Improving Cross-View Matching with Self-Supervised Learning

Master Thesis

Jianfeng Cui

Improving Cross-View Matching with Self-Supervised Learning

Master Thesis

by

Jianfeng Cui

Student Name	Student Number
Jianfeng Cui	5225256

Supervisor:	J.F.P. Kooij, Z. Xia
Project Duration:	February, 2022 - November, 2022
Faculty:	Faculty of Mechanical, Maritime and Materials Engineering (3mE), Delft

Cover:	Photo by Kayle Kaupanger on Unsplash (Modified)
Style:	TU Delft Report Style, with modifications by Daan Zwaneveld

TU Delft

Contents

1 Introduction **1**
- 1.1 Visual geo-localization and cross-view matching 1
- 1.2 Self-supervised learning 3
- 1.3 Research question and main contributions 5

2 Related Works **8**
- 2.1 Cross-view matching open challenges and methods 8
  - 2.1.1 Drastic change in viewpoint 9
  - 2.1.2 Orientation misalignment 10
  - 2.1.3 Center location misalignment 11
  - 2.1.4 Temporal scene changes 11
- 2.2 Self-supervised learning and contrastive learning 12
  - 2.2.1 MoCo, SimCLR and MoCov2 13
  - 2.2.2 BYOL, SimSiam 15
- 2.3 Related works: summary 17

3 Method **19**
- 3.1 Self-supervised pre-training by contrastive learning 20
  - 3.1.1 The contrastive learning framework: MoCov2 20
  - 3.1.2 Different pre-trained weights 23
- 3.2 Image Retrieval Architecture and Loss Function 24
  - 3.2.1 The image retrieval framework 25
  - 3.2.2 The soft-margin triplet loss 25
- 3.3 Feature distribution visualization and uniformity metric 25
- 3.4 Add uniformity loss and data augmentation directly 27
  - 3.4.1 Uniformity loss 27
  - 3.4.2 Data augmentation 27

4 Experiments **29**
- 4.1 Datasets and evaluation metrics 29
- 4.2 Implementation details 31
- 4.3 Cross-view matching with contrastive learning pre-training 31
- 4.4 Verification on swapped dataset 33
- 4.5 Feature distribution 34
  - 4.5.1 Settings 34
  - 4.5.2 Observations and findings 37
- 4.6 Uniformity loss and data augmentation 37

5 Conclusion **40**
- 5.1 Answers to the research questions 40
- 5.2 Discussion and future works 41

References **42**

6 Appendix 46
6.1 The network architectures 46
6.2 MoCov2 data augmentation 47
6.3 Localization heatmap 47
6.4 Visualization of feature map 48
6.5 A unified version of MoCov2 and cross-view matching 49

1 Introduction

In the past few years, there has been great interest and progress in the field of Intelligent Vehicles for both academia and industries. Intelligent Vehicles are considered a promising way that could reduce road accidents and traffic congestion, to achieve safer and more efficient transportation in the future.

According to SAE International (Society of Automotive Engineers), there are 6 levels of autonomous driving, ranging from no driving automation (level 0) to full driving automation (level 5) [8]. Currently, it is already very common to see the application of various driver assistant systems installed on modern vehicles, e.g., lane keeping, and adaptive cruise control. However, these driver assistant systems are limited to aid the driver, while the driver is responsible for overall control of the vehicle or under certain driving conditions. There are still many technical challenges to be solved.

One of the key challenges for vehicles with high driving automation is localizing the vehicle in the environment. An autonomous vehicle that aims to travel from one place to another need to be aware of its location globally. Many sensors could be utilized to help localization, such as by sensing the local environment (e.g., by the camera, Radar, or LiDAR) or the vehicle's ego-motion (e.g., odometry, Inertial Navigation System). GNSS (Global Navigation Satellite System), such as GPS, could provide absolute and global positioning information to determine the geographic position of an object (i.e., geo-localization). Furthermore, different information from different sensors could also be merged together to improve robustness.

In this chapter, we briefly describe two main topics related to our work: cross-view matching and self-supervised learning. And then we propose our research question and introduce our work in the last sub-section.

1.1. Visual geo-localization and cross-view matching

Geo-localization is the process of determining the geographic location of an object or person using various methods. Specifically, *visual geo-localization* is the task of identifying the location where the image was captured only based on its visual information [40]. Once the location of the image has been determined, it can be displayed on a map, allowing the user to see exactly where the photo was taken. This can be useful for a variety of applications, such as providing location-based information or services. It has great potential for noisy GPS correction [4, 56] and navigation [27, 32] in crowded cities.

A major technique to perform visual geo-localization tasks is *image retrieval*, which is based on query-reference image matching. The query image is the image that is being used to determine the location where it was captured. The reference image is an image with a known location that is being compared to the query image in order to determine the location. For example, if a person takes a picture with their smartphone and wants to determine the location, the query image would be the picture that he took, and the reference image would be a known image from a database of known locations. The query and reference images would be compared using image recognition algorithms or other techniques, and the location of the query image would be determined based on the correspondence between the two images. In general, the query image is the input to the geo-localization process, and the reference image is the source of information that is used to determine the location of where the query image was captured.

Visual geo-localization can be performed using different types of images, including satellite imagery and ground-level imagery. Due to the complete coverage and easy access of satellite images from Google Map API [1], a thread of works [61, 42, 49, 54, 23, 24, 55, 16] focuses on *cross-view geo-localization*, where the ground-level images are query images and satellite images are reference images. Figure 1.1 visually shows the idea of cross-view matching. The availability of abundant satellite images all over the world naturally forms a huge database for us to perform visual geo-localization at a very low cost. Researchers are able to establish their own databases for different regions and cities, including both rural [28, 57] and urban areas [50, 61]. Otherwise, collecting ground-level images to establish the reference image database requires us to navigate over the target area with well-equipped localization devices and subsequent data preparation, which consumes huge labor and financial resources.

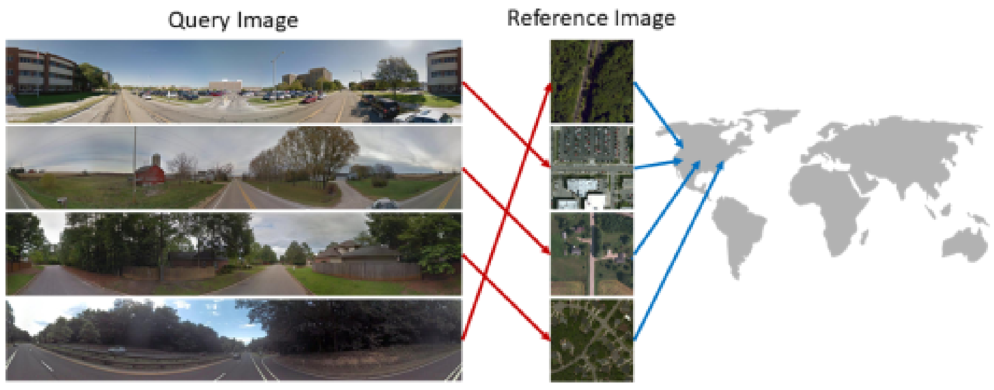


Figure 1.1: Cross-view matching: match query ground-level images to reference satellite images. Based on image appearances, query ground-level images are matched with reference satellite images (indicated by red arrows). Tagged locations of reference images are the final location predictions for the queries (indicated by blue arrows). The figure is taken from [52]

In order to find the correspondences between the query and reference images, *metric learning* [30] is introduced. In metric learning, the goal is to learn a distance function that maps objects from a dataset to points in an embedding space such that the distances between the points accurately reflect the similarity between the corresponding objects. This is typically done by defining a loss function that measures the quality of the learned distance metric and then using optimization algorithms to find the parameters of the distance function that minimize the loss. Once the distance function has been learned, it can be used to compute the similarity between objects in the dataset by mapping them to points in the embedding space and then measuring the distance between the points. For example, in image matching, the objects may be images and the embedding space may be a space in which the distance between two images reflects their similarity. By mapping the images to points in the embedding space, it becomes possible to compare their distances and determine which images are the most similar. Typically, the loss function is designed to pull the representations of similar images (e.g., a query and its positive reference sample) close together and push dissimilar ones (e.g., a query and its negative reference sample) far apart in the embedding space. By using an embedding space, the learned distance function can be applied to the objects in the dataset in a consistent and meaningful way. Figure 1.2 briefly shows the idea of cross-view matching by metric learning.

However, such cross-view matching systems suffer from many open challenges, in which a major one is the drastic viewpoint change between ground-level and satellite images. Ground and satellite

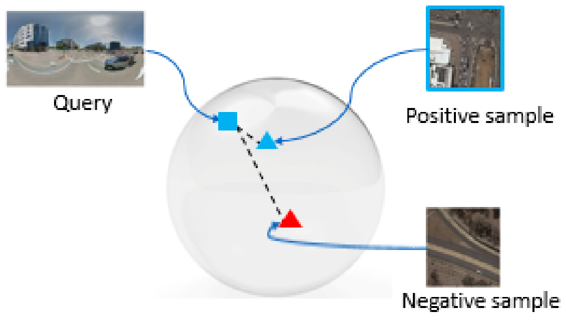


Figure 1.2: Cross-view matching by metric learning. The gray sphere represents the embedding space. The blue square, blue triangle, and red triangle represent the projected data point of the query ground-level image, positive reference satellite image, and negative reference satellite image, respectively. The figure is taken from [52]

views contain very different visual contents, i.e., building facades, trees, and cars occupy the majority of ground image scenes, while satellite images mainly contain buildings, tree tops, and road structures. This challenge causes problems in using traditional handcrafted features such as SIFT [29] to compare similarities [23]. Therefore, concerning the great success of image feature extractors utilizing neural networks [3, 10, 18], most trending cross-view matching methods adopt neural networks for learning image representations which are highly related to the topic of deep metric learning. Generally, researchers train a two-stream CNN (Convolutional Neural Network) framework and employ metric learning loss [23, 49] to train the network.

Apart from the drastic viewpoint change, in cross-view matching, there are also many other challenges including orientation misalignment [28, 43, 49], center location misalignment [61] and temporal scene changes [38]. These open challenges pose higher requirements for image representations generated by the neural network. A good representation in cross-view matching is expected to be semantically rich, aware of geometric information, and even invariant to the temporal scene changes.

1.2. Self-supervised learning

Another field of research area related to this work is self-supervised learning. In recent years, *self-supervised learning* methods [22, 36, 58, 20, 11, 34, 13, 53, 33, 21, 5, 7, 15] have achieved significant performance on various vision tasks by representation learning even without human-tagged labels for training.

To start with, *supervised learning* is a type of machine learning where a model is trained to make predictions based on a set of labeled data. In supervised learning, the data is labeled with the correct output or target, which the model uses to learn how to make predictions on new data. The label is the correct output or target for a given input, and it is used by the model to learn the relationship between the input and the correct output. For example, in image classification, the input might be an image and the label might be the correct category or class that the image belongs to (e.g. “cat” or “dog”). The model would then use the labeled data to learn how to make predictions on new images, such as whether a given image is of a cat or a dog. The goal of supervised learning is to minimize the difference between the predicted output and the correct label so that the model can make accurate predictions on new data.

Then, by definition, self-supervised learning is a version of unsupervised learning where data itself provides supervision [25]. In order to train deep neural networks, a large amount of labeled data is generally needed to achieve better performances in visual applications. To relieve the cost of annotating large-scale datasets, various self-supervised learning methods are designed to learn general image features from unlabeled raw images without using human-tagged labels. Technically, the difference between unsupervised and supervised learning is that the models in the first work with unlabeled datasets while the latter work with labeled data. Consequently, unsupervised learners have no idea about e.g.

the names, classes, or values the inputs are assigned with. While supervised learners learn on the basis of examples, unsupervised learners mostly try to find meaning or structure within the data. By design, unsupervised learners could be used to exploit (hidden) useful structures in the data which have the potential to be deployed as models that learn data-specific features, store large datasets efficiently or generate new realistic samples.

To intuitively interpret the idea of self-supervised learning, Figure 1.3 is presented to show an example that designs a self-supervised scheme for image depth estimation [12]. The framework is an auto-encoder, in which the encoder (part 1) is a traditional convolutional neural network with stacked convolutions and pooling layers that maps the left image I_1 of the rectified stereo pair into disparities (scaled inverse depth). The decoder (part 2) takes the right image I_2 and the encoder output as inputs, which synthesizes a backward warp image I_w by moving pixels from the right image I_2 along the scan-line. By constructing a reconstruction error (Part 3) to match the reconstructed image I_w with the encoder input, the network learns to predict correct depths. It could perform single-view depth estimation during test time. In this way, rather than learning with direct human-annotated labels, the model forms supervision by itself.

Self-supervised learning approaches like the example above are task-specific. In fact, most existing self-supervised learning methods follow a general scheme shown in Figure 1.4. Generally, it has two phases:

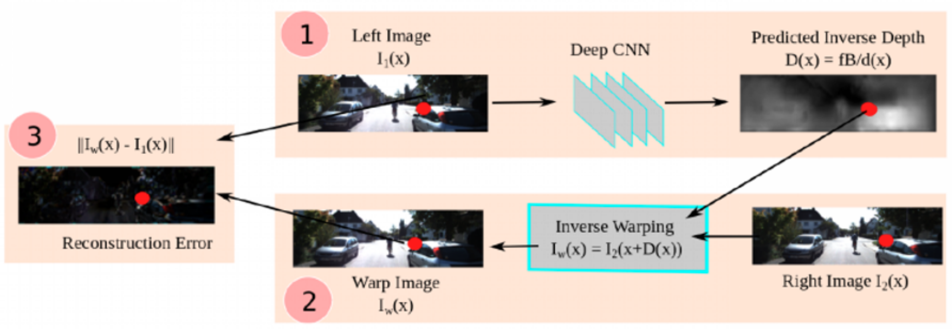


Figure 1.3: A self-supervised scheme for image depth estimation which utilizes stereo image pair. The figure is taken from [12]

1. A pretext task is designed for neural networks to solve and image representations could be learned through this process for accomplishing the pretext task. Pseudo labels for pretext tasks can be generated automatically without human tagging.
2. Transfer the pre-trained network to downstream tasks by fine-tuning with human-annotated labels.

Pretext tasks can be predictive tasks, generative tasks, contrasting tasks, or a combination of them. For example, it could be designed as pretending there is a part of the input we do not know and predicting it, e.g., hiding some patches over an image and recovering the original pixels [36]. When the pretext tasks are designed as contrasting tasks, it performs *contrastive learning*, which is one of the most representative methods of self-supervised learning. In contrastive learning, a model is trained to predict the relationship between two input items without the need for human-provided labels. One common technique used in contrastive learning is random data augmentation, where the input data is transformed in various ways to create multiple versions of the same data. For example, in image recognition, an image of a cat might be rotated, flipped, or cropped to create multiple versions of the same image. These augmented versions of the data can then be used as positive examples in the contrastive learning process, allowing the model to learn to recognize the same object despite differences in its appearance. Data augmentation can help improve the performance of contrastive learning by providing the model with more examples to learn from.

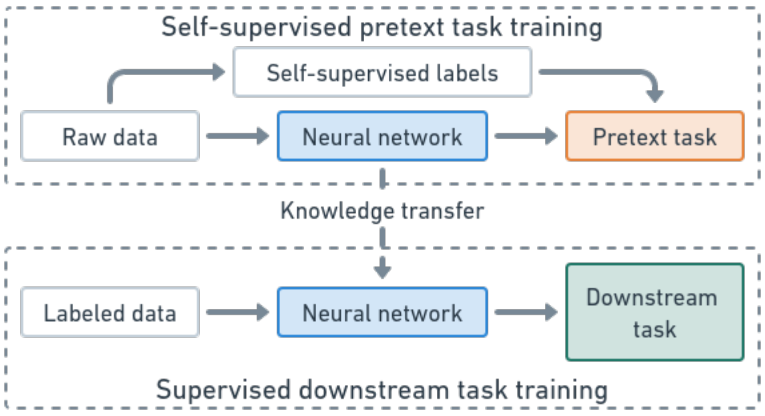


Figure 1.4: A typical pipeline of self-supervised learning. The image representation could be learned by training to solve a designed pretext task. After that, the learned parameters serve as a pre-trained model to solve other downstream visual tasks by fine-tuning

Downstream tasks are usually common computer vision tasks that we are trying to solve for specific applications, such as image classification and object detection, which are target tasks that we are trying to solve for applications. It could also be viewed as a way to evaluate the quality of features learned by self-supervised learning. These applications can greatly benefit from the pre-trained models when training data are scarce.

1.3. Research question and main contributions

The above introduction of cross-view matching and self-supervised learning brings us to the main topic of our work. The quality of the learned image features by self-supervised methods could be verified by the two stages in Figure 1.4: the learned parameters by self-supervised learning are employed as pre-trained models and then fine-tuned on downstream tasks. The performance of the transfer learning on these high-level vision tasks demonstrates the generalizability of the learned features. If networks gained from self-supervised learning can learn general good features, then the pre-trained models can be used as a good starting point for other vision tasks that require capturing similar features from images. Therefore, it seems that such self-supervised learning methods relying on pretext tasks could serve as a general approach to gain good pre-trained weights for plenty of different downstream tasks. However, although existing self-supervised learning methods have been proven successful in learning image representations for downstream tasks such as image classification, object detection, and so on, to the best of the authors' knowledge, there is currently no or little research verifying the effectiveness of self-supervised learning methods on cross-view matching. Moreover, notice that a major difference between cross-view matching and other downstream tasks that are already proven can benefit from self-supervised learning is that cross-view matching requires learning features within two different modalities (i.e., ground-level image and satellite image), while others are always extracting useful feature from a single-modality image which is consistent with the pretext tasks, despite of their specific downstream task objectives. This leaves a question of whether general self-supervised learning methods also work on cross-view matching. Figure 1.5 shows the brief idea of applying self-supervised learning methods on cross-view matching.

