2024/11/08

Fine-tuned SigLIP Image Embeddings for Similar Looks Recommendation in a Japanese C2C Marketplace

AI

Author:Sho Akiyama YadaYuki , 2024/11/08

日本語で読む →

Hello, we are Yuki and Sho, machine learning engineers on the AI/LLM team at Mercari.

In this tech blog, we dive into how we fine-tuned a large-scale Vision Language model on Mercari's product catalog to create foundational image embeddings for AI teams across the company.

By using the embeddings obtained from the model created this time, **we conducted an A/B test in the "Visually Similar Items" section on the product detail page.**

Originally, the "Visually Similar Items" section, internally known as "Similar Looks," utilized a 128-dimensional PCA-compressed embedding derived from a non-fine-tuned MobileNet model.

We conducted an A/B test on the "Similar Looks" feature, using image embeddings from our fine-tuned SigLIP model's Image Encoder in the treatment group. The results demonstrated significant improvements in key performance indicators:

- **1.5x increase in tap rate**
- **+14% increase in Purchase Count via Item Detail Page**

After confirming the positive results of the A/B test, we have released the fine-tuned SigLIP Similar Looks variant to 100% of the users. In this article, we will discuss about the defails of the project including the fine-tuning process, offline evaluation, and the end-to-end deployment infrastructure.

Fine-tuning of the SigLIP model using product data

Image Embedding

Image embedding is a core technique that expresses features such as the objects appearing in an image, their colors, and types as numerical vectors. In recent years, it has been used in various real-world application scenarios like recommendation and search. Within Mercari, its importance is increasing daily. Image embeddings are used in various contexts such as similar product recommendations, product searches, and fraudulent listing detection.

Recently, the AI/LLM team at Mercari worked on improving product image embedding using **a large-scale Vision Language Model: SigLIP.**

SigLIP

In recent years, models that have been pre-trained using contrastive learning with large-scale and noisy image-text pairs datasets, such as **CLIP [3]** and **ALIGN [4]** , are known for achieving high performance in zero-shot classification and retrieval tasks.

The **SigLIP** model was introduced in a paper presented at ICCV 2023. This Vision Language Model employs a novel approach to pre-training by replacing the conventional Softmax loss function used in CLIP with a **Sigmoid loss** function. Despite the simplicity of this modification, which solely involves altering the loss calculation method, the authors report **significant performance improvements on standard benchmarks** , including image classification tasks using ImageNet [6].

Let’s examine the implementation of the loss function that was developed for fine-tuning the model using Mercari’s internal dataset, which will be discussed in more detail later.

```
def sigmoid_loss(
    image_embeds: torch.Tensor,
    text_embeds: torch.Tensor,
    temperature: torch.Tensor,
    bias: torch.Tensor,
    device: torch.device = torch.device("cuda") if
torch.cuda.is_available() else torch.device("cpu")
):
    logits = image_embeds @ text_embeds.T * temperature +
bias
    num_logits = logits.shape[1]
    batch_size = image_embeds.shape[0]
    labels = -torch.ones(
        (num_logits, num_logits), device=device,
dtype=image_embeds.dtype
    )
    labels = 2 * torch.eye(num_logits, device=device,
dtype=image_embeds.dtype) + labels
    loss = -F.logsigmoid(labels * logits).sum() / batch_size

    return loss
```

We utilized google/siglip-base-patch16-256-multilingual as a base model. This model has been trained on a multilingual WebLI dataset [5], making it particularly suitable for our application as it supports Japanese, which is the primary language used in Mercari's service.

Fine-tuning Using In-house Data

In this section, we introduce a detailed setting of fine-tuning experiments of SigLIP using real-world service data. We conducted fine-tuning of the SigLIP model using approximately one million randomly sampled Mercari product listings (text-image pairs)

from items listed. The input data for SigLIP consisted of product titles (text) and product images (image), both of which were created by sellers on the Mercari platform.

The training code was implemented using PyTorch and the Transformers library. Due to the large scale of our dataset, we leveraged WebDataset to optimize the data loading process, ensuring efficient handling of the substantial amount of training data.

Model training was conducted on a **single L4 GPU** . We utilized Vertex AI Custom Training to construct a robust training pipeline. For experiment monitoring, we employed Weights & Biases (wandb), taking advantage of Mercari's enterprise contract with the platform. This setup allowed for comprehensive tracking and analysis of the training process, facilitating iterative improvements and model optimization.

