Exploring and fine-tuning CLIP and further extensions

Vinay Kumar
Deep Vision Project Report - MAI

Ostbayerische Technische Hochschule Amberg-Weiden

July 3, 2025

Abstract

Contrastive learning has gained significant attention for its ability to perform zero-shot classification across diverse visual domains. Models like CLIP and its variants, which use contrastive language-image pre-training, have shown impressive results in general computer vision tasks, but adapting these models for specialized, high-precision applications like satellite image analysis remains challenging. This report explains how a state-of-the-art model from Google, called SigLIP, can be successfully trained to classify different types of satellite images. SigLIP is based on a unique method, which uses a 'sigmoid' function to learn more efficiently by focusing on individual image-and-text pairs rather than comparing large groups of them. My approach was to build upon this existing knowledge and carefully guide the model to learn the specific features of satellite imagery. A key discovery was that even though the model was trained to pick just one label for each image, it could later be used in a new way to identify multiple related features.
Dataset used - https://huggingface.co/datasets/jonathan-roberts1/Million-AID
GitLab repo - [1]

Keywords
zero-shot classification, image-text pair, image classification.

1 Introduction

1.1 Multi-Modal ML

Multi-modal Deep Learning represents a significant leap forward in artificial intelligence, moving beyond single-source data to create more holistic models. This subfield focuses on training algorithms to process, interpret, and find relationships between disparate data types of "modalities" such as text, audio, images, and video [2].

The fundamental principle is that a more accurate and robust understanding of the world is achieved by integrating these streams. Often, critical information exists exclusively in one modality. So the process of fusing different modalities so that in the end model can learn from them is known as **multimodal fusion** and the models which utilities this fusion is known as **multimodal models**.

How does the model tackle with when it is combined with different modalities ? There are 3 general strategies for multimodal fusion:

- **Early Fusion:** It refers to combining different modalities at the input level.By merging all modalities at the input stage, this method avoids separate processing pipelines and handles everything through one unified model.
- **Late Fusion:** It works like combining multiple models, where each modality is independently processed by dedicated models before combining their outputs. This approach offers simplicity and allows each model to extract rich, modality-specific representations.
- **Hybrid Fusion:** It strategically mixes early and late fusion techniques, processing some modalities together while keeping others separate. This adaptable method optimizes performance by choosing the most effective fusion strategy for different modality combinations.

Joint representation learning creates unified

embeddings from multiple data types by mapping them into a shared semantic space. Unlike fusion techniques that combine modalities at specific stages, joint representation learning forces different modalities to learn compatible representations from the start. Text and image encoders are trained together to produce embeddings that capture cross-modal relationships and semantic similarities. This approach is particularly valuable for contrastive learning because it enables direct comparison between different modalities in the same embedding space. While fusion methods require additional processing steps to align modalities, joint representations naturally support contrastive objectives by ensuring that semantically similar content across modalities appears close together, regardless of the input type. This unified space makes contrastive learning more effective since the model can directly measure similarities and differences between text, images, and other modalities without complex fusion operations

1.2 Contrastive Learning

Contrastive learning is a joint representation learning technique that teaches models to understand the world by distinguishing between similar (positive pairs) and different (negative pairs) examples. This method learns meaningful feature representations from vast amounts of unlabeled data by creating a smart internal embedding space where semantically similar items are grouped closely together while dissimilar ones are pushed far apart. The core principle revolves around learning robust representations that capture underlying data patterns and relationships without requiring explicit labels. Through this process, the model develops rich internal representations that encode semantic meaning, making it highly effective for downstream tasks. This representation learning approach has proven incredibly successful, as it enables models to extract meaningful features that generalize well to various applications.[3]

Where this concept truly excels is in **Self-Supervised Contrastive Learning (SSCL)**, which addresses label-free learning by automatically generating comparative samples from unlabeled data. The approach creates positive pairs through augmented views of the same input, teaching the model to recognize their inherent similarity. Concurrently, negative pairs are formed using distinct data instances, training the model to distinguish them as dissimilar. Through massive repetition, the model autonomously learns to extract essential features without human annotations.

