Today

- Dimensionality Reduction
- Eigenfaces
- Viola-Jones Face Detector
- Bag-of-Visual-Words
Curse of Dimensionality

Keep in mind, our images are high dimension:

![Image of face with pixel values]

19x19 pixels = \(361\) vector (flat)

From a sampling viewpoint:

- As the number of dimension (linearly) grows the number of samples to cover the space grows \textit{exponentially}!

1D: we took 10 samples.

so from a k-NN perspective we can guarantee we will get a sample that’s 0.1 away.
Curse of Dimensionality

Keep in mind, our images are high dimension:

From a sampling viewpoint:

- As the number of dimension (linearly) grows the number of samples to cover the space grows \textbf{exponentially}!

2D: now we need \textbf{100} (10²) samples to cover with a 0.1 margin.
Curse of Dimensionality

Keep in mind, our images are high dimension:

From a sampling viewpoint:

- As the number of dimension (linearly) grows, the number of samples to cover the space grows exponentially!

3D: now we need 1000 ($10^3$) samples to cover with a 0.1 margin.
Curse of Dimensionality

Keep in mind, our images are high dimension:

From a combinatorics viewpoint:

- As the number of dimensions grows, the number of possibilities grows exponentially!

We have a single cube (i.e. 1D), we have 6 possibilities (and thus probability) for an outcome.

19x19 pixels = 361 vector (flat)
Curse of Dimensionality

Keep in mind, our images are high dimension:

19x19 pixels = 361 vector (flat)

From a combinatorics viewpoint:

- As the number of dimensions grows, the number of possibilities grows exponentially!

If we have 2 cubes (i.e. 2D), we have $36 \ (6^2)$ possibilities for an outcome.
Curse of Dimensionality

Keep in mind, our images are high dimension:

19x19 pixels = 361 vector (flat)

From a combinatorics viewpoint:

- As the number of dimensions grows, the number of possibilities grows exponentially!

If we have 3 cubes (i.e. 2D), we have $216 (6^3)$ possibilities for an outcome.
Curse of Dimensionality

Keep in mind, our images are high dimension:

- As the number of dimensions grows, the number of possibilities grows **exponentially**!

And so on...

19x19 pixels = 361 vector (flat)
Questions
So we should strive to work in lower dimensions as possible.

But how can we do that?

- Leverage on correlation between samples or dimensions
  - Clustering
  - PCA, LDA, ICA, etc.
  - Feature selection
  - Machine learning
- Throw away things at random (works surprisingly well)

Keep in mind, our images are high dimension:

19x19 pixels = 361 vector (flat)
Dimensionality Reduction

Images can be treated as high-dimension vectors.

We can apply arithmetics to vectors:

Or take an average: (over all the samples)

Insight: Faces all roughly look the same. I.e. their vectors have high correlation.
Dimensionality Reduction

We can translate the origin to the mean and rotate the “intensity” axes to a new linear basis. (what do we call such a rigid transformation?)

We can find new axis so the vectors will lie close to an axis that better represents the data than the original “intensity” axes.

In the new basis:
Axis 2 carries much less information than Axis 1.
So we can simply drop Axis 2.

How to find a new basis that maximizes the information on certain dimensions?
Dimensionality Reduction

Look at the (co-)variance across all the samples:

\[
\text{Var}(X) = \frac{1}{n} \sum_{i=1}^{n} (x_i - \mu)^2,
\]

\[
\text{var}(X) = \text{cov}(X) = E \left[ (X - E[X])(X - E[X])^T \right].
\]

\[
\Sigma = E \left[ (X - E[X]) (X - E[X])^T \right], \quad \mu = E(X)
\]

To maintain the most information we’d like to maintain the most (i.e. maximize) variance.

What is the vector that maximizes the var?
Dimensionality Reduction

Let's take the mean off of all the samples:

\[ \Sigma = \mathbb{E} \left[ (X - \mathbb{E}[X]) (X - \mathbb{E}[X])^T \right] \quad \mu = \mathbb{E}(X) \]

Now they have zero mean, we want to find an (orthonormal) transformation \( P \) so that \( P\hat{X} \) has diagonal covariance matrix \( \text{cov}(PX) \) - i.e. each dimension is uncorrelated with the others.

\[ \text{cov}(PX) = \mathbb{E}[PX (PX)^*] \]
\[ = \mathbb{E}[PX X^* P^*] \]
\[ = P \mathbb{E}[XX^*] P^* \]
\[ = P \text{cov}(X) P^{-1} \]

\( P \) diagonalizes \( \text{cov}(X) \)
Dimensionality Reduction

We’re looking for $P$ that diagonalizes $\text{cov}(X)$.

How do we find a diagonalization of matrices?
We're looking for $P$ that diagonalizes $\text{cov}(X)$.

How do we find a diagonalization of matrices?

Eigen decomposition!

$$\text{cov}(X) = \Sigma = PDP^{-1}$$

Verify $\Sigma$ is rectangular and positive definite

Where the columns of $P$ are the eigenvectors and eigenvalues are on the diagonal of $D$.