Moreover, an important property of cross-view matching is that satellite images are globally available and easily accessed. This differs from many other common computer vision tasks where the whole

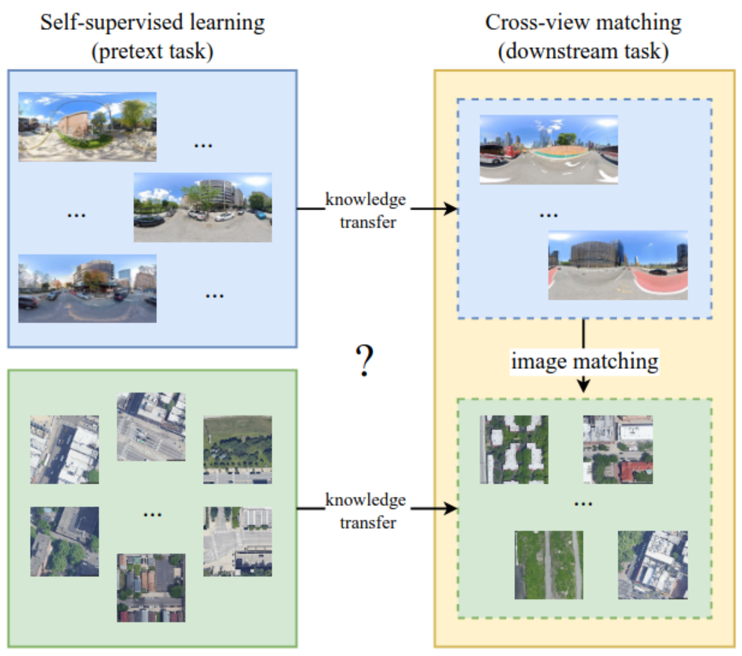


Figure 1.5: The research question: will cross-view matching benefit from self-supervised pre-training by contrastive learning? The left column: pre-training is performed using standard self-supervised learning methods on ground-level images (in the blue block) and satellite images (in the green block) separately. The right column: pre-trained models are then transferred to the downstream task training, which is cross-view matching

data distribution of the test set is unknown. This gives us the potential to make use of the raw satellite image data as a piece of prior knowledge to enhance the representation extraction. Motivated by the discussion above, we propose the following research question:

As a downstream task, will cross-view matching benefit from self-supervised pre-training by contrastive learning?

Furthermore, We divide this question into two sub-questions:

1. *Will the performance of cross-view matching be improved by pre-training using contrastive learning?*
2. *How will the image representations generated by the cross-view matching network be different by pre-training using contrastive learning?*

In this work, to answer the first sub-question, we experimented with cross-view matching methods with the self-supervised learning approach and empirically verified the performance improvement brought by self-supervised learning. Moreover, to answer the second sub-question and discover the interpretation, by visualizing the representation distribution in the feature space, we observed different behaviors with or without self-supervised pre-training. We summarized the main contributions as follows:

1. This work verifies the effectiveness of self-supervised contrastive learning on cross-view matching. Self-supervised pre-training brings stable performance improvement on different cross-view matching network settings.
2. An in-depth interpretation of analyzing the representation distribution in the embedding space. By self-supervised learning pre-training, before the downstream task fine-tuning epoch starts,

the network can already distinguish images from different cities and generate less homogeneous representations. A more uniform distribution is achieved through pre-training.

2 Related Works

2.1. Cross-view matching open challenges and methods

Visual geo-localization answers the question of where an image was captured in a region. The task is commonly addressed as a query-reference matching problem. Typically, the query image is an image with an unknown location, and references are from an image database with known GPS coordinates. The scenes captured in the image can experience huge variations, such as under different times and weather or containing dynamic objects. But useful information like landmarks and buildings does help the model identify places based on the image appearances.

Traditional visual geo-localization approaches are performed with both the query and reference image taken from the same viewpoint, usually, both from the ground view [39, 49]. However, the vehicle cannot assume it has been everywhere before. The ground-view image database with tagged GPS coordinates commonly requires crowd-sourcing, e.g., photos from Flickr, but the coverage is still very sparse. Gathering data to establish an area-specific database is also money- and time-consuming.

To address the limitation of ground-view image geo-localization, some researchers have explored methods using satellite images as references, which is called cross-view image geo-localization. Note that the satellite image database is almost globally available and easily accessed, by the application of the GNSS system and also thanks to free public releases of geo-tagged satellite images from companies such as Google. As a result, in terms of the data source, the matching and localization could be performed all over the world. Existing works for cross-view geo-localization methods generally adopt a two-stream CNN framework to extract different features for two views, then learn an embedding space where images from the same GPS location are close to each other.

However, such cross-view retrieval systems suffer from several open challenges. To conclude, there are mainly four open challenges for the cross-view matching task:

- Drastic change in viewpoint: the scene is contained in two different viewpoints (i.e., ground view and bird-eye view) in ground-level and satellite images respectively, causing different image contents.
- Orientation misalignment: satellite patch represents the same scene as that of the ground-level image, no matter what the relative orientation is.
- Center location misalignment: the center of the ground-level image and its corresponding ground-truth satellite image are not exactly aligned, but with offsets.
- Temporal scene changes: the satellite image database can be outdated compared with the captured ground-level image, causing it hard to be retrieved due to different scene appearances.

In the following sections, we introduce several representative methods in terms of their major targeted challenges.

2.1.1. Drastic change in viewpoint

The “cross-view” attribute and usage of satellite images pose the most tricky part of cross-view matching. In this case, the query is a ground-level image, taken from the ground facing front, but the reference image will be a satellite image patch, which is in bird-eye-view. Ground and satellite views contain very different visual contents, i.e., building facades, trees, and cars occupy the majority of ground image scenes, while satellite images mainly contain buildings, tree tops, and road structures. This is caused by the drastic viewpoint change between the two different image sources and serves as the main challenge for cross-view matching.

A representative approach that addresses this issue is SAFA [42]. By leveraging the prior geometric knowledge between the two domains, SAFA proposed a pre-processing technique that adopts a polar transformation on the satellite image. This polar coordinate transforms satellite images, making them approximately aligned with a ground-view panorama. Instead of enforcing neural networks to learn the geometric gap between two modalities implicitly, SAFA explicitly transforms the satellite images which roughly eliminates the gap. In doing so, the polar transformation eases the task of learning multiple correspondences (i.e., geometry and feature representations), which facilitates the network’s convergence. The polar transformer relies on such an observation: pixels lying in the same azimuth direction in a satellite image approximately correspond to a vertical image column in the ground-level panorama image. By intuition, polar transformation ”unwraps” the satellite image along the azimuth angles to convert the image contents in the bird-eye view to a pseudo-ground-level panorama view. In practice, the polar transformation takes the center of each satellite image as the polar origin and the north direction as the angle of 0 degrees. Therefore, the corresponding panorama image in the dataset of the satellite image is expected to align well with the center position. Otherwise, the polar-transformed satellite image will be significantly different from the ground-truth one. However, in practice, since the database of satellite patches is usually sampled from the satellite map in a certain resolution, for a ground-level image at a random location, it is not realistic that the perfectly aligned satellite image is included in the database. Therefore, it is inappropriate to apply polar transformation in real-world scenarios.

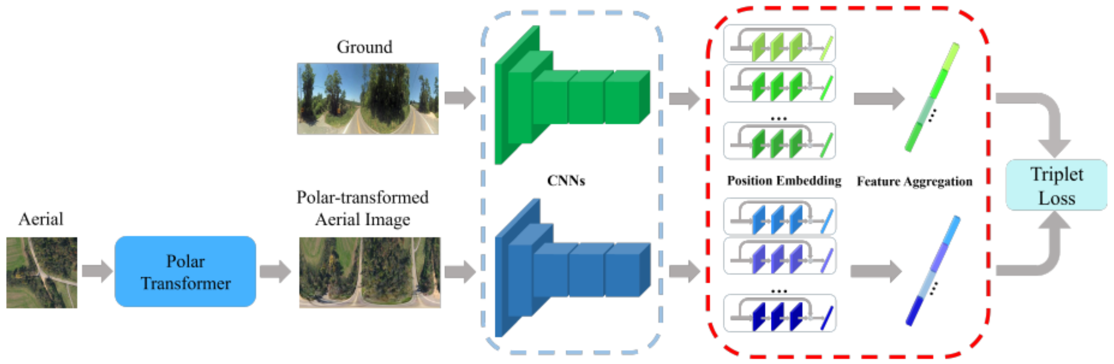


Figure 2.1: SAFA architecture. The figure is taken from [42].

Another approach that tries to solve the drastic viewpoint change is CVFT [41]. Instead of pre-processing the data samples like polar transformation, the authors of CVFT proposed a Cross-View Feature Transport (CVFT) layer for domain feature transferring which facilitates cross-domain feature matching. CVFT explicitly models the domain gap between satellite and ground-level images by a transport matrix. The method is inspired by the Optimal Transport (OT) theory, which was originally developed for finding the optimal transport plan that can best align two probability distributions. Traditionally, Optimal Transport is a linear programming problem that is computationally inefficient. To solve this problem, a different version of OT called Sinkhorn solver which is based on entropy regularization was adopted. Finally, the feature transport problem was converted into a convex problem and could be solved by a Sinkhorn solver.

Another idea of bridging the domain gap between ground-level and satellite images comes from image synthesizing networks. [57] synthesized ground-level information from top-view satellite inputs. They learned to map semantic labels from the satellite view to the ground view and use them to create ground view information. A more recent work [31] proposed to generate ground views from satellite images by utilizing depth maps and semantic labels. These works explored ways to generate realistic views and do not explicitly apply them to the geo-localization problem.

[37] proposed a feature fusion training strategy including the use of GAN [14] in which the features from synthesized satellite views are fused with corresponding ground view features. It is the first to use pre-trained synthesized images to train a retrieval network for geo-localization. However, it is done in three stages, not end-to-end. Moreover, they obtained less accurate retrieval results than methods based on polar transformations. This suggests that while networks like GAN create images that look more realistic, a transformation relying on geometric information like polar transformation is better to map the content of the images across the two domains for cross-view matching problems.

Therefore, SAFA-GAN [45] proposed a novel geo-localization method that is trained jointly for the multi-task setup of both synthesizing ground images from satellite images and retrieving cross-view image matches. The authors designed a single network for both of these tasks which can be trained in an end-to-end manner.

The proposed network is composed of a generator that maps the ground view to a synthetic satellite view image, a discriminator which learns to distinguish between the generated satellite views and real sampled satellite view images, and a SAFA-based retrieval sub-network that is used to extract the features of satellite and ground-level images for cross-view geo-localization. The intermediate features from the generator are then reused in a retrieval sub-network for cross-view geo-localization. By using the intermediate features, the two tasks (image retrieval and image generation) mutually learn from each other. Thus, the learned feature representation is informative for both tasks.

2.1.2. Orientation misalignment

Another issue related to cross-view matching is orientation misalignment. In a real-world scenario, typically we want the method to be able to retrieve the correct image patch with the same location as the query ground-level image, which means the satellite patch represents the same scene as that of the ground-level image, no matter what the relative orientation is. This issue is addressed as an orientation misalignment problem. Basically, cross-view matching methods can be divided into two streams on this problem: 1. simply ignore this misalignment, i.e., directly work with the dataset which includes misalignment and let the network learn orientation-invariant features 2. explicitly handle the misalignment by specific design aspects, e.g., predict the relative orientation.

Lending Orientation [28] proposed an efficient method to fuse the heading direction of ground-level images into a network and learn more robust features. The author borrowed ideas from the color-coded map and designed a similar mechanism to encode the pixel-wise orientation information. Ground view and aerial images are both encoded with orientation information separately and fed into a Siamese model.

Specifically, in the Lending Orientation method, the per-pixel orientations in ground-level are represented as azimuth and altitude (spherical angles), and those in satellite images are represented as azimuth and range (polar coordinates). These orientation values are encoded using HSV color channels, which are concatenated with the original image RGB channels as the total input.

Another novel method addressing the orientation misalignment issue is DSM [43]. DSM focused on cross-view geo-localization on limited field-of-view images and their heading orientation. The author proposed a Dynamic Similarity Matching (DSM) module to measure the feature similarity between two views in a sliding window manner and estimate the orientation at the same time. Specifically, DSM is fed with two feature maps from ground image and aerial image feature extractors. DSM slides on the ground features as to compute the inner product with the aerial features. The location of the highest inner product value indicated the orientation and similarity. Notice that the polar transformation is one of the pre-processing steps in this paper.

DSM is the first work that jointly estimates the position and orientation of a query ground image regardless of its Field-of-View. It is reasonable that the authors named it ”dynamic”: the network is able to predict the orientation, and then do the shift crop on the panorama image to align the two images such that they are looking in the same direction.

Learn Alignment [60] studied the effect of alignment information on previous cross-view geo-localization models. The authors found that (Table 1. in [60]):

- Training with randomly rotated aerial images (the alignment information is therefore not available) yields a performance drop on the aligned validation set compared with training with aligned images. But this trained model is able to perform well on the randomly rotated validation set.
- On the other hand, the model trained with aligned images has an extremely low accuracy on the randomly rotated validation set.

These results indicate that the alignment information indeed has a great impact on accuracy. And training without alignment information makes the model generalize better.

2.1.3. Center location misalignment

As mentioned before when discussing the polar transformation proposed by the SAFA network, there is normally a wrong assumption in existing datasets: each query ground-view image has one corresponding reference aerial-view image whose center is exactly aligned at the location of the query image. This is not practical for real-world applications, because the query image can occur at arbitrary locations in the area of interest.

However, few of the cross-view matching methods consider a sub-image level localization beyond the image-level retrieval or multiple reference images for training. VIGOR [61] is a recent work that aims to address this issue. VIGOR proposed a new dataset that has a many-to-one correspondence between ground and satellite views. This dataset densely sampled four cities in the U.S. and grabbed panoramic images from Google Street View (GSV) [2]. The authors proposed three categories to define cross-view matching: positive, semi-positive, and negative. The definitions of positive and negative pairs are the same as in previous datasets. In the semi-positive pairs, the ground view location appears in the aerial image but not in the center region. By doing so, cross-view geo-localization becomes a many-to-one retrieval problem rather than a one-to-one retrieval problem. This configuration is closer to real-life deployment.

The authors also proposed their method to solve the problem. For the image retrieval task, the proposed model adopted the same architecture as SAFA (without polar transformation). Moreover, to estimate the offset of the camera locations with respect to the center of the satellite image, an offset prediction sub-network was added. By feeding the concatenation of aerial features and ground features, this sub-network can predict the camera location on the satellite image. The author also proposed an IOU-based loss function to guide the network in learning features from semi-positive samples.

2.1.4. Temporal scene changes

Another challenge of referencing a satellite patch is the temporal variation of places. For instance, after the photographs have been captured, new buildings might appear or vegetation patterns might change. This means that to cover all variations that could occur in the same location, we need to update satellite images from every possible location and at every possible time, which is not practical. The temporal scene change limits the applicability of existing cross-view image geo-localization methods and challenges the generalization ability of cross-view matching networks.

Seeing the Unseen [38] proposed a novel data augmentation pipeline for cross-view geo-localization. The author utilized the segmentation map from existing models to cut out the objects (buildings, sidewalks, sky, etc.) in the ground view images to force the network to learn from the unseen objects and perform unseen object matching. Specifically, an off-the-shelf image segmentation module is used to obtain class-specific masks corresponding to the ground view image. These masks are used to create augmented samples in *keep* and *remove* mode respectively. Finally, the augmented samples along with their original versions are used for training in a Siamese pipeline. Note that the corresponding satellite image is kept unchanged.

2.2. Self-supervised learning and contrastive learning

Unsupervised learning is a type of machine learning where a model is trained to find patterns or relationships in a dataset without the need for human-provided labels. In unsupervised learning, the model is given a dataset and must discover the underlying structure or relationships within the data on its own. This is typically done using algorithms that can identify clusters or groups within the data, or that can learn to represent the data in a lower-dimensional space. As mentioned before, self-supervised learning is a type of unsupervised learning where a model is trained to predict the relationship between two views of the same input data. For example, in image recognition, a self-supervised model might be trained to predict whether two different views of the same image (such as the original image and a horizontally flipped version of the image) are the same or different. The goal of self-supervised learning is to learn useful representations of the input data that can be used for a variety of downstream tasks.

Following the general self-supervised learning scheme as shown in Figure 1.4, we could categorize different methods by their pretext tasks. An effective pretext task is reconstructing a corrupted version of an image so that the network is forced to learn useful features to recover the original image. For example, the corruption could be adding noise, hiding some part of the input, or separating image channels (e.g., depth and color information). Representative approaches include several early works: denoising autoencoder [47, 48], context encoder [36], and colorization [58]. Moreover, masked autoencoder [20] produces state-of-the-art performance and effectively makes use of the transformer.

Another effective method as a pretext task is based on visual common sense. For example, we have the ability to solve a jigsaw puzzle that depicts a scene or object based on our real-life experience. By designing such a pretext task, the network is expected to learn useful features to gain such visual common sense. Representative methods are reordering shuffled image patches (i.e., Jigsaw [34]) and predicting the image rotation (i.e., RotNet [13]).

Contrastive learning [17] is a discriminative approach that aims at grouping representations of similar samples closer and diverse samples far from each other, as shown in Figure 2.2. In recent years, many contrastive learning methods achieved state-of-the-art performances, and it is worth mentioning that yet even re-implemented with suitable architectures [26], the methods introduced in previous sections are being outperformed by contrastive methods [21, 5, 7]. Therefore, contrastive learning is considered the main self-supervised learning approach in our work.

Specifically, in a self-supervised manner for the visual application, the pretext task of contrastive learning is commonly chosen as instance discrimination: Firstly, one sample from the training dataset is taken and two transformed versions of the sample are gained by applying appropriate data augmentation techniques. During training, the one augmented images of the original sample are considered a positive sample of another, and the rest of the samples in the batch/dataset (depending on the method being used) are considered negative samples. Then, the model is trained to differentiate positive samples from negative ones. In doing so, basically, the model is forced to learn quality representations to identify that two different images are fundamentally containing the same object/scene. This knowledge is used later for transferring to downstream tasks.

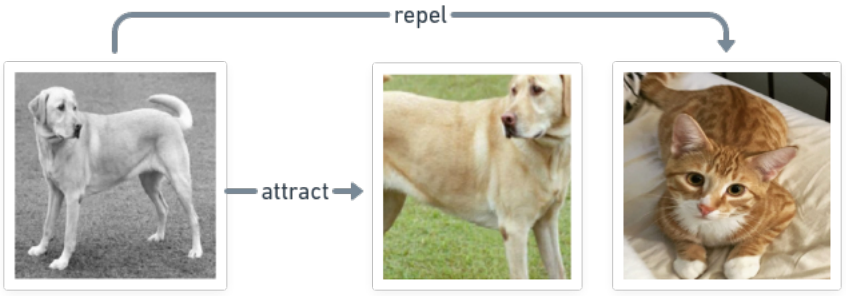


Figure 2.2: Contrastive learning pulls the representations of similar samples together and pushes dissimilar samples away

It could be identified that the notion of contrastive learning is somewhat similar to image retrieval, while the term is more commonly used in the field of self-supervised learning. CL trains encoders to perform a dictionary-look-up task: the encoded query should be similar to its matching key and dissimilar to others. The learning could be formulated as minimizing a contrastive loss [17]. The contrastive loss is a function whose value is low when the query is similar to its positive key and dissimilar to all other keys (negative keys for the query). In the following sub-sections, several representative contrastive learning self-supervised learning methods will be introduced.

2.2.1. MoCo, SimCLR and MoCov2

In practice, contrastive learning methods benefit from a large number of negative samples. Momentum Contrast (MoCo) [21] is a successful work that provided competitive results under the common linear protocol on ImageNet classification. Consider an encoded query q and a set of encoded samples $\{k_0, k_1, k_2, ...\}$ that are keys of a dictionary. There is a single key (denoted as k_+) in the dictionary that q matches. The architecture of MoCo is shown in Figure 2.3. In general, the query representation is $q = f_q(x^q)$ where f_q is the query encoder network and x^q is a query sample. In the same way, the key representation $k = f_k(x^k))$, where f_k is the key encoder and x^k is a key sample.