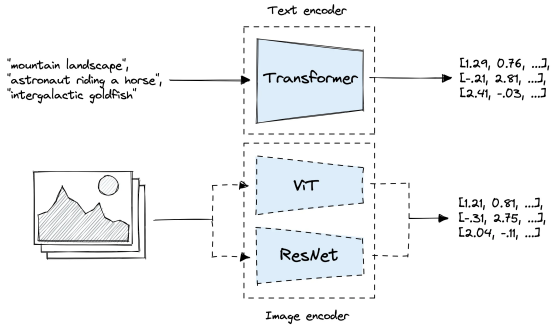

Figure 1: CLIP Architecture[5]

And this entire learning process is driven by, most famously the InfoNCE loss function also known as softmax cross entropy over similarities.

$$\mathcal{L}_{\text{InfoNCE}} = -\log\left(\frac{\exp\big(\text{sim}(q,k^+)/\tau\big)}{\sum_{i=1}^{K}\exp(\text{sim}(q,k_i)/\tau)}\right)$$

Where q is the **query** vector, k^+ is the **positive key** the correct match for q, k_i represents all possible keys (positives + negatives), $\text{sim}(q,k)$ is the similarity function (e.g., cosine or dot product), and τ is the temperature parameter which controls sharpness of the softmax.

2 Related Work

Contrastive language-image pre-training [4] has become popular because it is a model that connects images and text by mapping them into a shared vector space. Using contrastive learning, CLIP [4] learns to distinguish which image-text pairs are correctly matched and which aren't. This capability allows it to recognize new types of objects, even those it wasn't specifically trained on. Consequently, CLIP is particularly good at zero-shot classification, enabling it to accurately identify novel items based solely on textual descriptions

Zero-Shot Classification: This technique enables the model to categorize new items without requiring training examples. It's termed "zero-shot" because it needs no specific training data, using only text descriptions to generate predictions

CLIP's architecture (Fig.1) handles zero-shot classification through an dual-encoder design. At its core, two specialized encoders – one for images (vision transformer) and one for text (original transformer encoder) work at he same time. These convert input images and text prompts into high-dimensional embeddings within a shared vector space.

To perform zero-shot classification, CLIP first converts a set of text descriptions (like "a photo

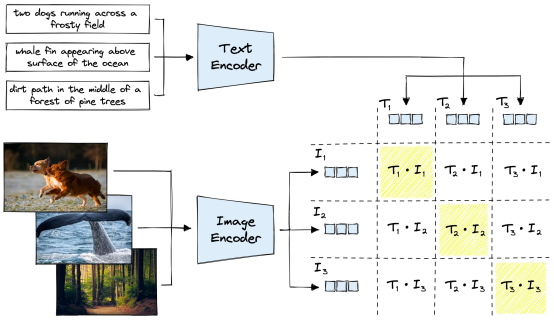

Figure 2: Contrastive pretraining with CLIP [5]

of a cat" or "a photo of a dog") into embeddings, while simultaneously transforming the input image into its own embedding. The model then calculates cosine similarity (Fig.2), which measures directional alignment between the image embedding and every text embedding. By selecting the text label with the highest similarity score, CLIP predicts the image's category. This state-of-the-art process enables image classification for entirely novel concepts without any task-specific training, relying purely on semantic relationships learned from language.

CLIP embedding similarities are represented by their angular similarity. Meaning we can identify similar pairs using cosine similarity:

$$\text{cossim}(A, B) = \frac{A \cdot B}{||A|| \cdot ||B||} \tag{1}$$

Or, if we have normalized the embeddings, we can use dot product similarity:

$$\text{dotproduct}(A, B) = A \cdot B \tag{2}$$

Despite CLIP faces two major scalability challenges stemming from its contrastive learning design. First, its loss function demands extremely large batch sizes (e.g., [32,768] examples) to expose the model to sufficiently diverse negative samples during training, necessitating expensive multi-GPU/TPU clusters. Second, those large batche sizes create a massive communication problem. Because CLIP needs to compare every single image with every single text description at the same time (using its core "softmax" math), all the GPUs must constantly share their data with each other. Every GPU ends up needing a full copy of every image and text snippet in the batch to calculate how well they match. This constant back-and-forth sharing between GPUs slows everything down tremendously, making CLIP very hard and expensive to train at large scale. The loss function for the CLIP is defined as :

```
# image_encoder - ResNet or Vision Transformer
# text_encoder  - CBOW or Text Transformer
# I[n, h, w, c] - minibatch of aligned images
# T[n, l]       - minibatch of aligned texts
# W_i[d_i, d_e] - learned proj of image to embed
# W_t[d_t, d_e] - learned proj of text to embed
# t             - learned temperature parameter

# extract feature representations of each modality
I_f = image_encoder(I) #[n, d_i]
T_f = text_encoder(T)  #[n, d_t]

# joint multimodal embedding [n, d_e]
I_e = l2_normalize(np.dot(I_f, W_i), axis=1)
T_e = l2_normalize(np.dot(T_f, W_t), axis=1)

# scaled pairwise cosine similarities [n, n]
logits = np.dot(I_e, T_e.T) * np.exp(t)

# symmetric loss function
labels = np.arange(n)
loss_i = cross_entropy_loss(logits, labels, axis=0)
loss_t = cross_entropy_loss(logits, labels, axis=1)
loss   = (loss_i + loss_t)/2
```

