Lecture 21: Recommendation systems

Data C182 (Fall 2024). Week 13. Tuesday Nov 19th, 2024

Speaker: Eric Kim

Announcements

- HW03 ("Transformers + NLP") out! Due: Fri Nov 22nd 11:59 PM PST
- Eric's Wed Nov 20th 3pm-4pm office hours will be held in-person at 110 Warren Hall! [link]
  - Come say hi! :)

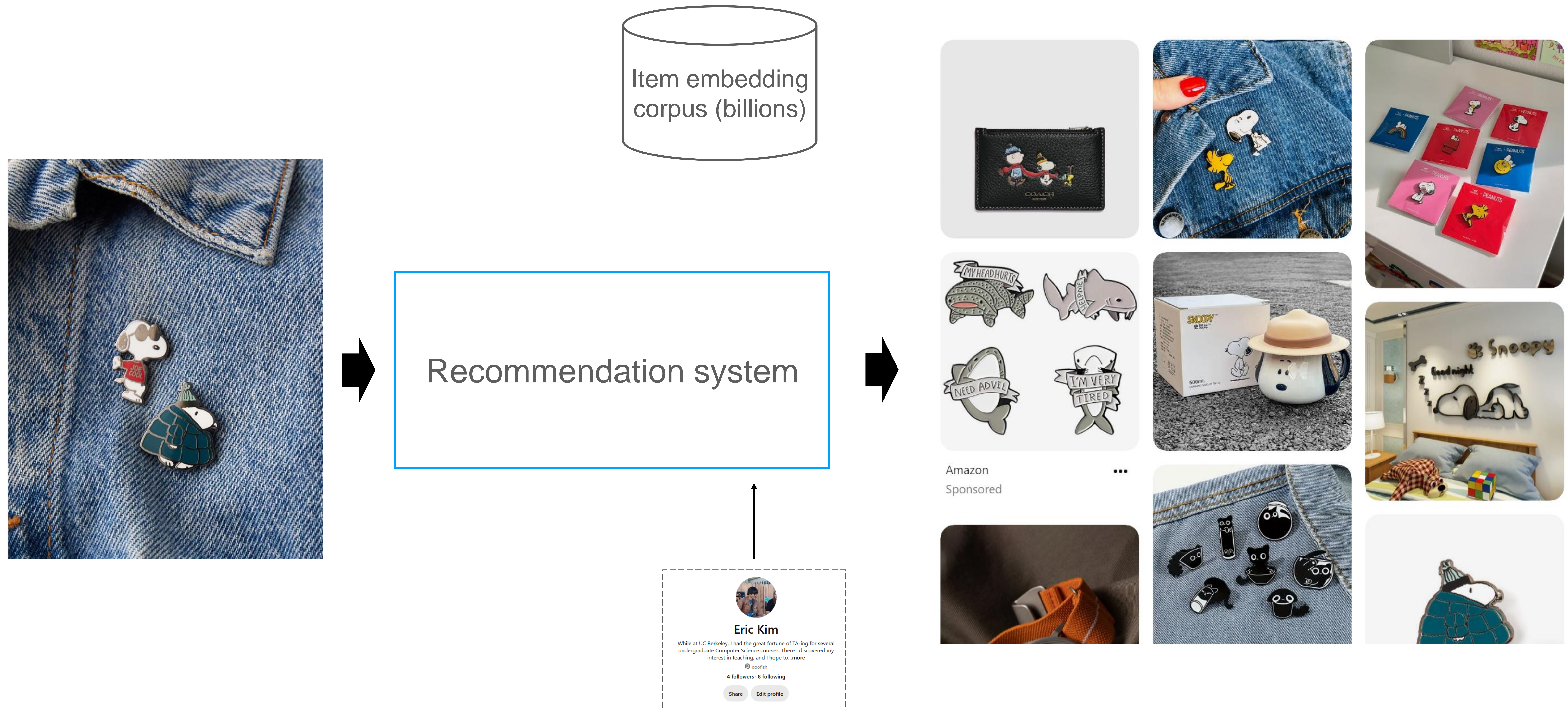
Today's lecture

- (Part 1) (finish off GPU slides from Lecture 19)
- (Part 2) Recommendation systems
  - ...and how do DNNs fit?

What is a recommendation system?

- Given a corpus of items (videos, websites, songs, products, etc): recommend the User something that they will like
- Examples:
  - Google: item is website
  - YouTube/TikTok: items are videos
  - Amazon: items are other products (shopping)
  - Pinterest: items are other Pins

High level: recommendation system



Query Item (websites, videos, products, Pins, etc)

User metadata (for user personalization)

Retrieved Results

https://www.pinterest.com/pin/553802704115502612/

Task: Related Content

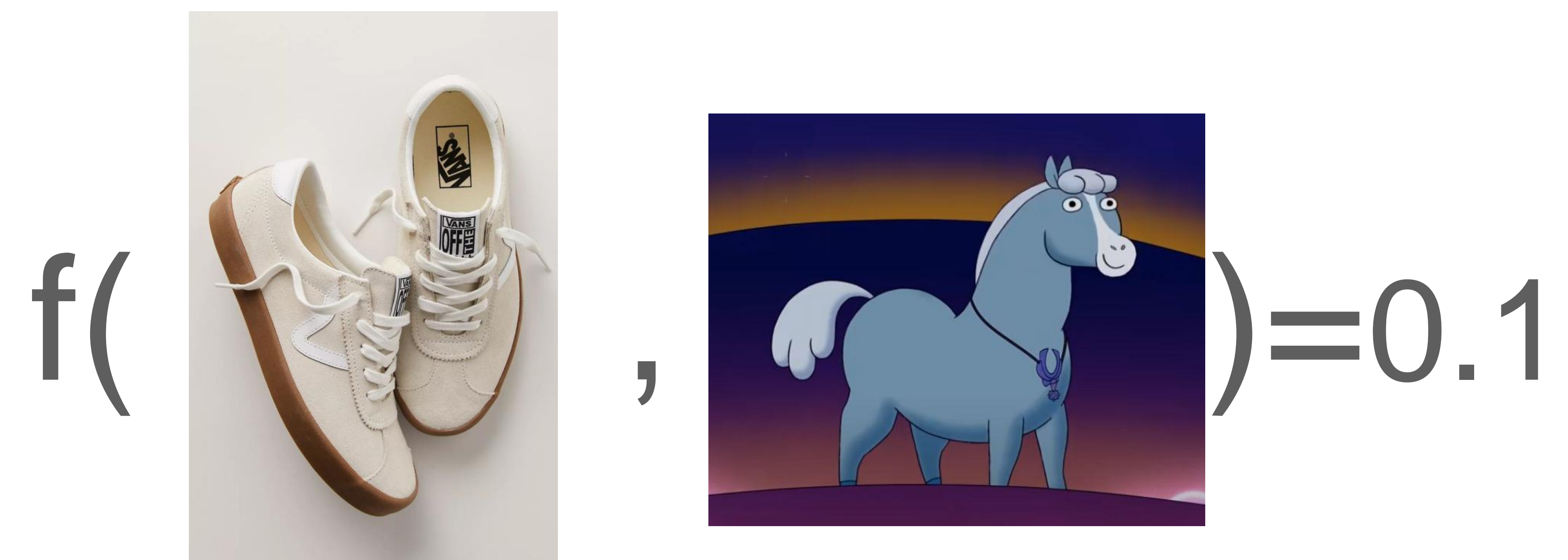
- Related content. Given a query item, recommend other related items
  - Goal: recommend content that the User is likely to engage with
  - Ex: if I'm shopping for white shoes, recommend me: other shoes from other brands, outfits that may go well with that shoe, etc.
  - (Optional) User personalization. How to recommend content that is tailored to a User's specific interest?

