UAV Geolocalization Using Satellite Imagery

A Project Report Submitted

in the Partial Fulfilment of the Requirements

for the Degree of

Bachelor of Technology - Master of Technology Dual

by

Abhinav Tripathi

15807023

to

Prof. Ketan Rajawat

DEPARTMENT OF ELECTRICAL ENGINEERING

INDIAN INSTITUTE OF TECHNOLOGY, KANPUR

June 2020

CERTIFICATE

It is certified that the work contained in the project report entitled "UAV Geolocalization Using Satellite Imagery" by "Abhinav Tripathi" has been carried out under my supervision and that this work has not been submitted elsewhere for a degree.

June 2020

Prof. Ketan Rajawat
Associate Professor,
Department of Electrical Engineering,
Indian Institute of Technology,
Kanpur-208016.

DECLARATION

This is to certify that the project report titled "UAV Geolocalization Using Satellite Imagery" has been authored by me. It presents the research conducted by me under the supervision of Prof. Ketan Rajawat.

To the best of my knowledge, it is an original work, both in terms of research content and narrative, and has not been submitted elsewhere, in part or in full, for a degree. Further, due credit has been attributed to the relevant state-of-the-art and collaborations (if any) with appropriate citations and acknowledgements, in line with established norms and practices.

Signature

Name: Abhinav Tripathi
Roll No: 15807023
Programme: BTech-MTech Dual
Department: Electrical Engineering
Indian Institute of Technology Kanpur
Kanpur-208016

ABSTRACT

Name of the student: **Abhinav Tripathi** Roll No: **15807023**

Degree for which submitted: **BT-MT Dual** Department: **Electrical Engineering**

Report title: **UAV Geolocalization Using Satellite Imagery**

Supervisor: **Prof. Ketan Rajawat**

Month and year of submission: **June 2020**

Unmanned aerial vehicles or UAVs mostly rely on GPS data to precisely determine their global pose. However, data from GPS can sometimes be unreliable such as in case of loss of line-of-sight, presence of jammers, unfavourable weather conditions etc. In this work, we assume a GPS denied environment and match the images from UAV camera with the available satellite imagery. Such a technique can be utilized to estimate the global pose of the UAV in the absence of GPS.

We train a deep convolutional neural network consisting of two networks with unshared weights and achieve an accuracy of over 89% in determining whether a pair of images, one each from UAV and satellite, are taken from the same scene or not. We show that the dual networks can achieve a better performance as compared to the widely used siamese networks, when the images to be matched come from two specific distributions. We also describe a systematic approach of handling a large dataset while training a deep neural network and show that our technique can reach an accuracy of over 78% when using less than 7% of the entire dataset.

Acknowledgements

I am deeply grateful to my advisor Prof. Ketan Rajawat, for his continuous support, guidance and constructive recommendations on this research. I thank him for providing the freedom to explore various research paths and giving valuable feedback on them. I am thankful to the members of SPiN Lab, in particular to Lavish Arora, Mohan Krishna and Shubhanshu Khandelwal for the countless help they have provided during my graduate studies. I would like to appreciate the fastai deep learning MOOC, that helped me gain the valuable insights in this field, and without which this project would not at all have been this successful. I am thankful to my wingmates and friends for their warm and cheerful company.

Finally, I thank my parents for the unconditional love and support they have shown, and the sacrifices they have made, throughout my educational endeavours.

Dedicated to my family and this wonderful institute

Contents

Acknowledgements iv

List of Figures ix

List of Tables x

1 Introduction 1

1.1 Motivation . . . 1

1.2 Related Work . . . 2

1.2.1 Bag of Visual Words . . . 2

1.2.2 Matching Image Pairs using SIFT . . . 3

1.2.3 Aerial Imagery Matching . . . 3

1.2.4 One Shot Matching . . . 4

1.3 Goal and Main Contributions . . . 4

1.4 Overview of the Report . . . 4

2 Theory and Preliminaries 6

2.1 Siamese vs Dual Network Architecture . . . 6

2.2 Initialization . . . 7

2.2.1 Kaiming Initialization for Training from Scratch . . . 7

2.2.2 ImageNet Pretrained Weights for Transfer Learning . . . 8

2.3 Training Techniques on Pretrained Models . . . 8

2.3.1 Discriminative Layer Training . . . 8

2.3.2 Gradual Unfreezing . . . 9

2.4 ResNet18 Architecture Modification . . . 9
2.5 One Cycle Fitting . . . 11
2.6 Contrastive Loss Function . . . 12
2.6.1 Accuracy Metric . . . 12
2.7 Finding a Good Learning Rate . . . 13
2.8 Improving Training Time . . . 13

3 Dataset **15**
3.1 *Aerial Cities* Dataset . . . 15
3.2 Train-Validation Set . . . 17

4 Our Systematic Approach to Model Design **18**
4.1 Creating a Small Representative Dataset . . . 19
4.1.1 *Aerial Atlanta* Dataset . . . 19
4.2 Overfitting on *Aerial Atlanta* . . . 19
4.2.1 Effect of Initialization . . . 19
4.3 Reducing Overfitting . . . 21
4.3.1 Reaching a Baseline with Data Augmentation . . . 21
4.4 Improving Baseline Performance by Making Tweaks . . . 22
4.4.1 Discriminative Layer Training . . . 23
4.4.2 Using SiamResNet18+ Architecture . . . 24
4.5 Training on the *Aerial Cities* Dataset . . . 25

5 Results on Full Dataset **26**
5.1 Performance on *Aerial Cities* . . . 26
5.1.1 DualResNet18+ . . . 27
5.1.2 SiamResNet18+ . . . 28
5.2 Analysis of Wrong Predictions . . . 28
5.3 Insights . . . 29

6 Conclusions and Future Work **31**
6.1 Key Contributions . . . 31

6.2 Open Source Contributions 32
6.3 Future Work 32

References **33**

List of Figures

1.1 SIFT based feature matching for UAV and satellite image pair (from the same scene) fails and shows random matches 2

2.1 A representation of network architectures used 7
2.2 Layer grouping for discriminative layer training 8
2.3 Standard ResNet18 architecture 9
2.4 ResNet18 with custom head, refactored into 3 layer groups 10
2.5 Cosine annealing of learning rate and momentum variation for one cycle fitting 12

3.1 Matching image pairs from *Aerial Cities* dataset 16
3.2 Distribution of images across cities in the *Aerial Cities* dataset 17

4.1 Effect of initialization on training error 20
4.2 DualResNet18 with data augmentation (100 epochs, ImageNet initialization) 21
4.3 Effect of using discriminative learning rates 23
4.4 Dual network vs siamese network performance on *Aerial Atlanta* 25

5.1 Dual network vs siamese network performance on *Aerial Cities* 27
5.2 Matching pair wrongly predicted as non-matching 29
5.3 Non-matching pair wrongly predicted as matching 29

List of Tables

4.1 List of transforms used to reduce overfitting 22

4.2 Effect of data augmentation 22

4.3 Best results on *Aerial Atlanta* 24

5.1 Final list of transforms used on *Aerial Cities* 26

5.2 Best results on *Aerial Cities* 27

Chapter 1

Introduction

1.1 Motivation

Recent advances in research in the field of unmanned aerial vehicles or UAVs have made them publicly accessible and relatively inexpensive. There are numerous applications of UAVs for civilian, commercial and military uses. Not only these flying robots have been extremely useful for surveillance and monitoring tasks, but have also proven to be effective in disaster management, precision agriculture and aerial photography. For such outdoor applications, UAVs mostly rely on the global positioning system or GPS. However, for accurate geolocalization using GPS, the UAV must be able to receive a direct line of sight from four or more GPS satellites. This can be an issue in case of presence of high buildings, mountains or jammers which can obstruct the signals from satellites. Moreover, in case of military enforcements, sometimes the availability GPS satellites can also be an issue.