Figure 3: pseudo code for clip

$$-\frac{1}{2|B|}\sum_{i=1}^{|B|}\left[\underbrace{\log\frac{e^{t\mathbf{x}_i\cdot\mathbf{y}_i}}{\sum_{j=1}^{|B|}e^{t\mathbf{x}_i\cdot\mathbf{y}_j}}}_{\text{image}\to\text{text}}+\underbrace{\log\frac{e^{t\mathbf{x}_i\cdot\mathbf{y}_i}}{\sum_{j=1}^{|B|}e^{t\mathbf{x}_j\cdot\mathbf{y}_i}}}_{\text{text}\to\text{image}}\right]$$

So the improvement for CLIP disadvantages was the introduction of SigLIP[6] it overcomes CLIP's efficiency limitations by fundamentally changing how it evaluates image-text relationships. Instead of treating matching as a complex ranking task where every possible pair in a batch must be compared simultaneously (like CLIP does), SigLIP simplifies the problem. It assesses each image-text pair individually, asking a straightforward question: "Do these specifically belong together, yes or no?" This independent, binary approach means SigLIP doesn't need to gather and analyze every single image and text embedding across a massive batch to understand one relationship. Consequently, SigLIP eliminates the need for CLIP's computationally heavy global comparisons and the large similarity matrix they require. This independence drastically cuts communication needs between computing units SigLIP only shares text data once, not both images and texts repeatedly. The result is a leaner, faster model: SigLIP achieves high performance like CLIP but trains more efficiently with smaller hardware demands, lower memory use, and significantly reduced communication overhead.

To address the extreme imbalance between rare matching pairs and abundant mismatches, SigLIP introduces two clever "calibration techniques": a temperature parameter (t) and a novel bias term (b) (Fig.4). During training, the model faces millions of negative pairs (mis-

Algorithm 1 Sigmoid loss pseudo-implementation.

```
# img_emb      : image model embedding [n, dim]
# txt_emb      : text model embedding [n, dim]
# t_prime, b   : learnable temperature and bias
# n            : mini-batch size

t = exp(t_prime)
zimg = l2_normalize(img_emb)
ztxt = l2_normalize(txt_emb)
logits = dot(zimg, ztxt.T) * t + b
labels = 2 * eye(n) - ones(n) # -1 with diagonal 1
l = -sum(log_sigmoid(labels * logits)) / n
```

Figure 4: SigLIP [6]

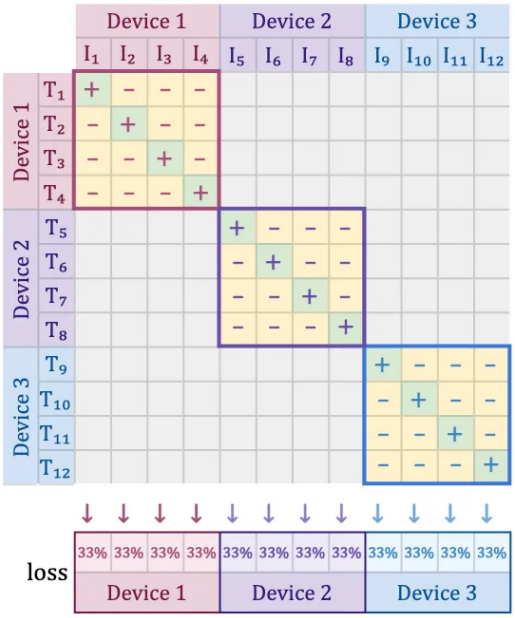

Figure 5: Computation of the loss [6]

matches) for every positive match, which would normally overwhelm early learning. The bias term b acts like a counterweight—initialized at -10, it deliberately lowers initial match scores to account for this imbalance, preventing drastic over-corrections. Meanwhile, the temperature t (initialized at log(10))[6] controls how sharply the model distinguishes between clear matches and borderline cases. Together, these parameters serve as "training stabilizers" they start the model in a balanced state aligned with real-world data distribution, allowing gradual refinement rather than chaotic early adjustments.