The combination of these technologies and platforms—PyTorch, Transformers, WebDataset, Vertex AI, and wandb—provided a scalable and efficient framework for fine-tuning the SigLIP model on our proprietary e-commerce dataset, while maintaining close oversight of the training progress and performance metrics.

Offline Evaluation

Prior to conducting A/B testing, we performed an offline evaluation using user interaction logs from the existing "visually similar products" feature. This evaluation utilized approximately 10,000 session data points.

Here is a specific example of an action log. The `query_item_id` holds the ID of the product displayed on the product detail page as the query image, `similar_item_id`

contains the ID of the product displayed in the "Similar Looks" section, and clicked is a flag indicating whether the product was viewed or not.

session_id	query_item_id	similar_item_id	clicked
0003e191…	m826773…	m634631…	0
0003e191…	m826773…	m659824…	1
0003e191…	m826773…	m742172…	1
0003e191…	m826773…	m839148…	0
0003e191…	m826773…	m758586…	0
0003e191…	m826773…	m808515…	1
...			

We formulated the evaluation as an image retrieval task, treating user clicks as positive examples. The performance was assessed using nDCG@k and precision@k as evaluation metrics. This approach allowed us to quantitatively measure the model's ability to rank relevant products in a manner consistent with user preferences.

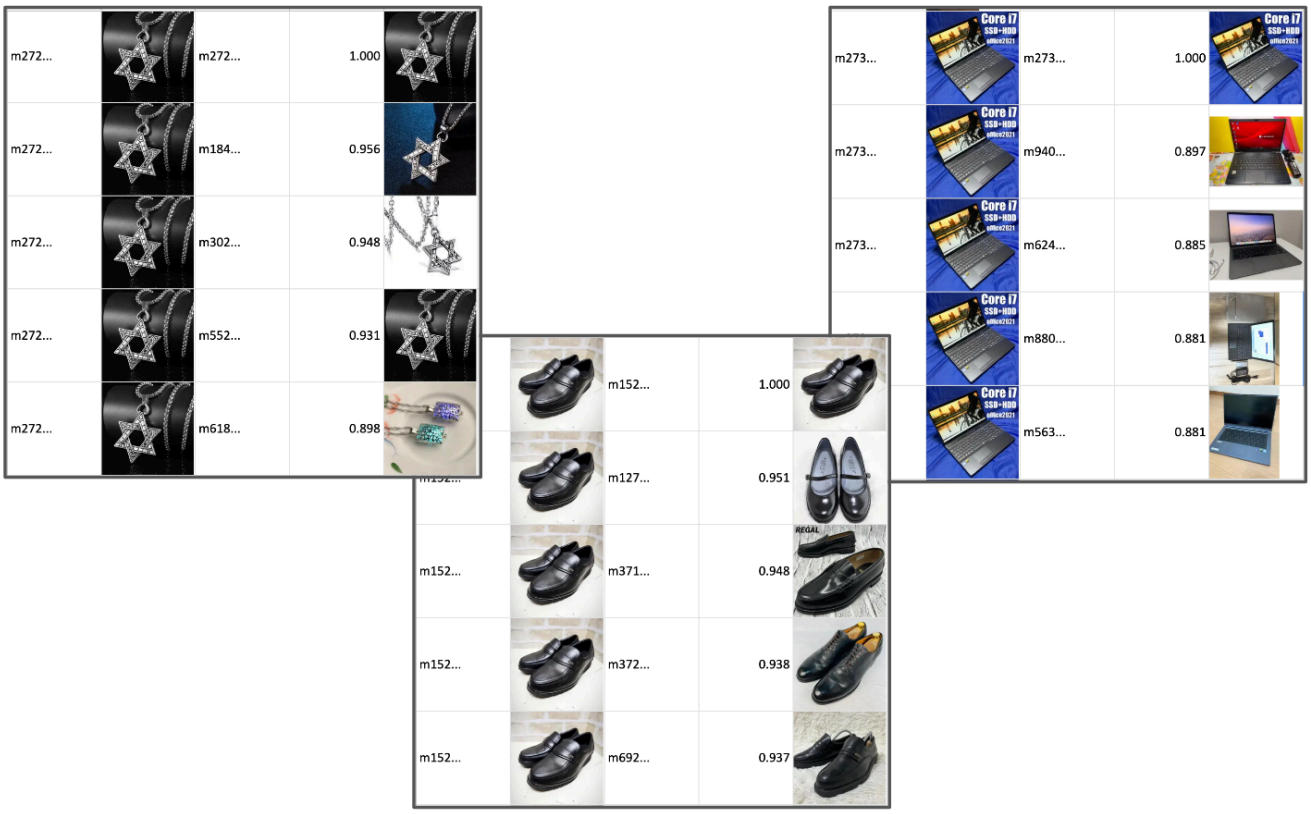
We conducted our evaluation using two baseline methods for comparison: random recommendation and image retrieval based on MobileNet, which is currently employed in the existing Similar Looks feature.