“Classic” recommendation system approaches

- PageRank (Google). Represent website inbound/outbound connections as a graph, and utilize graph theory to compute a quality score for each page.
  - Very neat application of spectral graph theory. Boils down to computing the largest eigenvalue of the graph’s adjacency matrix (can’t escape linear algebra!)
- Text-based information retrieval techniques
  - TF-IDF score, Bag-of-words models
- Collaborative filtering (aka matrix factorization)
- While older methods are still valid (and in use today at many companies), today we’ll focus on ML-based approaches

Scoring functions

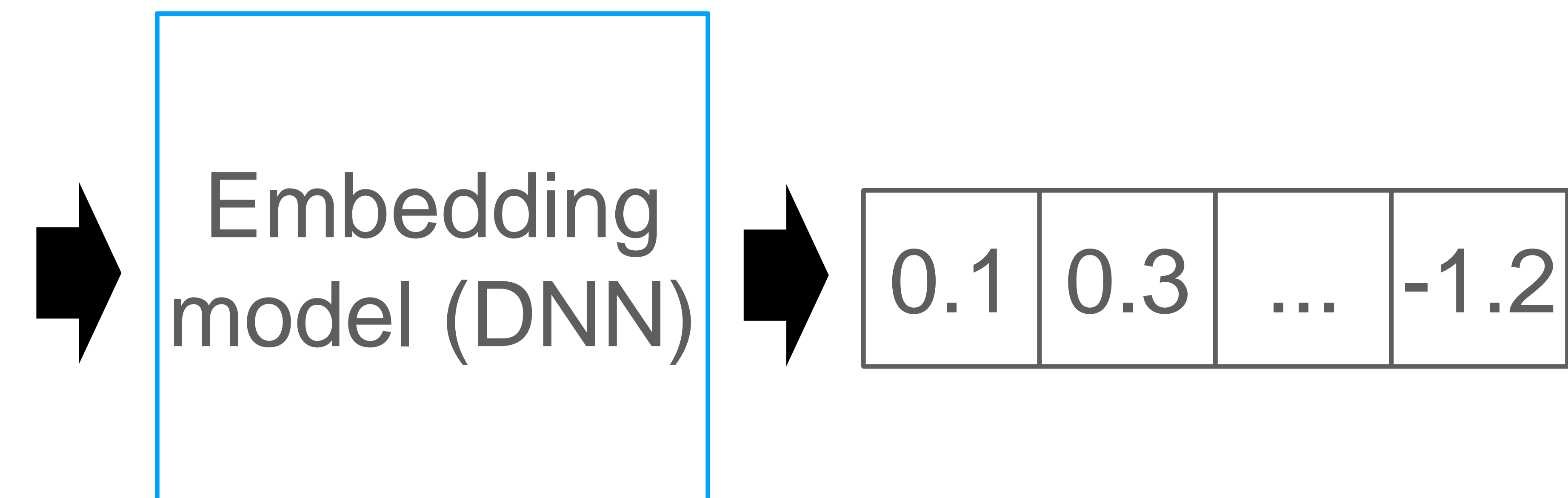
- Assume we have a **scoring function** that, given two items, outputs a similarity score between [0.0, 1.0]. 0.0 means “low similarity”, 1.0 means “high similarity”.
  - f(item_a, item_b) -> [0.0, 1.0]
- One simple recommendation system: given a query item, score all items, sort by score, and show top K=50 results to the User.
  - Sorted([f(item_query, item_candidate) for item_candidate in item_corpus])
- The game is: how to design a good scoring function `f()`?



Deep learning: aka representation learning

- In this course (Data C182), we've learned that deep learning boils down to learning strong (semantic) representations for downstream tasks (classification, object detection, text generation, etc).
- Idea: let's leverage DNNs to learn item embedding representations!
- Recall: an item embedding is a vector representation of some item, eg a Tensor with shape=[256]

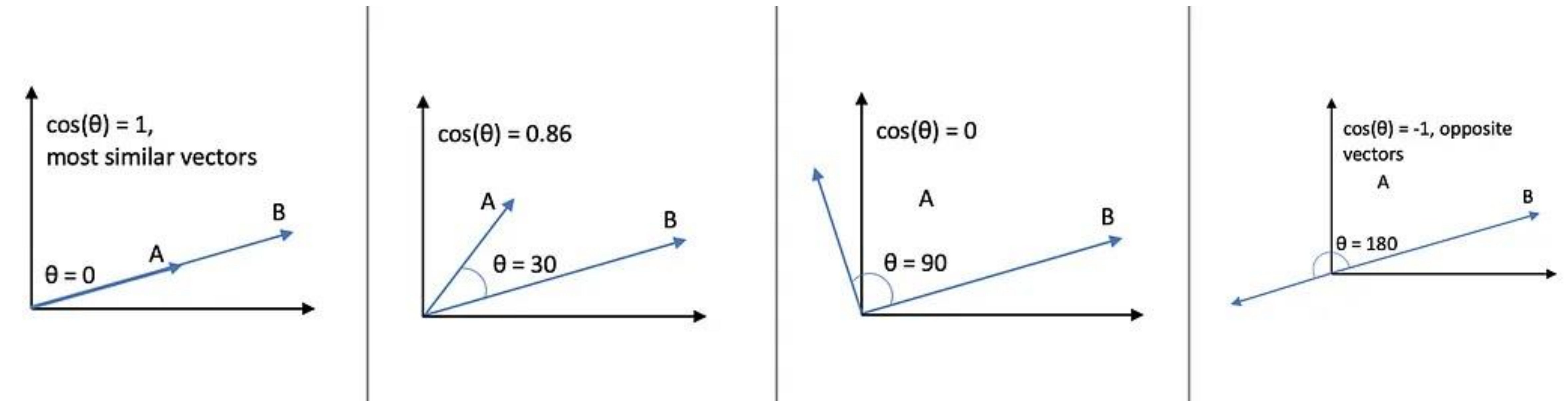
Item (ex: image, video, website, user, etc)



Item embedding ("representation")

Embedding metric spaces

- Popular approach: learn embeddings such that we can utilize simple similarity measures to easily compare two embeddings.
  - Ex: dot product, cosine similarity, L2
- So, we know what `f()` is (a simple similarity metric like cosine-similarity).
- Big question: how to learn the item embeddings?
  - In particular: how to learn such that embedding similarity metrics “works”?



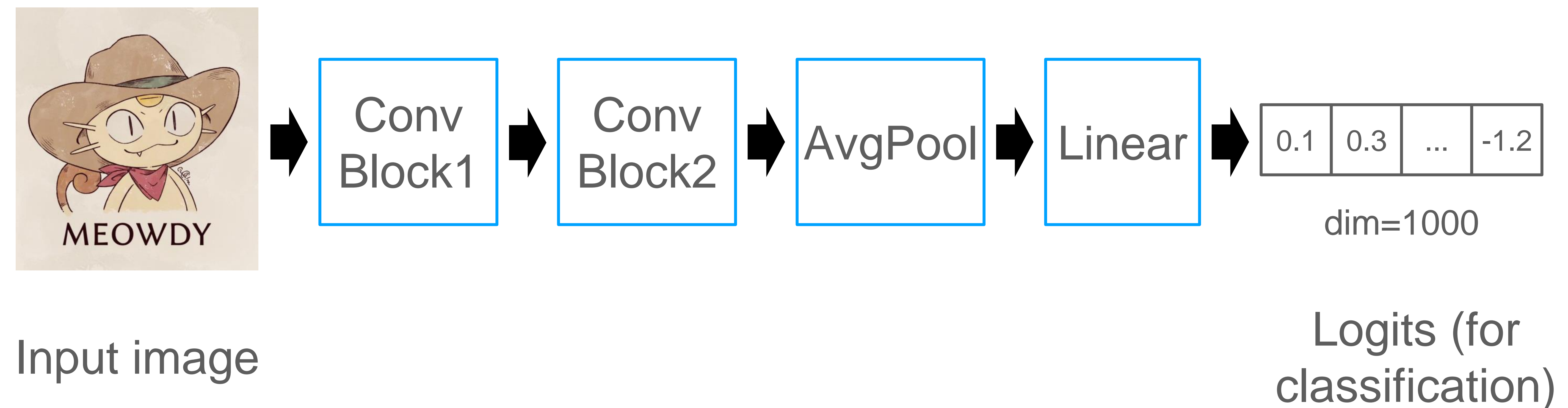
$$\cos(\theta) = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\|\|\mathbf{B}\|} = \frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} A_i^2}\sqrt{\sum_{i=1}^{n} B_i^2}}$$