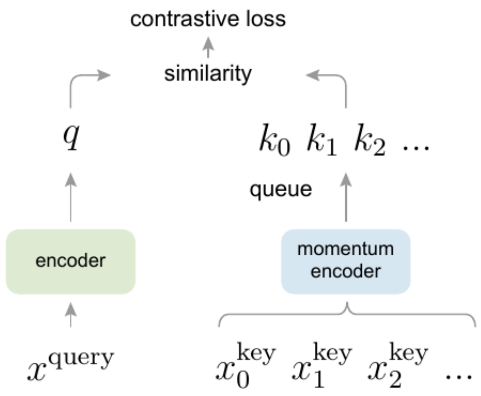


Figure 2.3: Momentum Contrast (MoCo) architecture. The figure is taken from [21].

The authors hypothesized two major properties of the learning method leading to good features: 1. the dictionary is *large* enough to cover a rich set of negative samples, and 2. the encoder for the dictionary keys is kept as *consistent* as possible despite its evolution during training. This motivation leads to the core design of the approach around the momentum encoder, which will be described next.

The dictionary of encoded keys is maintained as a queue, which allows us to reuse them from preceding mini-batches. The samples in the dictionary are progressively replaced: the current mini-batch is pushed to the dictionary, and the oldest mini-batch in the queue is removed. This method enables us to flexibly set the size of the dictionary to be an independent hyper-parameter, which could be much larger than a normal mini-batch size. In addition, removing the oldest mini-batch can be beneficial because its encoded keys are the most outdated and thus the least consistent with the newest ones.

However, the queue-like dictionary makes it intractable to update the key encoder f_k by back-propagation because the gradient should propagate to all samples in the queue. Instead of adopting a naive solution which could be simply copying query encoder f_q to f_k, MoCo proposed a momentum update to address this issue:

$$\theta_k \leftarrow m\theta_k + (1 - m)\theta_q \tag{2.1}$$

where θ_k, θ_q is the parameters of the key encoder f_k and query encoder f_q, respectively. $m \in [0, 1)$ is a momentum coefficient. This momentum updating makes θ_k evolve smoothly. As a result, although the keys in the queue are encoded by different encoders (in different mini-batches), the difference among these encoders can be made small.

In terms of the learning objective, MoCo uses a form of contrastive loss called InfoNCE [35], which measures the similarity by the dot product. InfoNCE loss is a method used to train a model to discriminate between a sample and a set of negative samples. It is commonly used in the field of natural language processing (NLP) to train language models but has also been applied to other areas such as computer vision and recommendation systems. The goal of training a model with NCE loss is to learn a function that can predict the probability that a given sample belongs to a target distribution, given a set of negative samples that do not belong to the target distribution. This is done by defining a probability distribution over the samples and minimizing the cross-entropy between the model's predictions and this distribution.

Another type of contrastive learning mechanism is naturally simplified in an end-to-end manner. In this case, the query encoder f_q and key encoder f_k are both updated end-to-end by back-propagation during training. It uses samples in the current mini-batch as the dictionary, so the keys are consistently encoded by the same set of encoder parameters. A successful end-to-end model is SimCLR [5], which basically means "a Simple framework for Contrastive Learning". Even with neither specialized architectures nor management on the memory of keys like MoCo, it outperformed previous works by that time. The core contribution SimCLR made is the systematical studying of what major components enable good contrastive representation learning. The key findings and major components of SimCLR will be introduced next. The architecture is shown in Figure 2.4.

Firstly, SimCLR proved that the composition of multiple data augmentation operations is crucial in defining the contrastive prediction tasks that yield effective representations. In addition, unsupervised contrastive learning benefits from stronger data augmentation than supervised learning. Same as MoCo as we mentioned before and most of the methods in contrastive learning for self-supervised learning, SimCLR learns representations by maximizing agreement between differently augmented views of the same data example via a contrastive loss in the latent space. In Figure 2.4, t and $t^{'}$ are two specific sampled augmentations from the same family $\mathcal{T}$. SimCLR sequentially applies three simple augmentations: random cropping followed by resizing back to the original size, random color distortions, and random Gaussian blur. By analysis, random cropping makes the pretext task involve predictions from the global to local view or adjacent view. Specifically, the authors also emphasize the effectiveness of combining random cropping with random color distortion, since most patches from an image share similar color distribution, which may lead to shortcuts to solve the task.

Secondly, introducing a learnable nonlinear transformation between the representation and the contrastive loss substantially improves the quality of the learned representations. Apart from the encoder $f(\cdot)$, a small neural network projection head $g(\cdot)$ is set, which maps representations to the space where contrastive loss is finally applied. The authors use an MLP with one hidden layer to obtain $z_i = g(h_i)$, and by experiments, they found it beneficial to define the contrastive loss on z_i instead of representations h_i which is commonly used in previous works.

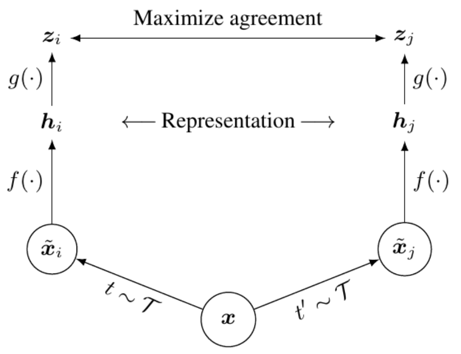


Figure 2.4: SimCLR architecture. The figure is taken from [5].

Thirdly, SimCLR benefits from larger batch sizes (and longer training time). Unlike MoCo which stores the negative samples in a queue, Since SimCLR adopts an end-to-end manner, the dictionary size is directly linked with the mini-batch size. In order to accumulate a greater number of negative samples, it needs large batch sizes. The authors found that in-batch negative example sampling suffices with larger batch sizes. Training longer also provides more negative examples.

However, in fact, SimCLR is still trained with a batch size of 4096 as a default value to provide good results, which is not accessible for every researcher since the batch size is limited by the GPU memory size. The scalability factor with the method remains an issue.

Note that in SimCLR, since the dictionary look-up task is performed within each mini-batch, samples could serve as both query and keys (i.e., we could flip what is the query and what is the key). Therefore, the contrastive loss of each batch is constructed as a sum of both ways.

Another finding of SimCLR is that unsupervised contrastive learning benefits more from bigger models than its supervised counterpart. A hypothesis is that the pretext task is much harder compared to that of supervised learning.

To summarize, SimCLR mainly proposed three design improvements: 1. substantially larger batch size (4k or 8k) that is able to provide enough negative samples, 2. replacing the output fully-connected layer with a small MLP head, and 3. stronger data augmentation. Later on, MoCov2 [7] is proposed with simple modifications from SimCLR's design improvements and achieved even better results than SimCLR. In the MoCo framework, a large number of negative samples are readily available. MoCov2 replaces the fully-connected layer (head) in MoCo with a 2-layer MLP head and extends the original augmentation by including the blur augmentation in SimCLR. MoCov2 with a batch size of 256 not only outperformed SimCLR with a batch size of 8192 but also costs much lower GPU memory and time. In fact, even with the same batch size, the end-to-end manner is still more costly because it back-propagates to both query and key encoders, while MoCo only back-propagates to the query encoder. The results suggest that large batches are not necessary for good accuracy, and state-of-the-art results can be made more accessible.

2.2.2. BYOL, SimSiam

From the approaches introduced in the previous sections, it seems that careful treatment of negative pairs is essential for contrastive learning, by either simply relying on huge batch sizes like SimCLR, or managing memory banks like MoCo or NPID. This prompts the question of whether using negative pairs is necessary, to make the framework even simpler.

Bootstrap Your Own Latent (BYOL) [15] answered this question, which surprisingly demonstrated that state-of-the-art results could be achieved even without negative pairs. Basically, the architecture of BYOL could be compared with MoCov2 as an introduction and for discussion, as shown in Figure 2.6b and Figure 2.6c. The architecture of BYOL is shown in Figure 2.5. The network's two branches in BYOL are termed as an *online* network and *target* network. BYOL builds on the concept of momentum network of MoCo: the parameters ξ of the target network's encoder f_ξ and projector g_ξ are updated by the exponential moving average (EMA) of parameters θ from the online network. The design aspects of BYOL will be introduced next.

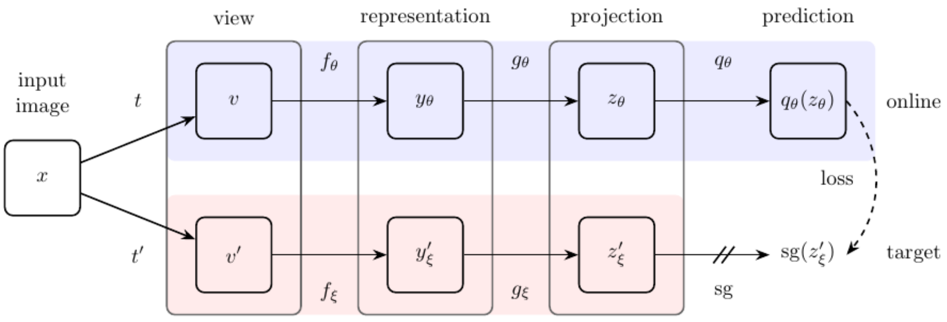


Figure 2.5: BYOL architecture. The figure is taken from [15].

Firstly, an MLP called *predictor* q_θ is additionally appended at the end of the online network. The predictor converts z_θ (the output of projector g_θ) to $q_\theta(z_\theta)$ which is finally involved in the loss.

Secondly, BYOL basically formulates the task as a regression task instead of a typical contrastive learning task since no negative samples are used. The loss function is set as a Mean Squared Error (MSE) between the normalized outputs of the online and target network:

$$\mathcal{L}_{\theta,\xi} = \|\overline{q_\theta}(z_\theta) - \overline{z}'_\xi\|_2^2 = 2 - 2 \cdot \frac{\langle q_\theta(z_\theta), z'_\xi \rangle}{\|q_\theta(z_\theta)\|_2 \cdot \|z'_\xi\|_2} \tag{2.2}$$

This means BYOL purely tries to convert two augmented versions of the original sample into the same representation vector, and not (explicitly) push other samples away as approaches using negative samples. Same as SimCLR, BYOL also symmetrizes the loss $\mathcal{L}_{\theta,\xi}$ by flipping the inputs: feeding v' to the online network and v to the target network to compute $\tilde{\mathcal{L}}_{\theta,\xi}$. Therefore, the overall loss is $\mathcal{L}^{\text{BYOL}}_{\theta,\xi} = \mathcal{L}_{\theta,\xi} + \tilde{\mathcal{L}}_{\theta,\xi}$.

Thirdly, the stop-gradient is adopted on the target network branch: the stochastic optimization step is only performed with respect to θ, but not ξ. Thus, the parameter dynamics are summarized as:

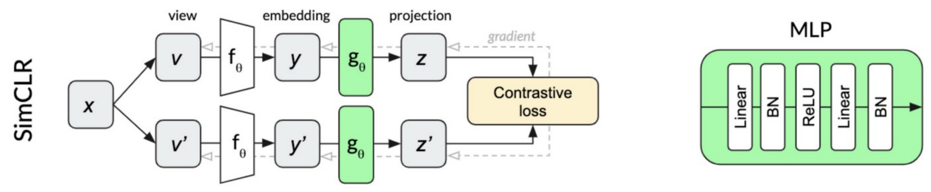
$$\theta \leftarrow \text{optimizer}(\theta, \nabla_\theta \mathcal{L}^{\text{BYOL}}_{\theta,\xi}, \eta) \tag{2.3}$$

$$\xi \leftarrow \tau\xi + (1-\tau)\theta \tag{2.4}$$

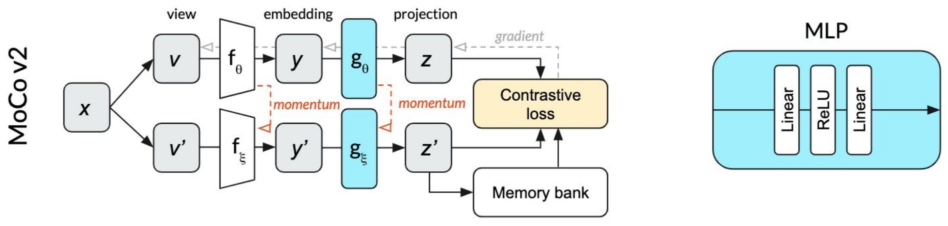
where optimizer is an optimizer and η is the learning rate.

After dropping the negative samples, it could be noticed that there exists apparently easy solution for the network to perform the task perfectly by generating *collapsed* representations. For example, if all projection vectors z_θ and z'_ξ are the same, then the network only needs to learn the identity function for q_θ in order to achieve perfect prediction. Surprisingly, although BYOL's objective admits such collapsed solutions, the authors empirically showed that BYOL does not converge to such solutions.

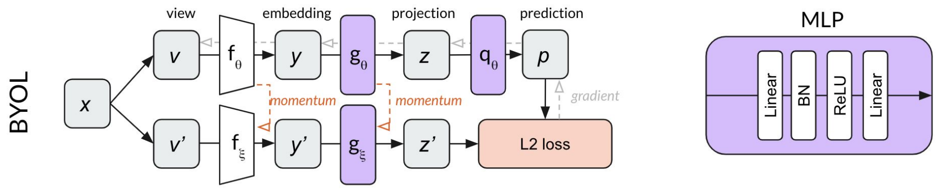
To summarize, a unified view of the architectures of SimCLR, MoCov2, and BYOL is shown in Figure 2.6.



(a) SimCLR



(b) MoCov2



(c) BYOL

Figure 2.6: A unified view of state-of-the-art self-supervised learning methods. (a) SimCLR, (b) MoCov2, (c) BYOL. The figure is taken from
`https://generallyintelligent.ai/blog/2020-08-24-understanding-self-supervised-contrastive-learning/`

SimSiam [6] later on provides an even simpler architecture. It can be thought of as "BYOL without the momentum encoder" subject to many implementation differences. The authors of SimSiam stated that the momentum encoder may be beneficial for accuracy, but it is not necessary for preventing collapse. Instead, SimSiam discovered that out of all main design aspects (e.g., stop-gradient, predictor, batch size, batch normalization, similarity function, and symmetrized loss), the stop-gradient operation is critical. This discovery can be obscured by the usage of a momentum encoder, which is always accompanied by a stop-gradient as it is not updated by its parameters' gradients. SimSiam achieved state-of-the-art results even with its simple architecture.

2.3. Related works: summary

To conclude, we give a brief summary of the cross-view matching methods in terms of the open challenges as introduced at the beginning of section 2.1:

- Drastic change in viewpoint: a representative method to address this problem is by transforming the contents between the two modalities either at an image level (e.g., polar transformation [42]). However, polar transformation works under the assumption that the ground-level and satellite images are perfectly center-aligned, which is not practical in a real-world application. [45] adds an auxiliary task of synthesizing images from one view to another and makes use of GAN to generate useful features, but in their method polar transformation is still applied and the generator network synthesizes the ground-level image on the base of a converted view of the satellite image. This issue remains the major open challenge for cross-view matching.
- Orientation misalignment: [43] explicitly predicts the orientation using the correlation between feature maps. However, this method also assumes that the ground-level and satellite images are

center-aligned. Moreover, most cross-view matching methods do not explicitly handle this issue and purely depend on the data inputs.

- Center location misalignment: one pioneer method proposed a new dataset with offset targets and predict the offsets in a supervised way [61]. [55] addresses this issue by combining the metric learning paradigm for image retrieval with dense probabilistic output via a U-Net-like decoder for local metric localization.
- Temporal scene changes: [38] addresses this issue by designing augmentations for training cross-view matching networks by masking objects predicted by standard image segmentation networks. However, during the data augmentation, the masked object classes are manually selected. This limits the generalization of this approach.

Moreover, the contrastive learning method leads to state-of-the-art performances in the field of self-supervised learning. Among the most representative ones, MoCo [21, 7] uses a memory bank and momentum encoder, SimCLR [5] simply uses an end-to-end pipeline with an extremely huge batch size to keep an identically large amount of negative samples. Then, BYOL [15] totally discards the use of negative samples and proved that it can also prevent collapsing and learn useful features.

In our work, rather than tackling one specific open challenge of cross-view matching with certain designs, we experimentally tested the effect of contrastive learning as pre-training to find out the possibility of generally improving cross-view matching methods by representation learning. In the next chapter, we explain the method in detail.

3 Method

In this work, corresponding to the first research sub-question, we experimentally tested the effect of pre-training by contrastive learning on cross-view matching. The problem definition, dataset, and overall method are introduced in section 3.1 and section 3.2. Apart from that, for the second research sub-question, we plotted and analyzed the feature distributions in the embedding space produced by different networks. They are described in section 3.3 and section 3.4.

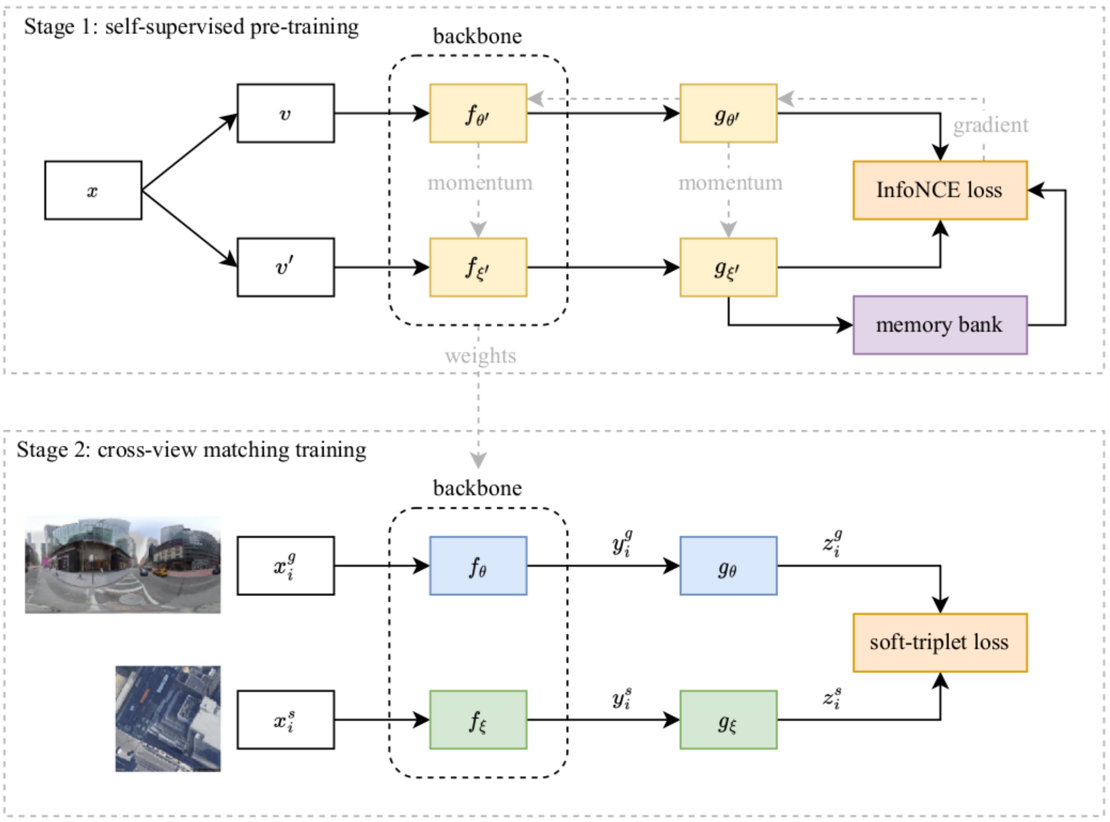


Figure 3.1: The overall method pipeline. The pipeline is composed of two stages: 1. self-supervised pre-training by contrastive learning and 2. cross-view matching downstream training

As shown in Figure 3.1, the network pipeline is composed of two stages: 1. self-supervised pre-training by contrastive learning and 2. cross-view matching downstream training. The image backbone pre-trained by contrastive learning is loaded in the training of the second stage as a starting point. In section 3.1, we introduce contrastive learning self-supervised learning (stage 1) and in section 3.2 the architecture used for cross-view matching (stage 2) is presented.

