CS 6351 DATA COMPRESSION

APPLICATIONS BEYOND COMPRESSION

Instructor: Abdou Youssef

OBJECTIVES OF THIS LECTURE

By the end of this lecture, you will be able to:

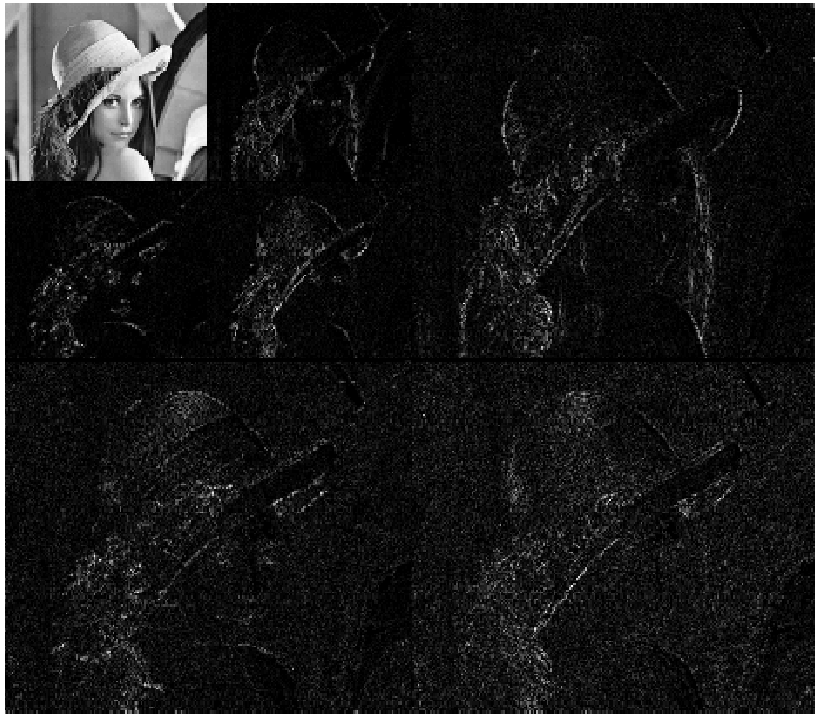
- Describe applications of many of the DC techniques, in other areas, such as:
  - Progressive transmission
  - Audio-visual query-by-example search
  - Differentiated error protection providing graceful recovery from bit errors
  - Watermarking
  - Feature extraction and feature selection in machine learning
  - Auto-encoders in deep learning (e.g., word embedding)
  - Feature learning in deep learning
  - Convolution Neural Networks (CNN) in deep learning

OUTLINE

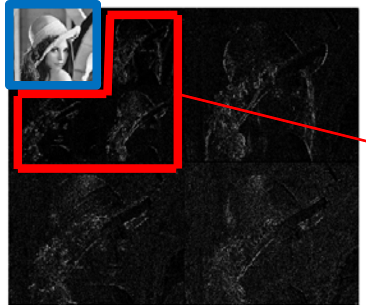
- Progressive transmission
- Indexing for query by example (look-like and sound-like search)
- Differentiated error protection
- Watermarking
- Feature extraction and feature selection in machine learning
- Auto-encoders in deep learning (e.g., word embedding)
- Feature learning in deep learning
- Convolution Neural Networks (CNN) in deep learning

PROGRESSIVE TRANSMISSION (1/5)

PROGRESSIVE TRANSMISSION (2/5)

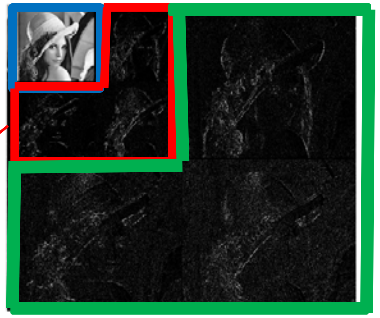


- How? Use subband coding
- Structure the coded bistream by subband
  1. The bistream of the LL subband,
  2. Then the bitstream of the next 3 HF subbands
  3. Then the bitstream of the next 3 HF subbands
  4. And so
- When transmitting, transmit the bitstream in that order
- The receiver decodes the received pieces, assuming the higher-pass subbands of the yet unreceived bitstream pieces to be zeros, and displays them
- As more pieces of the coded bistreams are received, the decoder decodes them, and algebraically adds that decoded part to the image in display
- By repeating this until all the bitstream pieces (i.e., subbands) are received, the image is finally displayed at full resolution



Lena received+reconstructed from **the LL band**

Improved after receiving **the next 3 HF bands**



Improved after receiving **the next 3 HF bands**

Improved after receiving **the next 3 HF bands**

PROGRESSIVE TRANSMISSION (5/5)

- Alternatively, you can use block DCT
- Encode the DCT coefficients of the blocks so that the lower-frequency components' encoding is distinguishable from the higher-frequency components' encodings.
- When transmitting, transmit the bitstream in that order (lower-frequency components are transmitted first, and later components transmitted successively later)
- The receiver decodes the received DCT components, assuming the yet unreceived components to be zeros, and displays.
- As later DCT components are received, they are decoded, assuming all other DCT components are zeros,
  - the new decoded "image" is added to the already displayed image
- By repeating this until all the DCT components are received, the image is finally displayed at full resolution

INDEXING FOR QUERY BY EXAMPLE (1/5)
-- LOOK-LIKE AND SOUND-LIKE SEARCH --

- Imagine there is a large collection of images
  - like all those on the Web, or
  - all those in your own personal collection
  - Or any collection of intermediate size
- Finding the image that resembles a given (query) image is of great interest
- To conduct such searches fast and with high accuracy (precision and recall):
  - It is too slow to compare live the query image with each image in the collection
  - Instead, we need ***indexing***, which is a common technique in all search and retrieval
- Indexing is a form of representing each object in your collection (e.g., text files, images, songs, etc.) with a small amount of differentiating information, called the index of the object