Hence, there is a need of having an accurate global pose determining strategy in GPS denied environments. Even though accurate systems like visual inertial simultaneous localization and mapping (VI-SLAM) exist but they are not flawless. They accumulate a lot of drift as time of flight increases, and this drift is corrected only when a loop closure is found, *i.e.*, the UAV revisits a place it has visited before.

The problem of geolocalizing a UAV can be solved by matching geo-referenced satellite images with the images coming from the UAV camera. A geo-referenced image is simply an image taken from a satellite with known pose. Matching the images from the feed of UAV camera

(a) UAV Image (b) Satellite Image (c) SIFT Matching

Figure 1.1: SIFT based feature matching for UAV and satellite image pair (from the same scene) fails and shows random matches [1]

with geo-referenced satellite images can help to get an estimate of the global pose of the UAV. In this work, we constrain ourselves to aerial image matching with wide baseline changes as in the case of images from satellite and UAV. *To this end, we train a deep learning model that can accurately match images from satellite and UAV camera.*

1.2 Related Work

1.2.1 Bag of Visual Words

In this technique, we first find points of interest in the images to be matched. These points are also known as keypoints, and this process is called as feature detection. These interest points are then described mathematically by a vector of fixed dimension. This vector is known as the descriptor of the interest point. There are a number of feature detector and descriptor algorithms like SIFT [1], SURF [2], and ORB [3]. There also exist deep learning based feature descriptors like LIFT [4], SuperPoint [5] and D2Net [6]. In order to find images that are similar to a given query image from a database, using feature descriptors, a bag of visual words approach [7] can be used where similar descriptors are clustered together and the mean of these descriptors is taken as one visual word. Each image is then reduced to a histogram of occurrences of visual words. The importance of each visual word is determined

[1] 1.1a and 1.1b have been taken from Google Earth and Google Maps respectively. No written permissions are required for printing these images in academic reports and presentations, as per the official guidelines (see https://www.google.com/permissions/geoguidelines/ for details).

by using TF-IDF (term frequency, inverse document frequency), which gives more weight to more distinct and expressive visual words. Finally, a cosine distance is used to find the distance between histograms of images. The images corresponding to lowest cosine distance of histograms are returned.

1.2.2 Matching Image Pairs using SIFT

SIFT algorithm is extremely popular and its features are invariant to rotation, illumination and uniform scaling. They are also partially invariant to affine transformation. However, for our task SIFT features do not perform well because the satellite and UAV images come from different distributions. Not only do the image pairs have a considerable change in viewpoint, but also have different hues and colours, which affect the performance of handcrafted features like SIFT. An example of such a case is shown in Figure 1.1.

1.2.3 Aerial Imagery Matching

There are a number of studies on matching aerial images. In [8], the authors propose an attention mechanism and produce a set of putative matches using a spatial transformer. Instead of matching entire images, they match the proposed regions using a siamese network architecture. In [9], the authors replace the SIFT detector with a denser detector that allows boundaries of superpixels as features. However, the final matching is done using SIFT descriptor under the assumption that the images have same scale and orientation. [10] use a triplet network (VGG architecture) with tied weights to mine deep features. This is followed by a hashing layer which converts the deep features into a binary feature vector of 128 dimensions. This allows fast nearest neighbour searching in Hamming space. In [11], authors use intersection of roads to match aerial images as road intersections are sparse and distinctive. Matching is done on a stored database of OSM intersection maps. However, for sufficiently low flying UAVs or near greenlands, there is a possibility of not having any road intersection in the UAV image. In [12], GANs have been used in order to generate aerial images from ground images. These generated images are used to match ground and aerial images. Dual AlexNets initialized with Places365 [13] weights are used in [14] to perform aerial image matching with wide-baseline changes.

1.2.4 One Shot Matching

In this technique, a convolutional neural network is used to encode an image into a vector. To compare it with another image, the same network with same parameters is used to encode the other image, producing a new representative vector. Now these encodings can directly be compared using a distance metric. To train these networks, parameters are learnt such that the distance between the encodings of similar images is lesser than the distance of the encodings of dissimilar images. Examples include [15] and [16].

1.3 Goal and Main Contributions

The goal of this work is to formulate a scheme for matching aerial images with wide baseline changes, coming from different distributions. This scheme can be used to match UAV images with geo-referenced satellite imagery and to find out the global pose of the UAV in a GPS denied environment.

In sum, we make **four main contributions**. (i) We train a deep CNN for matching UAV and satellite images. Our model can predict whether a pair of aerial images come from same scene or not with an accuracy of 89%. The images in the pair represent images taken from satellite and UAV, and have been captured from very different viewpoints. (ii) We show that dual networks can perform better than the widely popular siamese networks, when the images to be matched come from two specific distributions. (iii) We present a systematic method of handling large datasets where the insights from a small subset of dataset can be used to train the model on entire dataset, which saves both — time and resources. (iv) Finally, we show that as a result of the aforementioned method, our model can learn to predict whether a pair of images matches or not with an accuracy of over 78% even if there is limited amount of training data (less than 4k matching pairs) available.

1.4 Overview of the Report

In Chapter 2, we elucidate the concepts and tricks we have used for training our deep learning models.

In Chapter 3, we describe in detail the dataset we have used for aerial imagery matching.

In Chapter 4, we describe a systematic way of handling a large dataset, where the insights from a smaller dataset are used to train our models.

In Chapter 5, we list out our results on the entire dataset. We also provide an analysis of the failure cases with examples.

In Chapter 6, we conclude, highlighting our contributions, with directions to future work.

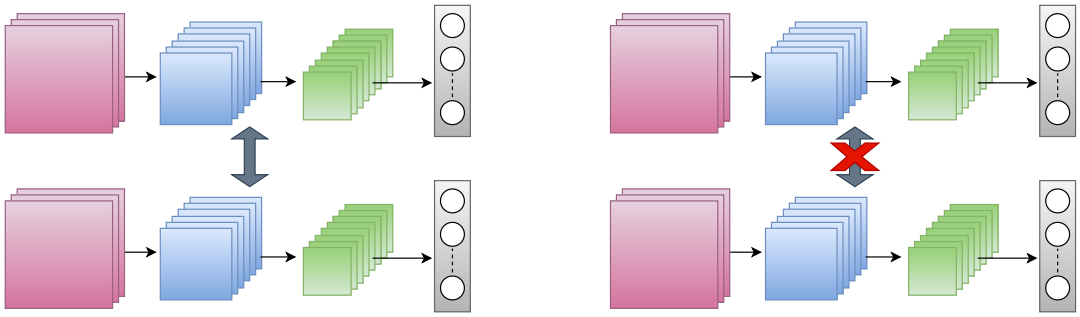
Chapter 2

Theory and Preliminaries

In this chapter we define some important concepts which we use while creating our models.

2.1 Siamese vs Dual Network Architecture

In our work, we use the following standard architectures for image matching. The first one is siamese network where the weights are shared between the two branches of the network. The second one, which we call dual network, is simply a siamese network with unshared weights. The siamese networks were introduced mainly for the task of facial recognition, however they have been adapted for a number of similar tasks. Since they share weights, it can be difficult for these networks to process an image pair in which both images represent the same object or scene, but are widely dissimilar looking. This is the reason that we use dual networks in our experiments. A dual network has twice the number of parameters as compared to a siamese network with the same backbone. It has a separate feature extractor for each image distribution, which allows the network to learn richer features for each of the distribution. Hence, the network can learn the similarities between a pair of images even if they look widely dissimilar to each other, as is the case for satellite and UAV imagery. While using a dual network, it must be made sure that the image from one distribution always enters the network from the branch specific to that distribution. Figure 2.1 shows the difference between the two types of networks for the task matching.