The following were our results:

Method	nDCG@5	Precision@1	Precision@3
Random	0.525	0.256	0.501
MobileNet	0.607	0.356	0.601
SigLIP + PCA	**0.647**	**0.406**	**0.658**
SigLIP	**0.662**	**0.412**	**0.660**

Evaluation results show that **image retrieval using embeddings from the fine-tuned SigLIP Image Encoder consistently outperformed MobileNet-based image search, even when SigLIP embeddings were compressed from 768 to 128 dimensions using PCA. This demonstrates the superior performance of our fine-tuned SigLIP model for product similarity tasks.**

In addition to quantitative evaluation, we also conducted qualitative evaluation through visual inspection. We created a vector store using FAISS, containing embeddings of approximately 100,000 product images. We then performed image searches for multiple products and compiled the results in a spreadsheet, as shown below, for visual inspection.

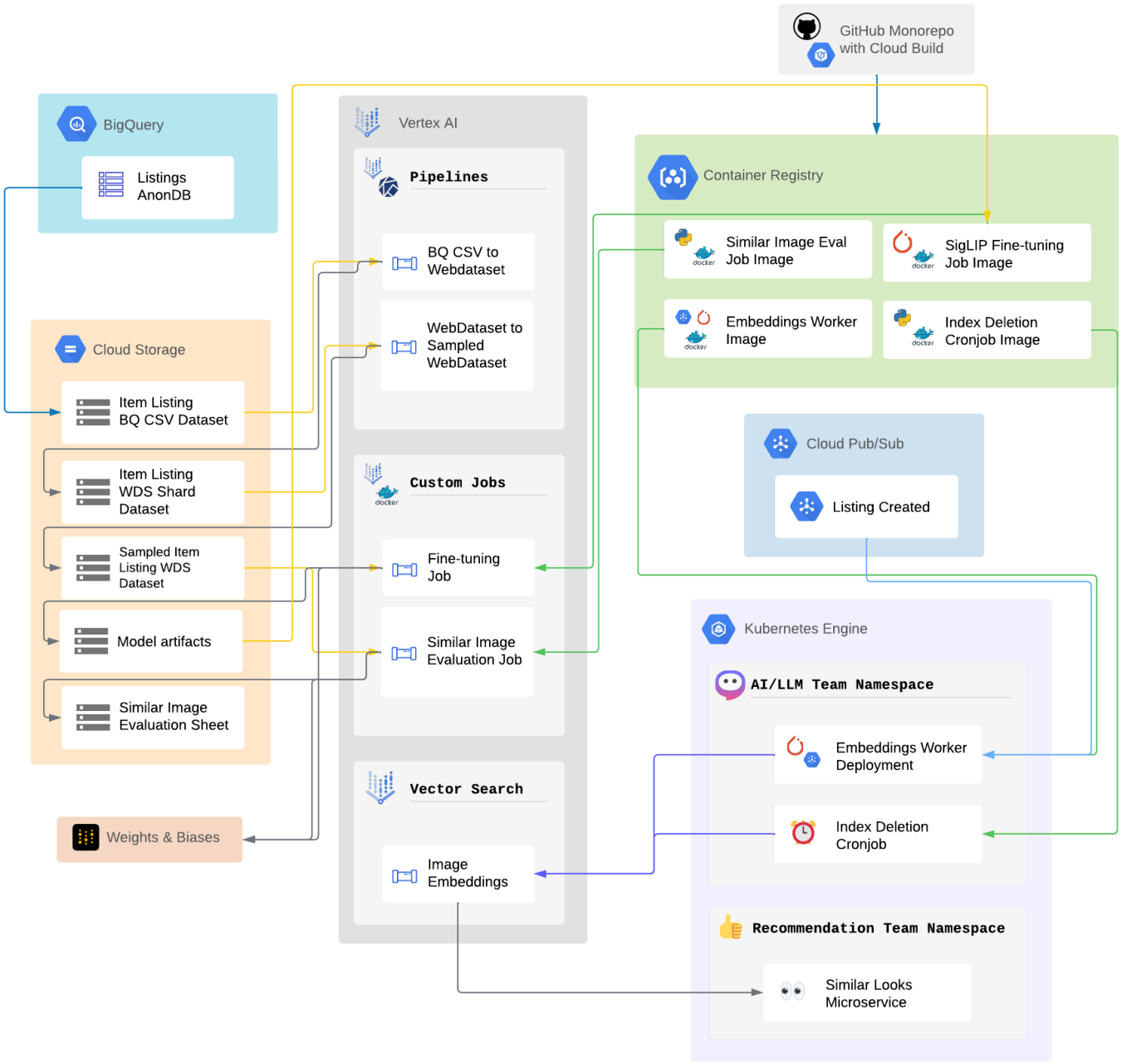


These results conclusively demonstrate that the Similar Looks Recommendation system, powered by the SigLIP Image Encoder fine-tuned on product data, outperforms the existing model both quantitatively and qualitatively. So, we decided to proceed with A/B test using the created model. In the following chapters, we will present the system design for deploying this model to production.

Deployment Architecture

End-to-End Architecture

Before diving into individual components, here's a high-level view of our architecture:



In the diagram above, you can see how data flows from the marketplace platform to our model services and how embeddings are stored and retrieved efficiently. While this is an initial version, this modular design ensures scalability and flexibility as we evolve the system.

Google Container Registry

Our model deployments are managed through **Google Container Registry (GCR)** , where Docker images of our microservices are stored. These images are continuously built and pushed to GCR from our GitHub repository via a CI/CD pipeline with Google Cloud Build.

By leveraging GCR, we ensure that our deployments in **Google Kubernetes Engine (GKE)** are always based on the latest versions of the code, offering seamless updates to the services that run in production.

Google Pub/Sub

