Intelligent Learning Agents for Board Games, Few-shot Translation of Satellite Images

Dual Degree Project Report

Submitted in partial fulfillment of the requirements of the degree of

Master of Technology

by

Mohd Safwan
Roll no: 17D070047

under the guidance of
Prof. Biplab Banerjee

and
Department of Electrical Engineering
INDIAN INSTITUTE OF TECHNOLOGY BOMBAY

2021-22

Declaration of Authorship

I, Mohd Safwan, declare that this written submission represents my ideas in my own words and where others ideas or words have been included, I have adequately cited and referenced the original sources. I also declare that I have adhered to all principles of academic honesty and integrity and have not misrepresented or fabricated or falsified any idea/data/fact/source in my submission. I understand that any violation of the above will be cause for disciplinary action by the Institute and can also evoke penal action from the sources which have thus not been properly cited or from whom proper permission has not been taken when needed.

[illegible]

Signature

17D070047

Roll Number

Date: June 16, 2022

Acknowledgments

I would like to express my sincere gratitude towards my guide for the project **Prof. Biplab Banerjee**, for his guidance and constant support throughout the work on my thesis. His regular inputs in the work and allowing me just the right amount of independence and guidance went a long way in making my first major pursuit of a research activity truly enriching and worthwhile.

I would like to thank my professors of IIT Bombay, with a special mention to those who taught me the concepts and nuances of Reinforcement Learning, Deep Learning and Computer Vision – igniting and enriching my interest in the subject.

Mohd Safwan
IIT Bombay
June 16, 2022

Contents

1 Introduction : Intelligent Learning Agents for Board Games 1

1.1 Abstract 1

1.2 Background 2

1.2.1 Indie Board Games 2

1.2.2 Learning Agent 3

1.2.3 Markov Decision Process and Value Function 3

1.2.4 Temporal Difference Learning 5

1.2.5 MiniMax Algorithm 5

1.3 Literature Review 6

1.3.1 State Space 6

1.3.2 Expert Player 6

1.3.3 ExpectiMinimax : Stochastic Tree Search 7

1.3.4 TD(λ) Algorithm 7

1.4 Methodology 8

1.4.1 State Features and Neural Network Evaluator 8

1.4.2 Learning and Inference Algorithms 9

1.4.3 Difficulty Setting and Sub-Optimal Play 10

1.5 Results 10

2 Introduction : Few-shot Translation of Satellite Images 13

2.1 Abstract 13

2.2 Background 13

2.2.1 Deep Learning and CNNs 13

2.2.2 Generative Adversarial Networks 15

2.3 Literature Review 17

2.3.1 Image to Image Translation 17

2.3.2 Non-Supervised Learning 19

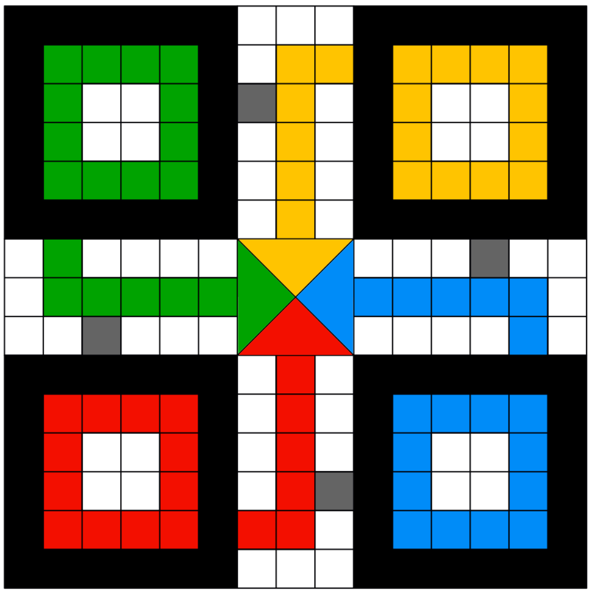
2.4 Methodology 21
2.4.1 Datasets 21
2.4.2 Polar Transformation 22
2.4.3 Model Architecture 23
2.5 Work for Phase 2 24

Chapter 1

Introduction : Intelligent Learning Agents for Board Games

1.1 Abstract

There has been a rise in popularity of board games like Ludo and Snakes and Ladders in past few years as these are now available as multiplayer mobile games. In phase 2 of my dual degree project, I have studied these board games in a reinforcement learning setting and I have developed a neural network based learning algorithm. The algorithm can be used to create game playing agents that play extremely well. The algorithm has low latency and can be used in realtime against human players. I also propose methods to tune the difficulty of the algorithm to match opponent's skill level and provide rich gameplay experience.