(a) Siamese Network (b) Dual Network

Figure 2.1: A representation of network architectures used

2.2 Initialization

There are a number of methods used to initialize the parameters of any neural network [17], [18], [19], [20]. It has been shown that it is not good to start with randomly distributed weights to train a deep neural network as this hurts variance propagation of activations to the deeper layers of the network [18] and hence it is very important to have properly initialized parameters. In this section we describe the two initialization strategies we used to train our network. We first use the Kaiming Normal initialization method which was introduced for the training of the deep residual networks [20] and compare its performance with transfer learning. In our experiments, we found out that for the networks not initialized with Kaiming or ImageNet weights (eg, default alexnet model in pytorch), our loss gets stuck around $2.5k$ (and the distance between all image pairs at 50). See Section 2.6 for details of the loss function.

2.2.1 Kaiming Initialization for Training from Scratch

It has been shown for a network with ReLU activation function that if the weights at the l^{th} layer are initialized with a zero mean Gaussian having variance $\sqrt{2/n_l}$, then the variance of activations does not die out, but remains close to unity as the activations move deeper into the network during a forward pass. A similar effect is seen on the gradients during a backward pass as well. Here n_l is the number of activations in layer l. This is generally known as Kaiming initialization scheme. We use this scheme for initializing the weights when training a neural network from scratch. We set the bias vectors to 0 during initialization.

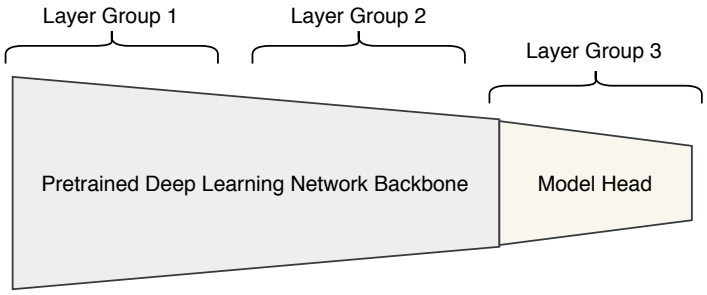


Figure 2.2: Layer grouping for discriminative layer training

2.2.2 ImageNet Pretrained Weights for Transfer Learning

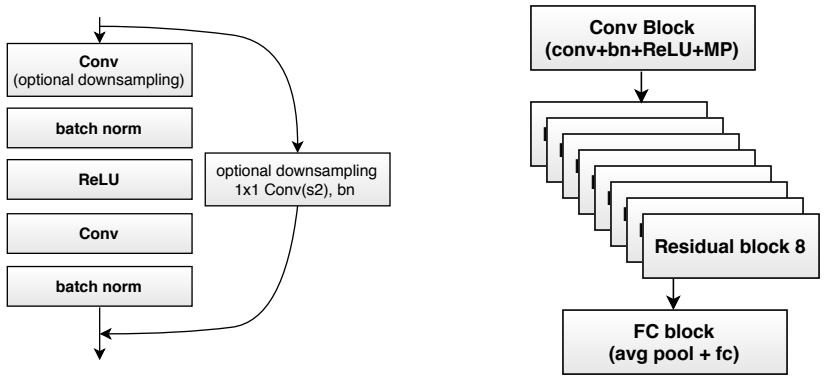
Instead of training a neural network from scratch, we can use the available pretrained ImageNet models for transfer learning. This allows us to use the previously learnt knowledge in the model (on ImageNet task) for the task of aerial imagery matching.

2.3 Training Techniques on Pretrained Models

In this section, we introduce two training techniques we use for tuning pretrained models. Both of them are inspired from the ULMFIT paper [21].

2.3.1 Discriminative Layer Training

Consider a network pretrained on ImageNet weights. In the initial layers of the network, it contains parameters that can identify low level features like blobs, corners, edges, colour gradients and simple patterns [22]. As we go deeper into the model, the layers contain parameters that can identify high level features like human faces, car tyres, written text etc. The idea of discriminative layer training is to not change the initial layer weights too much because they already contain weights which can be used for almost all the computer vision tasks (as most of the images contain some corners, edges and patterns).
In order to do this, we divide the model into three layer groups: the initial layer group, the middle layer group and the head of the model in order of increasing depth. While training the entire model, we do not train all the layer groups with the same learning rates. Instead, we train the head with a higher learning rate and decrease the learning rate for initial layers.



(a) Residual Block (b) ResNet18

Figure 2.3: Standard ResNet18 architecture

Figure 2.2 shows the strategy we use for grouping the layers. This makes sure that the fundamental features already learnt by the model are not disturbed.

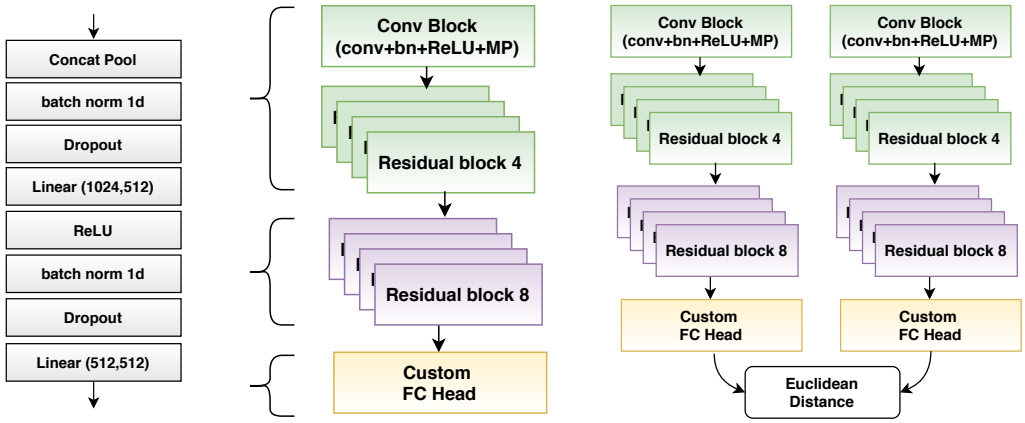
2.3.2 Gradual Unfreezing

While finetuning all the layers in transfer learning, the model can undergo catastrophic forgetting, where it can forget most of its knowledge from the parent task. Hence, it is not favourable to fine-tune all the layers at once. So we use the concept of gradual unfreezing where we unfreeze one layer group at a time for fine-tuning. We first train the head keeping the backbone frozen which is generally the case with transfer learning. Then we unfreeze the middle layer group and train the middle group and head with different learning rates as described above. Finally, we unfreeze the whole model and train all the layer groups using discriminative layer training.

2.4 ResNet18 Architecture Modification

The standard ResNet18 is shown in Figure 2.3 for reference. It consists of following three parts:

1. **Initial Convolutional Block:** This part consists of convolutional, batch norm, ReLU and maxpooling layers. Note that we use the 64 channel 7×7 convolutional filters in



(a) Custom FC Head (b) ResNet18+ (c) Dual/Siam ResNet18+

Figure 2.4: ResNet18 with custom head, refactored into 3 layer groups

this block as in the original paper [23].

2. **Residual Block:** There are 8 residual blocks in ResNet18. Each of these 8 blocks has an identity connection with an optional downsampling. Whenever there is downsampling in the main path, 1×1 convolutions with stride 2 are used in the identity connection along with a batch norm layer. The main path of the residual block consists of two convolutional layers (followed by batch norm layers) with a ReLU non-linearity in between them.

3. **Fully Connected Block:** After the 8 residual blocks, we get an output of 512 channels with 7×7 activations. This is flattened into a 512 dimensional vector using an average pooling layer. Finally, there is a fully connected layer which takes 512 long vector and outputs 1000 dimensional vector for each of the ImageNet categories.

We refactor the standard ResNet18 into three layer groups for discriminative layer training. We also modify the fully connected block of the network, such that the model is able to learn richer features. The layer grouping process is enumerated below:

1. **Layer Group 1:** It consists of the initial convolutional block and the first four residual blocks. The idea here is that these layers are going to contain most of the fundamental features like edges, corners, colour gradients etc. We can use this layer group without

much change for our task. This layer group is also referred to as the initial layer group.

2. **Layer Group 2:** It consists of the last four residual blocks of ResNet18. The learnt visual features in these layers are more specific to the ImageNet task. Hence, we will retrain and modify them to find features specific to our dataset. This layer group is also referred to as the middle layer group.

