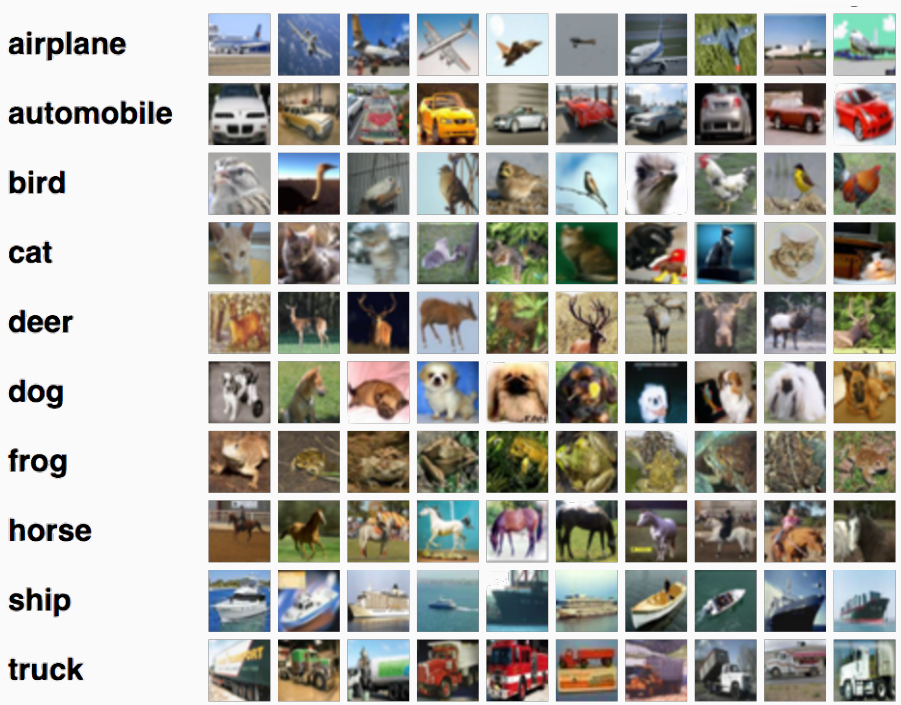
CS-UY 4563: Lecture 20
Auto-encoders, Dimensionality Reduction, Principal Component Analysis

NYU Tandon School of Engineering, Prof. Christopher Musco

COURSE LOGISTICS

Convolutional neural net demo: `demo_classification.ipynb`.

- Classification on CIFAR 10 data set. 60k images, 10 classes.
- **You're going to want to use a GPU.** Easiest way to access more compute power is through Google Colab.



TRICKS OF THE TRADE

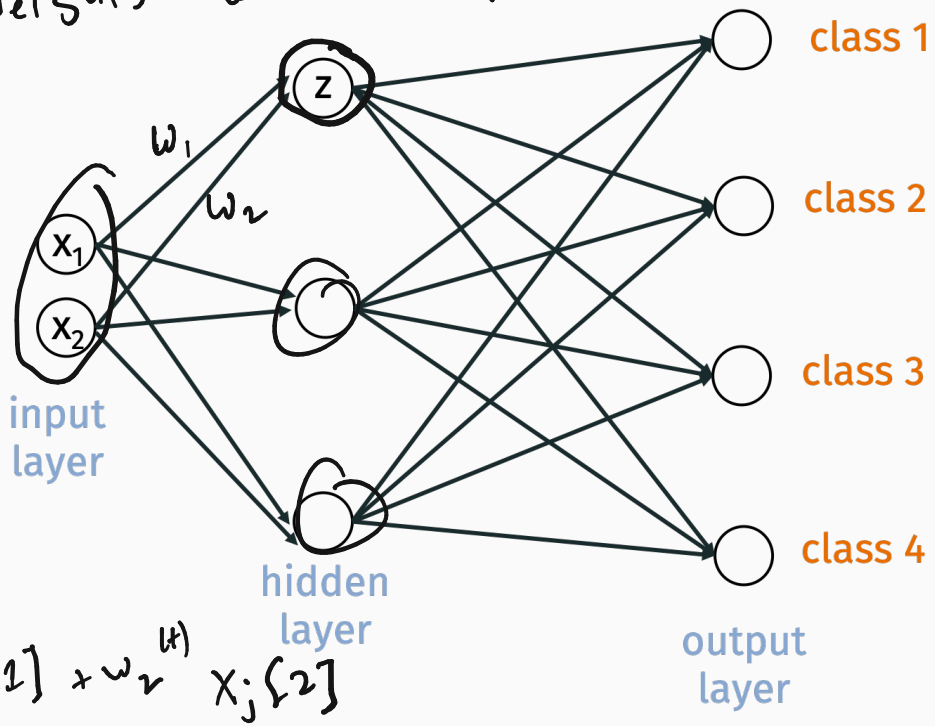
Beyond techniques already discussed (back-prop, batch gradient descent, adaptive learning rates) effectively training convolutional networks requires a lot of “tricks”. In the demo we use:

- Batch normalization (accelerate training).
- Dropout (prevent over-fitting)
- Data-augmentation.

BATCH NORMALIZATION

Start with any neural network architecture:

$w_1^{(t)}, w_2^{(t)}$ = weights at time t



$$\mu = \frac{1}{n}\sum_{j=1}^{n} z(\vec{x}_j)$$

$$= \frac{1}{n}\sum_{j=1}^{n} w_1^{(t)} x_j[1] + w_2^{(t)} x_j[2]$$

For input $\vec{x}$,

$\langle \vec{w}, \vec{x} \rangle$

$$\bar{z} = \vec{w}^T\vec{x} + b$$

$$z = s(\bar{z})$$

s = sigmoid, relu, ⋮

where $\vec{w}$, b, and s are weights, bias, and non-linearity.