The loss function of SigLIP is described as:

$$\frac{1}{|B|}\sum_{i=1}^{|B|}\sum_{j=1}^{|B|}\underbrace{\log\frac{1}{1+e^{z_{ij}(-t\mathbf{x}_i\cdot\mathbf{y}_j+b)}}}_{\text{log loss term}}$$

As shown in the (Fig.5) the sigmoid-based loss processes every image-text pair independently, effectively turning the learning problem into the standard binary classification on the dataset of all pair combinations, with a positive label for the matching pairs (I_i, T_i) and negative labels for all other pairs $(I_i, T_j,\ j \neq i)$.

3 Methods

3.1 Initial Exploration (Zero-Shot Inference)

This initial phase was designed to assess the capabilities of the pre-trained SigLIP model https://huggingface.co/google/siglip-base-patch16-224 on the specific task of satellite image classification without any prior training on the target dataset. The methodology involved creating a set of descriptive text prompts, one for each of the 51 potential classes (e.g., "a satellite image of a dam"). For each image in the dataset, the model independently computed a semantic similarity score between the image features and the features of every text prompt. The class corresponding to the prompt with the highest similarity score was considered the model's prediction. This process served the dual purpose of establishing a quantitative performance baseline against which all subsequent fine-tuning efforts could be measured, while also validating that the model's foundational visual-language understanding was indeed relevant and applicable to the remote sensing domain.

The (Fig.6) shows the result of just a simple zero-shot transfer learning on the given dataset. This highlights the model's performance on the dataset prior to any task-specific training. While the model demonstrates some capacity to identify general features, its predictions are frequently incorrect, leading to a high rate of misclassifications between visually similar or contextually related satellite image-text pair. This outcome establishes a critical need for fine-tuning, the next stage of this project, to adapt the model and significantly improve its classification accuracy for this specific task.

3.2 Data Preparation and Augmentation

To train the model effectively, getting the data ready was a key step. A model when trained on a small dataset, can easily cheat by simply memorizing the training images instead of actually learning to identify the objects within them. To solve this, a comprehensive data augmentation[7] was implemented. This involved applying a variety of transformations to

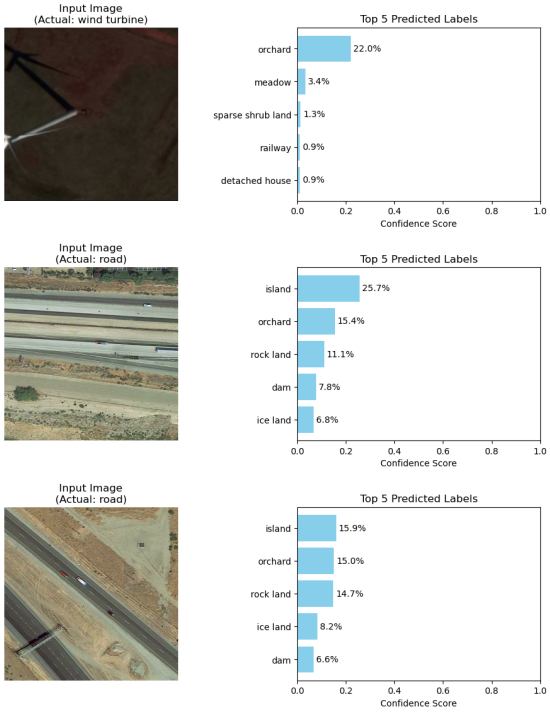

Figure 6: Zero-Shot Classification

the training images, including geometric augmentations like random resizing, cropping, flipping, and rotation, as well as color-based augmentations such as adjustments to brightness, contrast, and saturation. These techniques artificially expand the diversity of the training set, exposing the model to a wider range of visual scenarios and forcing it to learn more invariant and robust features.

A thing to keep in mind, this augmentation strategy was applied exclusively to the training data. The validation data, which is used to test the model's performance, in this augmentation was not applied because the purpose of the validation set is to provide a consistent and unbiased measure of the model's performance on unseen, real-world data. Applying random augmentations to it would corrupt the integrity of the evaluation, as it would be impossible to determine if a performance change was due to the model's learning or simply a random. By not touching the validation set it ensures that any measured increase in accuracy is a true reflection of the model's improved capabilities.