3. **Layer Group 3:** It is the custom head of the model which we have designed to replace the fully connected block of the standard architecture (see Figure 2.4a). We replace the average pooling layer of the standard ResNet18 with a "concat pooling layer", which concatenates the outputs from maxpooling and average pooling. It has been shown that it can yield a better performance than using either single one. It also doubles the number of features from 512 to 1024. It is followed by a linear layer which changes the number of activations from 1024 to 512. Finally, we use another linear layer which takes 512 inputs and gives 512 outputs. These 512 outputs are used as feature vectors of the input images. We use a ReLU non-linearity in between both the linear layers. Each of the linear layers is preceded by batch norm and dropout layers. The ordering of these layers is inspired from the fastai default ResNet models which achieve state of the art performance on a number of computer vision tasks. We call this layer group as the head or final layer group.

The refactored architecture along with the custom head is shown in Figure 2.4. We call this model as ResNet18+ in order to differentiate it from the standard model. Similarly, the dual and siamese networks with modified architectures are called DualResNet18+ and SiamResNet18+. These architectures are shown in Figure 2.4c.

2.5 One Cycle Fitting

In this policy, we find a progressively increase our learning rate to a maximum value, and then decrease it to zero as suggested in [24]. We do the opposite to the momentum and decrease it while the learning rate is increasing, and increase it while the learning rate is decreasing. Intuitively, the idea is to start the training slowly and not change the parameters

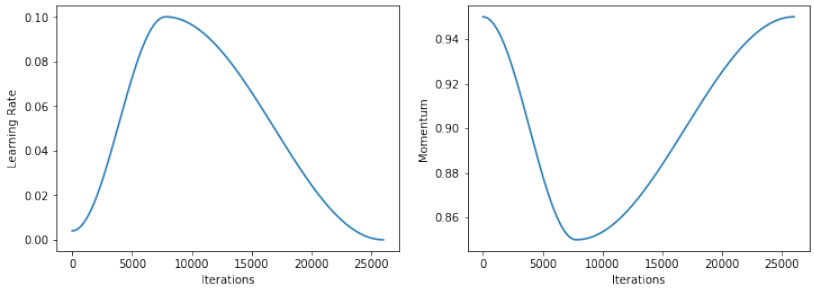


Figure 2.5: Cosine annealing of learning rate and momentum variation for one cycle fitting

too much so that the model does not have many unusable activations. Then the learning rate is gradually increased to the maximum value for faster training. This is also shown to have a regularization effect on the network. Finally, the learning rate is decreased which makes sure that the parameters are finetuned as is the general practice. An example of such a variation is shown in Figure 2.5.

2.6 Contrastive Loss Function

We use contrastive loss as shown in Equation (2.1) with a margin of 100. When the pair of images is matching, $l = 1$, and the loss is just square of the distance between the feature vectors of the image pair. When the image pair is non-matching, $l = 0$ and the loss is determined by how far is the distance between the image features from the margin. Note that if the distance is greater than the margin, no loss is incurred for a non-matching pair.

$$L = l \times d^2 + (1 - l) \times (\max(0, m - d))^2 \tag{2.1}$$

Here L is the loss function, l is the label, m is the margin, and d is the distance between the feature vectors of the images. By using this loss function, we make the network output a small distance for matching pair, and push the distance of the non-matching pairs up to m.

2.6.1 Accuracy Metric

We use the regular accuracy metric. Since we are using $m = 100$, we set a threshold of 50 on the output distance in order to check whether a prediction is correct or not. Hence, a

matching pair with $d < 50$ is considered as a true positive, and a non-matching pair with $d \geq 50$ is considered as a true negative.

2.7 Finding a Good Learning Rate

In our experiments we found that it is critical to find a good learning rate, a small learning rate would slow down our training, and using a high value of learning rate would make the network diverge. Hence to find a good learning rate, we increase the learning rate from a minimum value to a maximum value exponentially and feed each batch to the network with a different learning rate. We plot a graph between the loss and learning rate to get an idea of a good value of learning rate. We use the fastai `lr_find` function to do this. The rest of the parameters are set to fastai defaults.

2.8 Improving Training Time

We use three practical tricks which greatly improve the time to train our models:

1. **Saving Resized Images on Disk:** By resizing the images and saving them on the disk, we found out that we could reduce the time for each epoch down to its one-third on Nvidia V100. All the experiments were performed using a Solid State Drive.

2. **Batchwise Data Augmentation:** Instead of transforming the images on the fly, we found that it is better to perform data augmentation batchwise on GPU. This makes sure that the time spent during transforming the images is insignificant.

3. **Mixed Precision Training:** Mixed precision training [25] is a strategy where the forward pass is made using floating points with a precision of 16 bits. However, the backward pass and the gradient calculations are done using floating points with a precision of 32 bits. In general, we usually use the 32 bit floating point numbers while training. However, it was shown that we do not require such high precision while making a forward pass. High precision is required to calculate the gradients and update the weights. Hence mixed precision training can help to reduce the training time without sacrificing the model accuracy.

However, in our experiments we found that we could get a significant boost while using an earlier generation GPU (4GB Nvidia Geforce 940M) using this strategy, but there was no difference in the training time for the new generation 16GB Nvidia V100 GPU. We suspect that this was because either of these two reasons: (i) the VM instance with V100 being slowed down because of its full CPU usage, or (ii) the GPU is so powerful that it can perform calculations in high precision with minimal effects of training time. On 940M, the time for training the whole dataset was reduced from 1hr18min per epoch to 55min per epoch.

Chapter 3

Dataset

There are many datasets with aerial imagery specific to different tasks. INRIA labelling dataset [26] deals with binary semantic segmentation of buildings. ISPRS 2D and 3D datasets deal with semantic segmentation and reconstruction, but they not only include more categories like low vegetation, trees, cars, background, etc., but also but include digital surface models. DOTA [27] is a dataset for the purpose of object detection in aerial scenes and contains classes like planes, large-vehicles, small-vehicles, roundabouts, baseball fields, etc. The dataset used in [8] contains aerial images of same scene from different viewpoints, but the camera is not facing nadir, and hence not useful for our task; also it is not publicly available. Stanford drone dataset [28] contains videos taken from drones, annotated with bounding boxes for pedestrians, bicyclists, skateboarders, cars, etc. In our experience, there are not many datasets specifically for the task of aerial imagery matching. In this work, we use the dataset introduced by [14], referred to as *Aerial Cities* hereafter. The dataset is described here for completeness.

3.1 *Aerial Cities* Dataset

The training set in this dataset consists of satellite and UAV images from nine cities, and the test set consists of satellite and UAV images from three cities. All the cities are from the United States. Since the labels for the test set are not publicly available, we use the training set for training and validation, and the results are shown on the validation set. The UAV images are captured from Google Earth, and the satellite images are captured from Google

(a) Satellite Images

(b) UAV Images

Figure 3.1: Matching image pairs from *Aerial Cities* dataset

Maps. The satellite images are always oriented towards the North (0° heading), and the images are taken from the camera facing the nadir. The UAV images have a random heading (between 0° and 360°) and a random tilt from nadir (between 0° and 45°). The UAV images have been taken from an altitude between $100m$ and $200m$ above the ground. The satellite images have been taken from an altitude of 300m above the ground.[1] A pair of images is called matching if the distance of intersection line-of-sight of their cameras with ground is within 40m of each other.

All the images in the dataset have a size of 480×480 pixels. Examples of matching image pairs are shown in Figure 3.1. The distribution of images with respect to the cities in the dataset can be seen in Figure 3.2.

[1]Note that we found out that there are slight variations from the claimed altitudes. The reason is using a fixed average ground altitude for each city.