To handle real-time data streams, we rely on **Google Pub/Sub** . New listings created on our marketplace are published as messages to specific topics, such as topics for new listings. The relevant microservices subscribe to these topics, enabling the system to react dynamically to new product listings.

Whenever a seller uploads a new product image, a message is sent to Pub/Sub. This triggers our **Embeddings Worker** , which processes the image from the new listing and

updates the vector database with new embeddings. This asynchronous system allows us to scale effectively with the volume of marketplace activity.

Google Kubernetes Engine

The heart of our deployment lies within **Google Kubernetes Engine (GKE)** . This platform hosts several key services in our architecture:

Embeddings Worker

The **Embeddings Worker** is a critical service that listens to the new listings topic in Pub/Sub. For each new listing, the worker:

1. Fetches the corresponding image
2. Converts it into a fixed-length vector embedding using our fine-tuned **SigLIP** model
3. Runs **Principal Component Analysis (PCA)** to reduce the dimensions for improved latency on the similarity search and cost savings for storage (768 dim → 128 dim)
4. Stores the embedding in **Vertex AI Vector Search**

This process enables us to perform image similarity searches efficiently. Each embedding represents the visual content of the image, making it easy to compare and find visually similar listings across the platform.

Index Cleanup Cron Job

As the marketplace is highly dynamic, with new listings being added and old listings getting sold or removed, we needed a way to keep our embeddings up-to-date. For this, we implemented an **Index Cleanup Cronjob** . This cron job runs periodically to remove embeddings corresponding to outdated and sold listings from **Vertex AI Vector Search** .

While this batch cleanup process works well for now, we are exploring live updates for embedding management to improve efficiency further.

Similar Looks Microservice & Caching

The **Similar Looks Microservice** is the core of our image similarity feature. It takes a listing ID as input, retrieves the corresponding image embedding from **Vertex AI Vector Search** , and performs a nearest-neighbor search to find similar items in the marketplace.