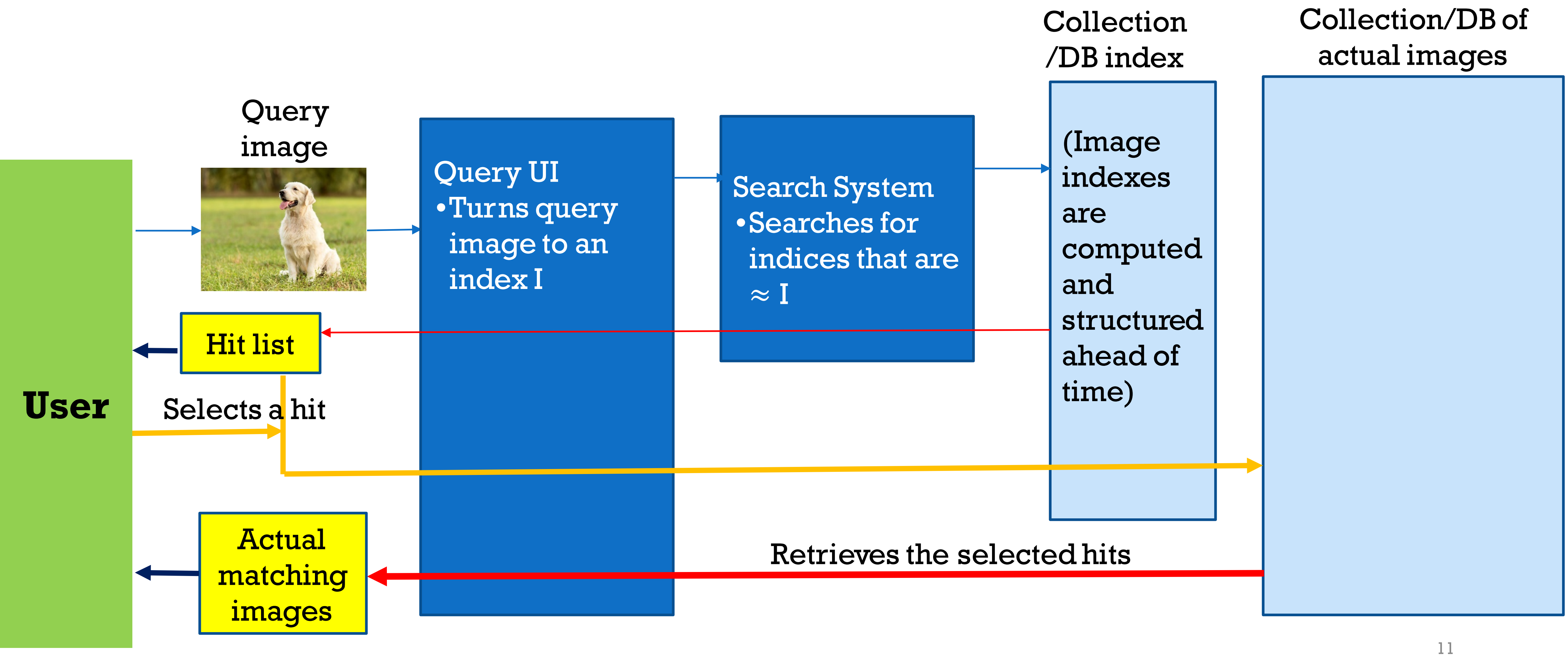
INDEXING FOR QUERY BY EXAMPLE (2/5)

-- LOOK-LIKE AND SOUND-LIKE SEARCH --

- The set of all the indices of the objects is called the index of the collection
- The index of the collection should be organized in such a way that searching against that index is as fast as possible
- Searching (for an image that resembles a query image) is done as follows:
  1. Compute the index of the query image
  2. Search for that index in the collection index, or, find in the collection index all the indices whose distance (e.g., MSE) from the query index is very small
  3. Take those matching indices from step 2, retrieve the corresponding images, and display them to the user as the hits (or matches) of the search

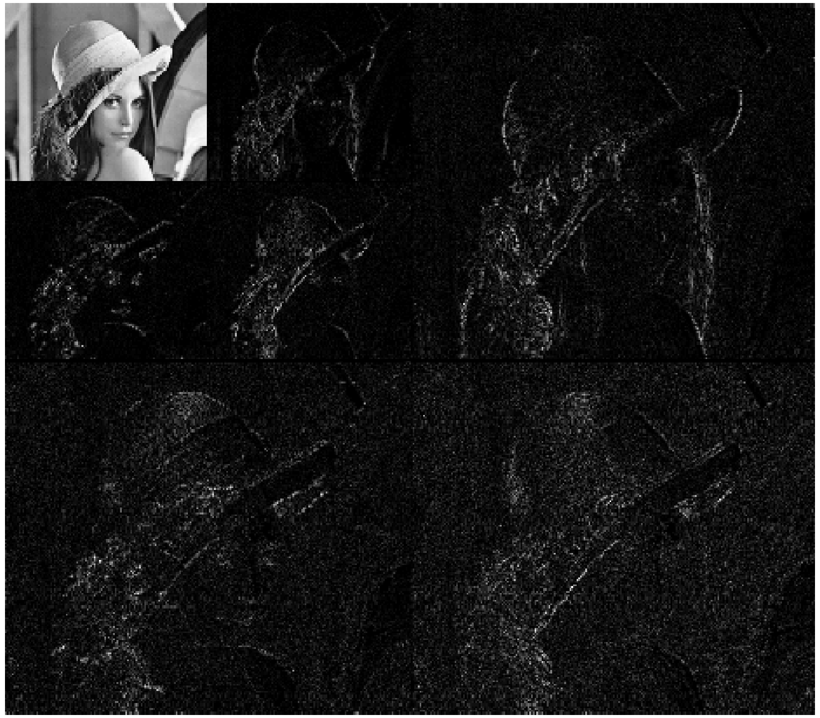
INDEXING FOR QUERY BY EXAMPLE (2/5)

-- LOOK-LIKE AND SOUND-LIKE SEARCH --



INDEXING FOR QUERY BY EXAMPLE (3/5)
-- LOOK-LIKE AND SOUND-LIKE SEARCH --

- The problem is then: how to compute a small index for a given image
- One approach is to have as an image index a thumbnail of the image