3.3 Fine-Tuning

For fine-tuning only the SigLIP image encoder part was touched i.e a vision transformer, as the project goal was to make an image and assign it a label, for this we needed only a image encoder part. As the goal is to that the trained model wouldn't need to read any text to make prediction, it must be self-capable to find and then decide on it's own the correct label and given them a score.

To do this, a new, small layer called a "classifier head" was attached to the end of the vision encoder. This new part's only job was to take the complex visual information from the encoder and learn how to map it to one of the 51 specific labels, given by original dataset.

As this SigLIP image encoder made up of 12 Vision transformer layers or blocks, so to avoid messing up the model's existing knowledge and to make training faster, freezing and unfreezing of certain layer had been performed. This means the layers which were termed as "frozen" were locked and won't contribute to the model performance during training means no weight will be updated, as it will contain it's old weights. And the "unfrozen" layer, means deciding how many layers to unfreeze was an experiment. The process was a trial-and-error to find what worked best. I started by unfreezing just the final 3 layers and tested the performance. I then increased this number, eventually finding that unfreezing the last 6 layers gave the best improvement in the model's accuracy on the validation set.

3.4 Hyperparameter

Choice of loss function: The foundational task for training the model was defined as single-label, multi-class classification, where the objective is to assign each image to one of 51 distinct and mutually exclusive categories. To align with this objective, *CrossEntropyLoss* was selected as the loss function. It s specifically for scenarios where only one class can be correct and softmax activation function is already being integrated inside this loss function. It was the great choice because, it forces a competitive relationship between the output classes by transforming the model's raw scores (logits) into a probability distribution that sums to one. An increase in the probability of one class necessitates a decrease in the probabilities of all others and also it calculates a loss that heavily penalizes the model if it fails to assign the highest probability to the correct ground-truth class.

And an alternative, *BCEWithLogitsLoss*, was also evaluated. However, it doesn't resulted into up-to the mark performance as expected. As is designed for multi-label problems where classes are independent, and its application to a single-label dataset provides a less efficient and more ambiguous learning signal, which block the effective model convergence and result it into underfitting problem means model doesn't learn any-

thing during the entire training process.

Optimizer: Choosing an right and appropriate optimizer is a fundamental component of the training algorithm, responsible for iteratively updating the model's weights to minimize the loss function. The process of training a neural network can be visualized as navigating a complex, to find the a global minimum, which represents the optimal set of model parameters. A well-chosen optimizer accelerates this process, enabling faster and more stable model convergence while avoiding getting stuck in suboptimal local minima and also doesn't let model to get stuck inside the saddle points means which is not a local maximum or minimum.

For this task, the Lion (Evolved Sign Momentum)[8] optimizer was selected, as used in SigLIP[6]. Lion uses a sign-based update rule, which makes it computationally efficient and well-suited for vision and large-scale Transformer models. While optimizers like Adam and its variant, AdamW, have long been the standard, recent research has demonstrated Lion's superior performance in this domain.

Lion significantly reduces GPU memory overusage a critical factor for large models. Furthermore, its simpler update rule, which relies on the sign of the gradient rather than its magnitude, has been shown to improve training stability and generalization for Transformer architectures. This choice is supported by empirical evidence demonstrating Lion's ability to achieve superior accuracy with a lower computational budget.

$$m_t = \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot g_t$$
$$\theta_{t+1} = \theta_t - \eta \cdot \text{sign}(m_t)$$

where:

- g_t is the gradient of the loss with respect to parameters at time step t
- m_t is the exponential moving average of past gradients (momentum)
- $\beta_1 \in [0, 1)$ is the momentum coefficient (typically around 0.9)
- η is the learning rate
- $\text{sign}(m_t)$ applies the sign function elementwise to m_t
- θ_t are the model parameters at step t

Learning Rate: The learning rate is crucial because it determines how quickly and effectively the neural network learns. In the optimization landscape, the learning rate controls how the model navigates through local and global minima during training. When training neural networks models, getting the learning rate right means finding the optimal balance for convergence - too high and the model may overshoot the global minimum or get stuck in poor local minima, too low and convergence becomes extremely slow as the model takes unnecessarily small steps in the loss landscape, thus convergence become very slow . A poorly chosen rate can either cause the model to diverge from optimal solutions by jumping between different local minima, or converge so slowly to a minimum that training becomes computationally inefficient. That's why tuning the learning rate is one of the most important parts of training neural networks successfully.