To reduce latency, we've implemented caching mechanisms in this microservice as well. This ensures a smooth user experience by delivering quick responses when users browse for similar products.

Vertex AI Vector Search

For storing and retrieving embeddings, we use **Vertex AI Vector Search** , a scalable vector database that allows us to efficiently search for similar embeddings. Each product image in the marketplace is mapped to a vector, which is then indexed by listing ID in **Vertex AI** .

The nearest-neighbor search algorithms built into Vertex AI enable fast retrieval of visually similar listings, even with a large amount of embeddings in the database.

Model Optimization with TensorRT

To optimize the performance of our fine-tuned **SigLIP** model and handle our high amount of listings created per second, we converted the model from PyTorch to **TensorRT** , NVIDIA's high-performance deep learning inference library. The conversion resulted in a **\~5x speedup** in inference times.

TensorRT

TensorRT optimizes deep learning models by performing precision calibration, layer fusion, kernel auto-tuning, and dynamic tensor memory allocation. Specifically, TensorRT converts the operations in the neural network into optimized sequences of matrix operations that can run efficiently on NVIDIA GPUs.

For our marketplace, this improvement was critical. With a massive amount of product listings being created per second, reducing inference time from hundreds of milliseconds to mere fractions enabled us to make sure that all new listings have their images almost instantly embedded into vectors to be ready in the Vertex AI Vector Search index for the Similar Looks component to use.

Next Steps

While our current deployment architecture is stable and scalable, we are constantly looking for ways to improve. Here are some of the next steps we are working on:

Live Updates of Embeddings

Currently, the **Index Cleanup Cronjob** is responsible for removing outdated embeddings from **Vertex AI Vector Search** . However, we plan to move to a more real-time solution where embeddings are updated as soon as a listing is removed or sold. This will eliminate the need for periodic cleanups and ensure that our index is always up-to-date.

Triton Inference Server

We are also exploring the use of Triton Inference Server to handle model inference more efficiently. Triton allows for the deployment of multiple models across different frameworks (e.g., TensorRT, PyTorch, TensorFlow) in a single environment. By shifting inference from the **Embeddings Worker** to Triton, we can decouple the model execution from the worker logic and gain greater flexibility in scaling and optimizing inference performance.

New Features Using the Fine-Tuned SigLIP Model

Lastly, we are working on new features that will leverage our fine-tuned **SigLIP** model. Stay tuned for updates on how we plan to enhance the user experience with advanced image search capabilities, potentially including multimodal search, where users can

combine text and image queries to find exactly what they are looking for, as well as apply the embeddings to a lot of different Mercari features and processes.

Conclusion

In this project, we fine-tuned the Vision-Language Model SigLIP using Mercari's proprietary product data to build a high-performance Image Embedding Model, improving the "Visually Similar Items" feature.

In offline evaluations, the fine-tuned SigLIP demonstrated superior performance in recommending "Visually Similar Items" compared to existing models. **Consequently, when we conducted an A/B test, we observed significant improvements in business KPIs.**

We hope that the content of this blog will be helpful to those interested in fine-tuning Vision Language Models, evaluation, and deploying deep learning models to real-world services.

Mercari is hiring Software Engineers who want to make impactful product improvements using Machine Learning and other technologies. If you're interested, please don't hesitate to apply!

References

[1] Sigmoid Loss for Language Image Pre-Training, 2023

[2] MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications, 2017

[3] Learning Transferable Visual Models From Natural Language Supervision, 2021

[4] Scaling Up Visual and Vision-Language Representation Learning With Noisy Text Supervision, 2021

[5] PaLI: A Jointly-Scaled Multilingual Language-Image Model, 2022

[6] ImageNet: A Large-Scale Hierarchical Image Database, 2009



0

AI Machine Learning Recommendation Vision & Language

Related article

2025/03/10

Tackling Knowledge Management

AI Organization

Author: raven

→

2025/02/06

LLM x SRE: Mercari's Next-gen Incident Handling Buddy

AI Infrastructure

Author: Tianchen Wang

→

2024/12/15

LLMs at Work: Outsourcing Vendor Assessment Toil to AI

AI Security

Author: simon danny

→

Mercari India

mercari R4D

Engineering blog
(USA)