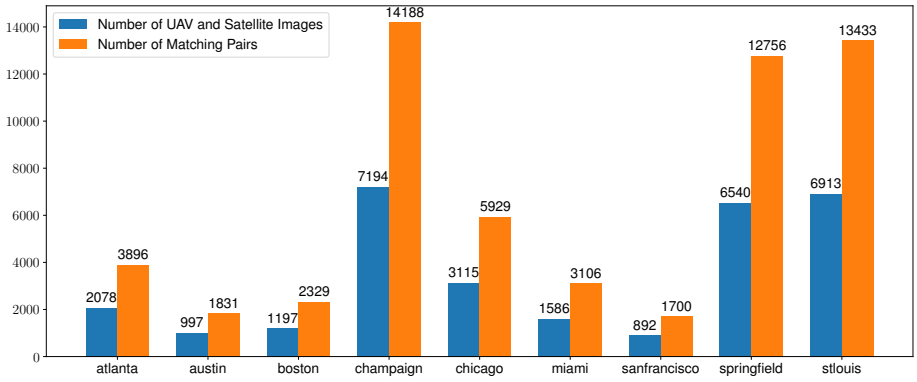


Figure 3.2: Distribution of images across cities in the *Aerial Cities* dataset

3.2 Train-Validation Set

We construct the matching image pairs using the given ground truth, and generate same number of non-matching images by randomly choosing satellite images for each UAV image. We then split these pairs in $4:1$ ratio for training and validation. We make sure that the non-matching pairs in the validation set contain the same UAV images as the matching pairs in the validation set. This makes sure that the model is validated on the UAV images it has never seen before. Hence, there is an implicit assumption that all the satellite images are already available beforehand. This split mimics the practical situation where this model is used live in the cities on which it has been trained using all the satellite imagery available.

Chapter 4

Our Systematic Approach to Model Design

Since our dataset is very large, we require a systematic approach for training our model in order to save time and resources. Hence, we do not start by training on the entire *Aerial Cities Dataset*. Instead we create the *Aerial Atlanta* dataset to test our different strategies. Finally, we use the best strategy on the entire dataset. We want to find an approach *which not only gives us good results, but also trains quickly.* We follow the following five step methodology to find the conditions under which the model trains fast and gives good results.

1. Creating a small representative dataset
2. Overfitting on the small dataset
3. Reducing overfitting
4. Improving baseline performance by making tweaks
5. Training on the entire dataset

In the following sections, we look at each one of these points in detail.

4.1 Creating a Small Representative Dataset

Working with a small subset of the main dataset is a good technique to quickly experiment before trying to train the entire dataset. This not only saves us time and resources, but also helps gain insights which can be extremely helpful while trying out the full dataset.

4.1.1 *Aerial Atlanta* Dataset

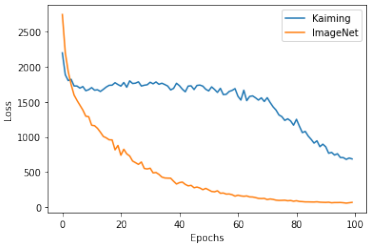
We create the *Aerial Atlanta* dataset which contains the images only from Atlanta city. In total, it consists of 6.81% of the images from *Aerial Cities*. We use the given ground truth to generate the matching pairs. In order to generate the non-matching pairs, we randomly select a non-matching satellite image for each UAV image. We use this dataset as a representative of *Aerial Cities*. We split all the image-pairs into train and validation set with a ratio of $4:1$. We also make sure that the UAV images in the validation set have not been seen by the model during the training. However, we allow the presence of satellite images in the non-matching pairs of the training set, in case they were selected while randomly choosing the satellite images. This is in accordance to a situation where such a model will be used in production. During training, the UAV images might not be available for the whole city, but the satellite images for the whole city can be used. The results mentioned are on the validation set itself.

4.2 Overfitting on *Aerial Atlanta*

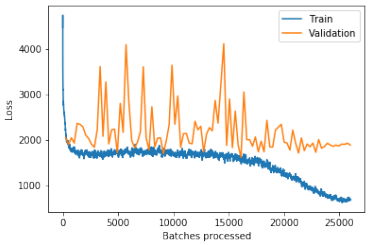
Overfitting demonstrates that the network actually has the capability to learn from the training set. It also serves as a sanity test, and checks if any errors have crept into our code. A model that cannot overfit will not be able to train well. We use no data augmentation in this case, and allow the model to learn the specific examples in the training set. Our goal in this case is to find out the technique which allows us to overfit quickly.

4.2.1 Effect of Initialization

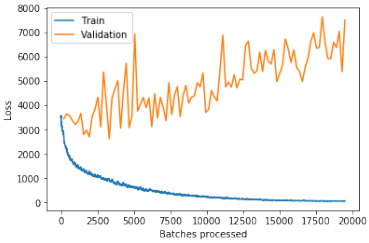
We use the DualResNet18 network architecture. First we use Kaiming initialization and then compare its performance with the same network initialized with ImageNet weights. In both the cases, we train our model for 100 epochs. We use One Cycle Policy and find the best



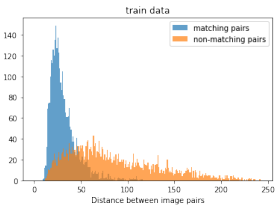
(a) Train error

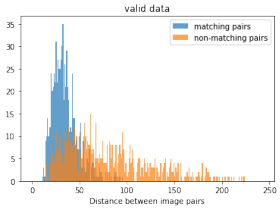


(b) Kaiming Initialization

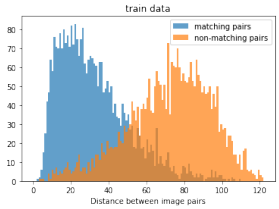


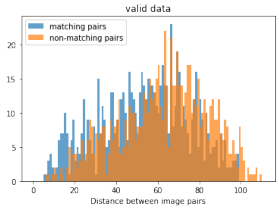
(c) ImageNet Initialization





(d) Distance Histograms (Kaiming Init)





(e) Distance Histograms (ImageNet Init)

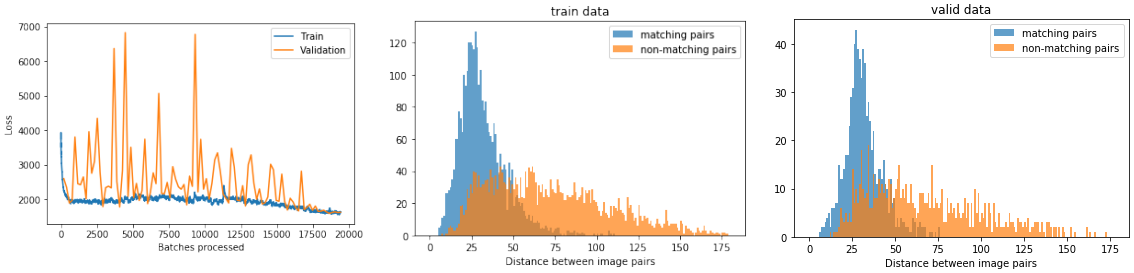
Figure 4.1: Effect of initialization on training error

learning rate using the technique discussed earlier in Chapter 2. We use the contrastive loss function with a margin of 100.

It can be seen in Figure 4.1a that the network initialized with ImageNet weights trains much faster compared to the one initialized with Kaiming initialization method. In case of ImageNet weights, the validation loss decreases upto epoch 16, and then increases rapidly, which is a sign of overfitting (see Figure 4.1c). We can also see the effect of overfitting on the distance between the matching and non-matching pairs (see Figure 4.1e). The distances between matching and non-matching pairs is well separated in case of examples from training set, but is mixed up for validation set. Even though the validation loss for ImageNet initialization is high, we do not care about it, at least for now.

The network with Kaiming initialization trains very slowly, the validation loss remains rather constant which implies that it takes more time for it to learn the features to distinguish matching and non-matching images (see Figure 4.1b). The distance histogram for validation set is better separated than ImageNet, the training set histogram is worse (see Figure 4.1d). It shows that it takes longer to overfit for a model initialized with Kaiming method.