So one of the technique known as **learning rate annealing** is a sophisticated optimization technique that adaptively adjusts the learning rate during training. It starts with a higher learning rate to allow rapid exploration of the loss landscape and escape potential local minima, then gradually decreases it to enable finer convergence towards the global minimum. This dynamic adjustment helps optimize the training process - the initial larger learning rate allows the model to traverse different regions of the parameter space quickly, while the subsequent reduction helps it settle into promising minima with greater precision. By automatically reducing the learning rate over time, annealing helps prevent both premature convergence to suboptimal solutions and the oscillation problems that can occur with fixed learning rates, ultimately leading to more robust and efficient model training.

For this task, *Cosine Annealing*is being used, it is a cyclical learning rate strategy that oscillates between maximum and minimum values following a cosine function. This approach allows the model to alternate between periods of exploration and exploitation - high learning rates help explore broader regions of the loss landscape, while low rates enable fine-tuning in promising areas. It's particularly effective in complex architectures like Transformers models, where the periodic nature of the learning rate helps escape local minima while ensuring thorough optimization of the model parameters.

The cosine annealing scheduler dynamically adjusts the learning rate following a cosine curve between η_{max} (initial learning rate) and η_{min} (minimum learning rate) over T_{max} epochs. At each step, the learning rate oscillates smoothly, starting high for exploration and gradually decreasing for fine-tuning, with T_{cur} tracking the

current epoch to determine the precise position in the cosine cycle.

$$\eta_t = \eta_{min} + \frac{1}{2}(\eta_{max} - \eta_{min})(1 + \cos(\frac{T_{cur}}{T_{max}}\pi))$$

3.5 Training and Validation:

The entire training process purpose is to check whether the model is learning the right lessons from the training data. The validation data doesn't directly influence the model's learning by causing weight updates; instead, it provides critical feedback that helps to fine-tune the overall training strategy. This was implemented by dividing each of the ten epochs into a learning phase and a testing phase.

The training phase itself was conducted using a mini-batch approach. Instead of feeding the entire large training dataset to the model at once, it was broken down into mini-batches. For each mini-batch, the model performed a forward pass to make predictions, the loss was calculated, and the optimizer updated the model's weights via backpropagation. This cycle was repeated for all the mini-batches until the model had processed the entire training set, which constitutes one full epoch. The model was presented with a different, randomized sequence of mini-batches at the start of every new epoch. While the overall set of training images remains the same, this shuffling is critical. It prevents the model from learning any coincidental patterns based on the order of the data, forcing it to learn more robust features directly from the content of the images themselves.

Once an epoch of training over all mini-batches was complete, the model was immediately validated to get a signal about its ability to generalize. This validation accuracy was the criterion for saving the model. The model's state was only saved if its performance on the validation set surpassed the best performance from all previous epochs.

4 Results & Discussions

4.1 Performance Analysis

These plots (Fig.7) track the model's loss and accuracy over the ten training epochs, providing insights into its learning behavior and generalization capabilities. The fine-tuning was highly successful, reaching in a peak validation accuracy of approximately 94%, which demonstrates a profound improvement over the zero-shot baseline, discussed in the above section 3.1.

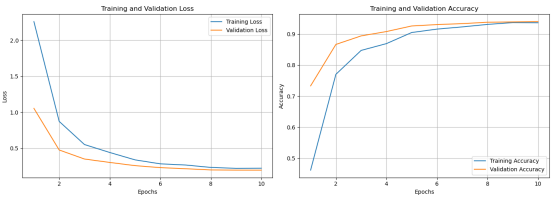

Figure 7: Prediction

The left plot (Fig.7), showing the loss, demonstrates an ideal convergence for the model. In the early epochs, both training and validation losses decrease sharply, indicating the optimizer is taking large, effective steps away from its poor initial state and descending towards a low-loss region. The validation loss then trying to get stabilized around epoch 6, indicating that the model has converged to a very effective local minimum for unseen data. While the training loss continues a slight downward trend, the stable validation loss confirms that further training would likely not lead to a better, more general solution, but would instead just be over-optimizing that could have led to effect the model to performs poorly on new, unseen data.