The EVecs are called the Principal Components, and this process is called Principal Component Analysis (PCA).
Here’s how the EVecs may look like:
Dimensionality Reduction

Since the EVecs are a new basis for the data, each sample can be represented as a linear combination:

Use the entire new basis to reconstruct the image precisely to the original - but that defeats the purpose of reducing dimension... What can we do?
Dimensionality Reduction

We can take just part of the EVecs, the heaviest ones (according to their EVals) - reduce the dimensionality, but --

We sacrifice some information.

But - how many EVecs should we take for our new base?
Dimensionality Reduction

How many EVecs should we take for our basis?

Original

~90% of variance covered by the first 10 EVs.

# EVs = 10  # EVs = 30  # EVs = 50  # EVs = 70  # EVs = 90  # EVs = 110  # EVs = 130  # EVs = 150
Detection, Recognition

What can we do with our PCs?

Object Detection:

- Sweep the image, take patch $x$
- Transform $x$ to the PC space, get the $a_i$ coefficients list.
- Check if lower than threshold

$$\|x - (\bar{x} + (a_1 v_1) + (a_2 v_2) + \ldots)\| < t$$

Object Identification:

- Use the $a_i$ coefficients list as a (NN) lookup vector in a reduced dimension space.
Questions
Say we have a 1000x1000 image.

For a sliding window detector we need to check $10^6$ patches. To be scale-invariant we need roughly $\frac{1}{2} 10^6$ additional locations.

Generally, faces in images appear much less than $10^6$ times (maybe 0-10 times?), so we have much more negative locations than positive ones.

Our False-Positive rate needs to be less than $10^{-6}$.

⇒ We should be fast about detecting.
Fast Object Detection
Viola-Jones
How can we speed up detection?

Think about our pain points (speed, accuracy):

- High-dimensions
- Calculation time
- Many more negative windows
Optimize for Speed

How can we speed up detection?

- Calculate features faster.
- Use less features.
- Reject negatives early.


Together the 2 papers garnered over 31,000 citations (!!) and were a cornerstone for real-time computer vision.

Key insights from their work:

- Use very simple features (effectively only summation and subtraction).
- Preprocess the image for feature calculation (the Integral Image)
- Spend most of the time training the classifier than in inference.
- Reject fast using a decision cascade.
Haar-like Features

Fast-to-calculate features:

- Sum the pixels in a number of rectangular areas
- Use area-sums as either “positive” or “negative” in a final summation
- The final summation gives a single number
- Called “Haar-like” - follow Haar’s signals proc. work

A very crude approximation of derivative filters we have seen in class, and others like Gabor and Sobel:

Figure 1: Example rectangle features shown relative to the enclosing detection window. The sum of the pixels which lie within the white rectangles are subtracted from the sum of pixels in the grey rectangles. Two-rectangle features are shown in (A) and (B). Figure (C) shows a three-rectangle feature, and (D) a four-rectangle feature.
Haar Features

In training:

- We take many thousands of such features
  - Problem: How many should we take?
- Pick the “best ones”
  - Problem: How can we determine what’s better?

Figure 1: Example rectangle features shown relative to the enclosing detection window. The sum of the pixels which lie within the white rectangles are subtracted from the sum of pixels in the grey rectangles. Two-rectangle features are shown in (A) and (B). Figure (C) shows a three-rectangle feature, and (D) a four-rectangle feature.
Haar Features

Problem: Speed.

Summing the pixels is $O(n)$. With several 1,000s of features this approaches $O(n^2)$...

What can we do to make this faster?

Figure 1: Example rectangle features shown relative to the enclosing detection window. The sum of the pixels which lie within the white rectangles are subtracted from the sum of pixels in the grey rectangles. Two-rectangle features are shown in (A) and (B). Figure (C) shows a three-rectangle feature, and (D) a four-rectangle feature.
If we could calculate the sums ahead of time for all possible feature-area combinations - we’d be golden.

Enter - the Integral Image.

\[
ii(x, y) = \sum_{x' \leq x, y' \leq y} i(x', y')
\]

Every pixel in the I.I. is the sum of all pixels above and to the left in the original image.

How can that help us in calculating the area-sums?
Integral Image

How can that help us in calculating the area-sums?

Figure 2: The sum of the pixels within rectangle $D$ can be computed with four array references. The value of the integral image at location 1 is the sum of the pixels in rectangle $A$. The value at location 2 is $A + B$, at location 3 is $A + C$, and at location 4 is $A + B + C + D$. The sum within $D$ can be computed as $4 + 1 - (2 + 3)$. 
How can we calculate the I.I. very fast? in $O(n)$
Integral Image

How can we calculate the I.I. very fast? in $O(n)$

Dynamic programming!

$$II(D) = X(D) + II(C) + II(A) - II(B)$$
Questions
Feature Selection

OK we can do it fast, but:

There are a few 100,000 feature possibilities for different sizes, types (2-rect, 3-rect and 4-rect), orientations and locations in reasonably sized images (1-2 MP).

We cannot calculate them all and make a decision in reasonable running time!