Pictured: cosine similarity between vectors A, B

https://towardsdatascience.com/cosine-similarity-how-does-it-measure-the-similarity-maths-behind-and-usage-in-python-50ad30aad7db

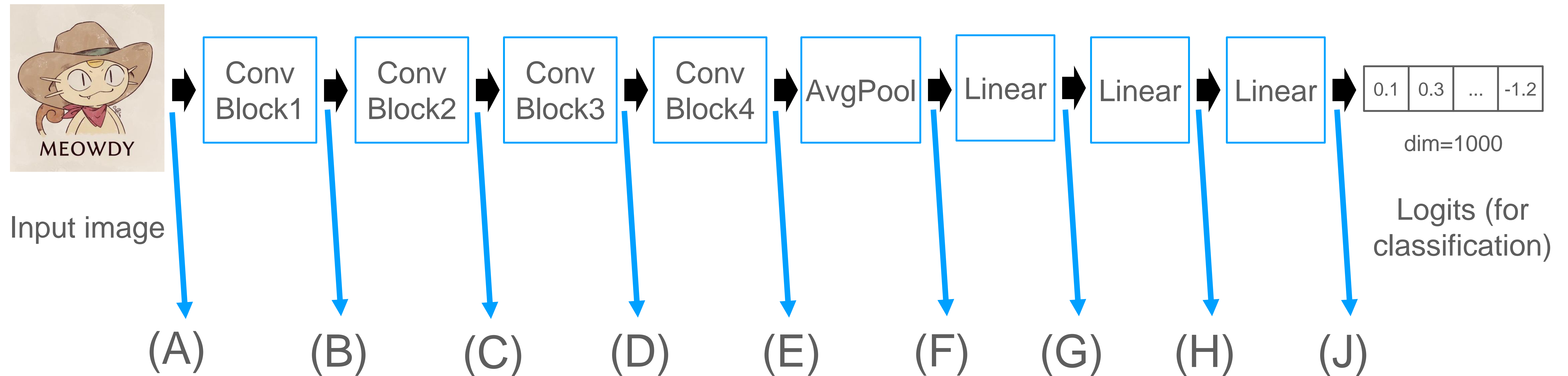
Designing embedding models

- Popular starting point: take an existing DNN model trained for some task, and use some intermediate feature as the embedding representation
- What is an "intermediate feature"?
    - Aka some intermediate activation map, say the output of some Linear/Conv2d/etc.
- Design question: which intermediate feature to use?



Designing embedding models

Suppose this model was trained on ImageNet-1k classification



Question: which of these choices might be the best for an embedding representation?

Answer: in my opinion G or H is best.
(A): using raw pixel values will not work well: too high-dimensional and has poor semantics
(B-D) Features are too low-level, eg edges.
(E) Better than (B-D), but likely too high-dimensional.
(F) Reasonable choice: has good semantics and has low-enough dimensionality to be useful as an embedding vector. But better options exist
(G,H) high semantics, and nice benefit is that it’s easy to explicitly define a target embedding size.
(J) Logits are too specialized for the classification task (eg ImageNet-1k), and likely throws away too much semantic “general purpose” information that would be useful for downstream tasks (like image similarity search).

Embedding visualization

- Heuristic: to check if your model is indeed learning a “healthy” embedding metric space, try clustering the embeddings + visualize them!
- How to project a high-dimensional embedding (eg 256-dim) to 2D? Lots of ways to do this
- Approach 1: PCA dimensionality reduction
- Approach 2: t-SNE (pictured applied to image embeddings) [link]

https://arxiv.org/abs/1702.04680

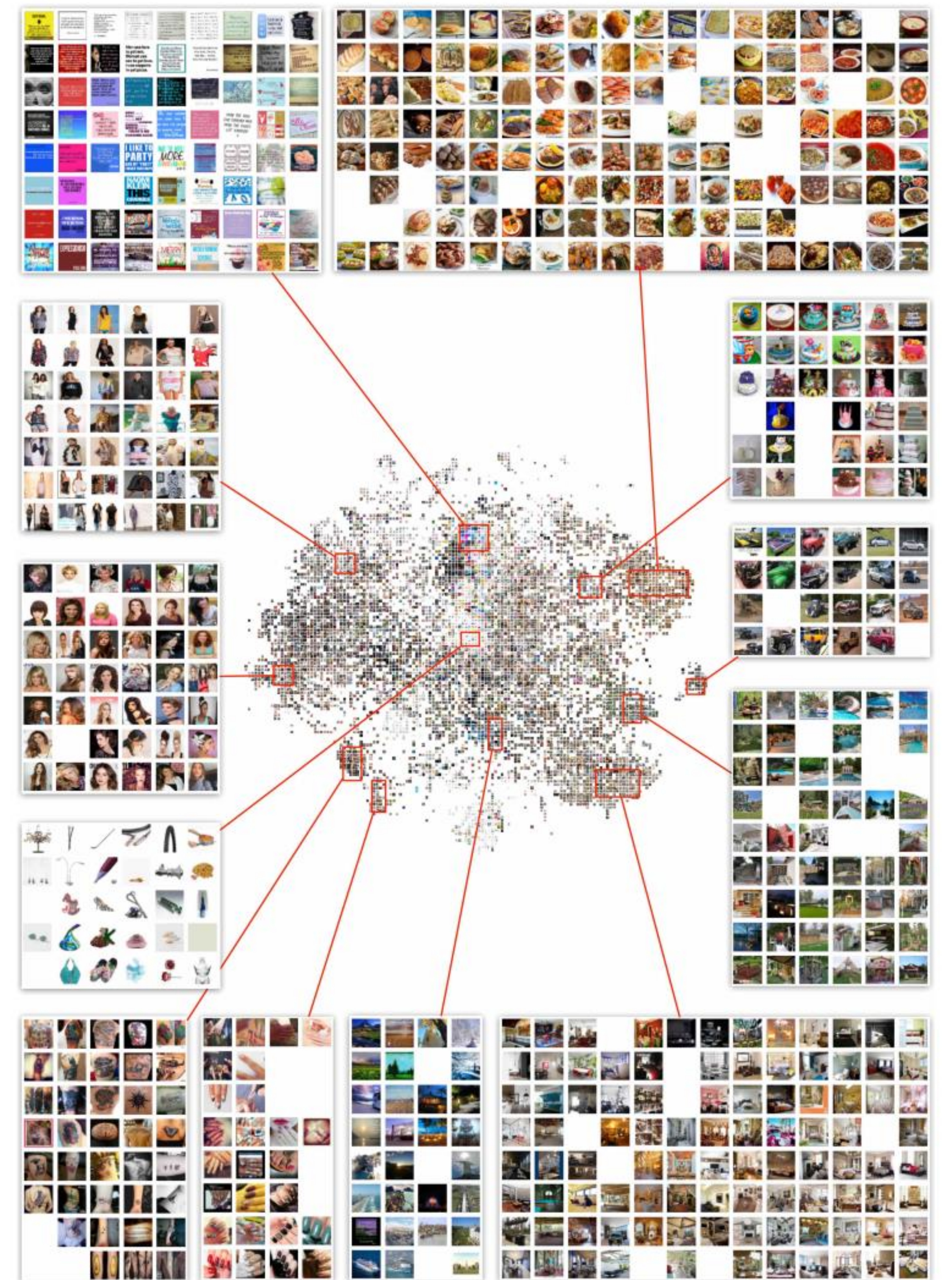
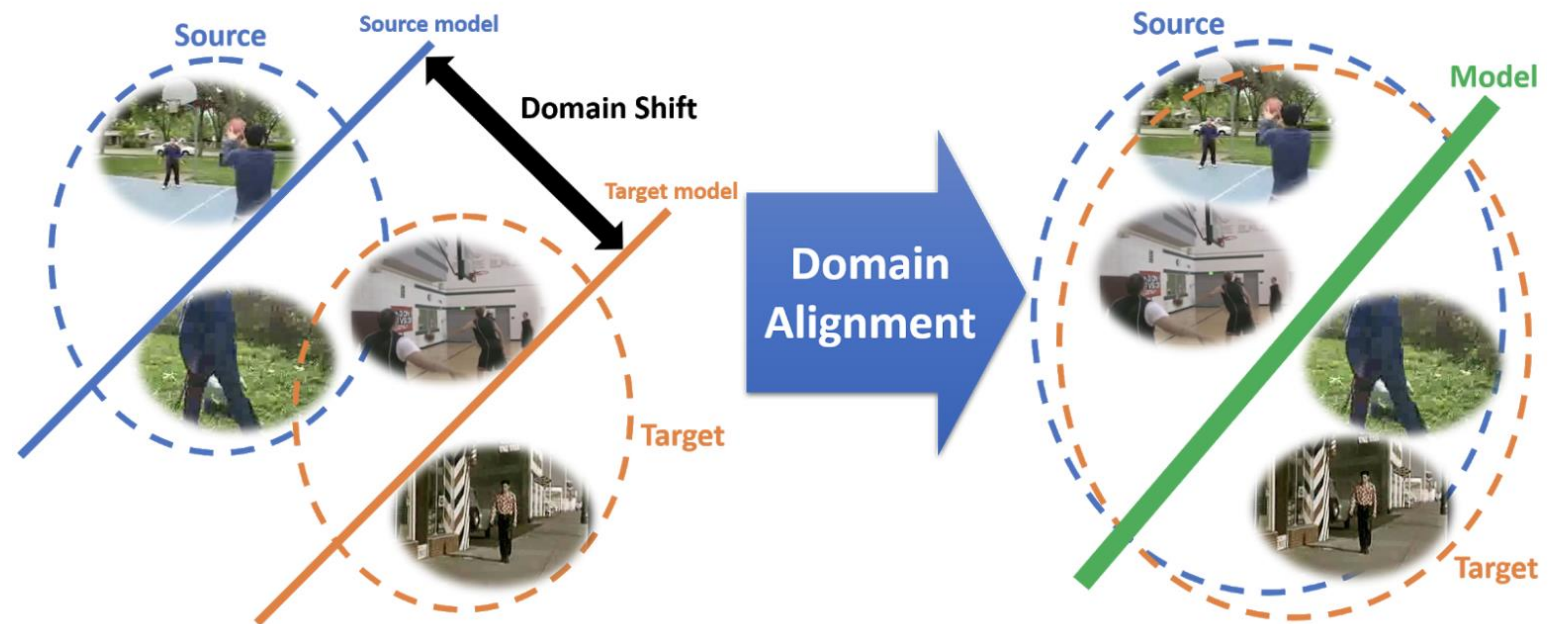