Since fast training is one of our priorities, we choose to make further experiments with



(a) Training Profile (b) Distance Histogram (Train) (c) Distance Histogram (Valid)

Figure 4.2: DualResNet18 with data augmentation (100 epochs, ImageNet initialization)

networks initialized with ImageNet weights.

4.3 Reducing Overfitting

The following techniques, mentioned in the order in which they should be used, can reduce overfitting:

1. Add more data
2. Use data augmentation
3. Use more regularization like dropout and weight decay
4. Reduce architecture complexity

It must be noted that the last two options might also hurt the training capacity of a deep learning model. Since our goal is to first obtain good results on *Aerial Atlanta*, we also refrain from using images from the other cities. Thus the best option we have is data augmentation.

4.3.1 Reaching a Baseline with Data Augmentation

We try out a variety of data augmentation techniques including random flips, rotations, zoom crops, lighting and contrast changes and symmetric warping. Data is augmented batchwise on the fly, by making an affine grid and multiplying by the rotation matrix on GPU in order to make it faster. The complete list of transforms and the probabilities with which they are applied can be found in Table 4.1. In this case, we found out that the network does not overfit.

Transform	UAV Images	Satellite Images
Random Horizontal Flipping	True ($p = 0.5$)	True ($p = 0.5$)
Random Vertical Flipping	False	True ($p = 0.5$)
Random Rotations (Reflection Padding)	Uniform $(-10°, 10°)$	Uniform $(-10°, 10°)$
Random Zooming	10% ($p = 0.75$)	10% ($p = 0.75$)
Random Lighting and Contrast Changes	$p = 0.25$	$p = 0.25$
Symmetric Warping	$p = 0.75$	$p = 0.75$

Table 4.1: List of transforms used to reduce overfitting

Network	Initialization	Data Aug	Total Epochs	Final Train Loss	Final Valid Loss	Best Valid Loss (Epoch)
DualResNet18	Kaiming	No	100	685	1886	1619 (ep 61)
DualResNet18	ImageNet	No	100	65	7509	2623 (ep 16)
DualResNet18	ImageNet	**Yes**	100	1628	1624	**1591 (ep 95)**

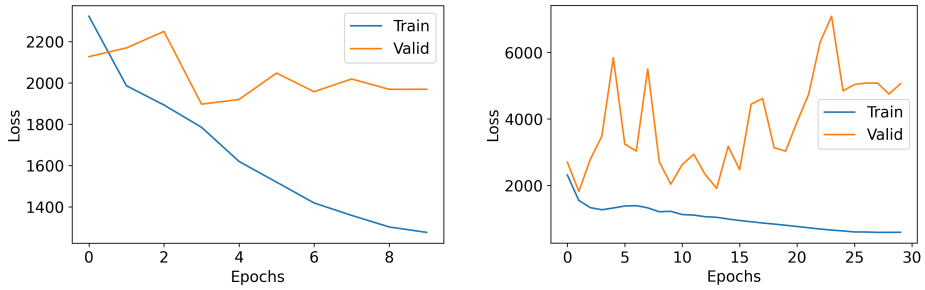
Table 4.2: Effect of data augmentation

The training loss profile along with distance histograms can be seen in Figure 4.2. We see that the validation error keeps on decreasing till the end, also it is much closer to the training error than it was before. From Table 4.2, it is clear that on using data augmentation, we get the best results. However, there are two main issues with this approach:

1. Training is way slower than it was before
2. The validation loss is very unstable and spiky

4.4 Improving Baseline Performance by Making Tweaks

We know that the first few layers in a convolutional neural network identifies low level features like edges, corners, blobs and other simple patterns and it is not a good idea to disturb them because even a slight change in the earlier layers can have a drastic impact on the performance of the model. We suspected that it was due to this reason that the previous model had very bumpy validation loss profile. To solve this issue, we use discriminative layer training with gradual unfreezing.



(a) Layer Group 3 trained (b) Layer Groups 2 and 3 trained

Figure 4.3: Effect of using discriminative learning rates

First of all, we add some fully connected layers on top of the DualResNet18. We call the new model with this customized head, as DualResNet18+. We group the layers in DualResNet18+ model and use gradual unfreezing with discriminative layer training. Then, we also experiment with the amount of data augmentation that should be done to find a sweet spot in between overfitting and underfitting.

4.4.1 Discriminative Layer Training

We show that this can make the validation loss smoother when compared to training all the layers at once. The training time is reduced as well.

Effect of Discriminative Layer Training: We found that discriminative layer training can drastically reduce the spikes in the validation loss. This supports the idea that the initial layers of a pre-trained network should not be trained with learning rates as high as the deeper layers. However, unfreezing of the layer groups should be done wisely:

1. On training only on the head of the network, the validation loss does not decrease and remains almost constant around $2k$. We suspect that this is because there are very few trainable parameters in this case. Moreover, all the convolutional layers are frozen, which might not allow the network to learn visual features specific to this dataset. The training profile is shown in Figure 4.3a.

2. On training only on the head and the second layer group, the validation loss decreases to $1.9k$, and then starts to increase, showing the effects of overfitting. This shows that

Network	Discr LR	Gradual Unfreezing	Data Aug	Total Epochs	Train Loss	Valid Loss	Best Valid Loss (Epoch)
DualResNet18+	Yes	Yes (10+20)	Yes	30	998	1617	1575 (ep 19)
SiamResNet18+	Yes	Yes (10+10)	Yes	20	641	**1486**	**1413 (ep 13)**

Table 4.3: Best results on *Aerial Atlanta*

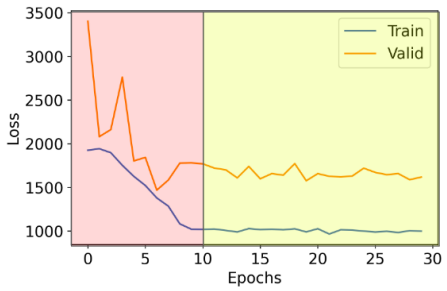
our previous suspicion was correct. By allowing the convolutional layers to be trained, the network is now able to learn visual features. The validation loss does not decrease beyond epoch 14, which means we should stop early. The training profile is shown in Figure 4.3b.

3. **Final strategy:** We train on middle layer group and the head for 10 epochs with discriminative learning rates. This corresponds to the red region in the figure (see Figure 4.4a). Then we unfreeze the whole model and train for 20 more epochs. We again use discriminative learning rates. This corresponds to the yellow region in the figure. The learning rate of head and middle layer group is kept higher than the initial layer group. This gives us the best results.

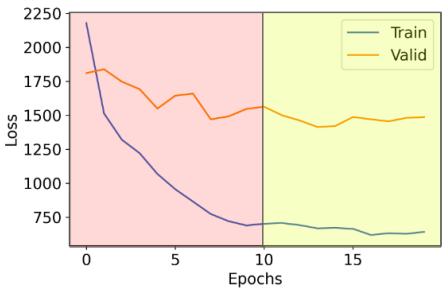
Data Augmentation: In all these three experiments, we add 90° rotations for the satellite images on top of flips. We also retain extra rotations in range $(-10°, 10°)$ with reflection padding. However, we remove symmetric warping for satellite images, as the satellite images are only taken from nadir facing camera.

4.4.2 Using SiamResNet18+ Architecture

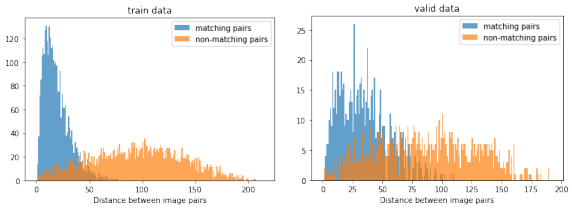
In our experiments, we found out that using a siamese network works slightly better when we train our model on *Aerial Atlanta*. It can be seen in Figure 4.4d that the validation histograms are much better separated when using a siamese network. This method gives us a final accuracy of 78.39%.