Take the LL subband as the thumbnail index

INDEXING FOR QUERY BY EXAMPLE (4/5)
-- LOOK-LIKE AND SOUND-LIKE SEARCH --

- But such a thumbnail is not robust against shifting or rotation of the images
- Also, the thumbnails can still be too big
- A better alternative so to compute the DCT of the LL subband, and keep only a few low-frequency components as the actual index
- This latter approach produces smaller indices, and is a little more robust to rotation/shifts in the images
- Another alternative is not to use subband coding, but instead to
  - divide each image into reasonably small blocks (though could be larger than 8x8),
  - apply DCT on each block,
  - take a few DCT low-frequency components from each block, and
  - finally take the combination of those selected components as an index for the image
- This allows for some shifting , and especially for sub-image search

INDEXING FOR QUERY BY EXAMPLE (5/5)

-- LOOK-LIKE AND SOUND-LIKE SEARCH --

- The previous approaches presented for image search can be easily modified to handle sound search
  - The query is a small sound (e.g., a hum, a whistle, piece of recording, …)
  - The search will find the full pieces of sound/music that resemble/contain the query sound
- Also, those same techniques can be adapted for indexing libraries of videos in order to do video search

DIFFERENTIATED ERROR PROTECTION (1/6)

- Imagine you compress an image/sound/video, and transmit the coded bistream (live)
  - During transmission, there is a high chance some of the bits get corrupted due to ambient noise in the atmosphere
- Or you stored the coded bitstream on disk, and days/years later , the file is a little corrupted due to some (perhaps minor) disk failure (e.g., a scratch)
- How can we protect against such mishaps?
- And can we protect in a way that achieves graceful degradation?
  - Graceful degradation: quality degradation proportional to the amount of error, rather than "all or nothing"

DIFFERENTIATED ERROR PROTECTION (2/6)

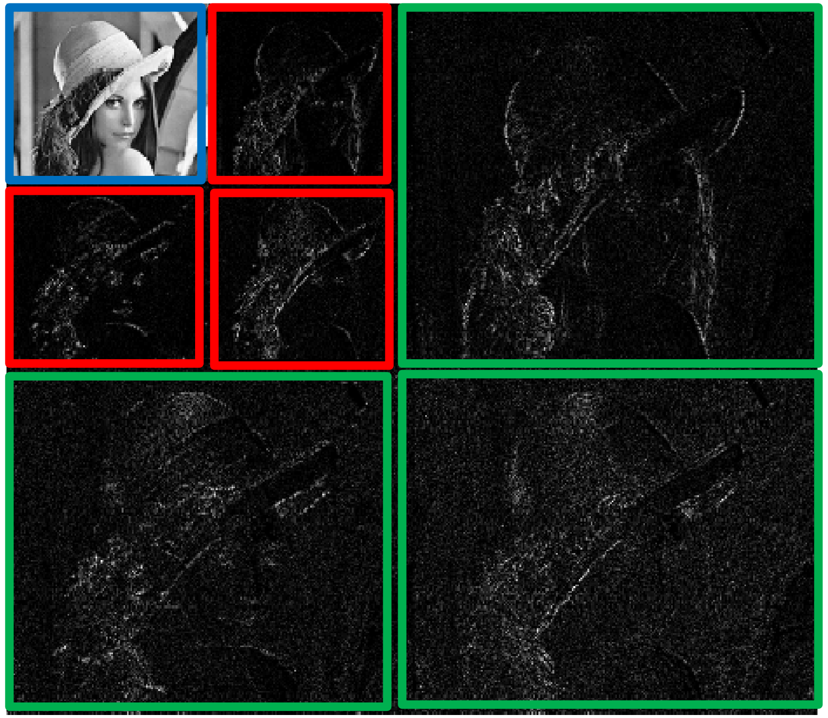
- In signal processing and information theory, there is a whole area called ***Error-Correcting Coding*** (ECC)
- The idea behind ECC is to
  - Divide the (long) binary string (e.g., the coded bitstream) into smaller substrings
  - Add some "redundancy" bits to each substring
  - The values of those redundancy bits are determined by the specific ECC scheme and by the values of the substrings
  - Depending on the ECC scheme and the length of the redundancy bits, one can detect and correct a number (say k) of flipped/corrupted bits in each substring

DIFFERENTIATED ERROR PROTECTION (3/6)

- Pros of this ECC scheme:
  - If the number of errors is $\leq k$, then all the errors can be corrected, and the receiver/decoder can decode the bitstream fully and correctly
- Cons:
  - If the number of errors in at least one substring is $> k$, then that substring cannot be corrected $\Rightarrow$ the whole coded bitstream will likely be undecodable $\Rightarrow$ losing the whole input
- This all-or-nothing situation is undesirable
- Rather, it will be good to be able to recover something
  - preferably the whole thing at a slightly lower resolution
- This is referred to in CS and in Engineering as *graceful degradation*
  - No matter how many errors, we will still be able to reconstruct/recover at a decreasingly lower resolution

DIFFERENTIATED ERROR PROTECTION (4/6)
-- GRACEFUL RECOVERY FROM BIT ERRORS --

- To achieve this graceful degradation, and even minimize the degradation, we can use JPEG/DCT compression, as well as subband coding
- In subband coding, for example
    1. Apply subband coding to get the subbands
    2. Quantize the subbands with scalar quantizers
    3. Turn each subband separately into a coded bitstream
    4. For each very high frequency subband, add a small number of redundancy bits to that band's bitstream
    5. As we get into lower frequency subbands, add increasingly more redundancy bits to their bitstreams
- This way, the more important data is protected more
- We are able to detect & correct more bit errors in the lower frequency subbands



DIFFERENTIATED ERROR PROTECTION (5/6)
-- GRACEFUL RECOVERY FROM BIT ERRORS --

- This is called differentiated error protection
  - Different parts of the data is protected with different levels of protection (i.e., with different numbers of redundancy bits)
  - The more important a data part, the more protection it gets
- There is a higher probability to fully recover from (even more errors) in the low-freq subbands
- If we can't recover from the errors in a high-frequency subband, we can replace it by zeros during the reconstruction/synthesis/decoding stage.
- This leads to a (slightly) lower resolution, but still the whole image is reconstructed
- Furthermore, even if more errors occur and more (probably high-freq) subbands are corrupted beyond the redundancy bits' ability to recover them, we can replace those corrupted subbands by zeros, and still recover a lower-resolution reconstruction