Figure 6: Visualization of binarized embeddings of Pinterest images extracted from fine-tuned VGG16 FC6 layer.

Domain shift

- In practice, we take a pre-trained model (eg image classifier trained on ImageNet-1k), and do another training run (“fine-tuning”) on our internal dataset (eg Pinterest/Instagram images).
- Reason: target images (eg Pinterest/Instagram) often have different characteristics than what the pre-trained model has seen (eg ImageNet-1k)
  - Finetuned embeddings usually perform much better than pretrained embeddings!
- In ML jargon, called **“domain shift”**

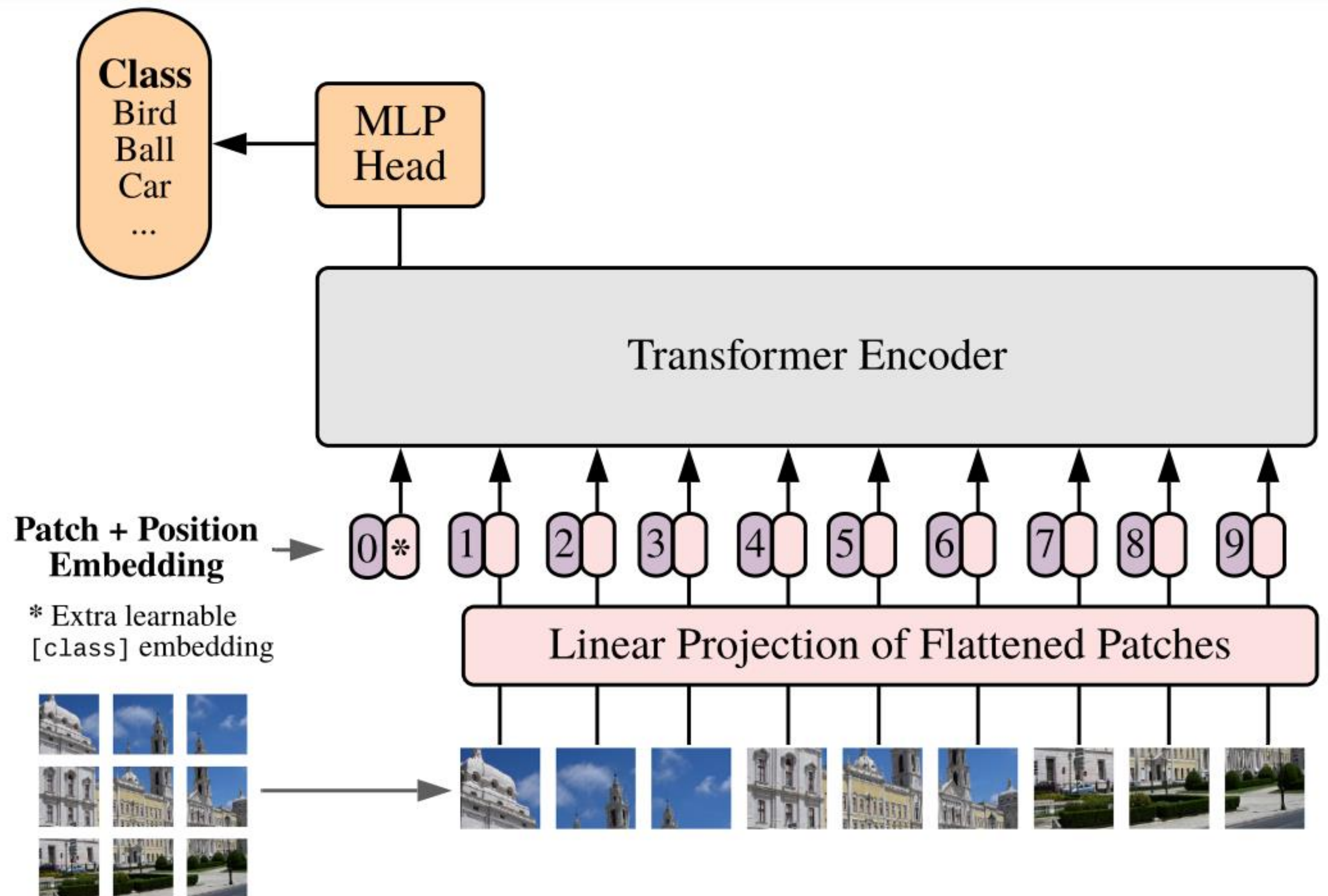


Example:
Source domain: ImageNet-1k
Target domain: Pinterest/Instagram images

https://mlatgt.blog/2019/09/10/overcoming-large-scale-annotation-requirements-for-understanding-videos-in-the-wild/