BATCH NORMALIZATION

$\bar{z}$ is a function of the input $\vec{x}$. We can write it as $\bar{z}(\vec{x})$. Consider the mean and standard deviation of the hidden variable over our entire dataset $\vec{x}_1 \ldots, \vec{x}_n$:

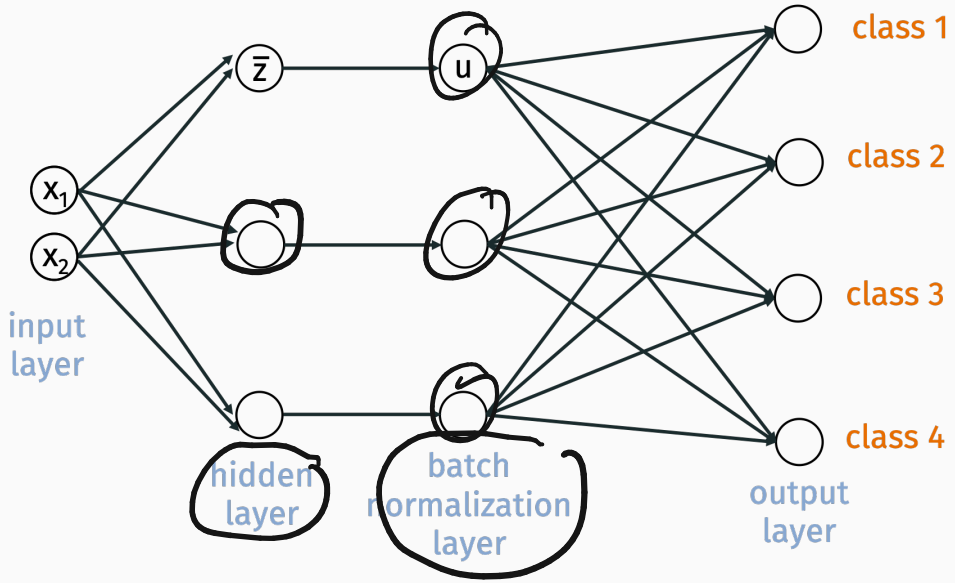
$$\mu = \frac{1}{n} \sum_{j=1}^{n} \bar{z}(\vec{x}_j) \qquad \text{sample mean}$$

$$\sigma^2 = \frac{1}{n} \sum_{j=1}^{n} (\bar{z}(\vec{x}_j) - \mu)^2 \qquad \text{sample variance}$$

Just as normalization (mean centering, scaling to unit variance) is sometimes used for input features, batch-norm applies normalization to learned features.

BATCH NORMALIZATION

Can add a batch normalization layer after any layer:



$$\bar{u} = \frac{\bar{z} - \mu}{\sigma}$$

$$u = s(\gamma \cdot \bar{u} + c).$$

$c = \mu$ $\quad \gamma = \sigma$ $\quad s(\bar{u})$

$u = s(\bar{z}) = 2$

Where γ and c are **learned parameters**. Has the effect of mean-centering/normalizing $\bar{z}$, and then mapping back to have a new mean and new standard deviation.

BATCH NORMALIZATION

Proposed in 2015: “Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift”, Ioffe, Szegedy.

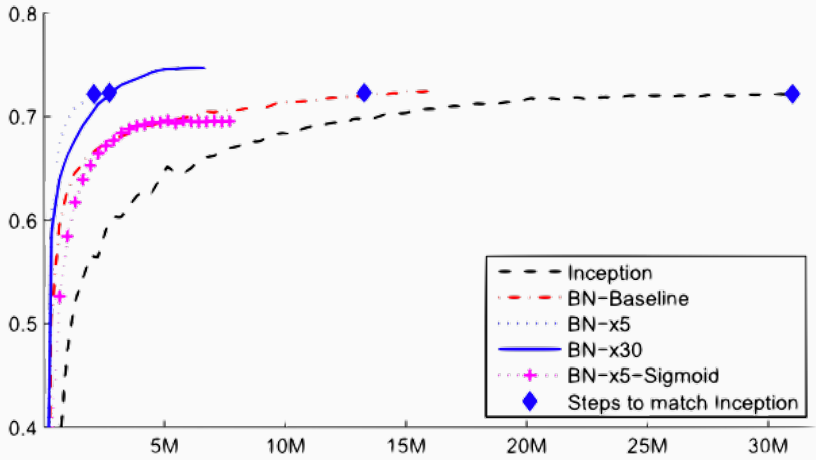
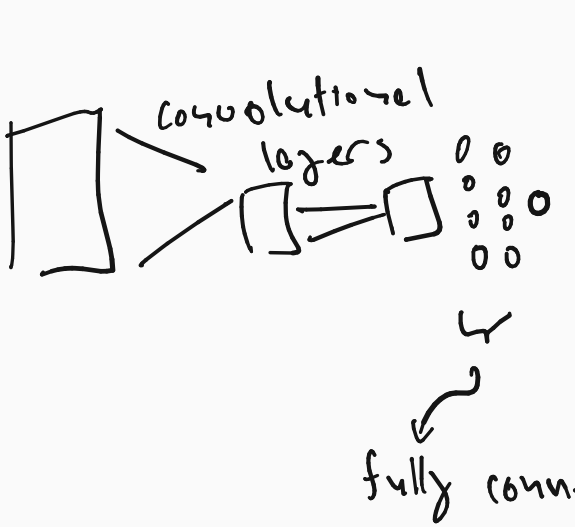


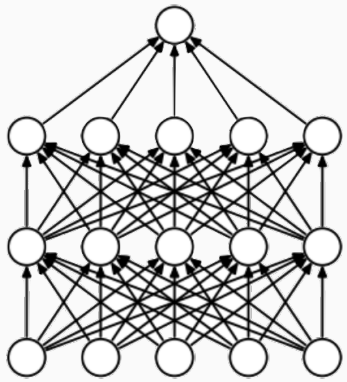
Figure 2: *Single crop validation accuracy of Inception and its batch-normalized variants, vs. the number of training steps.*

Doesn’t change the expressive power of the network, but allows for significant convergence acceleration. Authors and others have good intuition for why: happy to discuss this more offline.

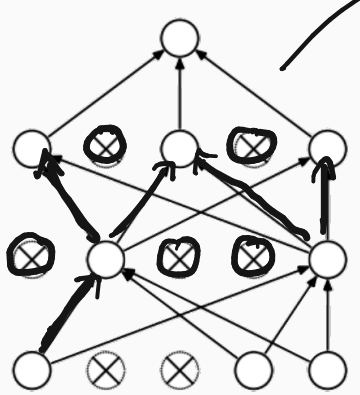
DROPOUT

Proposed in 2012: "Dropout: A Simple Way to Prevent Neural Networks from Overfitting", Srivastava et al.