DIFFERENTIATED ERROR PROTECTION (6/6)
-- IN BLOCK-DCT ENCODING --

- Use the same previous approach to bock-DCT-based encoding
- Simply protect the low-frequency components' portion of the bitstream with more redundancy bits
- As you move to higher and higher frequency components (AC terms), protect their portion of the coded bitstream with fewer and fewer redundancy bits
- At decoding time, any corrupted (likely higher-frequency) components that cannot be recovered are replaced by zeros before applying the inverse DCT
- This will result in recovering the whole image, at increasingly lower resolution, depending on the number of unrecoverable errors

WATERMARKING (1/4)
-- DEFINITION --

- Watermarking is the process of hiding information inside a file
- If the file is an image, the watermark information is not visible when the image is displayed
  - Occasionally, though, people want to <u>show</u> the watermark, like the word "draft" across the diagonal of the page, whether it is an image or a text
- If the file is audio, the watermark information is not "heard" when the file is played
- Same if the file is a video
- Furthermore, the presence of the watermark should not affect the quality of the file (whether image, audio or video)

WATERMARKING (2/4)

-- USES--

- Watermarking is used for copyright protection
- Suppose you have an image on your Web page and you want to sell it to customers such that
  - No customer is allowed to mass-copy it for profit
- You can embed (hide) a watermark in that image, where the watermark varies from buyer to buyer, like when the watermark codes the buyer’s id
- If a buyer makes and sells copies of the image they bought from you, do
  1. Grab one of those copies
  2. Recover the watermark from it
  3. Identify the original buyer from that watermark
  4. Sue that buyer

WATERMARKING (3/4)
-- DESIRABLE PROPERTIES --

- A watermark should be embeddable (hide-able) in any image
- A watermark should be undetectable
- A watermark should be indestructible (or at least should be quite robust to attacks)
  - The watermark should not be erased or distorted by various signal processing operations: filtering, compression, cropping, image enhancement, etc.
- If an image is printed on a piece of paper then digitized, it is desirable to still have the watermark preserved in that last digitized image
- Similar properties should hold for audio watermarks and video watermarks

WATERMARKING (3/4)

-- HOW TO EMBED WATERMARKS LIKE THAT--

- It is a challenging problem
- The more robust to attacks you want the watermark to be, the more challenging
- One simple way is to hide the watermark in high-frequency components
  - Modifying the HF components slightly will not produce ***perceptible*** changes
  - That is because we are less sensitive to high-frequency data
- But such a simple approach is not robust
- Other approaches involve hiding many replicas of the watermark in silent/blank spaces (like varying the spaces between words in some coded way where the code identifies the buyer)
- Other sophisticated ways code the watermark in a clever way in the HF components, such that even signal operation won't totally destroy it

WATERMARKING (4/4)

-- AN ART BY ITSELF --

- We will not say more about watermarking
- Suffice it to say that transforms could serve as one method of hiding watermarks
- Usually that method has to be combined with other methods to produce increasingly more robust watermarks

FEATURE EXTRACTION AND FEATURE SELECTION IN MACHINE LEARNING (1/2)

- In machine learning, objects are typically represented by a vector of certain measurements.
- The measurements are called ***features***
- The vector of the measurements is called the ***feature vector*** representing the object
- For example, facial recognition is a machine learning (classification) problem
  - Each human face is represented by one feature vector
  - The features can be anything that, collectively, would distinguish (and identify) a person
  - For example, features can be: length of ears, length of eyes, length of nose bridge, height of the forehead, distance between the inner corners of the eyes, etc.
  - Other types of features could be statistical features, such as "average' height of the forehead (reflecting the hairline), variance of the pixels in the forehead (capturing wrinkles), …

FEATURE EXTRACTION AND FEATURE SELECTION IN MACHINE LEARNING (2/2)

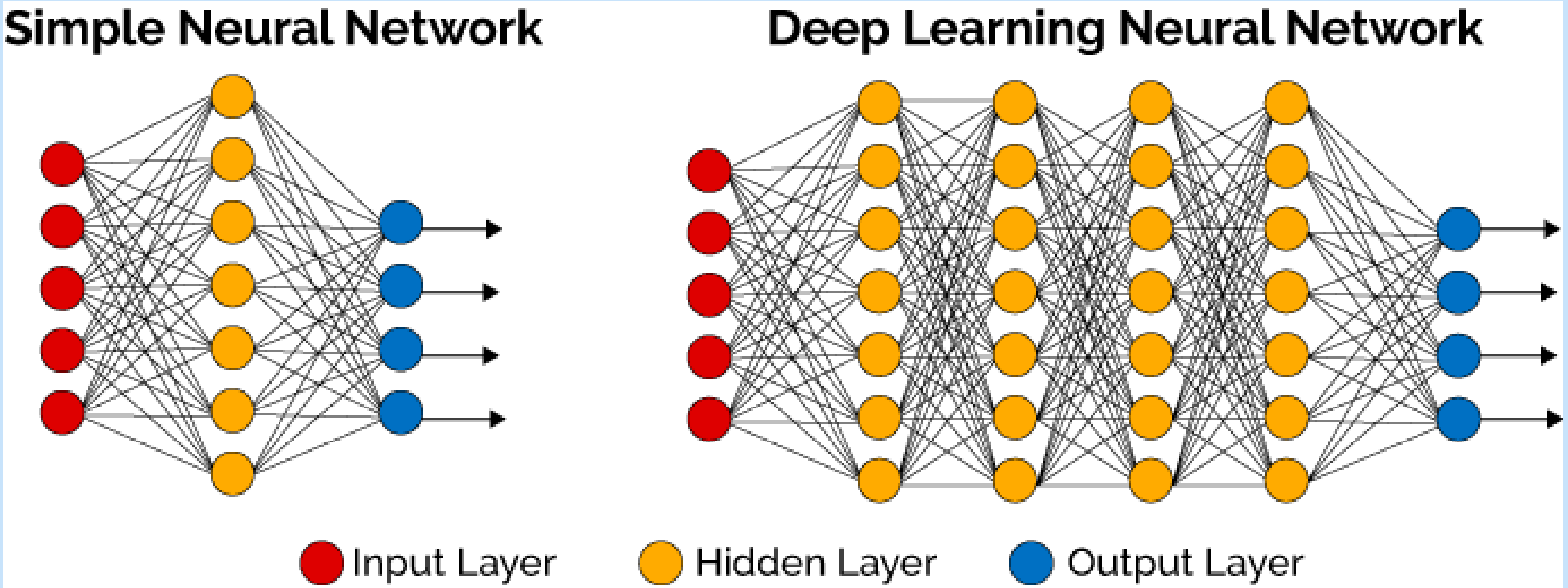
- DCT and DFT can yield features as well
- As we have seen, the frequency components, as features, have some interesting properties (including energy compaction)
- In particular, if a feature vector is quite long (e.g., many pixels), turning the features into frequencies and keeping only the lower-frequency components yield a smaller yet still representative/differentiating vector
- Note:
  - We're not saying that frequencies are the best or the only features
  - Rather, frequencies are potential features (among others) that have some desirable properties
  - So, one can combine a variety of features (e.g., frequencies, statistical, other measurements) into a single feature vector