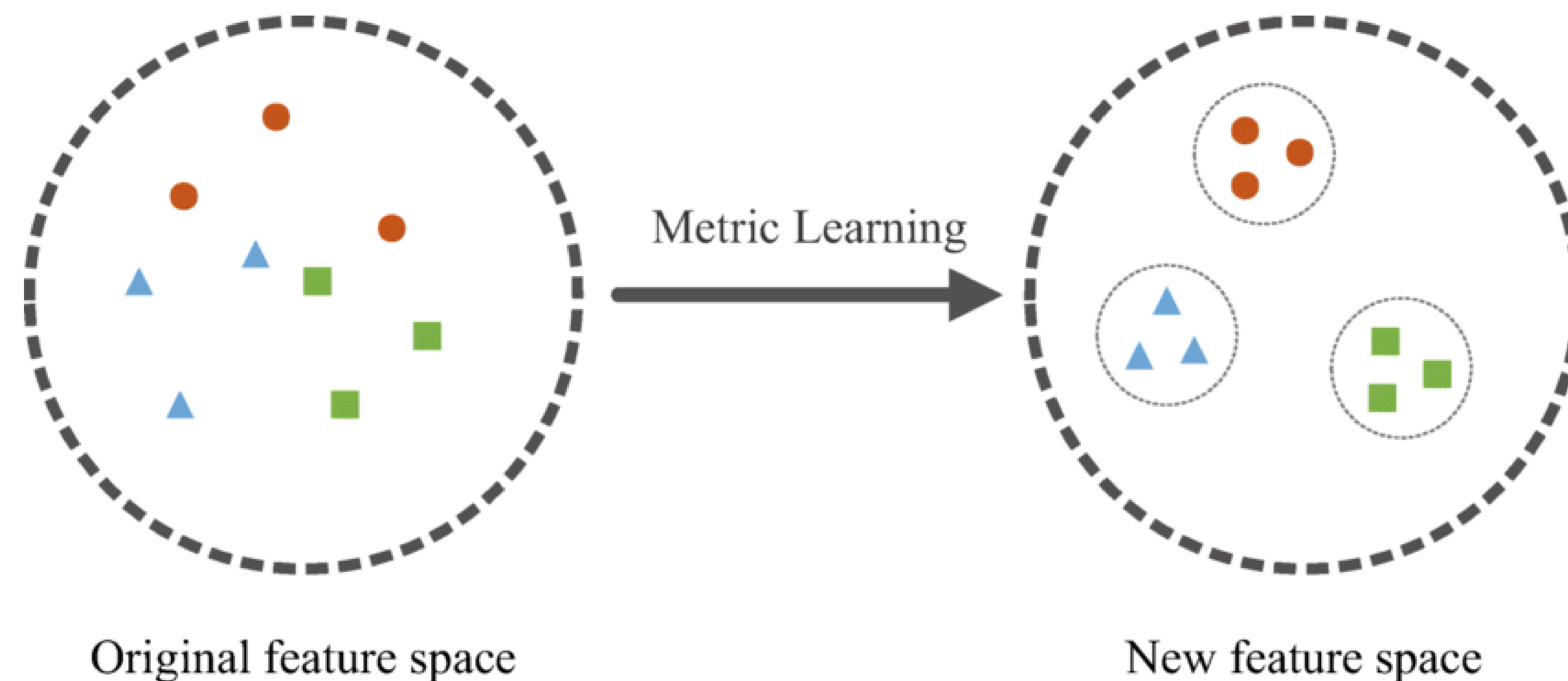
Image embeddings: ViT

- **Question**: suppose we want to use a Visual Transformer (ViT) to compute image embeddings. What should we use as the image embedding?
- **Answer**: the encoder's output embedding for the CLS token. Or: some intermediate embedding in "MLP Head" (if it has multiple Linear layers).



Pretrained models and metrics

- Funny enough: in practice, using image embeddings from pretrained image classification models works quite well even though there's no "metric learning" going on
    - Training loss is image classification, not anything "metric-y/distance-y"
- Idea: can we directly optimize for learning a good embedding that "behaves well" for some metric (eg cosine similarity)?

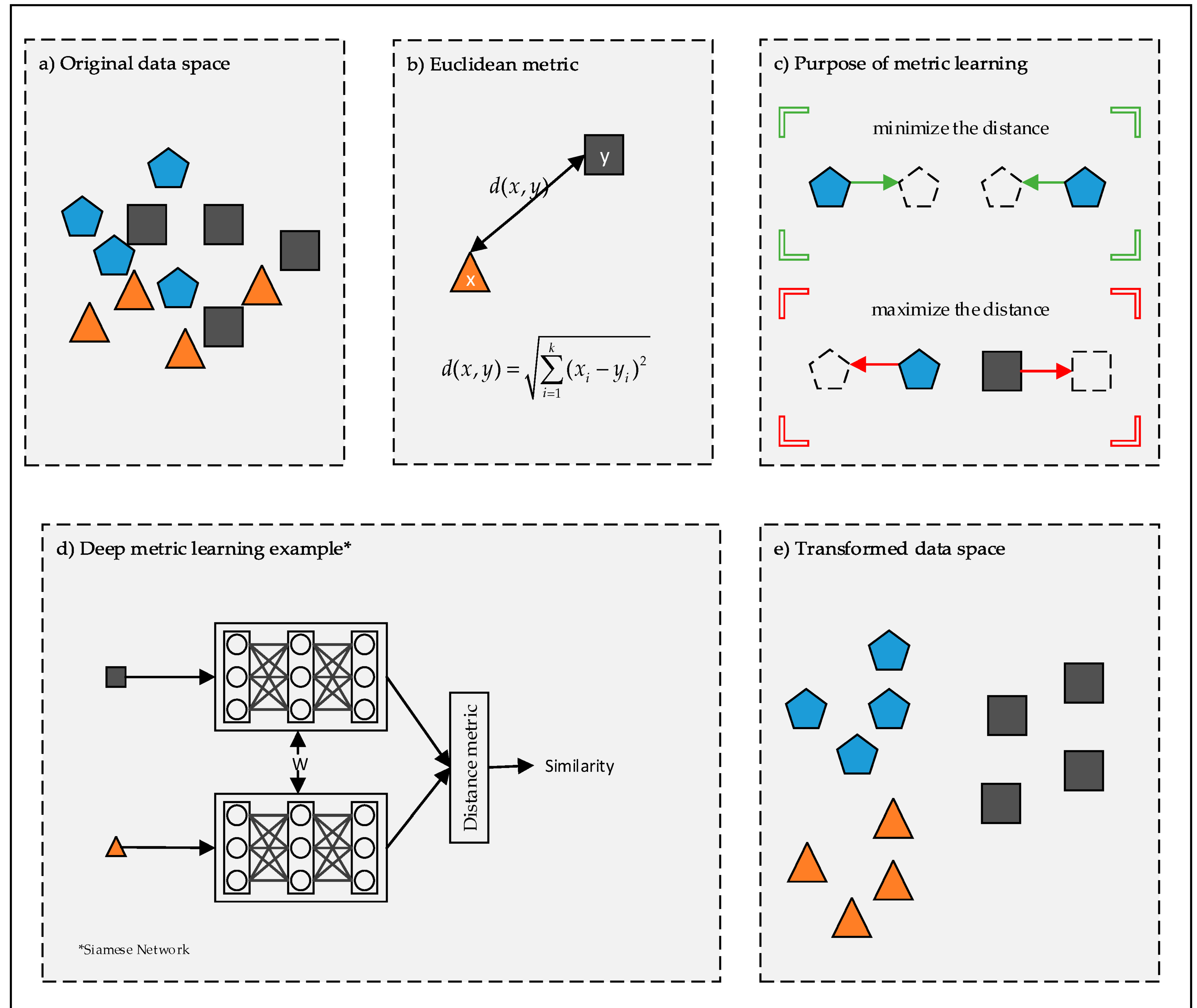


https://www.researchgate.net/figure/Working-mechanism-of-metric-learning-The-metric-learning-method-aims-to-find-a_fig3_355201453