(a) Standard Neural Net



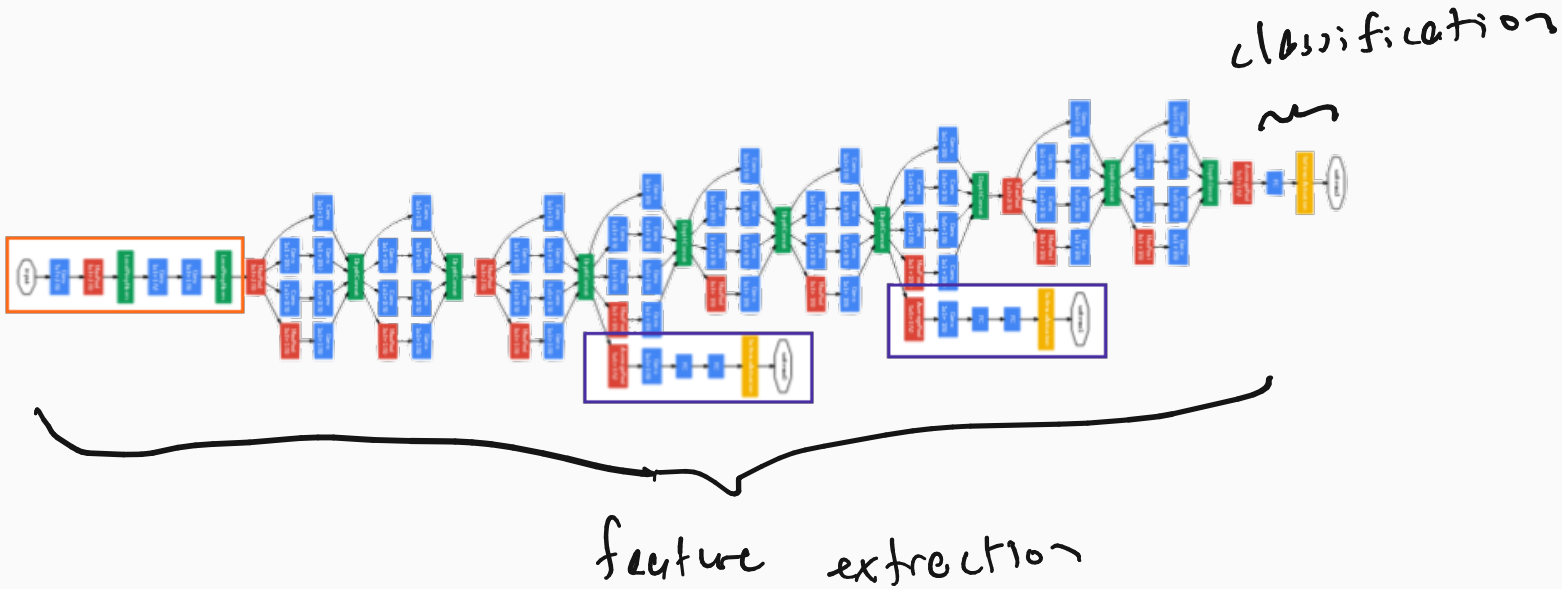
(b) After applying dropout.

still fully-connected but with less parameters

During training, ignore a random subset of neurons during each gradient step. Select each neuron to be included independently with probability p (typically $p \approx .5$). During testing, no dropout is used.

DROPOUT

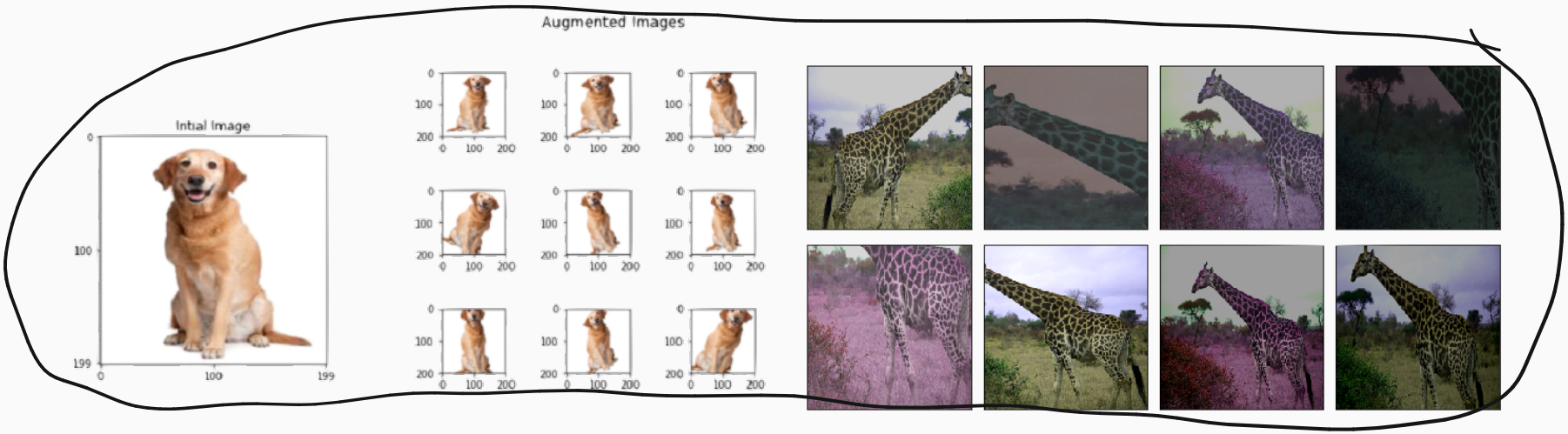
- Only used on fully connected layers.
- Simultaneously performs model regularization (model simplification) and model averaging.
- Has become less important in modern CNNs (convolutional neural nets) as the final fully connected layers become less important. But still a very helpful technique to know about!



DATA AUGMENTATION

Great general tool to know about. **Main idea:**

- More training data typically leads to a more accurate model.
- Artificially enlarge training data with simple transformations.