AUTOENCODERS IN DEEP LEARNING

- Autoencoders are a special case of deep neural networks
- They are meant to code/represent features and feature vectors better
  - Shorter (and thus faster to use and to store)
  - More effective way (i.e., lead to better trained classifiers)
  - More meaningful way (e.g., words can represented by vectors that capture semantics)
  - Perhaps at different levels of abstraction, for example:
    - From pixels to edges
    - From edge boundaries
    - From boundaries to basic but recognizable shapes such as circles and triangles, etc. ….
- Before we can understand autoencoders and their connection to compression/decompression (coding/decoding)
  - We need to understand neural networks at some high level

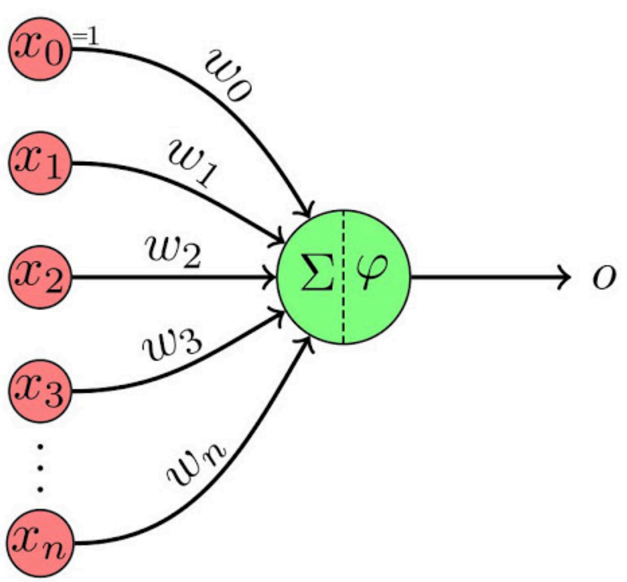
(DEEP) NEURAL NETWORKS (1/3)

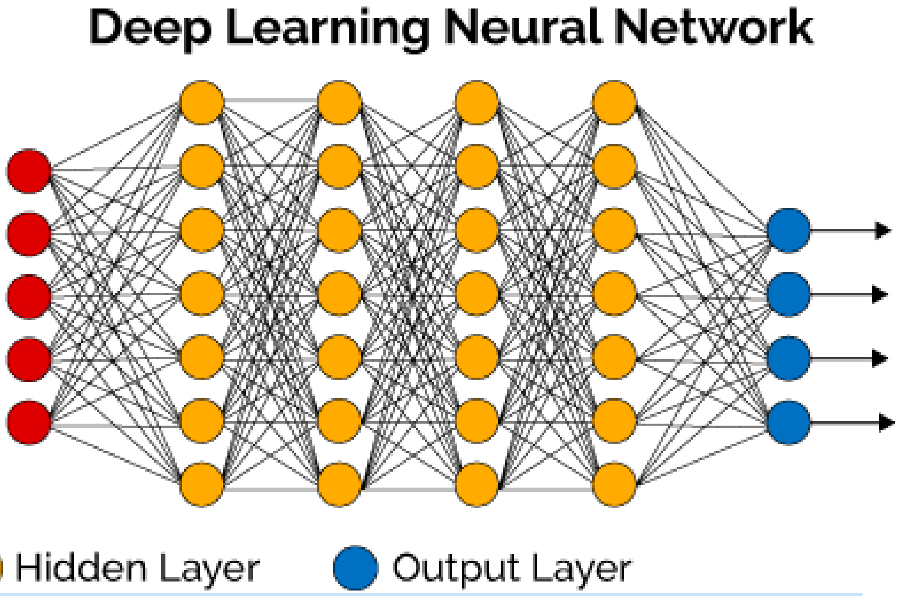
- General architecture of a (feed-forward) NN
- Every edge has a weight; the weights are the parameters of the NN
- Training a NN involves modifying the edge weights until the output of the NN agrees with the desired output to the fullest extent possible



DEEP NEURAL NETWORKS (2/3)

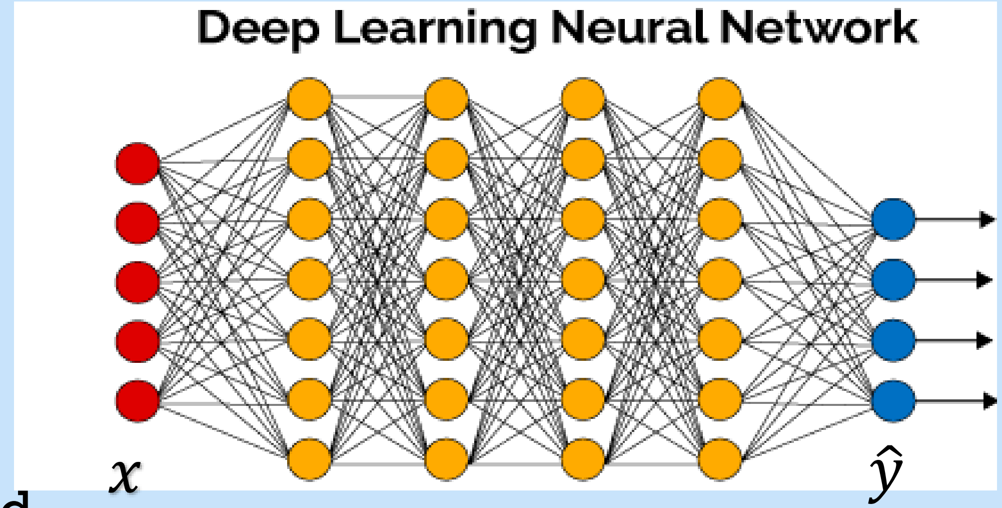
- Each feature vector (representing an object) is fed at the input layer:
  - The i^{th} feature is given to the i^{th} node in the input layer
- Each node in a layer (other than the input layer) receives n inputs from the n nodes in the previous layer, and computes its output as follows: $O = \phi(\sum_i x_i w_i)$
- ϕ is called the *activation function*
- ϕ can be the
  - Sigmoid: $\phi(t) = \frac{1}{1+e^{-t}}$
  - $\phi(t) = \tanh t$
  - ReLu: $\phi(x) = \max(x, 0)$
  - Etc.