Metric learning

- Metric learning: a well-studied problem in ML to learn a good feature representation where distance metrics "work well"
- "Deep" metric learning: train a DNN that learns a good embedding representation that works well with your desired distance metric (L2, cosine dist, etc)

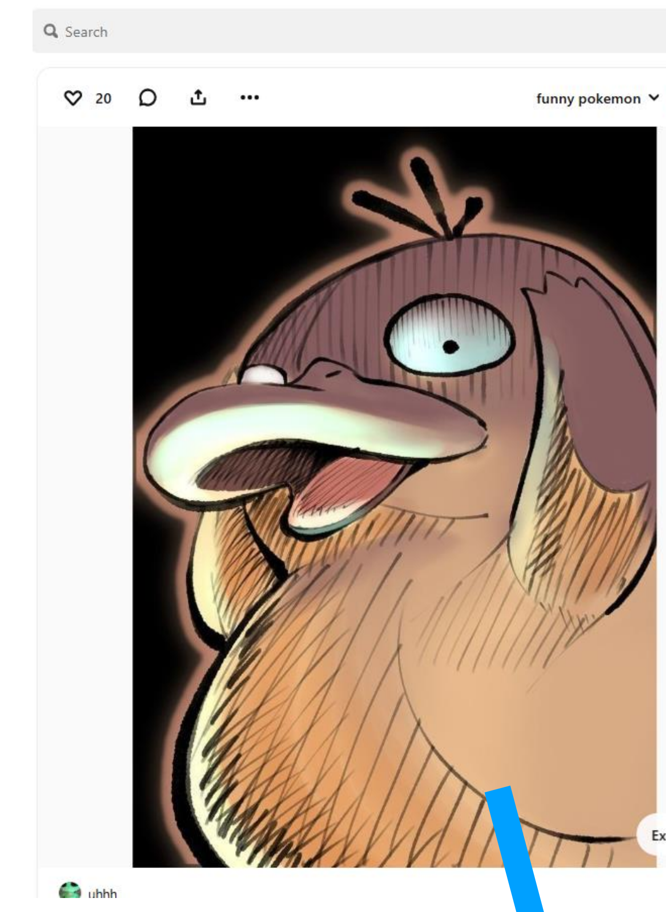
https://www.mdpi.com/2073-8994/11/9/1066



Dataset: triplets

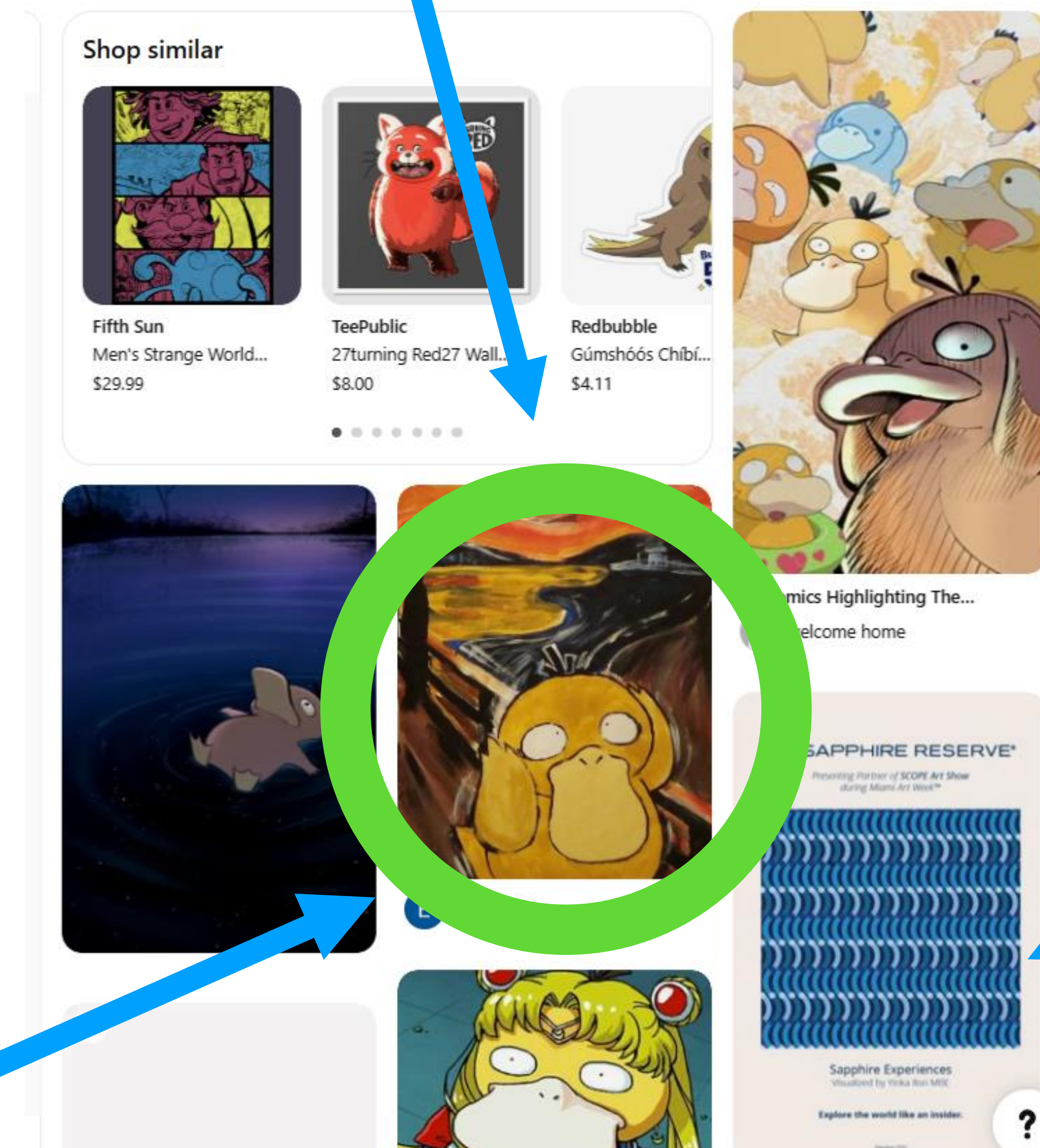
- Suppose we have a labeled dataset of (anchor, positive, negative)
- Example: user engagement logs.
  - **Anchor**: Query image/post/video that a User viewed
  - **Positive**: Next image/post/video the User clicked on next
  - **Negative**: An image/post/video that the User didn't click on
    - Or: random negatives works well in practice too

Query image (**Anchor**)



Engagement logs (some cloud DB): User "EricKim" viewed query image "anchor", and then clicked on this next image "positive"