Take training images and randomly shift, flip, rotate, skew, darken, lighten, shift colors, etc. to create new training images. **Final classifier will be more robust to these transformations.**

ONE-SHOT LEARNING

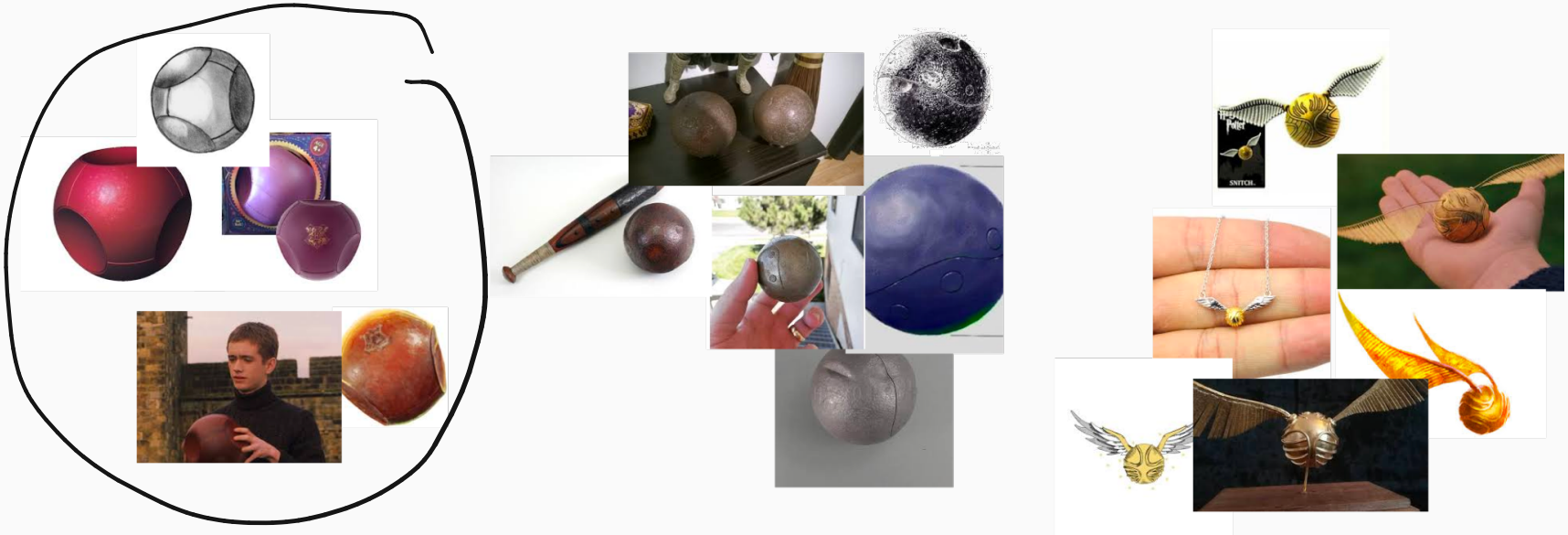
What if you want to apply deep convolutional networks to a problem where you do not have a lot of **labeled data** in the first place?

quaffle bludger snitch

Example: Classify images of different Quidditch balls.

ONE-SHOT LEARNING

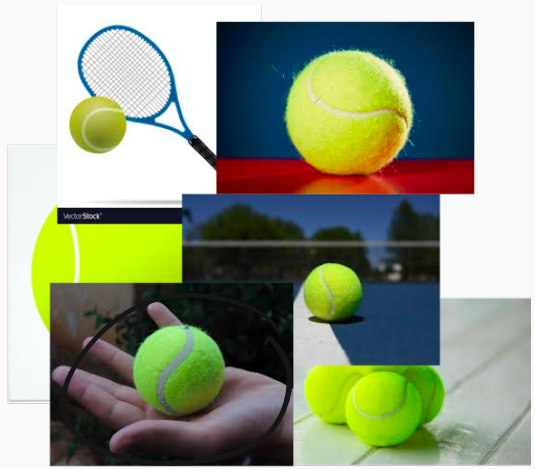
A human could probably achieve near perfect classification accuracy even given access to a **single labeled example** from each class:



Major question in ML: How? Can we design ML algorithms which can do the same?