DEEP NEURAL NETWORKS (3/3)

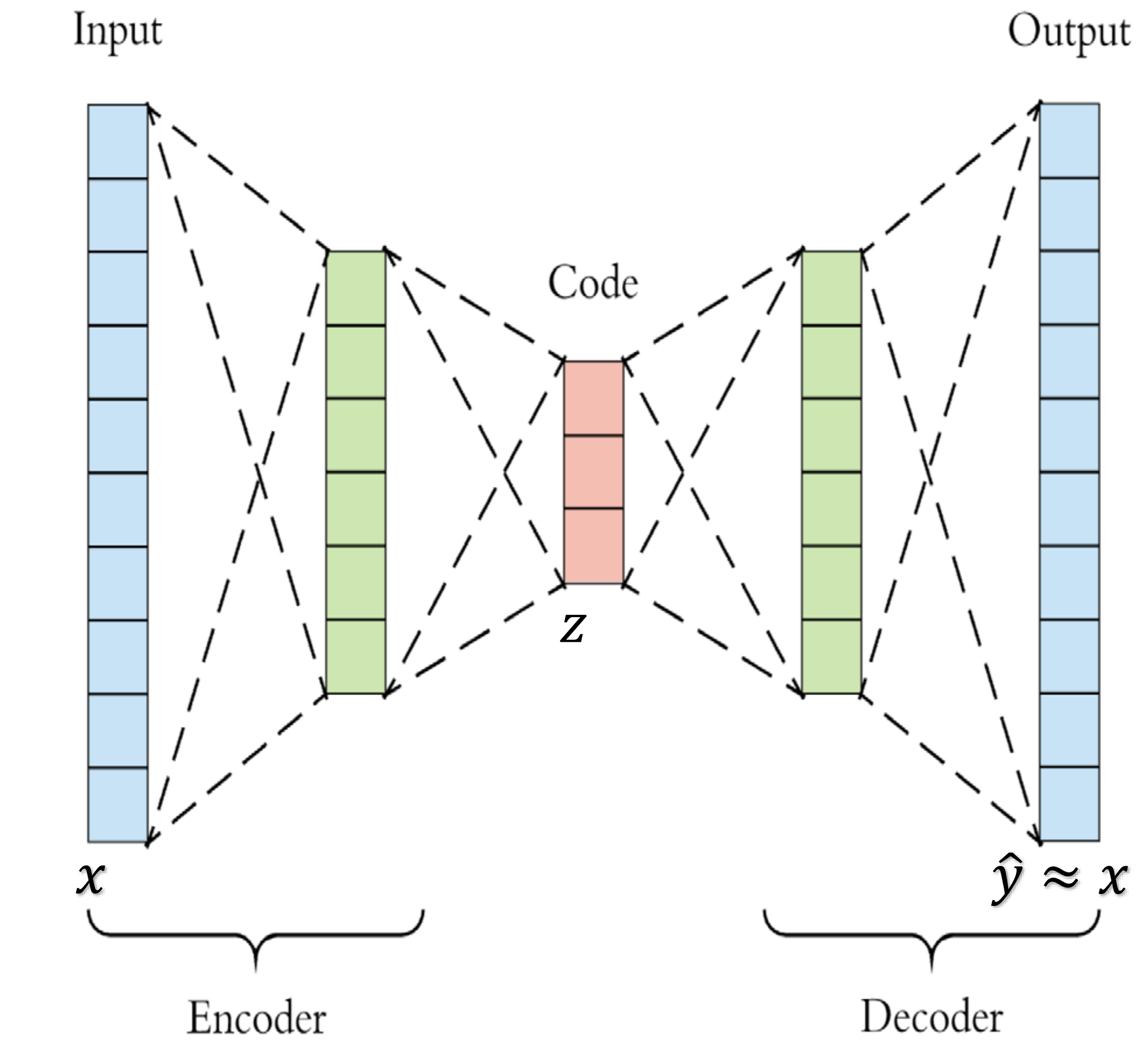
- The output of the whole DNN is a vector
- To "train" a DNN, you need a training dataset
- Each element (aka, instance/sample in the dataset is a pair (x, y) of vectors:
  - the input vector (a feature vector) x, and
  - the desired output vector y



- During training, instances (x, y) are used
  1. x is fed to the DNN as input
  2. The DNN produces as output a vector $\hat{y}$
  3. Some error function between y and $\hat{y}$ is computed
  4. Some backpropagation is performed to modify the edge weights based on the error, with the intent of reducing the error
- The previous step is repeated until the error is reduced/stabalized to some level