The right plot (Fig.7), the model achieves a high peak validation accuracy of around 94% on the dataset, which clearly shows that the model is not underfitting, stating it has successfully learned the complex features of the dataset. Furthermore, because the validation accuracy plateaus and does not begin to decrease as training continues, it also shows no signs of significant overfitting. An interesting point is that in the first epoch, the validation accuracy is higher than the training accuracy. This is a common and expected result when using heavy data augmentation, as the modified training images are more challenging for the model to classify than the new, unseen validation images.

4.2 Model Evaluation

Predicting all applicable labels for a given image is known as multi-label classification. Compared to the standard multi-class case, it is considerably more challenging to annotate training data for multi-label classification [9]. When the number of potential labels is large, human annotators find it difficult to mention all applicable labels for each training image. As a result, multi-label training data is often plagued by false negatives. The training and validation addresses the hardest version of this problem, where annotators provided only one relevant label for each image, resulting in training sets with only one confirmed positive label per image and a large set of unconfirmed negatives.

4.2.1 Phase 1: Single-Label Training

The core of the training strategy is to treat the model to become an expert at discriminating between a set of mutually exclusive categories. For any given image, the single provided label forms the sole positive pair. All other potential labels, when paired with that image, are treated as implicit negative pairs. The training objective is to maximize the model's confidence in the single positive pair while simultaneously suppressing its confidence for all negative pairs associated with that image.

This objective is achieved using the *CrossEntropyLoss* function. This loss function is critical because its internal Softmax operation creates a competitive environment among the output classes. The Softmax function normalizes the model's output scores into a probability distribution that sums to one. This means for the confidence of the positive pair to increase, the confidences of the negative pairs must proportionally decrease. This forces the model to learn the fine-grained features that not only define the positive class but also actively differentiate it from all other classes. This results in a highly refined internal feature space and a model that is an expert at discriminating between categories.

4.2.2 Phase 2: Multi-Label Inference

While the training was competitive, the goal during inference is collaborative: to understand all the plausible concepts the model "sees" in an image. To achieve this, the competitive Softmax function is replaced with the non-competitive *Sigmoid* function. Sigmoid evaluates each class independently, mapping its raw output score (logit) to an individual confidence score between 0 and 1. The score for one class has no bearing on the score of another.

This approach works precisely because the model has already undergone the rigorous discriminative training. The logits produced by this expert model are rich with information; visually or contextually similar classes will naturally have high logit values. The Sigmoid function acts as a calibrated lens, translating these high logits into high independent confidence scores, allowing multiple relevant labels to surface.

A highly confident model produces extreme raw outputs (logits). When these extreme scores are fed into the steep Sigmoid function, the resulting confidence scores are pushed to nearly 0 or 1. This "hardens" the output, hiding valuable information about plausible secondary labels, as their scores are also crushed towards zero.

Temperature scaling solves this by "softening" the logits before they enter the Sigmoid function. By dividing every logit by a temperature T (e.g., 4.0 in my case, this was obtained by just doing experiment), the scores are squashed closer to zero. This ensures the Sigmoid function produces a more nuanced spectrum of probabilities, allowing secondary, contextually relevant labels to appear with meaningful confidence scores instead of being lost. It effectively calibrates the output to be less binary and more interpretive.

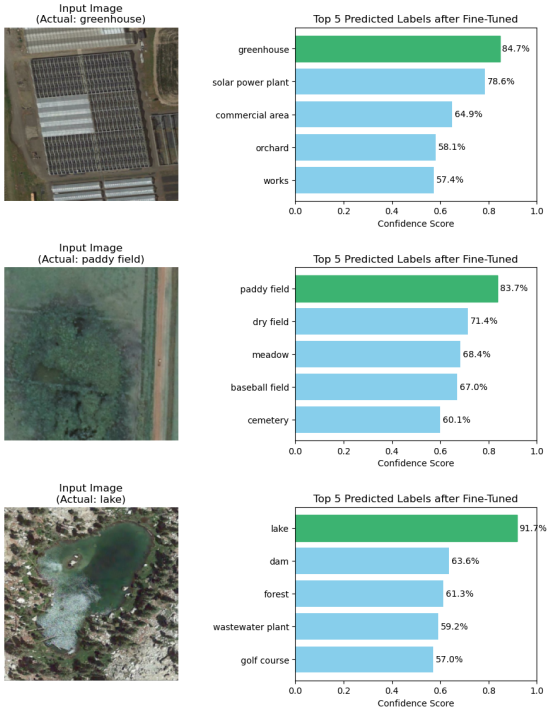

Figure 8: Performance Analysis

So the result obtained by performing this task (Fig.8) show that the model does more than just get the right answer, it also understands the context of the image, and hence that was ultimate goal for fine tuning the model to understand the context of the image, even though it's tough to figure out the right image label to classify when using satellite image, even sometime human's struggles in identifying problem with the image. As shown in the first set of image the original label for the image was "greenhouse" as it predicted correctly, but on the other hand it also look like solar plant image, even it somewhat look alike commercial area consist of buildings.