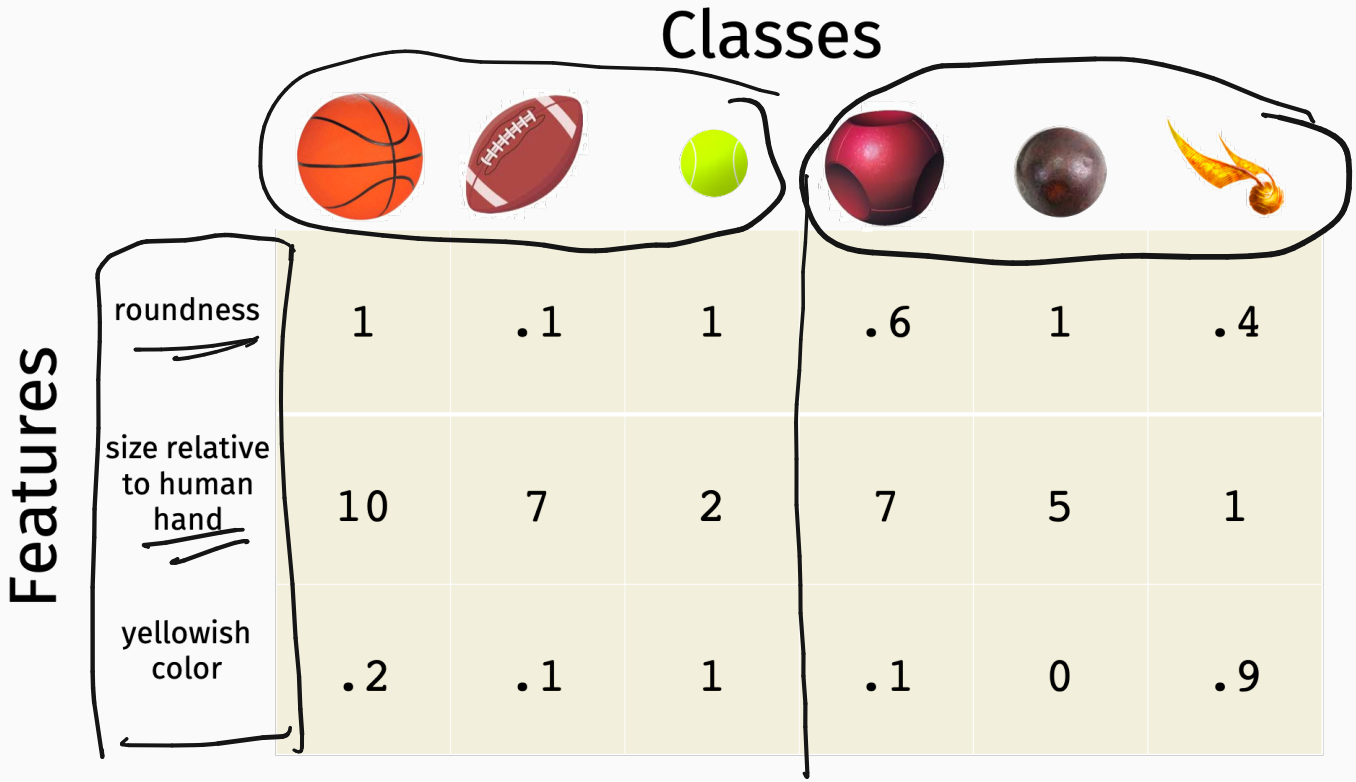
TRANSFER LEARNING

Transfer knowledge from one task we already know how to solve to another.



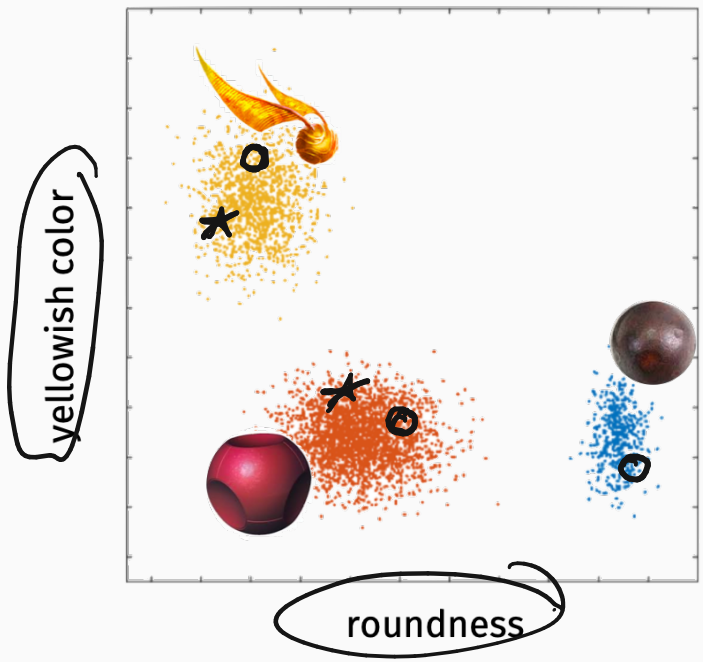
For example, we have learned from past experience that balls used in sports have consistent shapes, colors, and sizes. These features can be used to distinguish balls of different type.

Examples of possible high-level features a human would learn:



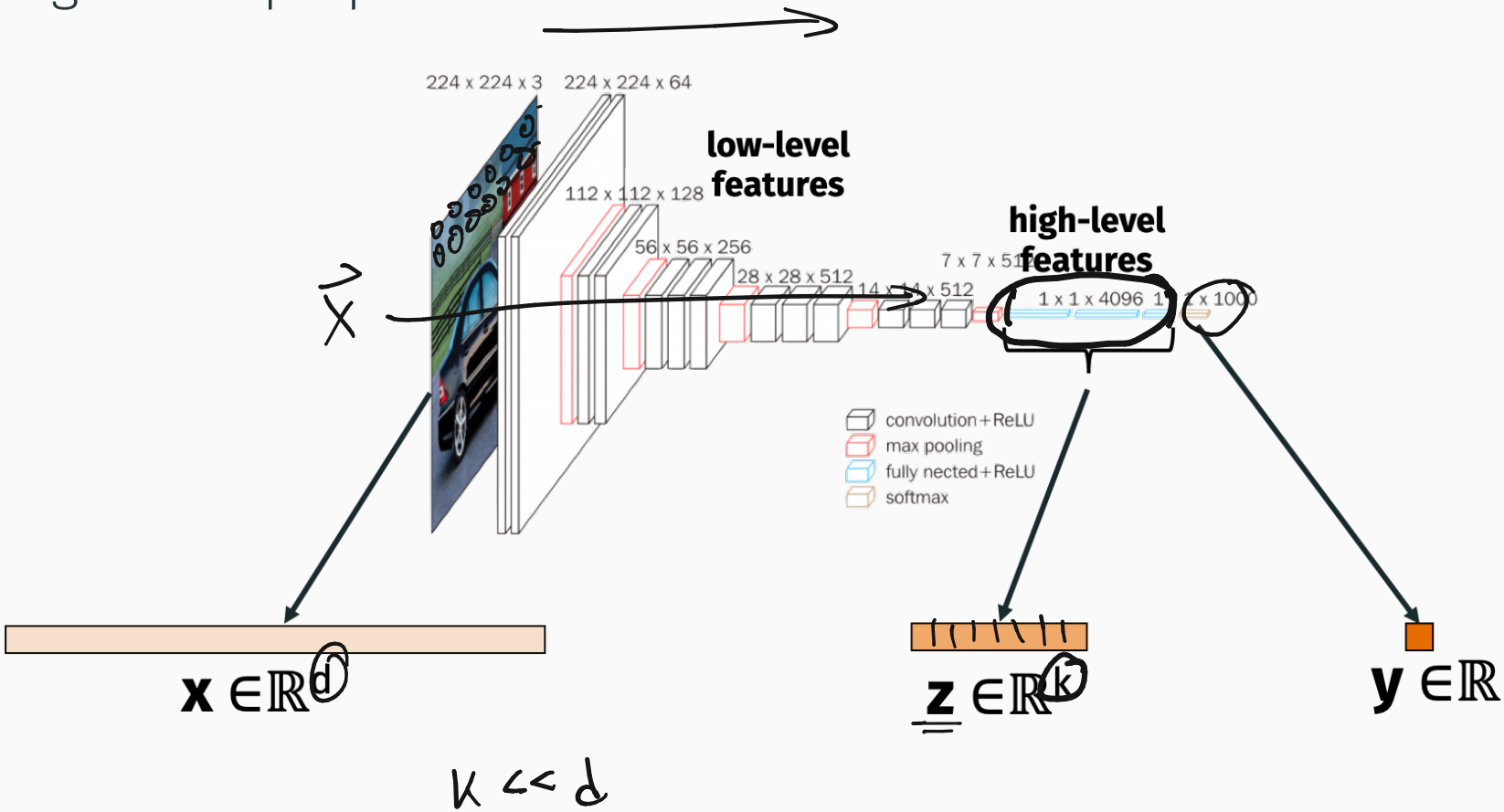
FEATURE LEARNING

If these features are highly informative (i.e. lead to highly separable data) few training examples are needed to learn.