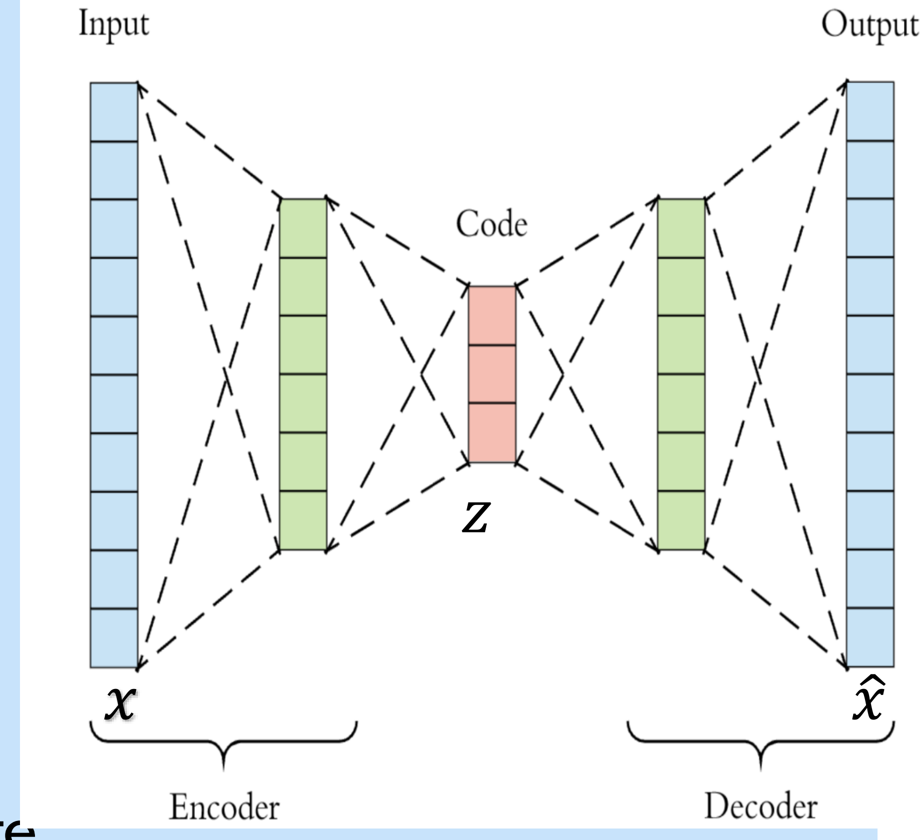
AUTOENCODERS (1/3)

- An autoencoder is a DNN trained on a dataset where each output vector = the input vector
- Think of the dataset as a set of basic/raw feature vectors x without actual outputs y
  - We turn each x into a pair (x,x), so y=x
  - Train the DNN with those pairs
- When the DNN is trained, the output z of the code layer (orange) is taken as a new feature vector representing x



AUTOENCODERS (2/3)

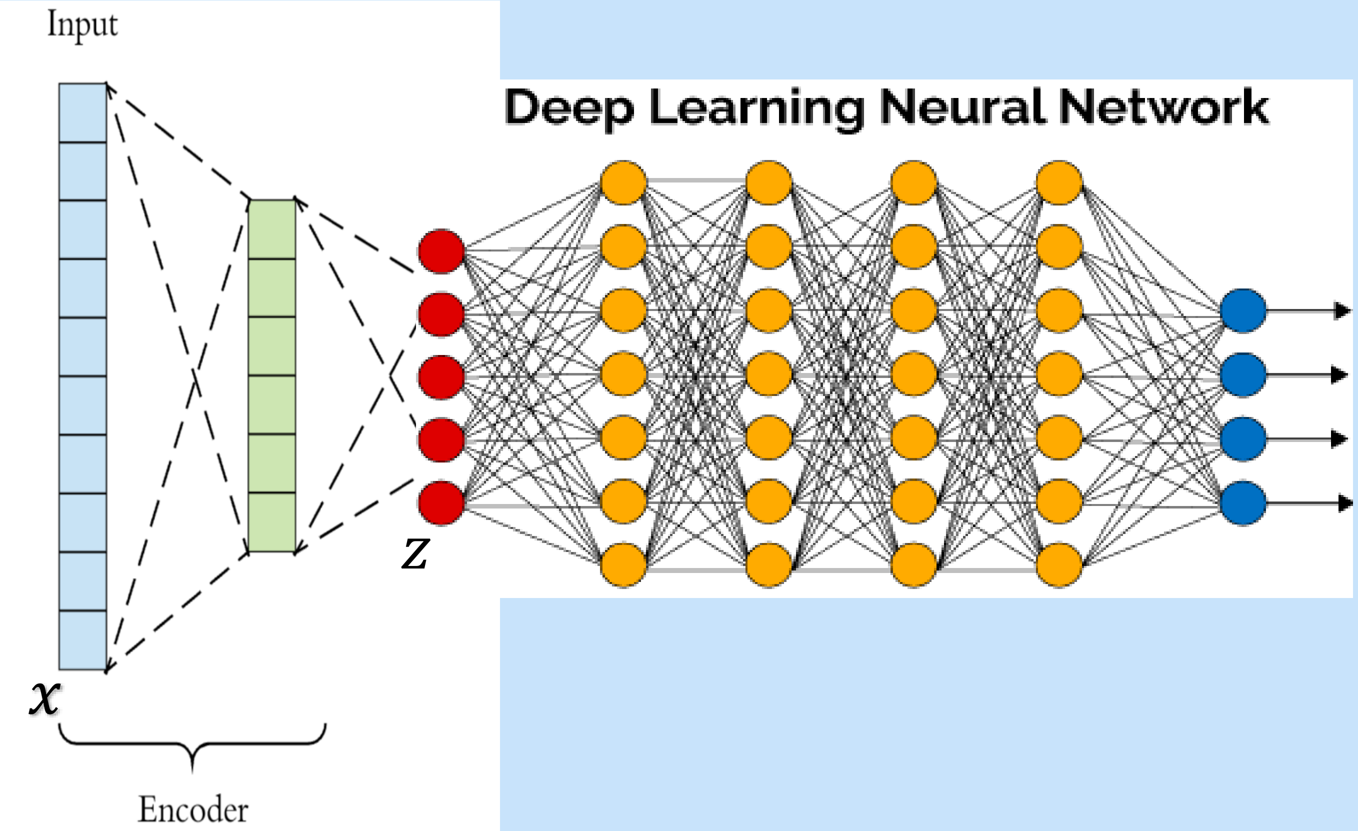
- When the DNN is trained, the output z of the code layer (orange) is taken as new feature vector representing x
- Typically, z is a better feature vector (than x) representing the underlying object
- By requiring the output to be equal to the input, the DNN is compression + decompression
- Due to error in the output, this compression is lossy
- The less loss there is, the more representative z is
- Once trained, the Encoder part is used as an input stage before any other DNN that will use the z vectors as input



AUTOENCODERS (3/3)

- Once trained, the Encoder part is used as an input stage before any other DNN that will use the z vectors as input

- That way, each original raw feature vector x gets transformed to be a feature vector z before it is fed to the DNN, either for training or for deployment after training



AUTOENCODER FOR NATURAL LANGUAGE PROCESSING (NLP) (1/3)