(a) Ludo

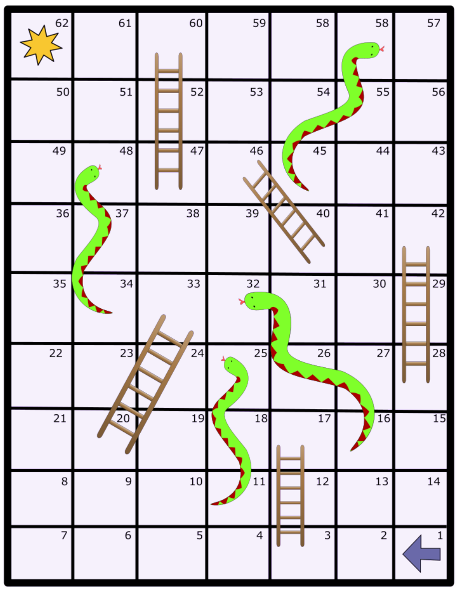

(b) Snakes and Ladders

Figure 1.1: Board Layouts

1.2 Background

1.2.1 Indie Board Games

I have chosen two board games for demonstration of the learning algorithm, Ludo, and Snakes and Ladders. These are turn based games with 4 and 2 players respectively where each player rolls a dice and then chooses from a few valid actions. The rules for these games have been slightly modified from traditional versions inspired by their mobile game version. These changes are as follows.

- Each player has **3 retries** where the player can re-throw the dice if he does not like the current dice roll.
- There is a game timer which is typically around 120-300 seconds, each move takes some time and that time is deducted from timer, except when a piece is captured/6 is rolled/retry is used.
- At the end of game timer, if there is no winner (all pieces at end square), then the player with highest score wins. Score is the sum of distance form staring square of all pieces of the player.

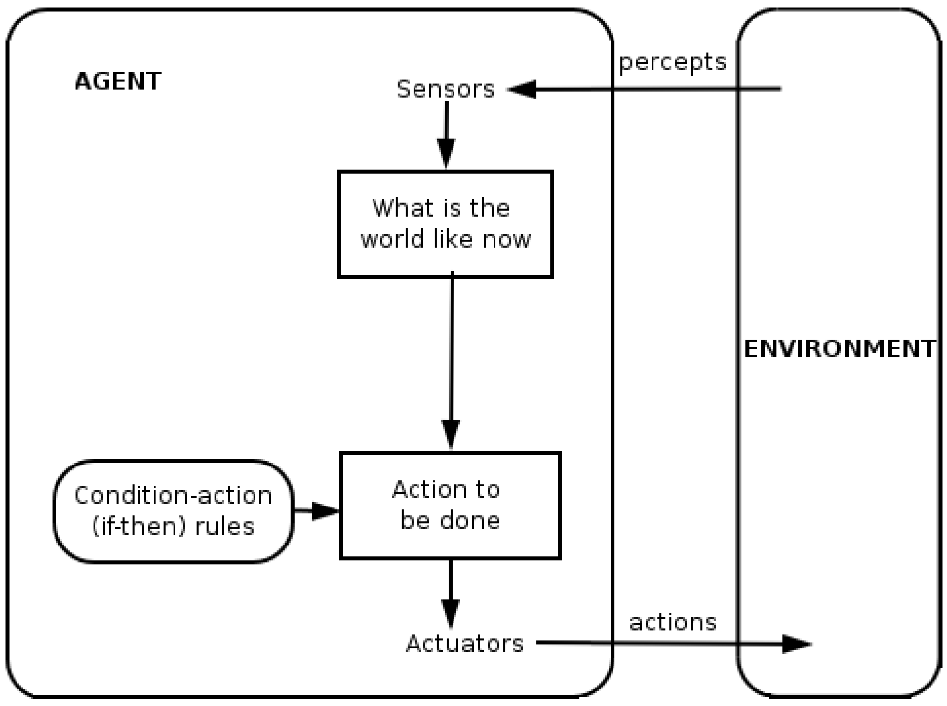


Figure 1.2: A Learning Agent

- In Ludo, there is no 6 roll required to release a piece, all pieces are 'unlocked' by default.