Might suffice to classify ball using nearest training example in feature space, even if just a handful of training examples.

Empirical observation: Features learned when training models like deep neural nets seem to capture exactly these sorts of high-level properties.



Even if we can't put into words what each feature in $\vec{z}$ means...

TRANSFER LEARNING

This is now a common technique in computer vision:

1. Download network trained on large image classification dataset (e.g. Imagenet).
2. Extract features $\vec{z}$ for amy new image $\vec{x}$ by running it through the network up until layer before last.
3. Use these features in a simpler machine learning algorithm that requires less data (nearest neighbor, logistic regression, etc.).

Train using few examples for new task.

This approach has even been used on the quidditch problem:
`github.com/thatbrguy/Object-Detection-Quidditch`

UNSUPERVISED FEATURE LEARNING

Transfer learning: Lots of labeled data for one problem makes up for little labeled data for another.

What if we don't even have much labeled data for irrelevant classes?

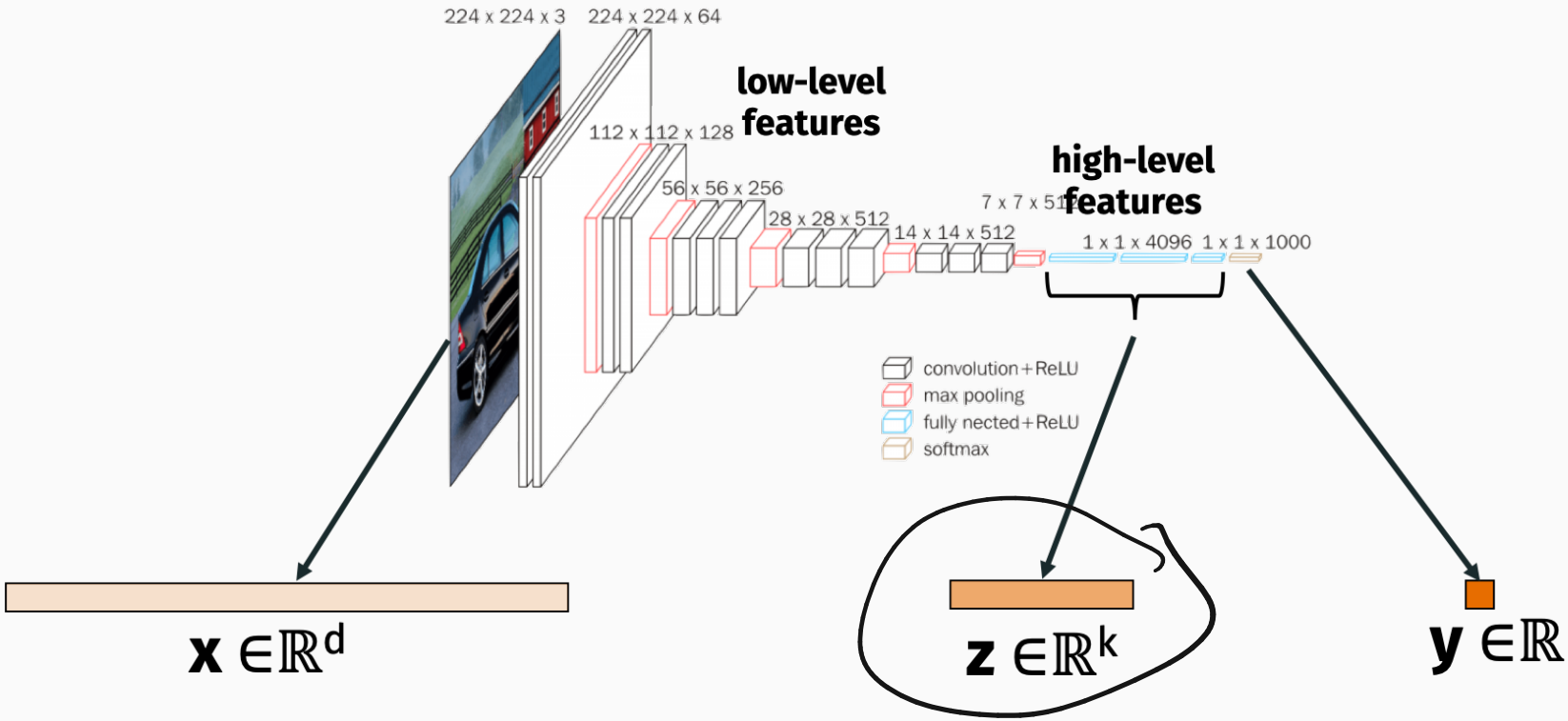
How to extract features in a data-driven way from unlabeled data is one of the central problems in **unsupervised learning**.

Unsupervised and semi-supervised learning will be the main topics of the next 2 weeks.

SUPERVISED VS. UNSUPERVISED LEARNING

- **Supervised learning:** All input data examples come with targets/labels. What machines are good at now.
- **Unsupervised learning:** No input data examples come with targets/labels. Interesting problems to solve include clustering, anomaly detection, semantic embedding, etc.
- **Semi-supervised learning:** Some (typically very few) input data examples come with targets/labels. What human babies are really good at, and we would love to make machines better at.

Back to the problem at hand: Want to extract meaningful features from an already trained neural network.



AUTOENCODER

Simple but clever idea: If we have inputs $\vec{x}_1, \ldots, \vec{x}_n \in \mathbb{R}^d$ but no targets $y_1, \ldots, y_n$ to learn, just make the inputs the targets.

