# Detecting image similarity using Spark, LSH and TensorFlow

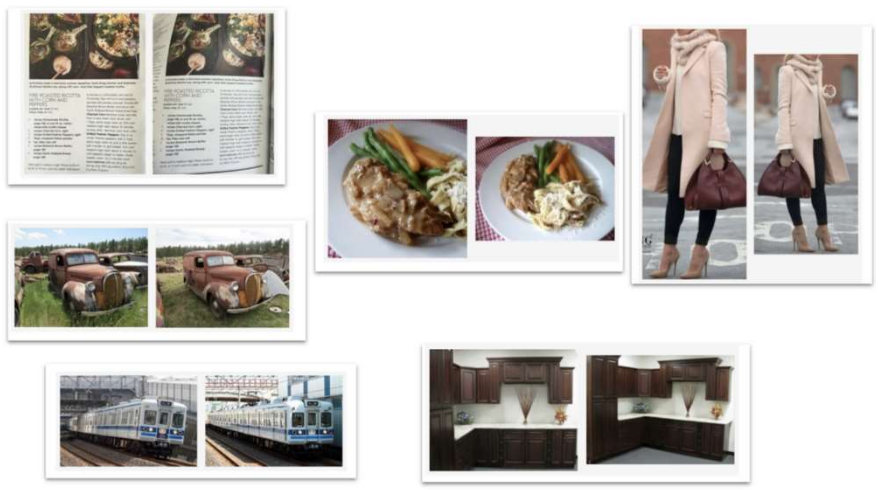
Pinterest Engineering Follow
Jun 15, 2018 · 5 min read

Andrey Gusev, Pinterest engineer, Content Quality

As a visual platform, the ability to learn from images to understand our content is important. In order to detect near-duplicate images we use the NearDup system, a Spark- and TensorFlow-based pipeline. At the core of the pipeline is a Spark implementation of batch LSH (locality-sensitive hashing) search and a TensorFlow-based classifier. Every day, the pipeline compares billions of items and incrementally updates clusters. In this post, we'll explain how we use this technology to better understand images and improve the accuracy and density of recommendations and search results across our production surfaces.

## Overview

We partition our image universe into classes of images that are nearly identical (as perceived by human observers). While this concept is somewhat subjective, the set of image pairs below will give you an idea of what falls within the NearDup threshold. Notice the image may not necessarily come from the same source photo (see the example image on the bottom right) or have the same background (see bottom left). It can have noticeable geometric distortions (see top left) or may be a rotational, crop or flip variant (see the middle and top right).



Finding an ideal partition over the universe of images is mathematically not well-defined, because the NearDup relation isn't transitive and therefore not an equivalence relation. To illustrate this point, imagine slowly morphing an image of a cat into an image of a dog over 1,000 iterations. It's expected that each iteration would fall well within the NearDup threshold, yet it's not clear how to partition the sequence: is there a cluster of cats, a cluster of dogs or perhaps a cluster of cat-dogs? We use a combination of transitive closure on selected candidates and a greedy k-cut to find an approximation to the partition that minimizes the k-cut over the graph. In the graph, edges represent image similarity, and nodes are the images.

## Candidate generation using batch LSH

### Embeddings and LSH terms

In order to understand the content of an image, we map an image into an embedded vector space. Visual embeddings are high-dimensional vector representations of images which capture visual and semantic similarity. They're typically produced via neural network architectures like VGG16 and Inception. To cluster images via the NearDup relation, each day we compare tens of millions of new images to billions of existing clusters. Approaching such a nearest neighbor search without an optimization would yield quadratic time complexity and a runtime proportional to more than 10 quadrillion (that's 16 zeros!) image comparisons. Instead, we use a reduction of an embedded representation of images into LSH terms to drastically improve the tractability of such a problem.

LSH is a modern technique used to reduce the dimensionality of high-dimensional data while preserving pairwise distances between individual points. Through a process based on random projections and bit sampling LSH, we first reduce the dimensionality of the original space. Next, the derived bits are grouped into LSH terms in a process that fundamentally trades off detection probability and runtime. The smaller the grouping, the more computationally expensive it is to run nearest neighbor search, but this increases the probability of accurate detection. This process uses LSH terms and their Jaccard overlap as an approximation of cosine similarity between vectors in the original embedded space.

### Batch LSH search

With every image being represented by a set of LSH terms, we proceed to build an inverted index and implement a batch-oriented search over images. At a high-level, we use a combination of functional transformations, compressed inverted indexes and joins to calculate a result set for all the query images at once. The pipeline is implemented in Spark and requires a series of optimizations to ensure we can handle the data volumes even in a much more computationally tractable LSH term space. Some of the optimizations include:

- **Dictionary encoding** so that everything is encoded via numeric primitives of the smallest possible width
- **Variable byte encoding** is used for all inverted indexes
- **Index partitioning** improves the balance of the inverted index
- **Cost-based optimizer** detects the density of the embedded space and determines the best runtime parameters
- **Primitive data packing** further improves memory utilization
- **Jaccard overlap counting** is done via low-level, high-performance collections
- **Off heaping** reduces GC overhead

## Candidate selection using transfer learning

Batch LSH is a powerful way to generate high recall while minimizing computational cost. However, it will generally not produce optimum precision and ranking of candidates. We use a pass over generated candidates via a supervised classifier to select the candidates that are similar enough to be considered NearDups. The classifier is an example of transfer learning over visual embeddings. It uses a TensorFlow feed-forward network and an AdamOptimizer. We've trained the classifier over a set containing more than one billion distinct pairs. The training set was derived from the output of a decision tree classifier over SURF visual features with geometric verification, which was used in prior iteration of the NearDup system. To improve learning and convergence for each pair of images, hamming bits (derived from visual embeddings) are XORed and fed into the input layer. The classifier is tuned for high precision and achieves over 99 percent precision on human-labeled data.

The inference over the trained network also happens in the Spark context. Using mapPartitions and grouped paradigm we can use large batches of predefined size to efficiently vectorize and reduce overhead. With a network that has almost 10 million parameters, we achieve an average of 2ms/prediction on a cluster of r3.8xlarge machines.

## Conclusion

NearDup detection requires a computationally costly pairwise comparison. By leveraging Batch LSH implementation in Spark, we drastically reduced computational complexity by skipping unlikely-to-be-similar pairs of images. The Spark-based implementation combines efficient distribution of workload as well as low-level optimization to minimize memory and CPU footprint. The subsequent fine-tuning step uses a supervised feed-forward network to select and rank image pairs that are above the NearDup similarity threshold. The combination of Spark and TensorFlow inference uses the best of distributed computations as well as vectorization per core to achieve both high throughput as well as low latency for prediction. The results of these two steps are then used to the cluster images that help power tens of billions of search results and recommendations on Pinterest every day.

*For more on this topic, check out my talk at the Spark+AI Summit 2018 below*

**Image Similarity Detection at Scale Using LSH and TensorflowAndrey Gusev (Pinterest)**

from **Databricks**

28:24

***Acknowledgements:*** *Thanks to the following members of the team for all their contributions on this project: Jiajing Xu, Vitaliy Kulikov, Jooseong Kim, Peter John Daoud, Andrew Zhai, Matthew Fang, Kevin Lau, Jacob Hanger, Zhuoyuan Li and Chao Wang*