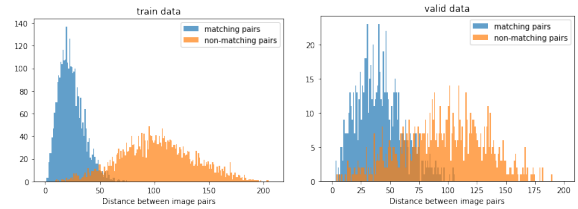
(a) DualResNet18+ training profile



(b) SiamResNet18+ training profile



(c) Distance Histograms (DualResNet18+)



(d) Distance Histograms (SiamResNet18+)

Figure 4.4: Dual network vs siamese network performance on *Aerial Atlanta*

4.5 Training on the *Aerial Cities* Dataset

We train the entire dataset using both the strategies which perform the best on *Aerial Atlanta*. These results are discussed in the next chapter.

Chapter 5

Results on Full Dataset

So far we have experimented with *Aerial Atlanta* to find the best training strategy. Using the insights from these experiments we train the entire dataset. We make two experiments, one with DualResNet18+, and the other with SiamResNet18+.

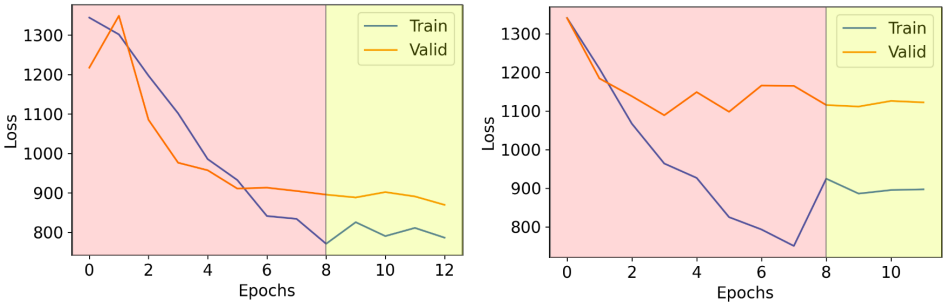
5.1 Performance on *Aerial Cities*

We train both - DualResNet18+ and SiamResNet18+ with discriminative learning rates.

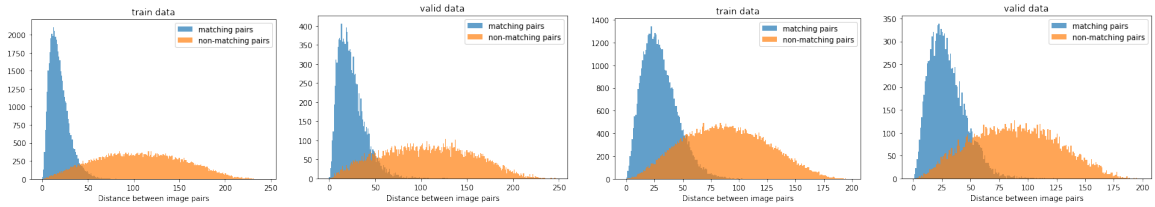
Transform	**UAV Image**	**Satellite Image**
Random Horizontal Flipping	$p = 0.5$	$p = 0.5$
Random Vertical Flipping	False	$p = 0.5$
Random Multiple 90° Rotations	False	Randomly chosen from $\{0°, 90°, 180°, 360°\}$
Random Rotations (Reflection Padding)	Uniform $(-10°, 10°)$	Uniform $(-10°, 10°)$
Random Zooming	10% ($p = 0.75$)	10% ($p = 0.75$)
Random Lighting and Contrast Changes	$p = 0.25$	$p = 0.25$
Symmetric Warping	$p = 0.75$	False

Table 5.1: Final list of transforms used on *Aerial Cities*

First, the layer groups 2 and 3 are trained. Then the entire model is unfreezed and all the layer groups are trained. We use data augmentation shown in Table 5.1. Hence, the training strategy is exactly the same as the best experiments on *Aerial Atlanta*.



(a) DualResNet18+ training profile (b) SiamResNet18+ training profile



(c) Distance Histograms (DualResNet18+) (d) Distance Histograms (SiamResNet18+)

Figure 5.1: Dual network vs siamese network performance on *Aerial Cities*

Network	**Cities**	**Discr LR**	**Gradual Unfreezing**	**Data Aug**	**Train Loss**	**Valid Loss**	**Acc**
DualResNet18+	All	Yes	Yes (8+5)	Yes	786.20	**869.34**	**0.8963**
SiamResNet18+	All	Yes	Yes (8+4)	Yes	897.26	1122.32	0.8569

Table 5.2: Best results on *Aerial Cities*

5.1.1 DualResNet18+

We get best results with this architecture. For training, first we train the layer groups 2 and 3 with learning rates 0.002/3 and 0.002 respectively for 8 epochs. Then we unfreeze the entire model, and train layer groups 1, 2 and 3 with learning rates $(7/3) \times 10^{-5}$, $(7/3) \times 10^{-5}$ and 7×10^{-5} respectively for 5 more epochs. The choice of learning rate is decided by running the learning rate finder as described in Section 2.7. In the first stage, the validation loss continuously decreases to 904.49. In the second stage, the validation loss further decreases to 869.34. The final accuracy achieved by this model is 89.63%, which means that given a pair of images, the model can predict whether they are from same scene or not 9 out of 10 times.

5.1.2 SiamResNet18+

The results in this case are slightly worse than the previous case. We follow a similar training strategy as we did before. First, we train the layer groups 2 and 3 with a learning rate 0.007/3 and 0.007 for 8 epochs. Then we unfreeze the whole model and train the layer groups 1, 2 and 3 with learning rates 10^{-5}, 10^{-5} and 3×10^{-5} respectively for 4 more epochs. Learning rates are chosen by running the learning rate finder. We find that in the first stage, the validation loss decreases upto epoch 4, and then increases slightly. There is insignificant effect of stage 2 on the validation loss. The final validation loss is 1122.32. The final accuracy achieved in this case is 85.69%.

5.2 Analysis of Wrong Predictions

In this section, we analyse the wrongly predicted examples in the validation dataset. In the figure, we have shown those examples which have the highest losses from our best model. Here we have shown only one example each for matching and non-matching image pairs (see Figure 5.2 and Figure 5.3), however, the analysis holds true for all the wrong predictions. The UAV image in Figure 5.2 seems to be taken right above the field with no distinctive features, and the satellite image shows its zoomed out scene. The network encodes both these images with very different vectors, and hence, they are predicted as non-matching images. This is a case of false negative. Figure 5.3 shows a case of false positive. Both the images show houses near areas populated with trees. This image pair looks like a matching one even to human eye, however, both the images are from different places in the same city. The cities in the United States have repeated housing structure, which confuses the model, making it to predict that these images have been taken over the same scene.

In a general case, the occurrences of false negative greatly depend on the features of the UAV images. If most of the portion in an image taken from UAV camera shows a single structure, *e.g.,* a field or a terrace, the model is unable to extract meaningful features, and hence the matching fails. On the other hand, the occurrence of a false negative is due to similarity in the images taken from different places, which may either be due to being over a tree dense area, or because of repeated housing structures.

(a) UAV Image (b) Satellite Image

Figure 5.2: Matching pair wrongly predicted as non-matching

(a) UAV Image (b) Satellite Image

Figure 5.3: Non-matching pair wrongly predicted as matching

5.3 Insights

Looking at the results, we draw some conclusions, which are listed below:

1. Even though network with siamese architecture performs better on a single city, it fails to beat dual network architecture on the entire dataset. We believe this is because the siamese network has exactly the same parameters for both - satellite and UAV images. Hence, it is unable to learn features specific to both distributions. The reason it performs better on *Aerial Atlanta* can be explained by the fact that the visual difference of UAV and satellite images is limited as compared to images from other cities.

2. The model can correctly predict whether two images are from the same scene or not

9 times out of 10. The examples on which the model fails are very confusing even to human eye. The model is confused when the major portion of a UAV image is covered with a single structure, or when there is a lot of greenery or repeated city structures in the images.