1.2.2 Learning Agent

An agent is an entity that can perceive environment, take actions based on some policy that is a part of the agent, and receives feedback (reward) from environment. The agent may use knowledge of previous actions and rewards to adjust its policy. The policy of an agent can be stochastic as well as non-stationary (evolves with time).

1.2.3 Markov Decision Process and Value Function

A Markov Decision Process or an **MDP** is an abstraction of an environment with fully observable states that an agent can interact with. The environment changes its states (possibly stochastically) when some action is taken and gives a feedback known as reward. An MDP can be mathematically represented as a tuple (S, A, T, R, γ) where

- S is the set of all possible states of the environment. A state contains all information about the environment.

- A is the set of all possible actions an agent can take.

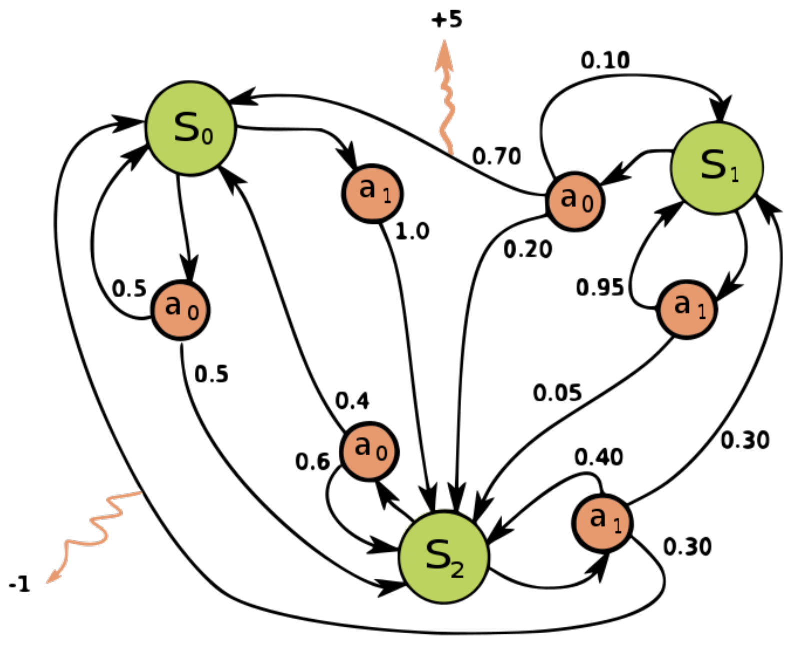


Figure 1.3: Example of a Markov Decision Process

- $T(s, a, s') \in [0, 1]$ is the probability of transition from state s to state s' after taking action a.

- $R(s, a, s') \in \mathbb{R}$ is a bounded real number called the reward given to the agent from the environment for the transition $s \to s'$ on taking action a.

- $\gamma \in [0, 1]$ also called discount factor is the factor by which previous reward are discounted in calculating the value of any state. I will be assuming $\gamma = 1$ for all the experiments.

An agent's policy $\pi : S \to A$ is a mapping from states to actions. Π is the set of all policies.

The value function $V^\pi : S \to \mathbb{R}$ is defined such that $V(s)$ is the cumulative expected reward when an agent starts at state s and follows policy π. By definitions, we get

$$V^\pi(s) = \sum_{s' \in S} T(s, \pi(s), s')[R(s, \pi(s), s') + \gamma V^\pi(s')]$$

Theorem 1 *For any MDP* (S, A, T, R, γ) *there exists a policy* $\pi^* \in \Pi$ *such that* $V^{\pi^*}(s) \geq$

$V^{\pi}(s) \quad \forall s \in S, \forall \pi \in \Pi$. *The value function* V^{π^*} *is known as the* ***Optimal Value Function*** V^*.

Note that V^* is obviously unique while π^* may not be unique. From now, when we talk about value of a state s, it will be the optimal value i.e. $V^*(s)$.

1.2.4 Temporal Difference Learning

There are two way to find π^*: searching the policy space or searching the value function space. Policy space search methods maintain explicit representations of policies and modify them through a variety of search operators, while value function search estimates V^*, through experience. There are two methods to this approach: Monte-Carlo and Temporal-Difference (TD) learning.