4.3 Finding: Evaluating the Base SigLIP Model

In addition to fine-tuning, a separate experiment was conducted to evaluate the general-purpose capabilities of the original, pre-trained SigLIP model. Using the Hugging Face pipeline [10], a

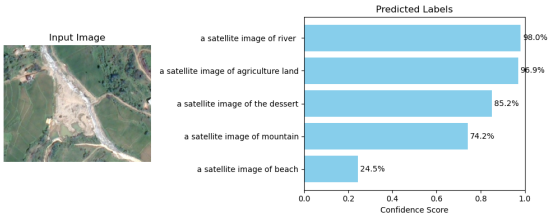

Figure 9: SigLIP Zero-shot Pipeline

random satellite image was tested against a set of arbitrary, pre-defined prompts that were not part of the main dataset .

This result state (Fig.9) the base model's powerful zero-shot classification ability. Without any specific training for these new labels, the model was able to produce a ranked list of confidences, demonstrating its flexibility to handle open-ended, user-defined classification tasks.

4.4 Challenges

During the experimental phase of this project, an alternative loss function, BCEWithLogitsLoss (Binary Cross-Entropy with Logits), was tested. While powerful for its intended purpose of multi-label classification, it proved to be unsuitable for this specific training setup and led to significant underfitting not learning atall. The core of the problem lies in a fundamental mismatch between the loss function's design and the nature of the training data. BCEWithLogitsLoss treats each class as an independent "yes/no" question, which is ideal for scenarios where an image can have multiple correct labels. However, the dataset was structured for a single-label task, where each image's label was represented as a one-hot vector, a vector with a single 1 for the correct class and 0s for all others. When training with this setup, the model received a confusing and unbalanced learning signal. For every single positive instruction, it received fifty negative ones, creating an overwhelming incentive to predict low scores for all classes to avoid penalties. This strategy prevented the model from ever becoming confident in the correct predictions, leading it to fail to learn the essential features from the data. The resulting poor performance on both the training and validation sets confirmed that the model was underfitting, making this loss function an inappropriate choice for this specific training task.

Conclusions

This result successfully demonstrated a complete transfer learning pipeline for adapting the large-scale, pre-trained SigLIP model to the specialized task of satellite image classification. Through a strategic approach involving various techniques, the model's performance was significantly improved from its zero-shot baseline, achieving a final validation accuracy of approximately 94%. Beyond simple accuracy, the project also revealed that by using a flexible inference strategy with a Sigmoid activation, the specialist model could provide context-aware predictions, identifying multiple relevant labels within a single scene. Ultimately, this work serves as a practical demonstration of how to effectively transform a general-purpose vision-language model into a high-performance specialist, tailored specifically for a target domain.

References

[1] Gitlab. Gitlab. https://git.oth-aw.de/6082/deepvision_siglip_finetuned#, 2025.

[2] Konstantinos Poulinakis. Multimodal deep learning guide, 2024. https://www.v7labs.com/blog/multimodal-deep-learning-guide.

[3] Nikolaj Buhl. Contrastive learning, 2023. https://encord.com/blog/guide-to-contrastive-learning/.

[4] Alec Radford et al. Learning transferable visual models from natural language supervision. 2021.

[5] pinecore. Openai clip. https://www.pinecone.io/learn/series/image-search/clip/, 2024.

[6] Xiaohua Zhai, Basil Mustafa, Alexander Kolesnikov, and Lucas Beyer. Sigmoid loss for language image pre-training. *arXiv preprint*, 2023.

[7] pytorch. Data augmentation. https://docs.pytorch.org/vision/main/transforms.html/, 2017.

[8] Google Brain. optimizer. https://github.com/lucidrains/lion-pytorch/, 2024.

[9] pytorch. Transfer learning for multi-label classification from a single-label model. https://discuss.pytorch.org/, 2018.

[10] Hugging Face. Siglip. https://huggingface.co/docs/transformers/en/model_doc/siglip, 2023.