Results

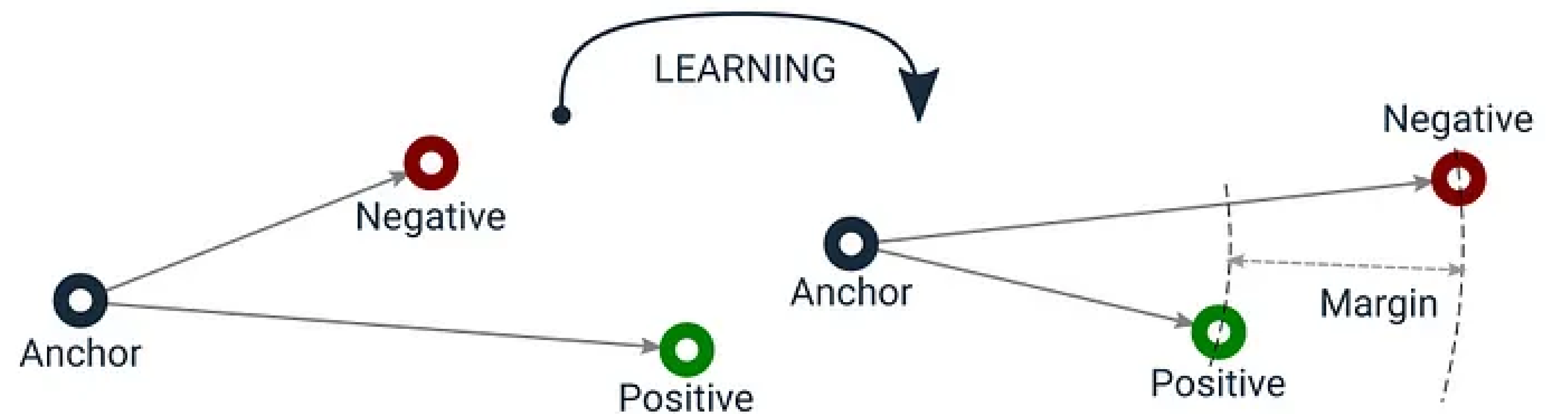


User clicked on this (**Positive**)

User did not click (**Negative**)

Metric learning: triplet loss

- Idea: design a training loss that pulls (anchor, positive) embeddings close to each other, and (anchor, negative) embeddings far away
- New loss! “Triplet loss”
- Pytorch: `torch.nn. TripletMarginWithDistanceLoss` [link]



The loss function for each sample in the mini-batch is:

$$L(a, p, n) = \max\{d(a_i, p_i) - d(a_i, n_i) + \text{margin}, 0\}$$

where

$$d(x_i, y_i) = \|\mathbf{x}_i - \mathbf{y}_i\|_p$$

This equation uses the Lp norm (eg L1, L2, etc) as the distance metric, but in principle you can use any metric like: cosine similarity, dot product, etc

Embedding retrieval at scale

- Armed with a good embedding model and an embedding metric, we're nearly there to a retrieval system!
- Algorithm: given query item, compute similarity between each query and all items in the corpus. Sort by similarity
  - Aka "nearest neighbor search"
- Problem: corpus can be very large (Billions!). Linear search is too slow: we want results in real time (eg <200ms latency)
- Solution: approximate distributed nearest neighbor!

f(,)=0.91

f(,)=0.1

Approximate nearest neighbor

- Idea: rather than compute “exact” nearest neighbor (too slow), compute approximate results (faster)
- Tradeoff: speed vs fidelity
- Popular algorithms:
    - Locality-sensitive hashing (LSH)
    - HNSW [link]

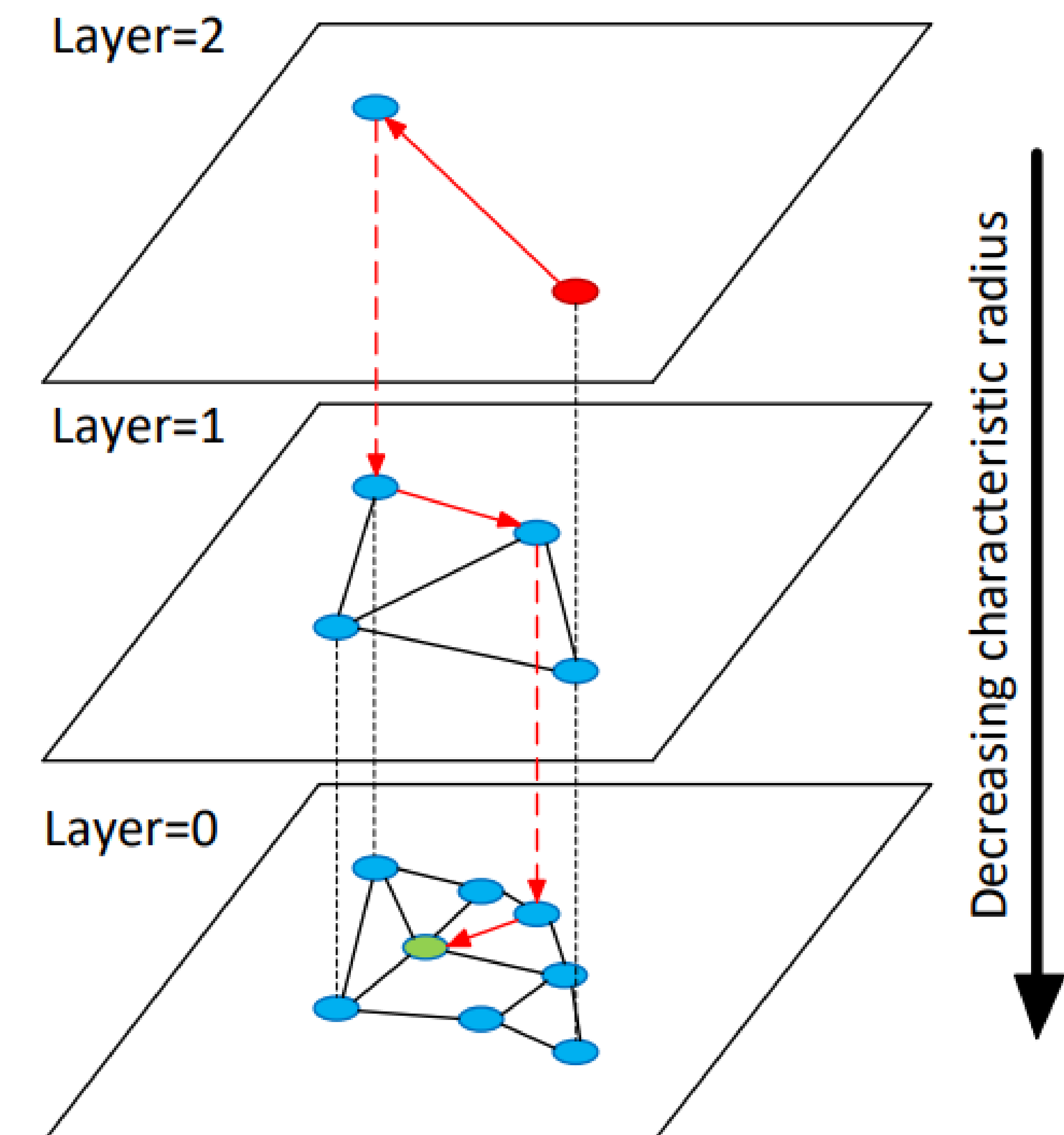


Fig. 1. *Illustration of the Hierarchical NSW idea.* The search starts from an element from the top layer (shown red). Red arrows show direction of the greedy algorithm from the entry point to the query (shown green).