- Let $f_{\vec{\theta}} : \mathbb{R}^d \to \mathbb{R}^d$ be our model.
- Let L be a loss function. E.g. squared loss: $L_{\vec{\theta}}(\vec{x}) = \|\vec{x} - f_{\vec{\theta}}(\vec{x})\|_2^2$.
- Train model: $\vec{\theta}^* = \min_{\vec{\theta}} \sum_{i=1}^{n} L_{\vec{\theta}}(\vec{x})$.

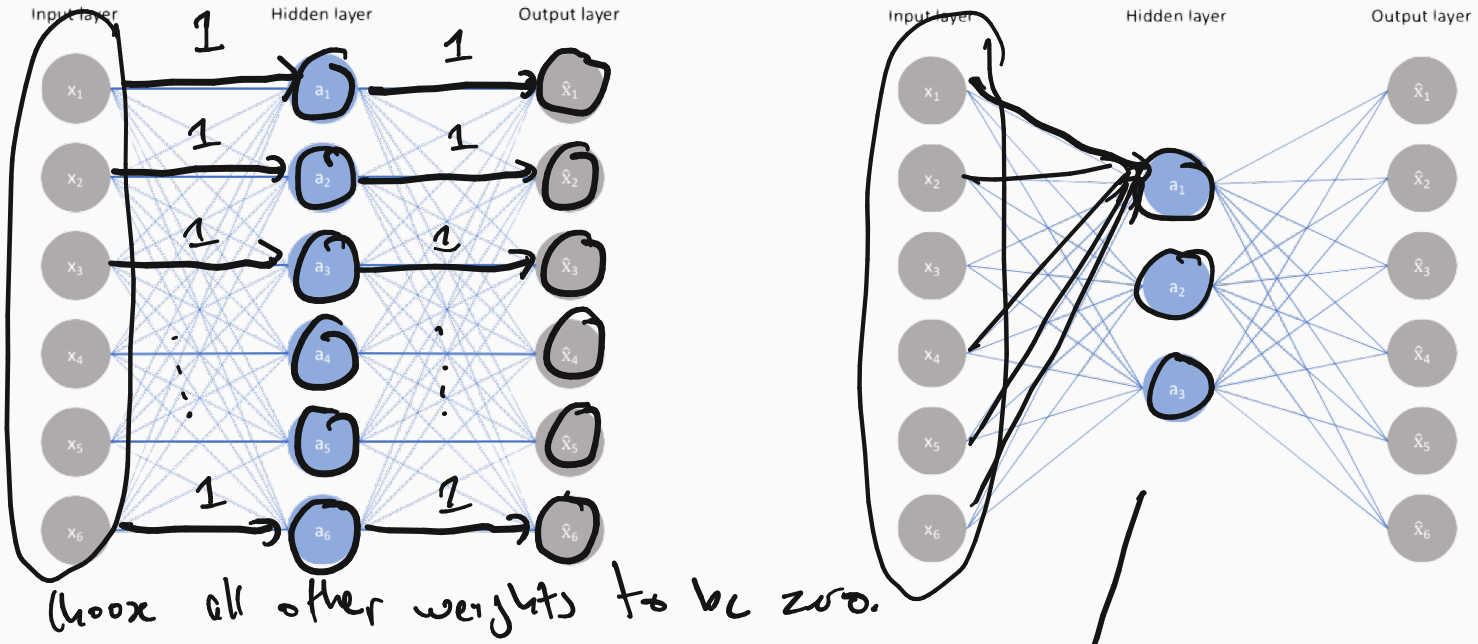
$d = 128 \times 128 \times 3$

$= \sum_{i=1}^{n} \|\vec{x} - f_{\vec{\theta}}(\vec{x})\|_2^2$

If $f_{\vec{\theta}}$ is a model that incorporates feature learning, hopefully these features will capture high-level meaning.

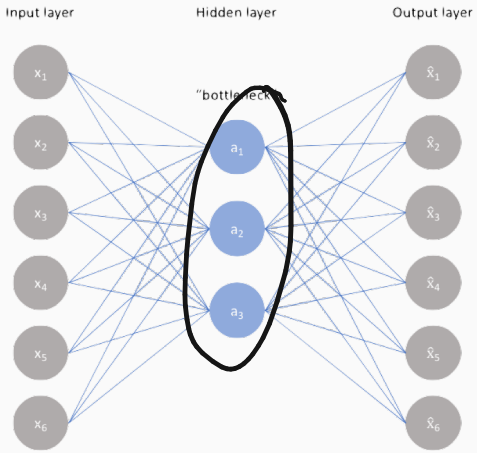
$f_{\vec{\theta}}$ is called an autoencoder. It maps inputs space to inputs space.

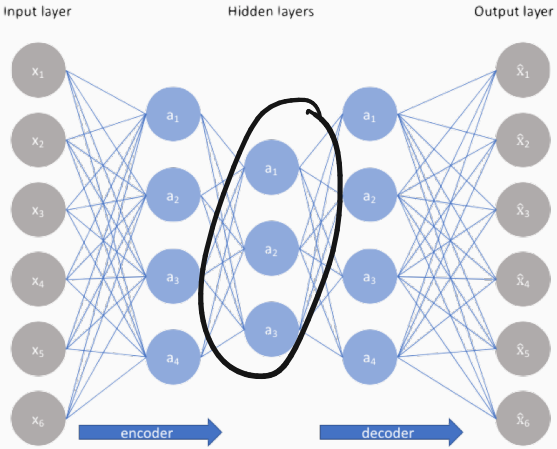
Two examples of autoencoder architectures:

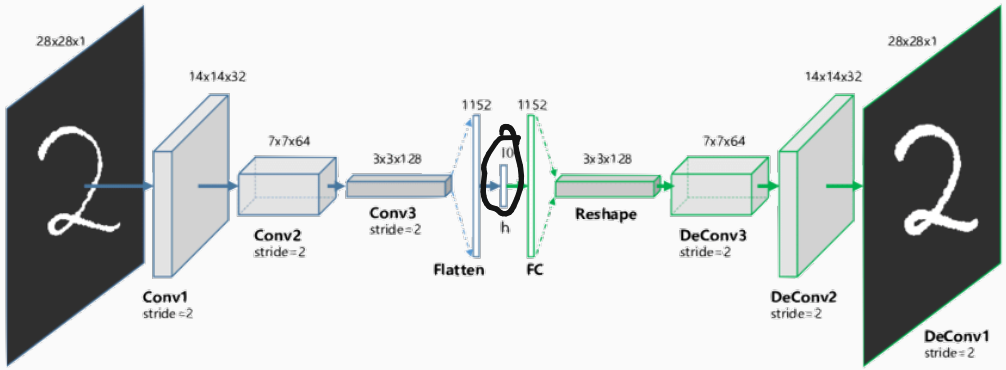


AUTOENCODER

Important property of autoencoders: no matter what architecture is use, there must always be a **bottleneck** with fewer parameters than the input. The bottleneck ensures information is “distilled” from low-level features to high-level features.