We define the dataset as a set of N sample pairs $\{(x_0^{\mathrm{g}}, x_0^{\mathrm{s}}), ..., (x_N^{\mathrm{g}}, x_N^{\mathrm{s}})\}$, each pair $(x_i^{\mathrm{g}}, x_i^{\mathrm{s}})$ consisting of a ground-level image and its corresponding ground-truth satellite image, where $x_i^{\mathrm{g}} \in \mathbb{R}^{3\times H_{\mathrm{g}}\times W_{\mathrm{g}}}$ and $x_i^{\mathrm{s}} \in \mathbb{R}^{3\times H_{\mathrm{s}}\times W_{\mathrm{s}}}$. H_{g} and W_{g} are the height and width of the ground-level image, and H_{s} and W_{s} are those of the satellite image.

3.1. Self-supervised pre-training by contrastive learning

In the first stage, we train the image backbone in a self-supervised way using contrastive learning. Specifically, the method we adopted is MoCov2 [7], which is a representative method that achieved state-of-the-art performances on ImageNet with many computer vision tasks. Compared to other methods like SimCLR [5], it does not need huge batch sizes (e.g., 4096) such that we are able to perform our experiments with limited computing resources. In the original work, MoCov2 works with a batch size of only 256. Moreover, rather than methods like BYOL [15] and SimSiam [6] which only use positive samples to learn representations, the risk of collapsing is more safely avoided by the use of negative samples in MoCov2. Although BYOL [15] and SimSiam [6] successfully avoid collapsing in their works, we consider MoCov2 [7] as a more stable baseline to start with since we will use it on new datasets and a new downstream task (i.e., cross-view matching). The main working process of MoCov2 will be introduced next.

3.1.1. The contrastive learning framework: MoCov2

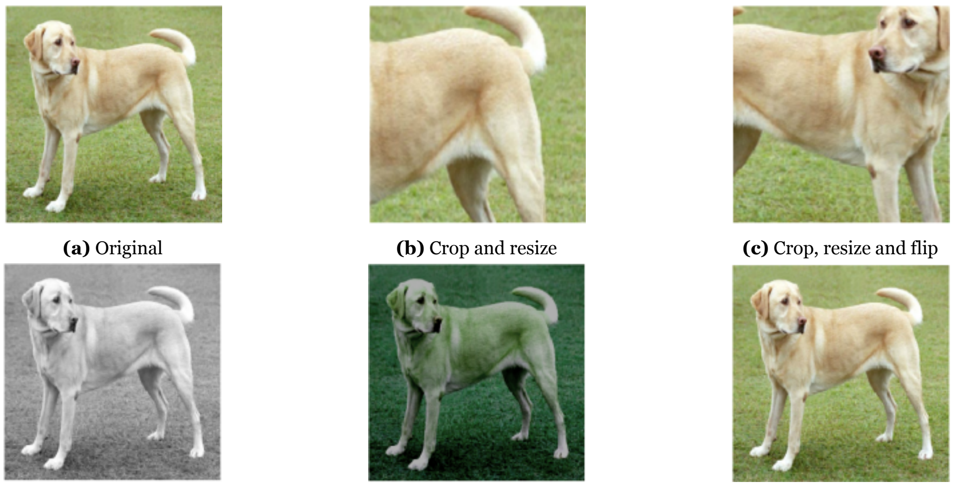
In this section, we introduce the overall contrastive learning framework used in stage 1 of Figure 3.1, including the data preparation, network architecture, momentum updating mechanism, and loss function.

The data preparation and network architecture

As shown in Figure 3.1, firstly, a data sample x is taken and applied with stochastic data augmentations, which generates two transformed versions (v and v') of the original sample. Note that at this stage the pre-training is performed independently on ground-level and satellite images. Therefore, x is either from $\{x_0^{\mathrm{g}}, ..., x_N^{\mathrm{g}}\}$ or $\{x_0^{\mathrm{s}}, ..., x_N^{\mathrm{s}}\}$. Then, we follow MoCov2's augmentation pipeline: *random cropping and resize*, *random color distortions*, *random grayscale*, *random Gaussian blur* and *random horizontal flip*. We exhibit Figure 3.2 to show the illustrations of these data augmentations operators. Figure 3.3 respectively shows two examples of the applied augmentations on the satellite and ground-level image from the dataset used in our work.

The task can be thought of as training a network for a dictionary look-up task. After the data augmentation step, one augmented sample is then encoded by a query encoder, which is composed of an image backbone $f_{\theta'}$ and projection head $g_{\theta'}$ sequentially. The backbone extracts feature maps from augmented data samples. The projection head extracts the representation vector from the feature map and then maps the representation to the space where the contrastive loss is calculated. We define the final encoded query representation as q. The other augmented sample is encoded by the key encoder, which is composed of $f_{\xi'}$ and $g_{\xi'}$ sequentially. We define the encoded query representation as k. In other words, $q = g_{\theta'}(f_{\theta'}(v))$ and $k = g_{\xi'}(f_{\xi'}(v'))$. The two augmented versions of the same original sample naturally form a positive pair, or equivalently, the current key is the corresponding positive key for the query sample. All key representations extracted from other augmented samples are considered as negative keys for the current query sample.

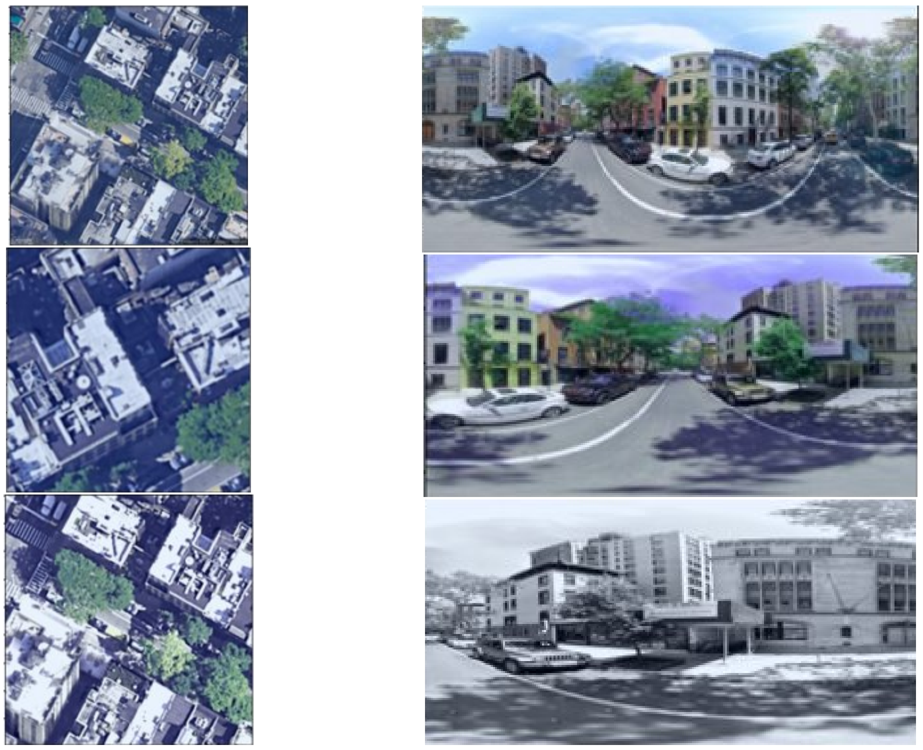
The framework allows different choices of the network, and in our work, for comparison with existing cross-view matching methods, the architecture of the image backbone ($f_{\theta'}$ and $f_{\xi'}$) is adopted as VGG16 [44] rather than the commonly used ResNet50 [19] in original contrastive learning methods



(a) Original **(b)** Crop and resize **(c)** Crop, resize and flip

(d) Grayscale **(e)** Color distortion **(f)** Gaussian blur

Figure 3.2: Illustrations of data augmentation operators. Each augmentation can transform data stochastically with some internal parameters, e.g., noise level. The figures are from SimCLR [5] (Original image cc-by: Von.grzanka)



(a) The original satellite image (top) and its two augmented versions (middle and bottom)

(b) The original ground-level image (top) and its two augmented versions (middle and bottom)

Figure 3.3: Examples of data augmentations used in Figure 3.1. The raw image sample is taken from VIGOR [61]

[21, 5, 7, 6], since VGG16 [44] is widely used in most cross-view matching methods [27, 42, 61]. The network blocks of VGG16 we used are shown in the appendix. For the projection head, we follow SimCLR [5] and MoCov2 [7] that use an MLP head. Specifically, the architecture of $g_{\theta'}$ and $g_{\xi'}$ is composed of an average pooling layer and an MLP head with one hidden layer. Therefore, the architecture of $g_{\theta'}$ and $g_{\xi'}$ could be shown in a PyTorch-like style as below:

```
(avgpool): AdaptiveAvgPool2d(output_size=(1, 1))
(classifier): Sequential(
  (0): Linear(in_features=512, out_features=512, bias=True)
  (1): ReLU()
  (2): Linear(in_features=512, out_features=128, bias=True)
)
```

The contrastive loss function and momentum updating mechanism

The authors of MoCo [21] hypothesized that good features can be learned by a large dictionary that covers a rich set of negative samples, while the encoder for the dictionary keys is kept as consistent as possible despite its evolution. Therefore, a memory bank is maintained as a queue of representations. This allows us to reuse the encoded keys from the immediate preceding mini-batches. The size of the memory bank can be much larger than a typical mini-batch size and can be flexibly and independently set as a hyper-parameter. The samples in the memory bank are progressively replaced. The current mini-batch is enqueued to the queue, and the oldest mini-batch in the queue is removed. The queue always represents a sampled subset of all data. Moreover, removing the oldest mini-batch can be beneficial, because its encoded keys are the most outdated and thus the least consistent with the newest ones. Based on the query and key representations, a form of contrastive loss function called InfoNCE [35] is defined as

$$\mathcal{L}_q = -\log \frac{\exp(q \cdot k_+/\tau)}{\sum_{i=0}^{K} \exp(q \cdot k_i/\tau)}, \tag{3.1}$$

where q is the query representation, k_+ is the positive key representation, and k_i is a key representation of the memory bank. All saved key representations in the memory bank serve as negative samples for the query in the current mini-batch. τ is a temperature hyper-parameter. The sum is over one positive and K negative samples. Intuitively, this loss is the log loss of a (K+1)-way softmax-based classifier that tries to classify q as q_+.

Using a queue can make the dictionary large, but it also makes it intractable to update the key encoder by back-propagation (the gradient should propagate to all samples in the queue). A momentum updating strategy is used to address this issue:

$$\xi' \leftarrow m\xi' + (1-m)\theta'. \tag{3.2}$$

Here ξ', θ' are the parameters of the query encoder $f_{\theta'}$, $g_{\theta'}$ and key encoder $f_{\xi'}$, $g_{\xi'}$, respectively. $m \in [0, 1)$ is a momentum coefficient. This momentum updating makes θ_k evolve smoothly. As a result, although the keys in the queue are encoded by different encoders (in different mini-batches), the difference among these encoders can be made small.

Overall, Algorithm 1 exhibits the pseudo-code of MoCo for stage 1 in Figure 3.1, as described in this section. For the current mini-batch, the queries and the corresponding positive keys are encoded by the network. The negative keys are directly from the queue. InfoNCE loss is then calculated and used for back-propagation. After that, the key network is updated by the query network with momentum, and the memory bank is updated with the keys of the current mini-batch.

Algorithm 1 Pseudo-code of MoCo in a PyTorch-like style (from [21])

```
# f_q, f_k: encoder networks for query and key
# queue: dictionary as a queue of K keys (CxK)
# m: momentum
# t: temperature

f_k.params = f_q.params # initialize
for x in loader: # load a minibatch x with N samples
    x_q = aug(x) # a randomly augmented version
    x_k = aug(x) # another randomly augmented version
    q = f_q.forward(x_q) # queries: NxC
    k = f_k.forward(x_k) # keys: NxC
    k = k.detach() # no gradient to keys
    # positive logits: Nx1
    l_pos = bmm(q.view(N,1,C), k.view(N,C,1))
    # negative logits: NxK
    l_neg = mm(q.view(N,C), queue.view(C,K))
    # logits: Nx(1+K)
    logits = cat([l_pos, l_neg], dim=1)
    # contrastive loss, Eqn.(1)
    labels = zeros(N) # positives are the 0-th
    loss = CrossEntropyLoss(logits/t, labels)
    # SGD update: query network
    loss.backward()
    update(f_q.params)
    # momentum update: key network
    f_k.params = m*f_k.params+(1-m)*f_q.params
    # update dictionary
    enqueue(queue, k) # enqueue the current minibatch
    dequeue(queue) # dequeue the earliest minibatch
```

3.1.2. Different pre-trained weights

After training, the backbone of the encoder in stage 2 will be loaded with the parameter weights of the backbone in the query encoder $f_{\theta'}$. In our experiments, we conducted the pre-training under various situations to see how the performance of stage 2 differs. These variants contain:

- If the pre-training starts with randomly initialized network weights or with loaded weights pre-trained on ImageNet [9]. It is a commonly adopted practice that initializes the network with weights gained from training on ImageNet [9] by supervised learning in many computer vision tasks, including most existing cross-view matching methods. Although original MoCov2 [7] and other self-supervised learning methods [21, 5, 15] do not load ImageNet [9] pre-trained weights to test the performance of the approach, here we adopt this option as a practical baseline, and achieve a fair comparison with existing cross-view matching methods. Since the data amount of the cross-view matching datasets we used for pre-training is extremely smaller than the scale of ImageNet [9], it is considered beneficial if our pre-training brings further performance improvement.
- If the pre-training is performed on ground-level or satellite images. Due to the drastic change in viewpoint, following most previous cross-view matching methods, the encoders for the ground-level and satellite images in stage 2 are not shared. Therefore, the corresponding encoder is naturally expected to load with pre-trained weights on the same image viewpoint. In our experiments, different combinations of pre-trained weights are tested on the two branches in stage 2.
- For pre-training on the satellite images, it could be performed on different dataset splits with three options: the training set, the validation set, or the whole dataset (the training set plus validation set). The availability of the validation set relies on the observation that the satellite images are all accessible. It is hypothesized that pre-training on the images from the validation set gives the network the possibility to perceive image features of the target area for evaluation and gain higher

performance. Moreover, making use of the whole dataset relies on the assumption that more data is generally beneficial for the network.

Based on the variants introduced above, we define different pre-trained backbone weights settings in our experiments on the two different datasets. Table 3.1 shows these combinations on the two network branches (ground and satellite), in which each row represents a pre-training strategy. Moreover, several terms are defined to represent the pre-training weights:

1. `/`: randomly initialized weights.
2. `ImageNet`: pre-trained weights by supervised learning on ImageNet [9].
3. `grd` or `sat`: pre-trained weights trained on ground-level or satellite images of the whole downstream dataset (e.g., CVACT or VIGOR) using MoCov2 [7]. The pre-training starts from scratch with randomly initialized weights.
4. `grd*` or `sat*`: pre-trained weights trained on ground-level or satellite images of the whole downstream dataset using MoCov2 [7]. The pre-training starts from loaded ImageNet [9] weights by marked with a star `*`.
5. `sat train*`: pre-trained weights trained on satellite images of the training set of the downstream dataset using MoCov2 [7]. The pre-training starts from loaded ImageNet [9] weights.
6. `sat val*`: pre-trained weights trained on satellite images of the validation set of the downstream dataset using MoCov2 [7]. The pre-training starts from loaded ImageNet [9] weights.

Note that we set more variations of the satellite image branch because as mentioned before, satellite images are publicly available and pre-training on them yields a more practical application, while ground-level images from a target area are not usually reachable. Therefore, in most settings, the pre-trained weights on the ground-level image branch are on the baseline ImageNet.

Table 3.1: Different combination of pre-trained weights in our experiments

Settings	Ground-level image branch			Satellite image branch						Note
	`/`	`ImageNet`	`grd*`	`/`	`ImageNet`	`sat`	`sat train*`	`sat val*`	`sat*`	
1	×			×						from scratch
2		×			×					baseline
3		×				×				
4		×					×			
5		×						×		
6		×							×	
7			×						×	

3.2. Image Retrieval Architecture and Loss Function

In this section, the architecture and loss function used in the downstream cross-view matching (stage 2) training is introduced. Given a set of query ground-level and satellite reference images, our task objective is to learn an embedding space in which each ground-level query is close to its corresponding ground-truth satellite image. Specifically, we expect a neural network that is able to perform a projection from the images to a representation vector. By calculating and comparing the distances between the representation vectors, we could find the closest satellite image for the given ground-level image. Similar to section 3.1, it could be also considered as a dictionary look-up task, but now the task is performed on crossed-view images.

Following the dataset definition, each ground-level image and its ground-truth satellite image are considered a positive pair, while others are considered negative. If there are multiple satellite images covering one ground-level image like VIGOR [61], we consider the nearest one as the positive and avoid sampling the other neighboring satellite images in the same batch to prevent ambiguous supervision.

3.2.1. The image retrieval framework

The network is composed of two networks with parameters θ and ξ that encode the ground and satellite images separately. Take the ground-level branch as an example, the backbone f_θ converts x_i^{g} to the feature map y_i^{g}. The pooling module aggregates the feature map and generates the final representation $z_i^{\mathrm{g}} \in \mathbb{R}^k$, where k is the vector's dimension. This process is the same for the satellite branch on the input satellite image, producing the representation of satellite image $z_i^{\mathrm{s}} \in \mathbb{R}^k$.