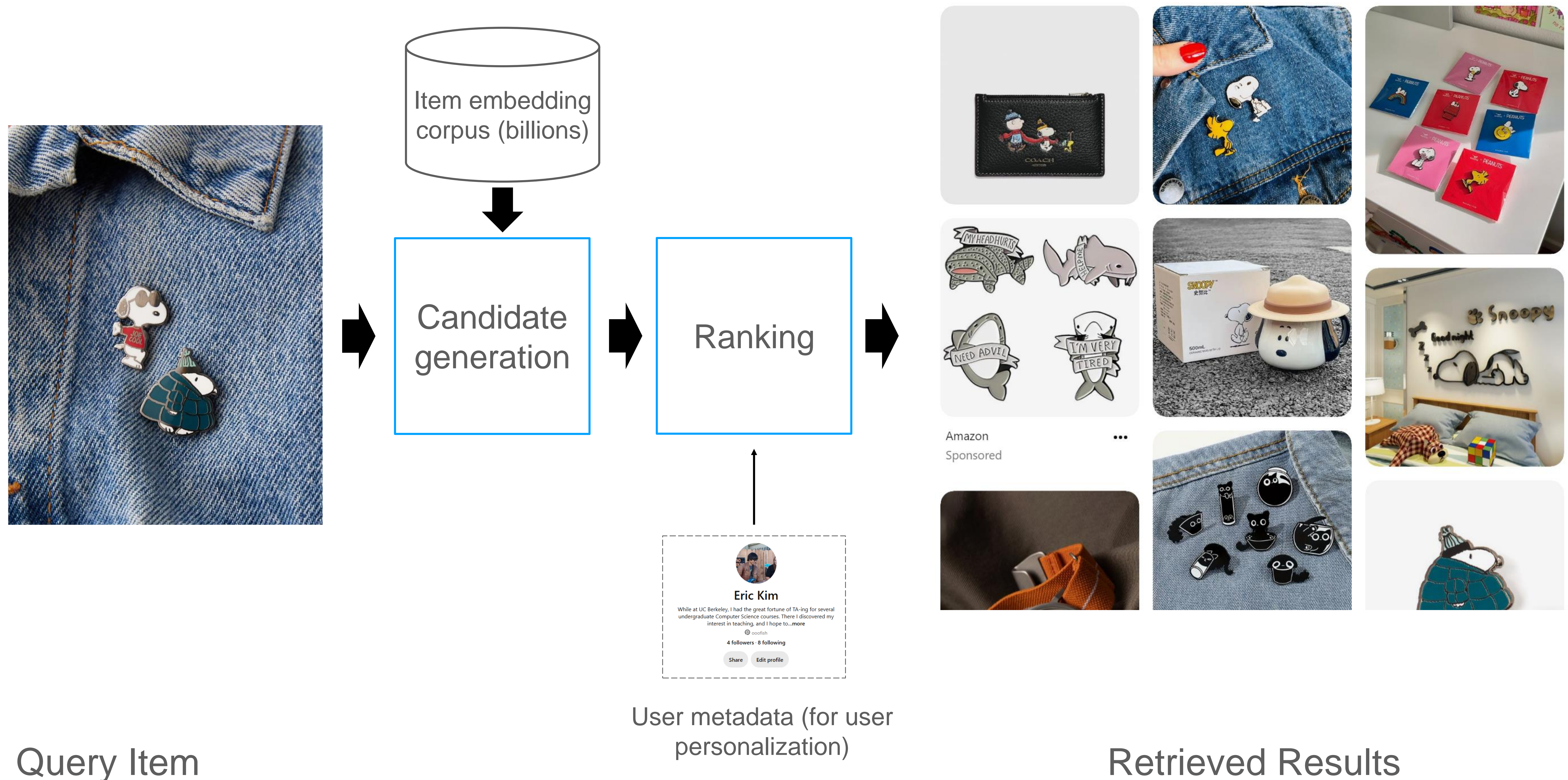
Candidate generation

- Jargon for: “initial lightweight retrieval”
- Goal: filter from Billions of corpus items down to hundreds.
- Popular choice: embedding model + ANN

Ranking stage

- Given hundred's of candidates from candidate generator: rerank them via a ML model
- Since we have fewer candidates (hundreds, instead of billions), we can use heavier-duty ML models
- Optimize for business metrics (ex: user clickthrough rate, ad impressions, etc)
- Can inject User personalization here too!

Putting it all together: a recommendation system



https://www.pinterest.com/pin/55380270411502612/

Putting it all together: a recommendation system

Candidate Generation

Note: must precompute item embeddings ahead of time, eg in an offline ML inference job, store to some fast key-value database (ex: rocksdb [link]).

Query Item

Item embedding corpus (billions)

Compute Embedding

Query embedding

0.1 | 0.3 | ... | -1.2

Approx. Nearest Neighbor

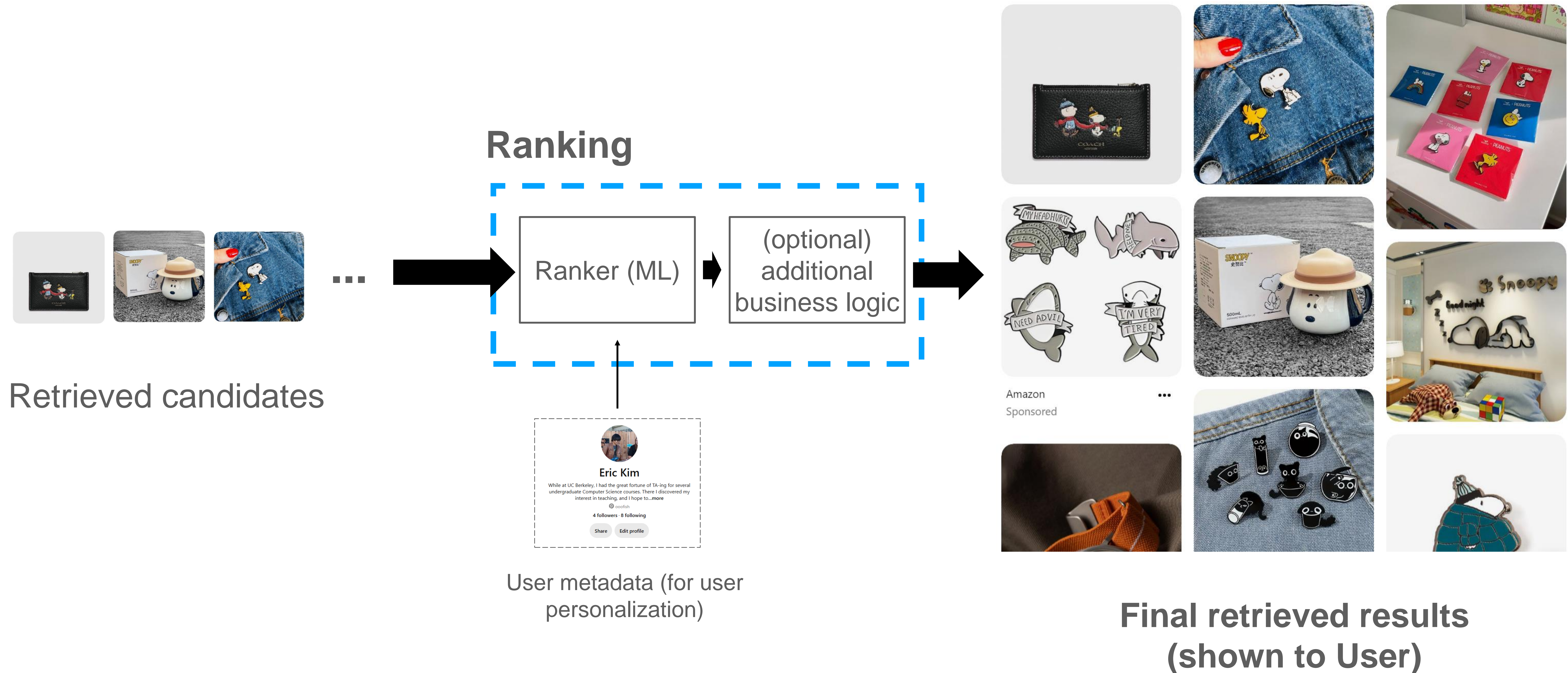
Embedding-based retrieval

0.85 0.84 0.81 ... 0.04 0.02

dist(query, candidate)

Retrieved candidates ("lightweight scoring")

Putting it all together: a recommendation system



To learn more about a real-world retrieval system, see: "Related Pins at Pinterest: The Evolution of a Real-World Recommender System" [link]