**How can we select just the best features?** So we can make a fast decision.

We’ve seen similar things...

Which one is the best?
Feature Selection

Treat each feature as a binary classifier.

Pick a threshold $\theta$ that minimizes the FP rate.
Feature Selection

Treat each feature as a binary classifier.

Repeat for all features...

$h_i(x) > \theta_i$

Pick the best one (least error)
Feature Selection

Treat each feature as a binary classifier.

Re-weight the samples, s.t misclassified ones are stronger, and successfully classified ones are weaker:

\[ h_i(x) > \theta_i \]
Feature Selection

Treat each feature as a binary classifier.

Next round, look for minimal (weighted) error again.

Repeat...

$h_i(x) > \theta_i$
Boosting Weak Classifiers

Algorithm:

- Calculate the (weighted) feature response for all features $h_i$ on all samples
- For each calculated feature:
  - Select an optimal threshold value, s.t. It minimizes the number of misclassified samples
- Pick the best feature
- Increase the weights for misclassified samples
- Repeat until done

This process is called Adaptive Boosting (AdaBoost), and is widely used in machine learning.
Questions
Even Faster

Look at the types of features, what is the fundamental difference?

Think about what we need for calculation...
Even Faster

Look at the types of features, what is the fundamental difference?

Some are faster to calculate!

2-rect ones need: $3 \times 2 = 6$ sum/sub ops

3-rect ones need: $3 \times 3 = 9$ sum/sub ops

4-rect ones need: $3 \times 4 = 12$ sum/sub ops

Insight: We can use the cheaper features first, but obviously they will not do as good as the general pool of features.

$D = 4 + 1 - (2 + 3)$
Cascade of Classifiers

A two-feature classifier that has 100% detection rate (finds all of the faces), but also has a 40% FP rate:

This is a very weak but very fast classifier.

Idea: Improve the FP/TP rates in successive feature-group classifiers, but optimizing also for complexity rather than just accuracy.
Results
Failures

Algorithmic “racism”
Failures

Face identification is much harder...

From latest Apple event: 9/12/2017
Questions
Issues

Indeed boosted Haar-like features are very fast.

But they have some problems:

- Features may be too generic, too low-level
  - Don’t capture a lot of information
- May only work for faces?
  - Faces are very easy! They don’t vary much
- Features are not translation invariant
  - We rely on expensive sliding window
- We model the class boundary in the parametric space of feature responses - it’s a **discriminative model**
  - It’s not necessarily a bad thing
Discriminative vs. Generative Models

Discriminative Models

- Find boundaries between classes
- Model $p(y|x)$ - “probability of class $y$ given data $x$”
- Examples: SVM, Linear Classifiers

Generative Models

- Model the class probabilities directly
- Model $p(x|y)$ - “how does data $x$ look like within class $y$”
- Examples: Naive Bayes
Bag-of-Visual-Words
Orderless representation of text documents.

Treat a document as a list of separate words, then count frequencies and weight.

TERM-WEIGHTING APPROACHES IN AUTOMATIC TEXT RETRIEVAL

GERARD SALTON and CHRISTOPHER BUCKLEY
Department of Computer Science, Cornell University, Ithaca, NY 14853, USA

(Received 19 November 1987; accepted in final form 26 January 1988)

Abstract—The experimental evidence accumulated over the past 20 years indicates that text indexing systems based on the assignment of appropriately weighted single terms produce retrieval results that are superior to those obtainable with other more elaborate text representations. These results depend crucially on the choice of effective term-weighting systems. This article summarizes the insights gained in automatic term weighting, and provides baseline single-term-indexing models with which other more elaborate content analysis procedures can be compared.

# of citations: ~9150
Bag-of-Visual-Words

Idea:

- Learn visual characteristic features of objects
- Represent an image as the frequencies of the appearance of these features (histograms)

The learned features - “Visual Vocabulary”: 
Bag-of-Visual-Words

Pipeline:

- Extract Features (e.g. SIFT)
Bag-of-Visual-Words

Pipeline:

- Extract Features (e.g. SIFT)
- Cluster features from all images - learn vocabulary
Bag-of-Visual-Words

Pipeline:

- Extract Features (e.g. SIFT)
- Cluster features from all images - learn vocabulary
- Represent images as a histogram of the vocabulary

Match to vocabulary

Use the histogram as the feature vector
Bag-of-Visual-Words

Pipeline:

- Extract Features (e.g. SIFT)
- Cluster features from all images - learn vocabulary
- Represent images as a histogram of the vocabulary
- Train a classifier
Large-scale image matching

- Bag-of-words models have been useful in matching an image to a large database of object *instances*

11,400 images of game covers (Caltech games dataset)

how do I find this image in the database?
Large-scale image search

• Build the database:
  – Extract features from the database images
  – Learn a vocabulary using $k$-means (typical $k$: 100,000)
Example bag-of-words matches

Query image

Top 16 matches
Example bag-of-words matches

Query image

Top 16 matches
Example bag-of-words matches

Query image

Top 16 matches
Wrap Up