Specifically, we performed experiments on two existing architectures Siamese-VGG [60] and SAFA [42]. Both of them use the backbone (i.e., f_θ and f_ξ) VGG16 [44] which is consistent with the backbone in section 3.1 so that it could be loaded with pre-trained weights. The two architectures differ in the aggregation module (i.e., g_θ and g_ξ). Siamese-VGG [60] serves as a baseline method that simply uses an average pooling module and a fully-connected layer after the image backbone to produce the final representation vector. SAFA [42] designed a spatial-aware embedding module to perform the aggregation. In our work, we re-implemented these methods in PyTorch for better integration with the pre-training stage and to avoid the risk of accuracy drop due to model conversion between different platforms. The details of the implementation of the architecture could be found in the appendix.

3.2.2. The soft-margin triplet loss

We adopt the soft-margin triplet loss in [23]. The loss is calculated on a triplet, which is composed of an *anchor* z_a, *positive* z_p, and *negative* z_n. The chosen metric to compute the distance between representations is L2-Norm. The loss function is defined as

$$L_{\text{triplet}}(z_a, z_p, z_n) = \log\big(1 + \gamma \cdot \exp\left(d_{\text{pos}} - d_{\text{neg}}\right)\big), \tag{3.3}$$

where $d_{\text{pos}} = \|z_a - z_p\|_2$ and $d_{\text{neg}} = \|z_a - z_n\|_2$ are the distances of the positive pair (anchor and positive) and negative pair (anchor and negative). γ is a hyper-parameter adjusting the gradient of the loss, thus controlling the convergence speed. The triplet loss aims at pulling the positive pair closer and pushing the negative pair further away. Based on the soft-triplet loss, the total loss is computed as

$$\mathcal{L}_{\text{total}} = \sum_{i=0}^{N} \sum_{\substack{j=0 \\ j \neq i}}^{N} \big(L_{\text{triplet}}(z_i^{\mathrm{g}}, z_i^{\mathrm{s}}, z_j^{\mathrm{s}}) + L_{\text{triplet}}(z_i^{\mathrm{s}}, z_i^{\mathrm{g}}, z_j^{\mathrm{g}})\big), \tag{3.4}$$

which pulls the corresponding pair of ground-level and satellite representations closer and pushes all others from other pair samples away. Intuitively, each ground-level and satellite image representation can either serve as a query or a key in other triplets. Figure 3.4 gives an illustration of how the total loss is composed. In total, it sums over $N(N-1)$ single triplet loss. In practice, the loss is calculated within each mini-batch of size B during the training process.

3.3. Feature distribution visualization and uniformity metric

Pre-training weights provide a starting point for the network parameters that will be optimized in downstream tasks. In order to interpret the results in previous sections, we performed an in-depth analysis by visualizing the features generated by different models in the embedding space.

Specifically, the analysis is performed on the experiments using Siamese-VGG [60]. Note that for visualization, we use t-SNE [46] to perform dimensionality reduction, which converts the output feature vector z_i^{g} and z_i^{s} to 2-d dimensional vectors. t-SNE (t-Distributed Stochastic Neighbor Embedding) is a machine learning algorithm for visualizing high-dimensional data. It is commonly used to reduce the

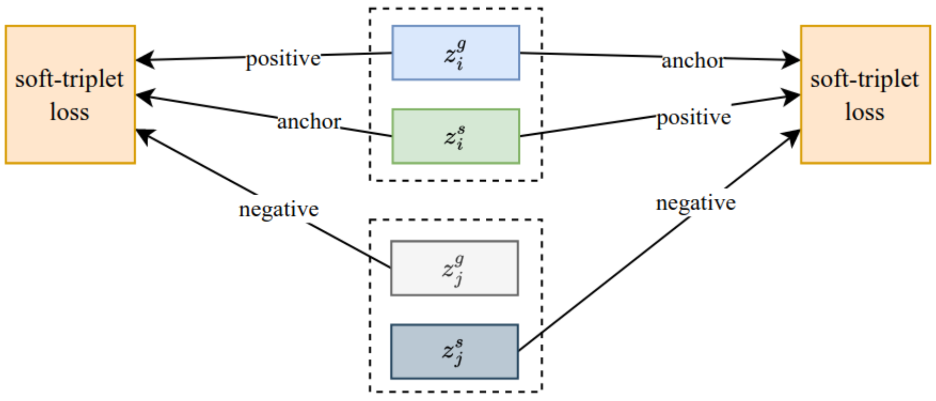


Figure 3.4: Illustration on the composition of triplet losses used for image retrieval training

dimensionality of data to two or three dimensions so that it can be plotted and visualized in a scatter-plot. It works by first constructing a probability distribution over pairs of high-dimensional data points, and then minimizing a divergence between the joint probabilities of the low-dimensional embedding and the high-dimensional data. The embedding is constructed such that similar points in the high-dimensional space are nearby in the low-dimensional space, while dissimilar points are far apart. It is is a powerful tool for data visualization and has been widely used in a variety of applications. Using this technique, we plot the representations of the validation set of the target dataset before the downstream task training epoch launches, to see the effect of pre-training.

Moreover, in order to analyze the effect of different pre-training, we selected three typical pre-training combinations:

- Both the ground-level and satellite image branches are loaded with ImageNet [9] pre-trained weights. This serves as the baseline method.
- The ground-level image branch is loaded with ImageNet [9] pre-trained weights, and the satellite image branch is loaded with contrastive learning pre-trained weights on the satellite images of the corresponding dataset. In this case, the encoder for satellite images is expected to be aware of the features of the satellite images.
- Both the ground-level and satellite image branches are loaded with contrastive learning pre-trained weights on the ground-level and satellite images of the corresponding dataset, respectively. In this case, both encoders for ground-level and satellite images are expected to be aware of the dataset.

Additionally, we adopt the methods in [51] to analyze the feature distribution properties in terms of how uniform the feature distribution is in the embedding space. Two extra plots are generated: 1. features in $\mathbb{S}^1$ by l2-normalizing features in $\mathbb{R}^2$. 2. features density estimation by Mises-Fisher (vMF) KDE on angles (i.e., arctan2(y, x) for each point $(x, y) \in \mathbb{S}^1$). These two plots further show the distribution in the embedding space.

We also reported the uniformity metric [51] of the representations to quantify how uniform the distribution is. The authors of [51] defined the uniformity metric based on the Gaussian potential kernel (i.e., the Radial Basis Function (RBF) kernel) under the consideration that the metric should be both asymptotically correct (i.e., the distribution optimizing this metric should converge to uniform distribution) and empirically reasonable with a finite number of points. Specifically, the uniform metric is defined as the logarithm of the average pairwise Gaussian potential kernel:

$$\mathcal{L}_{\text{uniform}}(f;t) \triangleq \log \mathbb{E}_{x,y\sim p_{\text{data}}} \exp\left(-t\|f(x)-f(y)\|_2^2\right). \tag{3.5}$$

Here p_{data} is the data distribution over $\mathbb{R}^n$, and $f:\mathbb{R}^n \to \mathbb{S}^{m-1}$ is an encoder that maps data to L2-normalized representation vectors of dimension m. Therefore, in our case, three values are reported: the uniformity metric within the ground-level image features, satellite image features, and the overall features. They could be defined as

$$\mathcal{L}_{\text{uniform-ground}} = \log \sum_{i=0}^{N} \sum_{\substack{j=0 \\ j\neq i}}^{N} \exp\left(-t\|z_i^{\text{g}} - z_j^{\text{g}}\|_2^2\right), \tag{3.6}$$

$$\mathcal{L}_{\text{uniform-satellite}} = \log \sum_{i=0}^{N} \sum_{\substack{j=0 \\ j\neq i}}^{N} \exp\left(-t\|z_i^{\text{s}} - z_j^{\text{s}}\|_2^2\right), \tag{3.7}$$

$$\mathcal{L}_{\text{uniform-all}} = \log \sum_{i=0}^{2N} \sum_{\substack{j=0 \\ j\neq i}}^{2N} \exp\left(-t\|z_i - z_j\|_2^2\right), \tag{3.8}$$

where N is the number of sample pairs, $z_i^g, z_j^g \in \{z_1^g, ..., z_N^g\}$ are the representation vectors of the ground-level images, $z_i^s, z_j^s \in \{z_1^s, ..., z_N^s\}$ are the representation vectors of the satellite images, and $z_i, z_j \in \{z_1^g, ..., z_N^g, z_1^s, ..., z_N^s\}$ are among all representation vectors. Note that although [51] uses uniform metric directly as a loss function for optimizing, here the metric value is only for analysis to show how uniform the features are distributed. Normally, the value is negative, and a lower value indicates that the features are distributed more uniformly and make use of more space in the embedding space.

3.4. Add uniformity loss and data augmentation directly

We identified two elements that could be beneficial during pre-training: uniformity within single-modality images and strong data augmentation. We try to add these two elements directly in downstream task training to see if the performance also gets improvements.

3.4.1. Uniformity loss

The learning objective of contrastive learning is identifying two augmented versions of the original image from the same image, pulling their representations closer, and pushing other samples away. This process internally makes the representation of each sample more unique and might enlarge the representations' occupation in the embedding space letting them be more uniform. We already used the uniformity metric in the last section to analyze the feature distribution. In this section, we try to directly add it to the downstream task training as an auxiliary loss to find out its effect.

Specifically, We define the uniformity loss function as

$$\mathcal{L}_{\text{uniformity}} = \log \sum_{i=0}^{N} \sum_{\substack{j=0 \\ j\neq i}}^{N} \exp\left(-t\|z_i^{\text{g}} - z_j^{\text{g}}\|_2^2\right) + \log \sum_{i=0}^{N} \sum_{\substack{j=0 \\ j\neq i}}^{N} \exp\left(-t\|z_i^{\text{s}} - z_j^{\text{s}}\|_2^2\right), \tag{3.9}$$

which tries to separate ground-level and satellite image features within their modalities respectively. The uniform loss is added to the total triplet loss to generate the final loss used in this section:

$$\mathcal{L}_{\text{total}}^{'} = \mathcal{L}_{\text{total}} + \alpha \mathcal{L}_{\text{uniformity}}. \tag{3.10}$$

Here α is a hyper-parameter that weights the uniform loss. We tuned this weight to see the influence of uniform loss directly applying to the downstream task.

3.4.2. Data augmentation

The strong data augmentations applied in contrastive learning pre-training enable the network to learn richer features. Moreover, there are few cross-view matching works using data augmentations during training, although it is a frequently used technique within many computer vision tasks. Therefore, we try to add it directly to the downstream task to see the effect.

We adopt the data augmentation applied in MoCov2 [7] pre-training directly on the downstream task. The augmentations include *random cropping and resize*, *random color distortions*, *random grayscale*, *random Gaussian blur* and *random horizontal flip*. Note that *random cropping and resize* crops the image and resizes it to the original size. For cross-view matching training, this process destroyed the alignment between the ground-level image and its corresponding ground-truth satellite image in the VIGOR [61] dataset with semi-positive samples. Therefore, we also test applying augmentations without *random cropping and resize*.

4 Experiments

In this section, we present the experiment results and analysis. section 4.3 and section 4.4 show the main results on the performance of cross-view matching with self-supervised pre-training. Then, section 4.5 and section 4.6 present further analysis and experiments.

4.1. Datasets and evaluation metrics

We perform the experiments based on two cross-view matching datasets: CVACT [28] and VIGOR [61]. The datasets contain images of both ground-level and satellite views and their correspondences.

CVACT dataset [28][1] is a GPS-tagged cross-view dataset covering 300 square miles of road in Canberra, Australia. To collect ground-level and satellite images, the GSV API[2] and Google Maps API were employed. All ground-level images (panoramas) were captured at zoom level 2 at a resolution of 1664×832 pixels and satellite view images were captured at zoom level 20 at a resolution of 1200×1200 pixels. Two sample images in this dataset are presented in the bottom two rows of Figure 4.1. In total, this dataset contains 128,334 ground-satellite image pairs in which 35,532 pairs are used for training, 8,884 for validation, denoted as `CVACT_val`, and 92,802 for testing, denoted as `CVACT_test`. In our experiments, the results are conducted on `CVACT_val`.

We also conduct experiments on the VIGOR [61] dataset and its `cross-area` split, to evaluate the methods in an urban scenario. VIGOR [61] originally contains 238,696 panoramas and 90,618 aerial images from four cities, i.e. Manhattan, San Francisco, Chicago, and Seattle. A balanced sampling is applied to select only two positive panoramas for each satellite image, resulting in 105,214 panorama images. VIGOR assumes that the queries can belong to arbitrary locations in the target area, and thus are not spatially aligned to the center of any aerial reference images in both training and test sets. For each query ground-level image, there are four corresponding ground-truth satellite images, with one of them tagged as "positive" and the rest three as "semi-positive". The dense sampling strategy also hugely increases the difficulty of the retrieval task. VIGOR has two evaluation protocols [36]: `same-area` and `cross-area`, depending on how the training and validation set is divided. Under the `same-area` setting, all images are from the same areas. Under the `cross-area` protocol, the training and validation set are extracted from non-overlapped, different cities: the training set consists of two cities Seattle and NewYork, and the validation set consists of two cities SanFrancisco and Chicago.

Table 4.1 gives a brief summary of the two datasets we used in our work. Through the experiments on the two different datasets, the result could be evaluated under both rural and urban image scenery. Moreover, based on the previous benchmarks, sampling strategy, and data split as introduced above, we consider image retrieval on CVACT [28] as a relatively easier task compared to VIGOR [61].

[1]`https://github.com/Liumouliu/OriCNN`
[2]`https://developers.google.com/maps/documentation/streetview/overview`

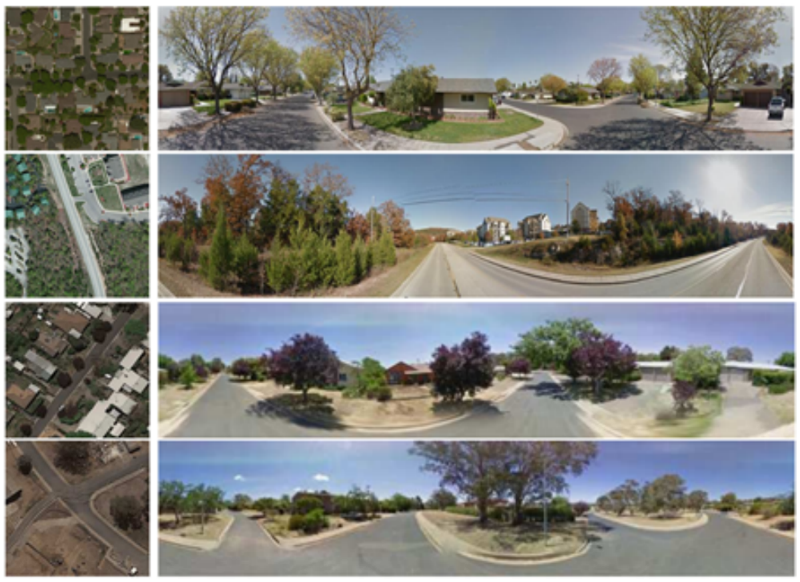


Figure 4.1: The CVACT dataset. On the left is a satellite image and on the right side is its corresponding ground-level panorama. The figure is taken from [43].

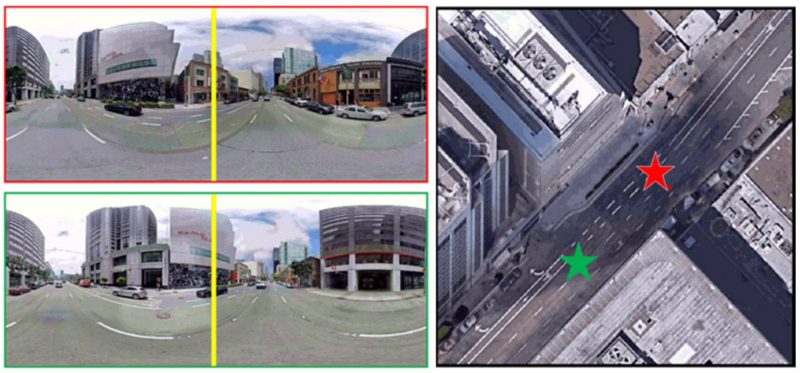


Figure 4.2: Sample images from VIGOR dataset. On the left are two ground-level images covered by the satellite image on right. The yellow line in the ground-level images represents the north direction. The color of the star in the satellite image represents the location of the ground-level image with the same color border. The figures is taken from [61].

Table 4.1: Comparison between the two datasets used in our work

dataset	CVACT [28]	VIGOR [61]
data split	`CVACT_val`	`cross-area`
data amount	training set: 35532, validation set: 8884	training set: 51520, validation set: 53694
area	mostly rural	mostly urban
generalization	The training and validation set cover same or adjacent areas	The training and validation set are totally from different cities

We will report the retrieval performance in terms of top-k recall accuracy, denoted as “R@k”. The k nearest reference neighbors in the embedding space are retrieved based on cosine similarity for each query. If the ground-truth reference image appears in the top k retrieved images, it is considered as correct.

4.2. Implementation details

In this section, we list the implementation details of the two stages in our method. During self-supervised pre-training (stage 1), we use SGD as the optimizer. The SGD weight decay is 0.0001 and the SGD momentum is 0.9. We use a mini-batch size of 128 in 2 GPUs, and the initial learning rate is 0.015. We train for 200 epochs with the learning rate multiplied by 0.1 at 120 and 160 epochs, taking 14 hours for training. Following MoCov2 [7], the encoder is composed of a backbone and a projector. The backbone is chosen as VGG16, and the projector is a 2-layer MLP head with an output dimension of 128. The size of the memory bank is 16384. The updating momentum for the key encoder is 0.999. The temperature τ in InfoNCE loss is 0.2. For CVACT dataset, the ground-level image size is 112×616 pixels and the satellite image size is 256×256 pixels. For VIGOR dataset, the ground-level image size is 224×448 pixels and the satellite image size is 224×224 pixels.

During cross-view matching training (stage 2), we use Adam as the optimizer. The weight decay is 0.0001 and the initial learning rate is 0.0001. We use a mini-batch size of 96 in a single GPU. We train for 100 epochs with the learning rate, taking 21 hours for training. The backbone is consistent with stage 1, chosen as VGG16 [44], and the aggregation part uses two settings: 1. global average pooling plus a shared fully-connected layer as in [60] and 2. spatial-aware embedding module as in [42]. The output dimension of the representation is 4096. The settings for the image sizes are the same as those during pre-training.

4.3. Cross-view matching with contrastive learning pre-training

This section exhibits the experiment results corresponding section 3.1 and section 3.2, and mainly answers the first sub-question proposed in the introduction. We performed cross-view matching experiments on the network initialized with different pre-trained weights.