Architecture typically split into two parts:

Encoder: $e : \mathbb{R}^d \to \mathbb{R}^k$

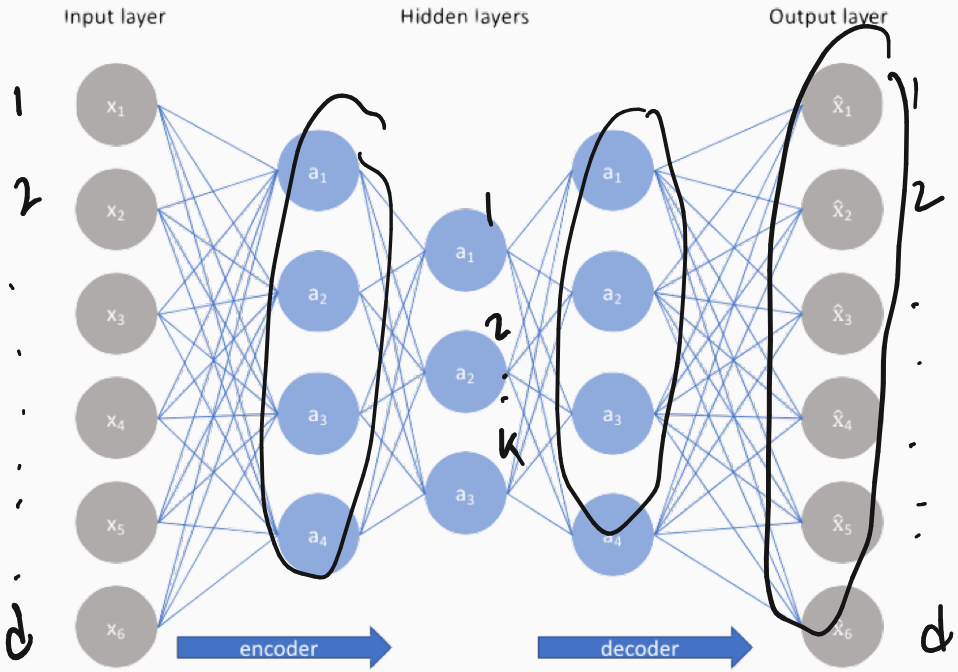
Decoder: $d : \mathbb{R}^k \to \mathbb{R}^d$

$k \ll d$

$f : \mathbb{R}^d \to \mathbb{R}^d$

express f using e, d

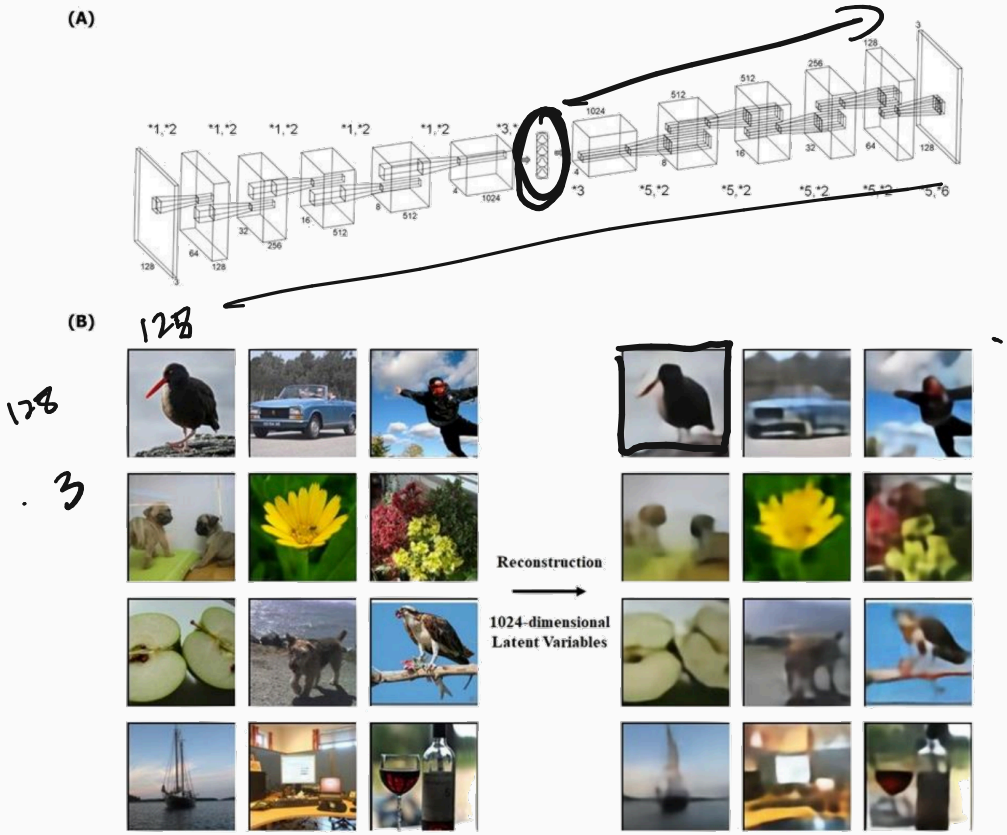
$$f(\vec{x}) = d(e(\vec{x}))$$



Often symmetric, but does not have to be.

AUTOENCODER RECONSTRUCTION

Example image reconstructions from autoencoder:



https://www.biorxiv.org/content/10.1101/214247v1.full.pdf

Input parameters: $d = 49152$.

Bottleneck "latent" parameters: $k = 1024$.

AUTOENCODERS FOR FEATURE EXTRACTION

At least for now, the best autoencoders do not work as well as for feature extraction as supervised methods. But, they have many other applications.

- Image segmentation.
- Learned image compression.
- Denoising and in-painting.
- Image synthesis.