The central idea of temporal difference learning is bootstrapping. We can get a better estimate of the value of current state using estimate of next state. For a transition (s, a, s', r), the update would be

$$V(s) \leftarrow V(s) + \alpha[r + \gamma V(s') - V(s)]$$

where α is the learning rate that can be annealed slowly.

So if we are given a batch of transitions from an MDP - Agent interaction, and given that the agent is sufficiently stochastic (there is rigorous definition for this but out of scope of this work) we can actually learn the value function (in the limit) by continuous updates.

1.2.5 MiniMax Algorithm

Minimax algorithm is used to play 2 player games optimally. The players try to find the action which has the best 'worst outcome', i.e. the move which has best outcome for them assuming opponent also plays optimally. It's named minimax because it minimizes the maximum loss scenario.

Each end state of the game has some value, higher if player 1 wins and lower if player 2 wins, generally this is taken to be in $1, 0, -1$ corresponding to win, draw, lose or $1, 0$ corresponding to win, lose. Then each non-leaf state of the game is the value according to the minimax principle.

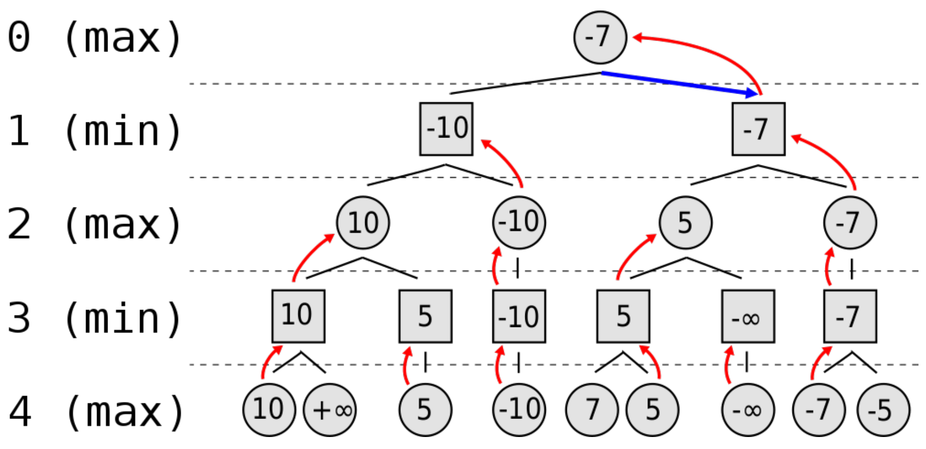


Figure 1.4: Minimax Algorithm

1.3 Literature Review

1.3.1 State Space

The authors of [2] have carefully analyzed the game of Ludo and have estimated the number of states to be 10^{20} which is extremely infeasible to solve for. Similarly using their methods, the number of states in Snakes and Ladders is 10^8. More clearly, in any board game with n tiles and m pieces, we can expect the state space complexity to be roughly $\mathcal{O}(n^m)$.

1.3.2 Expert Player

In the paper [2], the authors propose a rule-based *Expert* player for evaluating learning agents against. They provide analysis to claim that in the game of Ludo there is a hierarchy of playstyles based on most advantageous

$$\text{DEFENSIVE} \geq \text{AGGRESSIVE} \geq \text{RANDOM}$$

I have enhance the said expert player with some more rules as well as given it ability to perform RETRY moves with some probability based on some conditions. These conditions are dependent on factors like

- Is any piece attacking an opponent?
- Is any piece is attacking distance of any opponent's piece?
- Is any piece able to move to a safe position from an unsafe position?