The results of Siamese-VGG [60] and SAFA [42] on CVACT dataset are shown in Table 4.2 and Table 4.3. The results of Siamese-VGG [60] and SAFA [42] on VIGOR [61] dataset are shown in Table 4.4 and Table 4.5. The tables follow the definitions as described in subsection 3.1.2.

Table 4.2: Cross-view matching performance of Siamese-VGG [60] using different sets of pre-trained weights on the ground-level and satellite network branch. The (downstream) training and evaluation are performed on CVACT [28] dataset using `CVACT_val` split

Exp.	Ground-level image branch			Satellite image branch					R@1	R@5	R@1%
	`/`	`ImageNet`	`grd*`	`/`	`ImageNet`	`sat train*`	`sat val*`	`sat*`			
1	×			×					22.29	51.65	70.17
2		×			×				53.20	77.36	95.05
3		×				×			63.26	83.51	96.02
4		×					×		60.46	81.79	96.07
5		×						×	64.90	84.22	96.29
6			×					×	**65.00**	**85.21**	**96.60**

Exp 1, 7, 11, and 18 serve as baseline settings that use ImageNet pre-trained weights, which is a common practice in most of the implementations for cross-view matching. From the comparisons of the experiments above, the main findings are listed as follows:

- Exp 12. and exp 13.: on the VIGOR dataset, initialized weights that are pre-trained using con-

Table 4.3: Cross-view matching performance of SAFA [42] using different sets of pre-trained weights on the ground-level and satellite network branch. The (downstream) training and evaluation are performed on CVACT [28] dataset using `CVACT_val` split

Exp.	Ground-level image branch			Satellite image branch					R@1	R@5	R@1%
	/	`ImageNet`	`grd*`	/	`ImageNet`	`sat train*`	`sat val*`	`sat*`			
7		×			×				76.96	88.98	96.58
8		×				×			79.01	89.88	96.79
9		×					×		78.31	89.24	96.60
10		×						×	**79.27**	**90.27**	**96.99**

Table 4.4: Cross-view matching performance of Siamese-VGG [60] using different sets of pre-trained weights on the ground-level and satellite network branch. The (downstream) training and evaluation are performed on VIGOR [61] dataset using the “Cross-Area” split setting (the training and validation set are located in separate cities).

Exp.	Ground-level image branch			Satellite image branch						R@1	R@5	R@1%
	/	`ImageNet`	`grd*`	/	`ImageNet`	`sat`	`sat train*`	`sat val*`	`sat*`			
11	×			×						0.50	1.75	29.13
12		×			×					3.28	9.21	62.16
13		×				×				3.75	10.40	66.45
14		×					×			5.29	13.88	69.61
15		×						×		5.82	14.64	71.51
16		×							×	5.78	14.63	70.87
17			×						×	**7.35**	**18.25**	**76.91**

Table 4.5: Cross-view matching performance of SAFA [42] using different sets of pre-trained weights on the ground-level and satellite network branch. The (downstream) training and evaluation are performed on VIGOR [61] dataset using the “Cross-Area” split setting (the training and validation set are located in separate cities).

Exp.	Ground-level image branch			Satellite image branch						R@1	R@5	R@1%
	/	`ImageNet`	`grd*`	/	`ImageNet`	`sat`	`sat train*`	`sat val*`	`sat*`			
18		×			×					4.45	11.44	65.02
19		×				×				4.89	12.07	68.36
20		×					×			7.09	17.06	74.85
21		×						×		8.12	18.8	**76.68**
22		×							×	**8.18**	**19.36**	75.32

trastive learning from scratch on satellite images of the target (downstream) dataset produce an on-par and slightly higher cross-view matching performance than that of ImageNet pre-trained weights (3.28 vs. 3.75 on Recall@1).

- Exp 12. and exp 16.: initialized weights that are pre-trained using contrastive learning from ImageNet pre-trained weights on satellite images of the target dataset produce higher performance than that of ImageNet pre-trained weights (3.28 vs. 5.78 on Recall@1). This is also shown by the comparison between exp 2. and 5, 7 and 10, and 18 and 22, respectively.
- Exp 14. and exp 15.: on the VIGOR dataset, initialized weights that are pre-trained using satellite images exactly from the validation set of the downstream task achieve higher performance than that of the training set (5.29 vs. 5.82 on Recall@1). This is also shown by the comparison between exp 20. and 21. However, the result on the CVACT dataset is the opposite by looking at the comparison between exp 3 and 4, and 8 and 9. Possible reasons are 1. the training and validation set in VIGOR cover non-overlapped areas and the appearances differ, while in CVACT the appearances are similar. 2. the data amounts of the training and validation set in VIGOR are equivalent, while in CVACT the validation set contains fewer samples.
- Exp 16. and exp 17.: initialized weights that are pre-trained using both ground-level and satellite images from the whole dataset give us another performance improvement than that of only using the satellite images (5.78 vs. 7.35 on Recall@1). This is also shown by the comparison between exp 2. and 6. However, different from pre-training on satellite images that could be globally accessed as prior, pre-training on ground-level images lacks the possibility of a real-world application since ground-level images from a specific area are not always available.

These findings lead us to draw several conclusions or hypotheses as below:

1. On top of the widely applied ImageNet pre-trained weights, it is an effective way to perform self-supervised learning using standard contrastive learning methods on the cross-view matching dataset to generate pre-training weights for downstream task training. It is hypothesized that training on satellite images gives the network new knowledge on features in an aerial view for identifying places, while they are relatively rare in the normal ImageNet data.
2. If the localization task is aimed at a specific target area or city, it could be beneficial to collect dense satellite image patches covering the area to create a training set that is large enough for self-supervised pre-training, rather than using data from other areas. In a real-world scenario, the satellite images are all available to fetch from the remote sensing satellites, using e.g., GSV [2]. Self-supervised pre-training on the target area or city satellite images can sometimes give a best practice for training a cross-view matching network.

4.4. Verification on swapped dataset

In order to verify the main findings in section 4.3, extra experiments are performed using Siamese-VGG [60] on a swapped version of VIGOR [61] dataset. Specifically, the downstream task training is performed on the validation set of VIGOR and evaluated on the training set. The results are shown in Table 4.6.

Note that in Table 4.6 the pre-trained weights are termed the same as that of Table 4.4, e.g., “sat train*” still means the original training set split in VIGOR, but in fact, serves as the set for evaluation in the four experiments in this section. The results are consistent with the main finding in section 4.3: the performances of the network pre-trained directly with the satellite images from the validation set (Exp 24.) is higher than those from the training set (Exp 25.).

Besides, one detailed different result is: by comparing exp 25. and exp 26., this time initialized weights that are pre-trained using satellite images from the whole dataset achieve slightly better performance than that of only the validation set (6.41 vs. 6.46 on Recall@1).

Table 4.6: Cross-view matching performance of Siamese-VGG [60] using different sets of pre-trained weights on the ground-level and satellite network branch. The (downstream) training and evaluation are performed on VIGOR [61] dataset using a swapped "Cross-Area" split setting (the training and validation set are swapped on the original split).

Exp.	Ground-level image branch			Satellite image branch						R@1	R@5	R@1%
	/	`ImageNet`	`grd*`	/	`ImageNet`	`sat`	`sat train*`	`sat val*`	`sat*`			
23		×			×					3.87	10.01	57.91
24		×					×			6.41	15.12	68.83
25		×						×		5.58	13.40	64.52
26		×							×	**6.46**	**15.39**	**69.11**

4.5. Feature distribution

This section exhibits the experiment results of section 3.3 and mainly answers the second sub-question proposed in the introduction.

4.5.1. Settings

Overall, for both datasets, CVACT [28] and VIGOR [61], we plotted the feature distributions of 3 experiments respectively to see the effect of pre-training on certain branches. They include settings of:

- Both image branches are loaded with baseline ImageNet pre-trained weights: exp 2. for CVACT [28] and exp 12. for VIGOR [61].
- The ground-level image branch is loaded with ImageNet pre-trained weights, and the satellite image branch is loaded with self-supervised pre-trained weights on satellite images: exp 5. for CVACT [28] and exp 16. for VIGOR [61].
- Both image branches are loaded with self-supervised pre-trained weights on ground-level and satellite images respectively: exp 6. for CVACT [28] and exp 17. for VIGOR [61].

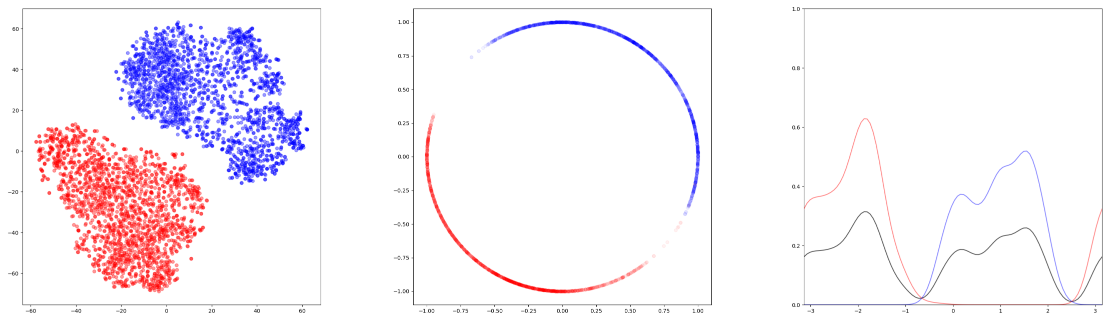
Furthermore, for each experiment, as introduced in section 3.3, we generated 3 plots:

- Features in $\mathbb{R}^2$ by t-SNE dimensionality reduction
- Features displayed in $\mathbb{S}^1$ by l2-normalizing on features in $\mathbb{R}^2$
- features density estimation by Mises-Fisher (vMF) KDE on angles (i.e., arctan2(y, x) for each point $(x, y) \in \mathbb{S}^1$)

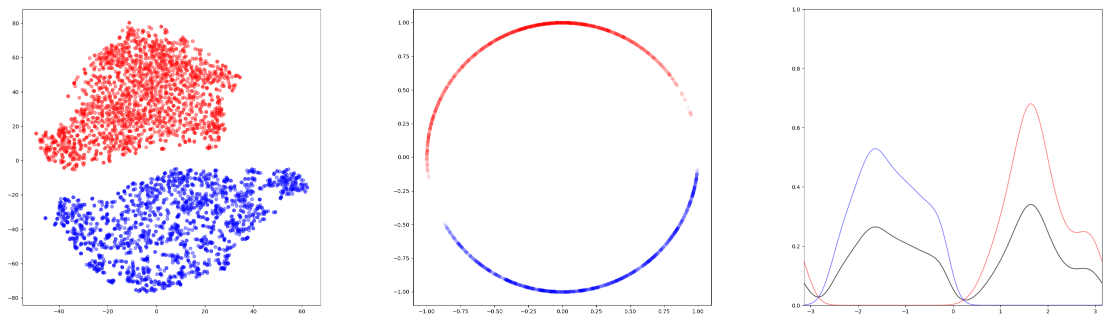
And the calculated uniformity metric is reported in the text of the corresponding sub-figure's caption. A lower value indicates a more uniform feature distribution.

The plots are shown in Figure 4.3 and Figure 4.4 for the two datasets ([28] and [61]) respectively. Note that in Figure 4.3, the red dots represent feature vectors of ground-level images and the blue dots represent satellite images. In Figure 4.4, since VIGOR [61]'s validation set contains two different cities, the plot considers features of two image viewpoints (i.e., ground-level and satellite) and two cities in the validation set (i.e., SanFrancisco and Chicago):

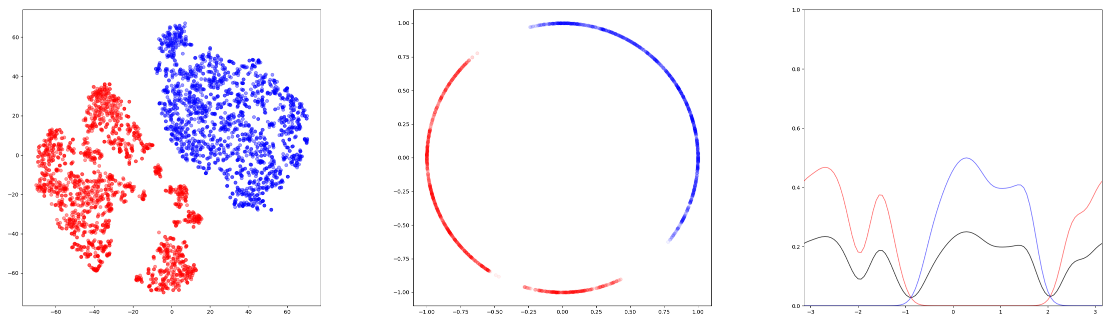
- Red: ground-level image features of SanFrancisco
- Blue: satellite image features of SanFrancisco
- Orange: ground-level image features of Chicago
- Green: satellite image features of Chicago



(a) Pre-training on ImageNet (exp 2.).
$\mathcal{L}_{\text{uniform-ground}}$: -0.92, $\mathcal{L}_{\text{uniform-satellite}}$: -1.33, $\mathcal{L}_{\text{uniform-all}}$: -1.48

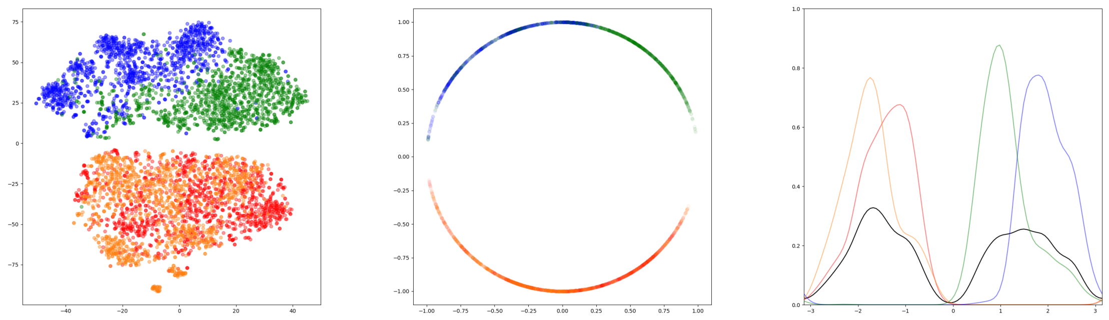


(b) Pre-training on ImageNet and satellite images of CVACT dataset (exp 5.).
$\mathcal{L}_{\text{uniform-ground}}$: -0.93, $\mathcal{L}_{\text{uniform-satellite}}$: -2.76, $\mathcal{L}_{\text{uniform-all}}$: -1.92

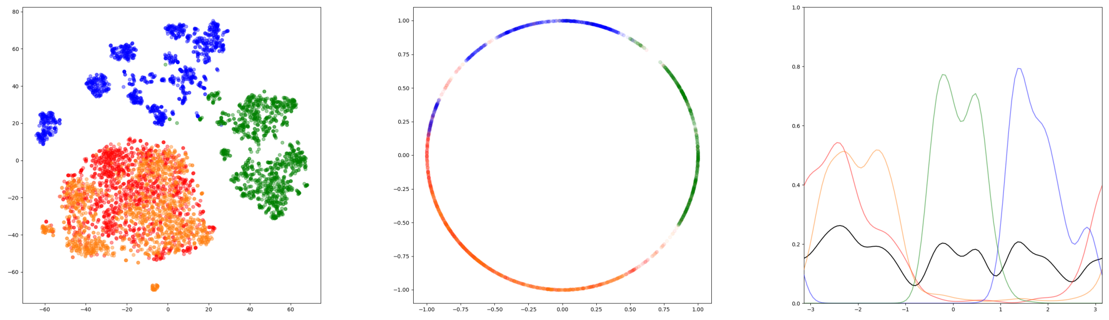


(c) Pre-training on ImageNet and both ground-level and satellite images of CVACT dataset (exp 6.).
$\mathcal{L}_{\text{uniform-ground}}$: -2.21, $\mathcal{L}_{\text{uniform-satellite}}$: -2.75, $\mathcal{L}_{\text{uniform-all}}$: -2.64

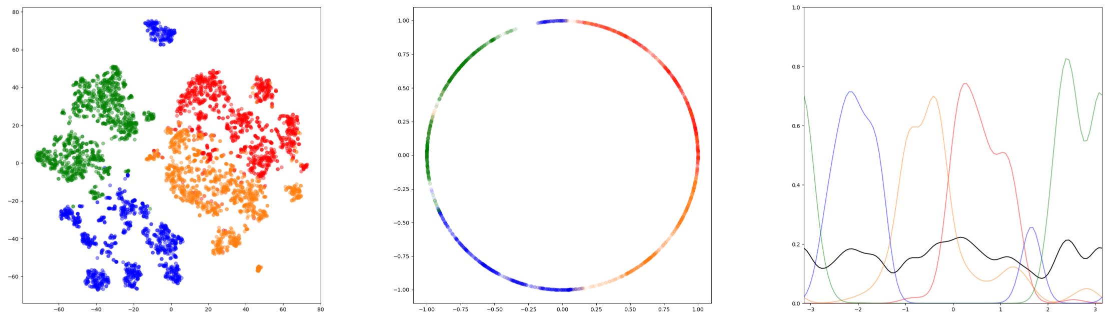
Figure 4.3: Representations of CVACT [28] validation set on $\mathbb{R}^2$ and $\mathbb{S}^1$ before downstream task training epochs. Column 1: features in $\mathbb{R}^2$ by t-SNE dimensionality reduction. Column 2: features displayed in $\mathbb{S}^1$ by l2-normalizing on features in $\mathbb{R}^2$. Column 3: features density estimation by Mises-Fisher (vMF) KDE on angles (i.e., $\text{arctan2}(y, x)$ for each point $(x, y) \in \mathbb{S}^1$). Red: ground-level image features. Blue: satellite image features



(a) Pre-training on ImageNet (exp 12.).
$\mathcal{L}_{\text{uniform-ground}}$: -1.08, $\mathcal{L}_{\text{uniform-satellite}}$: -1.35, $\mathcal{L}_{\text{uniform-all}}$: -1.51



(b) Pre-training on ImageNet and satellite images of VIGOR dataset (exp 16.).
$\mathcal{L}_{\text{uniform-ground}}$: -1.04, $\mathcal{L}_{\text{uniform-satellite}}$: -2.53, $\mathcal{L}_{\text{uniform-all}}$: -1.93



(c) Pre-training on ImageNet and both ground-level and satellite images of VIGOR dataset (exp 17.).
$\mathcal{L}_{\text{uniform-ground}}$: -2.05, $\mathcal{L}_{\text{uniform-satellite}}$: -2.53, $\mathcal{L}_{\text{uniform-all}}$: -2.66

Figure 4.4: Representations of VIGOR validation set on $\mathbb{R}^2$ and $\mathbb{S}^1$ before downstream task training epochs. Column 1: features in $\mathbb{R}^2$ by t-SNE dimensionality reduction. Column 2: features displayed in $\mathbb{S}^1$ by l2-normalizing on features in $\mathbb{R}^2$. Column 3: features density estimation by Mises-Fisher (vMF) KDE on angles (i.e., arctan2(y, x) for each point $(x, y) \in \mathbb{S}^1$). Red: ground-level image features of "SanFrancisco" split. Blue: satellite image features of "SanFrancisco" split. Orange: ground-level image features of "Chicago" split. Green: satellite image features of "Chicago" split

4.5.2. Observations and findings

Overall, the three different types of plots reflect the feature distribution qualitatively and the calculated uniformity metric evaluates it quantitatively. From Figure 4.3, the main observations are:

- Compare Figure 4.3 (a) with (b): from the first column, it could be found that features from satellite images (blue dots) are more uniform and less homogeneous. This could be shown in the first column and less obvious in the other two plots. Furthermore, the uniformity metric verifies this observation, with -2.76 in (b) and -1.33 in (a). A more uniform satellite image features distribution also brings the overall feature distribution to be more uniform (-1.92 vs -1.48).
- Compare Figure 4.3 (b) with (c): from the first column, it could be found that features from ground-level images (red dots) are less homogeneous. In (c), the red dots are clustered into different parts. This could also be shown in the third column: the features' occupation in the hyper-sphere is more average, rather than with peaks in (a) (the red curve). Furthermore, the uniformity metric verifies our observation, with -2.21 in (c) and -0.93 in (b). A more uniform satellite image features distribution also brings the overall feature distribution to be more uniform (-2.64 vs -1.92).