3. One must be careful while scaling from a small subset of dataset to the entire dataset. A model which performs the best on the smaller dataset, may not always perform the best on the entire dataset. However, one can get around this issue by choosing to experiment with all the good performing models on the entire dataset.

Chapter 6

Conclusions and Future Work

In this work, we train a convolutional neural network to predict whether two images taken from UAV and satellite are from the same scene or not. We achieve an accuracy of 89.63% on this task using a dual network architecture of ResNet18+ on *Aerial Cities* dataset. We also achieve an accuracy of 78.39% on *Aerial Atlanta* (containing only 6.81% of images from *Aerial Cities*) using siamese architecture of ResNet18+.

6.1 Key Contributions

This work has following main contributions:

- We show a smart way to use ImageNet pretrained models on a computer vision task with a dataset that contains images considerably different from the ImageNet images. This is made possible by using the concept of gradual unfreezing and discriminative layer training, which allows the fundamental features of the initial convolutional layers to remain in their places.
- We show a systematic way to handle large datasets, where the insights from a smaller subset of the training data can be used as guidelines for training the entire dataset.
- We show that the performance of dual networks is superior to that of siamese networks when the comparison is made on images which come from two distinct distributions as in case of UAV and satellite images.

6.2 Open Source Contributions

The pre-trained models on *Aerial Cities* along with all the experiments mentioned in this report are available on https://github.com/abhinavtripathi95/geolocalization/.

6.3 Future Work

We have only considered the problem of image matching in this work. However, it can be further extended in one or more of the following ways:

- Using satellite images, the accurate global pose of the UAV could be inferred by training a camera localization network.
- Improving robustness of aerial imagery matching against seasonal variations could also be explored.
- The dataset uses images of cities from the United States, where the satellite images are available in relatively higher resolution as compared to India. The effect of using satellite images from Indian cities can be explored.
- The choice of when two aerial images should be considered as matching can be explored. The dataset used in current setting uses a threshold of $40m$, which is, in our opinion, arbitrary.
- The robustness of the proposed method could be evaluated on the problem of geolocalization of a UAV when its pose is known with certain error, say $100m$.

References

[1] D. G. Lowe, "Distinctive image features from scale-invariant keypoints," *International Journal of Computer Vision*, vol. 60, no. 2, pp. 91–110, 2004.

[2] H. Bay, T. Tuytelaars, and L. Van Gool, "Surf: Speeded up robust features," in *Computer Vision – ECCV 2006* (A. Leonardis, H. Bischof, and A. Pinz, eds.), (Berlin, Heidelberg), pp. 404–417, Springer Berlin Heidelberg, 2006.

[3] E. Rublee, V. Rabaud, K. Konolige, and G. Bradski, "Orb: An efficient alternative to sift or surf," in *2011 International Conference on Computer Vision*, pp. 2564–2571, 2011.

[4] K. M. Yi, E. Trulls, V. Lepetit, and P. Fua, "Lift: Learned invariant feature transform," in *Computer Vision – ECCV 2016* (B. Leibe, J. Matas, N. Sebe, and M. Welling, eds.), (Cham), pp. 467–483, Springer International Publishing, 2016.

[5] D. DeTone, T. Malisiewicz, and A. Rabinovich, "Superpoint: Self-supervised interest point detection and description," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 224–236, 2018.

[6] M. Dusmanu, I. Rocco, T. Pajdla, M. Pollefeys, J. Sivic, A. Torii, and T. Sattler, "D2-net: A trainable cnn for joint detection and description of local features," *arXiv preprint arXiv:1905.03561*, 2019.

[7] L. Fei-Fei and P. Perona, "A bayesian hierarchical model for learning natural scene categories," in *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*, vol. 2, pp. 524–531 vol. 2, 2005.

[8] H. Altwaijry, E. Trulls, J. Hays, P. Fua, and S. Belongie, "Learning to match aerial images with deep attentive architectures," in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 3539–3547, 2016.

[9] T. Koch, X. Zhuo, P. Reinartz, and F. Fraundorfer, "A new paradigm for matching uav and aerial images," *ISPRS Annals of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, vol. 3, pp. 83–90, 2016.

[10] S. Chen, X. Li, Y. Zhang, R. Feng, and C. Zhang, "Local deep hashing matching of aerial images based on relative distance and absolute distance constraints," *Remote Sensing*, vol. 9, no. 12, p. 1244, 2017.

[11] D. Costea and M. Leordeanu, "Aerial image geolocalization from recognition and matching of roads and intersections," *arXiv preprint arXiv:1605.08323*, 2016.

[12] K. Regmi and M. Shah, "Bridging the domain gap for ground-to-aerial image matching," in *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*, pp. 470–479, 2019.

[13] B. Zhou, A. Lapedriza, A. Khosla, A. Oliva, and A. Torralba, "Places: A 10 million image database for scene recognition," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 40, no. 6, pp. 1452–1464, 2018.

[14] A. Shetty and G. X. Gao, "Uav pose estimation using cross-view geolocalization with satellite imagery," in *2019 International Conference on Robotics and Automation (ICRA)*, pp. 1827–1833, 2019.

[15] Y. Taigman, M. Yang, M. Ranzato, and L. Wolf, "Deepface: Closing the gap to human-level performance in face verification," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 1701–1708, 2014.

[16] F. Schroff, D. Kalenichenko, and J. Philbin, "Facenet: A unified embedding for face recognition and clustering," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 815–823, 2015.

[17] D. Mishkin and J. Matas, "All you need is a good init," *arXiv preprint arXiv:1511.06422*, 2015.

[18] X. Glorot and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks," in *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, pp. 249–256, 2010.

[19] H. Zhang, Y. N. Dauphin, and T. Ma, "Fixup initialization: Residual learning without normalization," *arXiv preprint arXiv:1901.09321*, 2019.

[20] K. He, X. Zhang, S. Ren, and J. Sun, "Delving deep into rectifiers: Surpassing human-level performance on imagenet classification," in *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, pp. 1026–1034, 2015.

[21] J. Howard and S. Ruder, "Universal language model fine-tuning for text classification," *arXiv preprint arXiv:1801.06146*, 2018.

[22] M. D. Zeiler and R. Fergus, "Visualizing and understanding convolutional networks," in *Computer Vision – ECCV 2014* (D. Fleet, T. Pajdla, B. Schiele, and T. Tuytelaars, eds.), (Cham), pp. 818–833, Springer International Publishing, 2014.

[23] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 770–778, 2016.

[24] L. N. Smith, "A disciplined approach to neural network hyper-parameters: Part 1–learning rate, batch size, momentum, and weight decay," *arXiv preprint arXiv:1803.09820*, 2018.

[25] P. Micikevicius, S. Narang, J. Alben, G. Diamos, E. Elsen, D. Garcia, B. Ginsburg, M. Houston, O. Kuchaiev, G. Venkatesh, *et al.*, "Mixed precision training," *arXiv preprint arXiv:1710.03740*, 2017.

[26] E. Maggiori, Y. Tarabalka, G. Charpiat, and P. Alliez, "Can semantic labeling methods generalize to any city? the inria aerial image labeling benchmark," in *2017 IEEE International Geoscience and Remote Sensing Symposium (IGARSS)*, pp. 3226–3229, 2017.

[27] G. Xia, X. Bai, J. Ding, Z. Zhu, S. Belongie, J. Luo, M. Datcu, M. Pelillo, and L. Zhang, "Dota: A large-scale dataset for object detection in aerial images," in *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 3974–3983, 2018.

[28] A. Robicquet, A. Sadeghian, A. Alahi, and S. Savarese, "Learning social etiquette: Human trajectory understanding in crowded scenes," in *Computer Vision – ECCV 2016* (B. Leibe, J. Matas, N. Sebe, and M. Welling, eds.), (Cham), pp. 549–565, Springer International Publishing, 2016.