General Tasks - Image Classification: Predict the classes of objects present	

in an image - **Single-object Localization**: Image classification + draw bounding box around one example of each object present - **Object Detection**: Image classification + draw a bounding box aoround each object present	
Microsoft COCO - 330 000 images - Segmentation, object recognition, key points ...	 Microsoft COCO: Common Objects in Context. Lin et al, 2014.

Buzzword List:

- Fully convolutional networks
- Recurrent neural networks
- Batch normalization
- Dropout
- Residual blocks
- Generative Adversarial networks
- Training algorithms: RMSProp, ADAM, Adagrad

Further Reading:

Neural Networks and Deep Learning by Michael Nielsen

Pattern Recognition and Machine Learning by C. Bishop

A guide on convolutional arithmetic for deep learning by V. Dumoulin and F. Visin