In Figure 4.4, since the experiments are performed on VIGOR [61] where there includes specific cities in the data splits, there are more information we could get from it. The main observations are:

- Compare Figure 4.4 (a) with (b): from the first column, it could be found that features from satellite images (blue and green dots) in (b) clearly separated between each other and form small blobs. Moreover, the blue and green dots also further distribute into different parts, while in (a) they stick together and are less centered. This indicates that image features from different cities are nicely separated. Another spot is that in the second column, more parts of the circle are occupied, producing a more uniform distribution. Also, in the third column, the black curve is with smaller peaks, which indicates a more uniform overall distribution. Furthermore, the uniformity metric verifies our observation, with -2.53 in (b) and -1.35 in (a). A more uniform satellite image features distribution also brings the overall feature distribution to be more uniform (-1.93 vs -1.51).
- Compare Figure 4.4 (b) with (c): we got similar observations for ground-level image features like those above. From the first column, it could be found that features from ground-level images (red and orange dots) in (c) are clearly separated from each other and form small blobs. Furthermore, the red and orange dots also clearly distribute into different parts, while in (b) they are centered together. Similarly, in the third column, the black curve is even flatter and shows a more uniform distribution. Then, the uniformity metric verifies these observation, with -2.05 in (c) and -1.04 in (b). A more uniform ground-level image features distribution also brings the overall feature distribution to be more uniform (-2.66 vs -1.93).

These observations within the experiments lead to consistent conclusions. It could be found that pre-training by contrastive learning on single-modality images helped the network separate them apart from each other and produces a more uniform distribution. Therefore, it is hypothesized that a more separated feature distribution at the beginning of training improves the downstream task performance.

4.6. Uniformity loss and data augmentation

This section exhibits the experiment results of section 3.4. In the previous results, standard contrastive learning pre-training improves the downstream task performance. We identify two key elements that could be beneficial during pre-training: uniformity within single-modality images and strong data augmentation. By Figure 4.4 and Table 4.4, it has been shown that more uniform representations as starting points generate eventually better cross-view matching performances. Moreover, the strong data augmentation used in contrastive learning could hugely increase the data amount delivered to the network.

We added these two elements directly in downstream task training to see if the performance also gets improvements. The experiments are also applied on using Siamese-VGG [60] on the VIGOR dataset [61]. The results are shown in Table 4.7. Note that these experiments are based on ImageNet pre-training (exp 12.). Therefore, exp 12. is again added in the tables below for easier comparison.

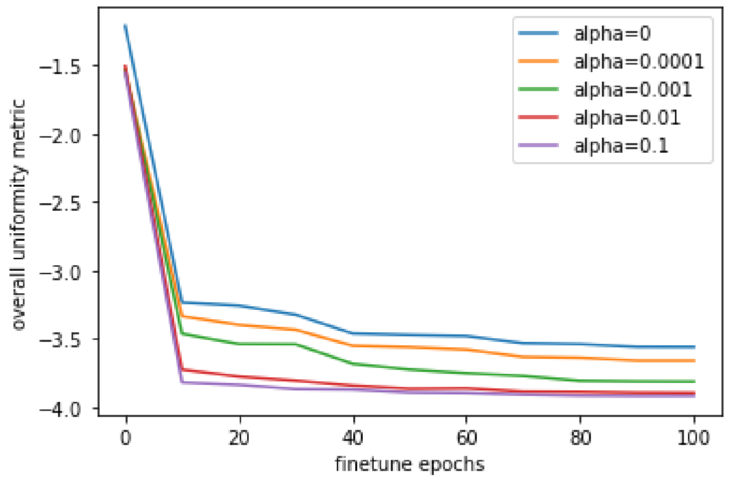


Figure 4.5: Overall uniformity metric on validation set along fintune epochs

Table 4.7: Cross-view matching performance of Siamese-VGG [60] with uniformity loss. The (downstream) training and evaluation are performed on VIGOR [61] dataset using the "Cross-Area" split setting (the training and validation set are located in separate cities).

Exp.	α	R@1	R@5	R@1%
12	0	3.28	9.21	62.16
27	0.0001	2.94	8.76	63.77
28	0.001	2.69	7.99	60.95
29	0.01	1.2	4.1	46.11
30	0.1	0.24	0.96	23.40

For different α, like in section 4.5, we calculated the overall uniformity metric $\mathcal{L}_{\text{uniform-all}}$ on the validation set to see the effect of adding more uniformity loss on the unified loss for optimization. This is shown in Figure 4.5. It could be seen that the distribution is more uniform (with a lower uniformity metric value) along with larger α. However, as shown in Table 4.7, uniformity loss makes the cross-view matching performance even worse. It failed to serve as an auxiliary loss to boost the performance. The main conclusion from it is that during the finetuning phase, the uniformity objective within each image viewpoint is not an ideal target for the network to optimize.

Table 4.8: Cross-view matching performance of Siamese-VGG [60] with augmentations. The (downstream) training and evaluation are performed on VIGOR [61] dataset using the "Cross-Area" split setting (the training and validation set are located in separate cities).

Exp.	Augmentation	R@1	R@5	R@1%
12	No aug-	3.28	9.21	62.16
31	MoCov2 aug-	0.45	1.61	30.40
32	MoCov2 aug- w/o *random cropping and resize*	4.13	11.70	69.42

The results of adding data augmentations are shown in Table 4.8. From the results in Table 4.8, it is shown that MoCov2 [7] augmentations without *random cropping and resize* improves cross-view matching performance by a small margin (4.13 in exp 32. and 3.28 in exp 12. at Recall@1). This meets our common sense that a richer set of data helps the network to improve its performance, and also verifies that the data augmentations used in self-supervised learning could be a positive element during training. Till now, since most cross-view matching methods do not use data augmentation and only a few works are exploring useful augmentation methods (e.g., adopting image segmentation as in [38]), there could be more future efforts to be paid in this direction.

5 Conclusion

In this chapter, we summarize and conclude our work in terms of the research questions proposed at the beginning, together with subsequent discussions and future works as well.

5.1. Answers to the research questions

In this work, we experimentally tested the effect of pre-training by performing contrastive learning on cross-view matching datasets separately for the two branches of the Siamese network, especially for the branch that encodes satellite images. For interpretation, we also visualized the feature distribution of different settings in the embedding space to see their behaviors. A major difference is that with self-supervised pre-training, the features of different image samples or from different cities are more separated and less homogeneous, resulting in a larger occupation in the embedding space.

Therefore, corresponding to the research question: As a downstream task, will cross-view matching benefit from self-supervised pre-training by contrastive learning? The overall answer is yes. We further discuss it in terms of the two sub-questions as below.

The first sub-question is: *Will the performance of cross-view matching be improved by pre-training using contrastive learning?* The answer is yes. The main experiments are presented in section 4.3 with quantitative results. We spotted that on top of the widely applied ImageNet pre-trained weights, it is an effective way to perform self-supervised learning using standard contrastive learning methods on the cross-view matching dataset to generate pre-training weights for downstream task training. This gives a hint that self-supervised pre-training may serve as a common approach to improve the existing network method in the application. Compared to the data amount contained in ImageNet (i.e., 1,281,167 images), although the added data for pre-training is much fewer (i.e., 35,532 images in CVACT [28] and 53,694 images in VIGOR [61]), it yields clearly visible performance improvement. This is very beneficial given the condition that satellite images can be easily accessed with few costs. It is hypothesized that training on satellite images provides the network with new knowledge on features in an aerial view for identifying places, while they are relatively rare in the normal ImageNet data.

The second sub-question is: *How will the image representations generated by the cross-view matching network be different by pre-training using contrastive learning?* The answer is another yes. The main works are shown in section 4.5 with qualitative visualizations. By visualizing the feature distribution in the embedding space, we found that self-supervised pre-training by contrastive learning on single-modality images (i.e., ground-level view or satellite view) helped the network separate them apart from each other. We have seen that adding the uniformity loss directly on the downstream task training does not improve or even results in a worse performance. Therefore, it is hypothesized that a more separated feature distribution at the beginning of downstream cross-view matching training improves the downstream task performance.

Moreover, another main conclusion from our experiments is that if the localization task is aimed at a specific target area or city, it is beneficial to collect dense satellite image patches covering the area to create a training set that is large enough for self-supervised pre-training, rather than using data from other areas. This could be shown from the result comparisons between pre-training on the training, validation and the whole dataset for VIGOR [61] dataset since it provides totally separated cities contained in the data splits. In a real-world scenario, the satellite images are all available to fetch from the remote sensing satellites, using e.g., GSV [2]. Self-supervised pre-training on the target area or city satellite images can sometimes give a best practice for training a cross-view matching network. Possibly this is because the data distribution and important features are already learned during the pre-training stage by exhibiting the images for the network.

5.2. Discussion and future works

There are still several limitations to our work. The experiments in our work are performed on only two network architectures and two datasets. The generalizability of this practice remains to be verified on more complicated methods, e.g., transformers used in TransGeo [59] or other methods that yield the most state-of-the-art performances. Moreover, the self-supervised learning method is chosen as MoCov2 [21] which uses manageable batch sizes and avoids the risk of collapsing by negative samples. There might be a better approach for self-supervised pre-training or specifically designed pretext tasks for cross-view matching. Our work provides a basic exploration of the effect of self-supervised pre-training on cross-view matching and there is still a very wide research area to be studied.

Another limitation of our work is that we focused on the self-supervised scheme by firstly pre-training the network with pretext tasks and then fine-tuning it with the downstream task. However, there might be better ways to improve cross-view matching, e.g., in an end-to-end manner. This relies on the observation that the architectures of cross-view matching and contrastive learning as shown respectively in the two stages in Figure 3.1 clearly share a similar Siamese-like paradigm, as they are both pulling positive pairs together and pushing negative pairs away. In such a sense, cross-view matching is essentially a type of contrastive learning, although this term is usually used in the context of self-supervised learning. However, although cross-view matching and contrastive learning share a similar architecture, the major difference reflects in the way how they define the input pairs. In cross-view matching, the query is a ground-level image and the key is the corresponding satellite image, so the data is labeled because we know the correspondence between these two image sources. In contrastive learning, the query and key are two augmented versions of the original image, so the correspondence is automatically formed, thus no human labeling is needed.

During the project, in order to find out the possibility as introduced above, we tried to combine contrastive learning architecture with cross-view matching to produce a unified version. However, this was left unfinished due to the time limits of the project. More information could be found in the Appendix. We hope the current information is still helpful for future works.

References

[1] URL: `https://developers.google.com/maps/documentation/maps-static/overview`.

[2] Dragomir Anguelov et al. "Google street view: Capturing the world at street level". In: *Computer* 43.6 (2010), pp. 32–38.

[3] Relja Arandjelovic et al. "NetVLAD: CNN architecture for weakly supervised place recognition". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 5297–5307.

[4] Eli Brosh et al. "Accurate visual localization for automotive applications". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops*. 2019, pp. 0–0.

[5] Ting Chen et al. "A simple framework for contrastive learning of visual representations". In: *International conference on machine learning*. PMLR. 2020, pp. 1597–1607.

[6] Xinlei Chen and Kaiming He. "Exploring simple siamese representation learning". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2021, pp. 15750–15758.

[7] Xinlei Chen et al. "Improved baselines with momentum contrastive learning". In: *arXiv preprint arXiv:2003.04297* (2020).

[8] SAE On-Road Automated Vehicle Standards Committee et al. "Taxonomy and definitions for terms related to on-road motor vehicle automated driving systems". In: *SAE Standard J* 3016 (2014), pp. 1–16.

[9] Jia Deng et al. "ImageNet: A large-scale hierarchical image database". In: *2009 IEEE Conference on Computer Vision and Pattern Recognition*. 2009, pp. 248–255. DOI: `10.1109/CVPR.2009.5206848`.

[10] Daniel DeTone, Tomasz Malisiewicz, and Andrew Rabinovich. "Superpoint: Self-supervised interest point detection and description". In: *Proceedings of the IEEE conference on computer vision and pattern recognition workshops*. 2018, pp. 224–236.

[11] Carl Doersch, Abhinav Gupta, and Alexei A Efros. "Unsupervised visual representation learning by context prediction". In: *Proceedings of the IEEE international conference on computer vision*. 2015, pp. 1422–1430.

[12] Ravi Garg et al. "Unsupervised cnn for single view depth estimation: Geometry to the rescue". In: *European conference on computer vision*. Springer. 2016, pp. 740–756.

[13] Spyros Gidaris, Praveer Singh, and Nikos Komodakis. "Unsupervised representation learning by predicting image rotations". In: *arXiv preprint arXiv:1803.07728* (2018).

[14] Ian Goodfellow et al. "Generative adversarial nets". In: *Advances in neural information processing systems* 27 (2014).

[15] Jean-Bastien Grill et al. "Bootstrap your own latent-a new approach to self-supervised learning". In: *Advances in Neural Information Processing Systems* 33 (2020), pp. 21271–21284.

[16] Yulan Guo et al. "Soft Exemplar Highlighting for Cross-View Image-Based Geo-Localization". In: *IEEE Transactions on Image Processing* 31 (2022), pp. 2094–2105.

[17] Raia Hadsell, Sumit Chopra, and Yann LeCun. “Dimensionality reduction by learning an invariant mapping”. In: *2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR’06)*. Vol. 2. IEEE. 2006, pp. 1735–1742.

[18] Stephen Hausler et al. “Patch-netvlad: Multi-scale fusion of locally-global descriptors for place recognition”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2021, pp. 14141–14152.

[19] Kaiming He et al. “Deep residual learning for image recognition”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 770–778.

[20] Kaiming He et al. “Masked Autoencoders Are Scalable Vision Learners”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. June 2022, pp. 16000–16009.

[21] Kaiming He et al. “Momentum contrast for unsupervised visual representation learning”. In: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 2020, pp. 9729–9738.

[22] Geoffrey E Hinton and Ruslan R Salakhutdinov. “Reducing the dimensionality of data with neural networks”. In: *science* 313.5786 (2006), pp. 504–507.

[23] Sixing Hu et al. “Cvm-net: Cross-view matching network for image-based ground-to-aerial geo-localization”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2018, pp. 7258–7267.

[24] Wenmiao Hu et al. “Beyond Geo-localization: Fine-grained Orientation of Street-view Images by Cross-view Matching with Satellite Imagery”. In: *Proceedings of the 30th ACM International Conference on Multimedia*. 2022, pp. 6155–6164.

[25] Longlong Jing and Yingli Tian. “Self-supervised visual feature learning with deep neural networks: A survey”. In: *IEEE transactions on pattern analysis and machine intelligence* 43.11 (2020), pp. 4037–4058.

[26] Alexander Kolesnikov, Xiaohua Zhai, and Lucas Beyer. “Revisiting self-supervised visual representation learning”. In: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 2019, pp. 1920–1929.

[27] Ang Li et al. “Cross-view policy learning for street navigation”. In: *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 2019, pp. 8100–8109.

[28] Liu Liu and Hongdong Li. “Lending orientation to neural networks for cross-view geo-localization”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2019, pp. 5624–5633.

[29] David G Lowe. “Distinctive image features from scale-invariant keypoints”. In: *International journal of computer vision* 60.2 (2004), pp. 91–110.

[30] Jiwen Lu, Junlin Hu, and Jie Zhou. “Deep metric learning for visual understanding: An overview of recent advances”. In: *IEEE Signal Processing Magazine* 34.6 (2017), pp. 76–84.

[31] Xiaohu Lu et al. “Geometry-aware satellite-to-ground image synthesis for urban areas”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2020, pp. 859–867.

[32] Piotr Mirowski et al. “Learning to navigate in cities without a map”. In: *Advances in neural information processing systems* 31 (2018).

[33] Ishan Misra and Laurens van der Maaten. “Self-supervised learning of pretext-invariant representations”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2020, pp. 6707–6717.

[34] Mehdi Noroozi and Paolo Favaro. “Unsupervised learning of visual representations by solving jigsaw puzzles”. In: *European conference on computer vision*. Springer. 2016, pp. 69–84.

[35] Aaron van den Oord, Yazhe Li, and Oriol Vinyals. “Representation learning with contrastive predictive coding”. In: *arXiv preprint arXiv:1807.03748* (2018).

[36] Deepak Pathak et al. “Context encoders: Feature learning by inpainting”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 2536–2544.

[37] Krishna Regmi and Mubarak Shah. “Generative adversarial for ground-to-aerial image matching”. In: *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 2019, pp. 470–479.

[38] Royston Rodrigues and Masahiro Tani. “Are These from the Same Place? Seeing the Unseen in Cross-View Image Geo-Localization”. In: *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*. 2021, pp. 3753–3761.

[39] Torsten Sattler et al. “Large-scale location recognition and the geometric burstiness problem”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 1582–1590.

[40] Paul Hongsuck Seo et al. “Cplanet: Enhancing image geolocalization by combinatorial partitioning of maps”. In: *Proceedings of the European Conference on Computer Vision (ECCV)*. 2018, pp. 536–551.

[41] Yujiao Shi et al. “Optimal feature transport for cross-view image geo-localization”. In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 34. 07. 2020, pp. 11990–11997.

[42] Yujiao Shi et al. “Spatial-aware feature aggregation for image based cross-view geo-localization”. In: *Advances in Neural Information Processing Systems* 32 (2019).

[43] Yujiao Shi et al. “Where am i looking at? joint location and orientation estimation by cross-view matching”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2020, pp. 4064–4072.

[44] Karen Simonyan and Andrew Zisserman. “Very deep convolutional networks for large-scale image recognition”. In: *arXiv preprint arXiv:1409.1556* (2014).