- When the input is individual words, and you want to use DNN's for some text related tasks (like telling if a word is a verb or noun or preposition, etc.)
  - The word has to be represented as a numerical feature vector
  - That is because DNNs can only take numerical feature vectors as input
  - DNN's don't take symbolic input!
- A very basic way of representing a word with a numeric feature vector is the so-called 1-hot vector
  - Imagine your language having N words (e.g., N=1 million)
  - Order the words of the language like in a dictionary: $w_1, w_2, \ldots, w_N$
  - Represent w_i with a 1-hot vector $[0, \ldots, 0, 1, 0 \ \ldots, 0]^T$ where 1 is in the i^{th} position
- The 1-hot vectors are really raw-feature vectors, and are utterly meaningless – only referential

AUTOENCODER FOR NLP (2/3)

- Train an autoencoder on the words of the language: input/output (x,x) are 1-hot vectors
- After training, every 1-hot vector x is encoded into vector z by the encoder part
- The z vectors are much shorter, and tend to be more meaningful
- In actual practice, the autoencoder for natural languages involves sequences of words at a time so that the autoencoder captures relational semantics between the words that tend to co-occur in the same sentences
- But the main idea is very much the same
- The adjustment just mentioned ends up producing feature vectors z for the words where the z's of related (e.g., synonymous) words are quite similar!!!!
- Those z vectors of words are called ***word embeddings***

AUTOENCODER FOR NLP (3/3)

- In short: the idea of lossy compression/decompression (or encoder/decoder)
  - Turned words into algebraically meaningful numerical vectors (word embeddings)
- Examples:
  - $z_{king} - z_{queen} \approx z_{man} - z_{woman}$
  - $z_{France} - z_{Paris} \approx z_{Italy} - z_{Rome}$

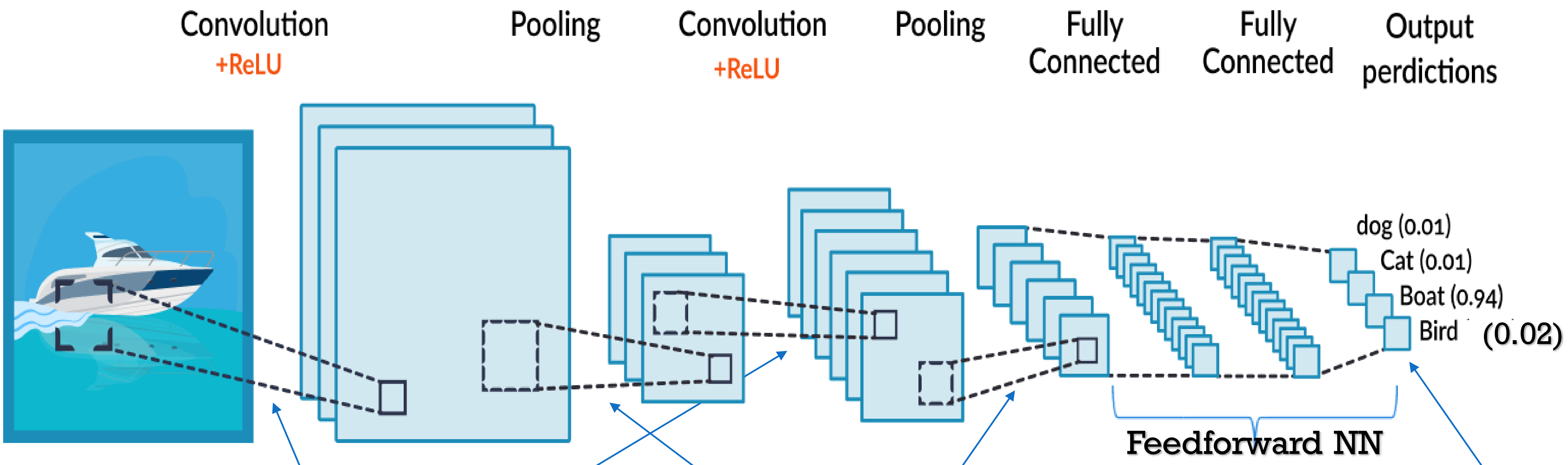
- Thus, if someone doesn't know the word for "female king",
  - they can compute $z_{queen} \approx z_{king} - z_{man} + z_{woman}$, and
  - get the corresponding word
- Similarly if they want to know the capital of France, assuming they know the capital of Italy

- That enabled the use of DNNs for natural language processing, such as translation from English to Chinese, or finding similar words (or similar concepts), etc.
- The use of DNNs in NLP is revolutionizing that whole field

FEATURE LEARNING IN DEEP LEARNING

- The same idea of autoencoder works in other fields besides NLP
- For example, for image understanding (e.g., OCR):
  - The raw features are simply the dumb pixels
  - Autoencoders can be used to convert the raw features (pixels) of an image into a more meaningful feature vector

CONVOLUTIONAL NEURAL NETWORKS (CNN) (1/2)



- Several filters, one per matrix
- Each filter is a 2D filter, not 1D
- So each filter is defined by a small rectangle of coefficients
- The coefficients are model parameters, and thus **are learned in training**

- Pooling = downsampling+avg

  or
- Pooling = downsampling+max
- That is, each m x m window W in the input is reduced ↓ to one pixel = avg(W) or max(W)

- The output is a probability vector (p_k)
- p_k is the probability that the input is class k
- The class of the input is selected to be the class with the largest p_k

CNN (2/2)

- The filters are **<u>not</u> human-designed**
- Rather, the filters are **<u>learned</u>** during training, using the training dataset
- Therefore, the filters are customized to the application
- The customization is automated
- Otherwise, we're seeing that filtering is very useful in feature extraction/refinement/learning
  - In moving from raw pixel features to refined pixel features
- CNN's have beaten human performance at certain image understanding tasks
  - ImgeNet contests (image classification)

CLOSING THOUGHTS

- We saw many applications of DC techniques outside of Data Compression
- Can you think of other applications?
  - Maybe from your own experiences and backgrounds
- We saw that DC techniques are being used in some aspects of machine learning
- Can machine learning be used to improve/customize compression?
  - Can we train a ML model to do a better transform?
  - Can we train a ML model to predict the next frame, and then compress the residuals (i.e., the differences between the actual frames and the predicted frames)?
- Think of other applications of DC techniques, and feel free to share them with me