[45] Aysim Toker et al. “Coming down to earth: Satellite-to-street view synthesis for geo-localization”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2021, pp. 6488–6497.

[46] Laurens Van der Maaten and Geoffrey Hinton. “Visualizing data using t-SNE.” In: *Journal of machine learning research* 9.11 (2008).

[47] Pascal Vincent et al. “Extracting and composing robust features with denoising autoencoders”. In: *Proceedings of the 25th international conference on Machine learning*. 2008, pp. 1096–1103.

[48] Pascal Vincent et al. “Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion.” In: *Journal of machine learning research* 11.12 (2010).

[49] Nam Vo, Nathan Jacobs, and James Hays. “Revisiting im2gps in the deep learning era”. In: *Proceedings of the IEEE international conference on computer vision*. 2017, pp. 2621–2630.

[50] Nam N Vo and James Hays. “Localizing and orienting street views using overhead imagery”. In: *European conference on computer vision*. Springer. 2016, pp. 494–509.

[51] Tongzhou Wang and Phillip Isola. “Understanding contrastive representation learning through alignment and uniformity on the hypersphere”. In: *International Conference on Machine Learning*. PMLR. 2020, pp. 9929–9939.

[52] Daniel Wilson et al. “Visual and Object Geo-localization: A Comprehensive Survey”. In: *arXiv preprint arXiv:2112.15202* (2021).

[53] Zhirong Wu et al. “Unsupervised feature learning via non-parametric instance discrimination”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2018, pp. 3733–3742.

[54] Zimin Xia et al. “Cross-View Matching for Vehicle Localization by Learning Geographically Local Representations”. In: *IEEE Robotics and Automation Letters* 6.3 (2021), pp. 5921–5928.

[55] Zimin Xia et al. “Visual cross-view metric localization with dense uncertainty estimates”. In: *European Conference on Computer Vision*. Springer. 2022, pp. 90–106.

[56] Amir Roshan Zamir and Mubarak Shah. “Accurate image localization based on google maps street view”. In: *European Conference on Computer Vision*. Springer. 2010, pp. 255–268.

[57] Menghua Zhai et al. “Predicting ground-level scene layout from aerial imagery”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2017, pp. 867–875.

[58] Richard Zhang, Phillip Isola, and Alexei A Efros. “Colorful image colorization”. In: *European conference on computer vision*. Springer. 2016, pp. 649–666.

[59] Sijie Zhu, Mubarak Shah, and Chen Chen. “TransGeo: Transformer Is All You Need for Cross-view Image Geo-localization”. In: *arXiv preprint arXiv:2204.00097* (2022).

[60] Sijie Zhu, Taojiannan Yang, and Chen Chen. “Revisiting street-to-aerial view image geo-localization and orientation estimation”. In: *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*. 2021, pp. 756–765.

[61] Sijie Zhu, Taojiannan Yang, and Chen Chen. “Vigor: Cross-view image geo-localization beyond one-to-one retrieval”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2021, pp. 3640–3649.

6 Appendix

6.1. The network architectures

The image backbone architecture we used in section 3.1 and section 3.2 is VGG16 [44]. The architecture is listed below in a PyTorch style.

```
(vgg16): Sequential(
    (0): Conv2d(3, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): ReLU(inplace=True)
    (2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (3): ReLU(inplace=True)
    (4): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (5): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (6): ReLU(inplace=True)
    (7): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (8): ReLU(inplace=True)
    (9): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (10): Conv2d(128, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (11): ReLU(inplace=True)
    (12): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (13): ReLU(inplace=True)
    (14): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (15): ReLU(inplace=True)
    (16): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (17): Conv2d(256, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (18): ReLU(inplace=True)
    (19): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (20): ReLU(inplace=True)
    (21): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (22): ReLU(inplace=True)
    (23): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (24): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (25): ReLU(inplace=True)
    (26): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (27): ReLU(inplace=True)
    (28): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (29): ReLU(inplace=True)
    (30): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
)
```

The aggregation module of Siamese-VGG [60]:

```
(avgpool): AdaptiveAvgPool2d(output_size=(1, 1))
(classifier): Sequential(
  (0): Linear(in_features=512, out_features=512, bias=True)
  (1): ReLU()
  (2): Linear(in_features=512, out_features=128, bias=True)
)
```

The re-implemented sptial-aware embedding module of SAFA [42]:

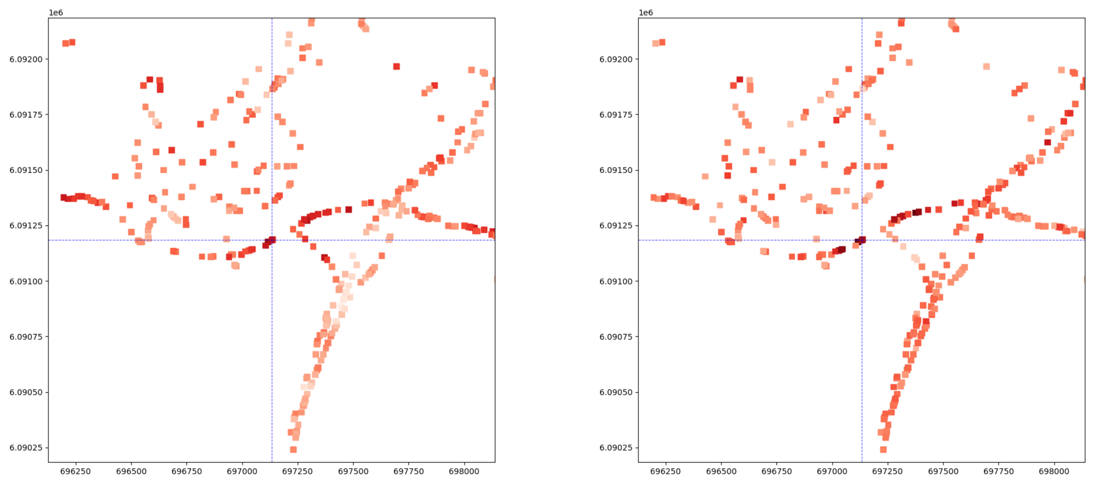
```
class SPE(nn.Module):
    def __init__(self, fmp_size):
        super(SPE, self).__init__()
        H, W = fmp_size
        self.fc1 = nn.Linear(H*W, H*W//2, bias=True)
        self.fc2 = nn.Linear(H*W//2, H*W, bias=True)

    def forward(self, fmp):
        B, C, H, W = fmp.shape
        # max pool
        fmp_pooled, _ = fmp.max(axis=-3, keepdim=False) #(B, H, W)
        # spatial-aware improtance generator
        x = fmp_pooled.flatten(start_dim=-2, end_dim=-1) #(B, H*W)
        x = self.fc2(self.fc1(x)).reshape(B, -1, H*W) #(B, D, H*W)
        # aggregate
        fmp = fmp.flatten(start_dim=-2, end_dim=-1) #(B, C, H*W)

        feat = torch.einsum('bci,bdi->bdc', fmp, x) #(B, C, D)
        feat = feat.flatten(start_dim=-2, end_dim=-1) #(B, C*D)
        return feat
```

6.2. MoCov2 data augmentation

We listed the data augmentation pipeline in a PyTorch style as below:

```
augmentation = [
    transforms.RandomResizedCrop(img_size, scale=(0.2, 1.)),
    transforms.RandomApply([
        transforms.ColorJitter(0.4, 0.4, 0.4, 0.1)  # not strengthened
    ], p=0.8),
    transforms.RandomGrayscale(p=0.2),
    transforms.RandomApply([moco.loader.GaussianBlur([.1, 2.])], p=0.5),
    transforms.RandomHorizontalFlip(),
    transforms.ToTensor(),
    normalize
]
```

6.3. Localization heatmap



(a) Localization heatmap of a sample in exp 2.

(b) Localization heatmap of a sample in exp 5.

Figure 6.1: Comparison between two experiments on an example of localization heatmap

Given a query, we could plot the localization heatmap by the calculated similarity during image

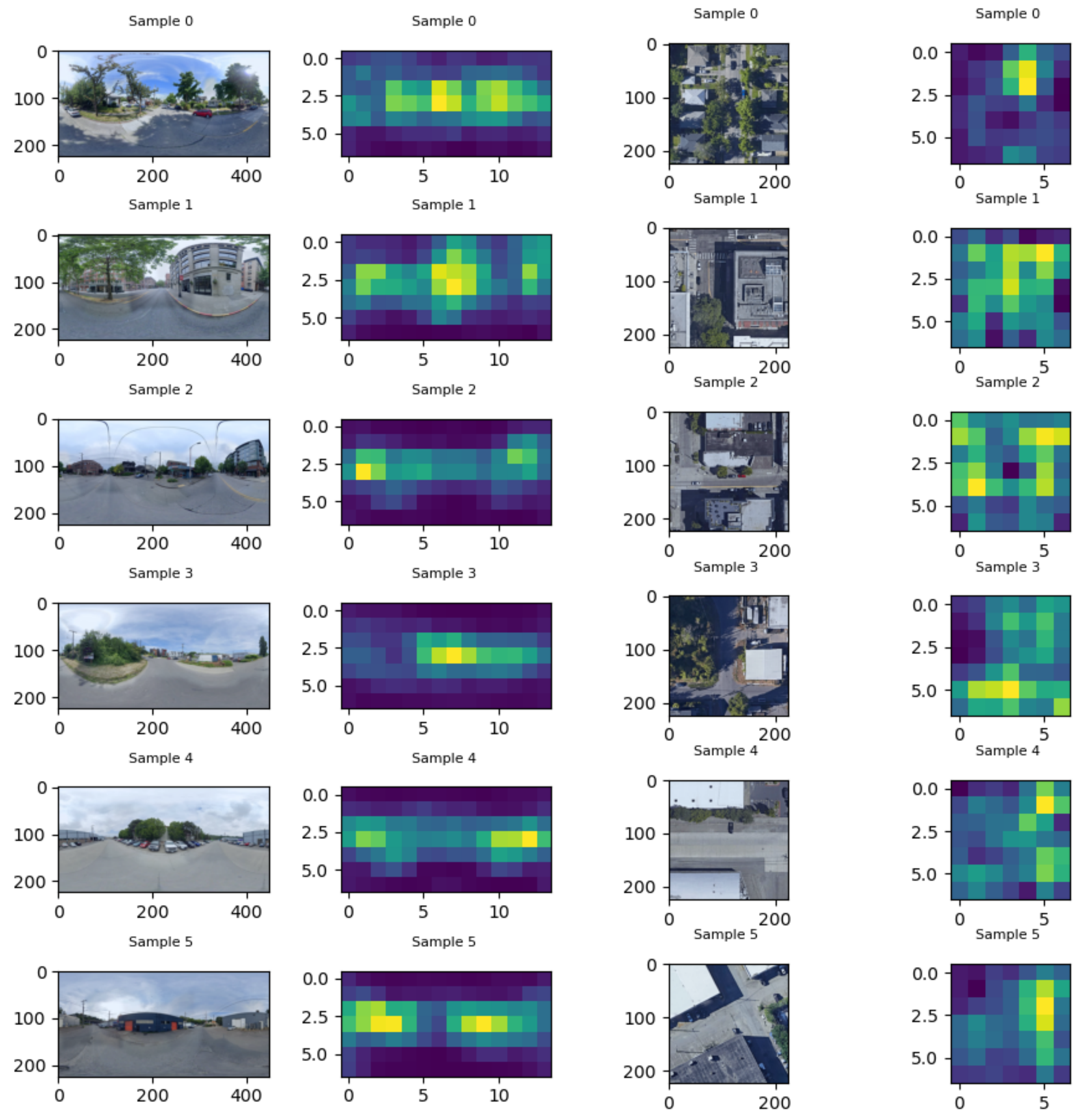


Figure 6.2: Example of a batch image samples and their corresponding feature map

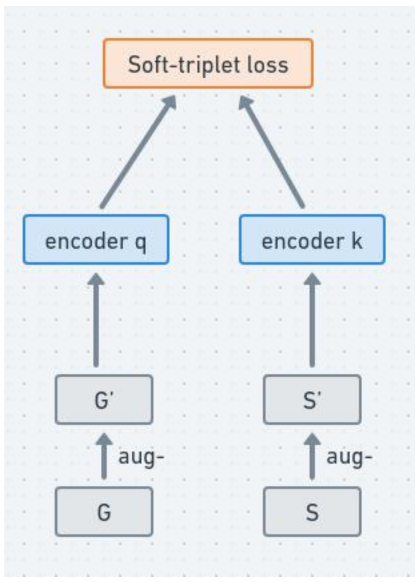
retrieval. An example is given in Figure 6.1. The cross of the two perpendicular blue dotted lines is the location of the query ground-level image, and the small red squares are the locations of satellite images on the map. The deepness of the red color indicates the similarity value. It could be shown that compared to exp 2., the satellite images around the query position have higher similarity values in the localization heatmap of exp 5.

6.4. Visualization of feature map

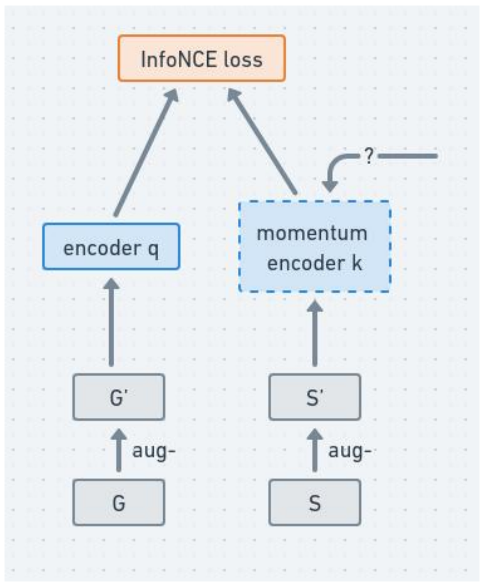
We could plot the feature map generated by the image backbone in cross-view matching (i.e., last layer output of f_θ and f_ξ) to have an insight of where are the important parts of the image that have high activation values. A group of samples of VIGOR [61] of exp 12. is shown in Figure 6.2. It could be roughly seen that many highlights are located at the road structure on both the ground-level and satellite images, which could be an important clue for image matching.

6.5. A unified version of MoCov2 and cross-view matching

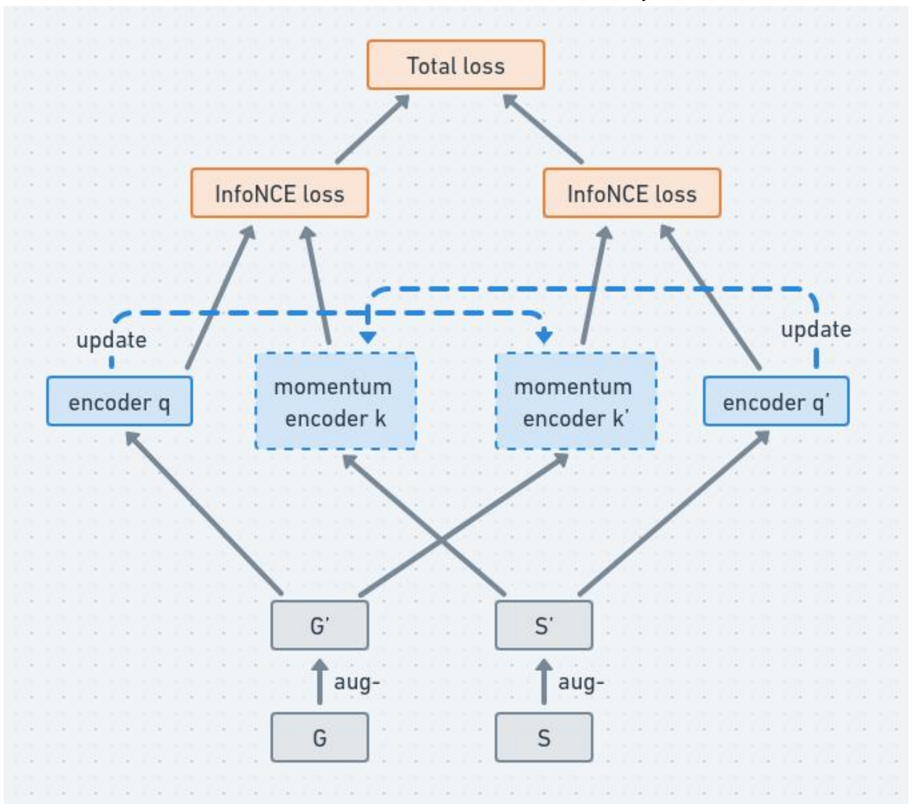
As mentioned before, we have another stream of work that tries to combine contrastive learning and cross-view matching in an end-to-end manner, but is left unfinished. In this section, we provide our implementation as a reference for future researchers.



(a) Original cross-view matching



(b) A natural way to add momentum encoder



(c) The proposed unified architecture of cross-view matching

Figure 6.3: Illustrations of introducing MoCov2 [7] momentum encoder to cross-view matching

Specifically, we combined MoCov2 [7]'s design with our cross-view matching architecture. The motivation is that the large negative memory buffer adopted in MoCov2 [7] plays an important role in contrastive learning, and since cross-view matching also follows such a Siamese-like framework, it could also be beneficial. Another aspect of the design is that complex data augmentation is also added to the pipeline.

To achieve this, we introduce the idea step-by-step. As shown in Figure 6.3 (a), the original cross-view matching uses two encoders q and k without weight-sharing. This is because their data inputs are different: one is augmented ground-level images G' and another is augmented satellite images S'.

To consider using a momentum encoder and memory buffer as in Mocov2 [7], a natural way will be to replace one of the two encoders with a momentum encoder that update weights using the other one, instead using back-propagation to itself, as explained in section 3.1. However, in terms of cross-view matching, it does not make sense to update the encoder for satellite images with the one for ground-level images. Therefore, as shown in Figure 6.3, the source of weights for encoder k is missed (that is what the question mark stands for).

Therefore, we tried the architecture as shown in Figure 6.3 (c). Essentially, it doubles MoCov2 [7] so that the momentum encoder in each one could be updated by the query encoder of the other pair. For example, in Figure 6.3 (c), the left two blue blocks form the first MoCo pair, with the ground-level image G' as the query input and the satellite image S' as the key. The right two blue blocks form the second pair, where the input data pair is flipped (the query is S' and the key is G'). Therefore, the momentum encoder k could be updated by the weights from the query encoder of the other pair, q', and in the same way, the momentum encoder k' could be updated by q. Moreover, in this architecture, two memory buffers are managed: one for the representations produced by k and another for k'.

The experiments are performed with Siamese-VGG [60] on VIGOR [61] dataset. Training with 40 epochs, the evaluation result is very low (R@1 = 0.28%), while the baseline method is 3.28% by exp 12. Since there still might be implementation issues, this result does not lead to concrete conclusions. We hope this section provides extra information about our work and leads to future works.