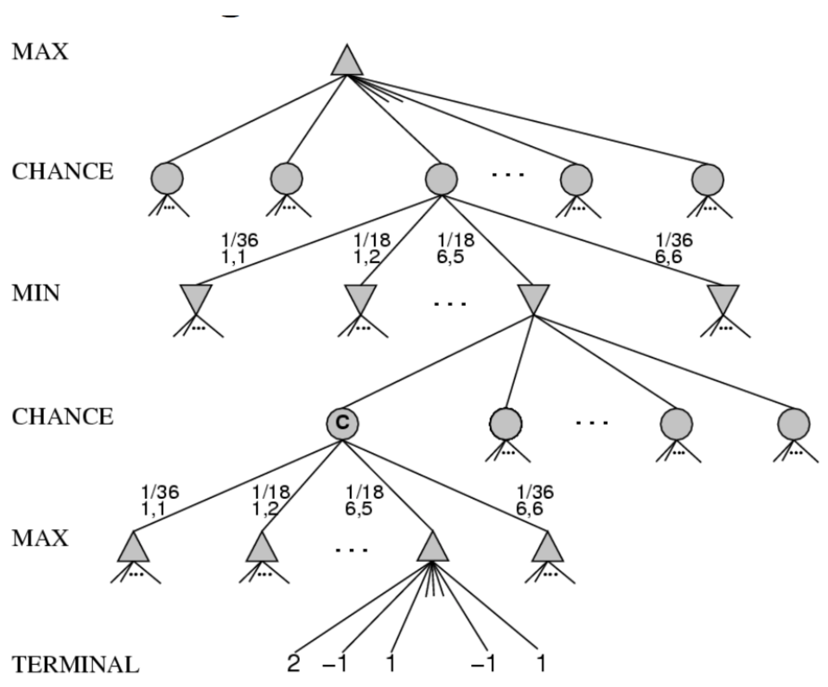


Figure 1.5: ExpectiMinimax Algorithm

- Is any piece able to go to the end square?

1.3.3 ExpectiMinimax : Stochastic Tree Search

To address the randomness introduced by the dice, we can reason in terms of expectation. So instead for minimizing the maximum loss, we can minimize the expected maximum loss. To implement this, while rolling out the game tree, we can assume the dice roll and then roll out for all dice roll values and then average the outcomes. This is the ExpectiMinimax algorithm as described in [3].

1.3.4 TD(λ) Algorithm

This algorithm described in [1] interpolates between the traditional bootstrapping based learning and the Monte Carlo estimate, where the value of state is averaged out over many episodes. The hyperparameter λ allows us to control the amount of bootstrapping we want. TD(λ) is usually implemented using eligibility traces also known as the backward view.

Algorithm 1 TD(λ) with explicit states

```
**Initialise** $V : S \to \mathbb{R}$ arbitrarily
for each episode do
    Set $z \to \mathbf{0}$ //Eligibilty Trace Vector
    Assume Agent born in state $s$
    for each step in episode do
        Take action $a$, get reward $r$, go to state $s'$
        $\delta \leftarrow r + \gamma V(s') - V(s)$
        $z(s) \leftarrow z(s) + 1$
        for all $s \in S$ do
            $V(s) \leftarrow V(s) + \alpha\delta z(s)$
            $z(s) \leftarrow \gamma\delta z(s)$
        end for
        $s \leftarrow s'$
    end for
end for
```

1.4 Methodology

To play the game optimally, one would think we can just do a expectiminimax tree search and find optimal moves. But, a game tree search is infeasible due to the shear number of states. Hence we need an **Evalutator** i.e. a function $EV : S \mapsto [0,1]^{\text{num\_players}}$ that can evaluate a game state. In 4 player Ludo, the output of this function can be interpreted as probability of winning of all 4 players $[p_1, p_2, p_3, p_4]$. This also introduces a constraint that $\sum p_i = 1$ which we can enforce while designing the function.

1.4.1 State Features and Neural Network Evaluator

For solving an MDP problem, there are many methods such as Value Iteration, Policy Iteration, and Linear Programming. But, we can't apply these methods to MDPs when state space ($|S|$) is large, as the computation becomes infeasible.

This is the case in our board games, hence we will be using a compact representation of the game state using neural networks. For this, we need to represent the game state as a feature vector, which can then be input to the neural network.

We can represent the game state of a 4 player Ludo game as a 257-dimensional vector. For both players, we use 57 elements for the 57 board squares, 1 element for start square, 1 element for the end square, 4 elements for one hot encoded representation of the number of retries left, and 1 element for the score. Then all 4 vectors are concatenated.

The vector of each player is shifted such that the current player's vector is placed first, then next, etc. This ensures that the representation is invariant to the board position (as it should be). These quantities are normalized appropriately. Then we also pass the normalized timer value as the last element.

1.4.2 Learning and Inference Algorithms

Let out neural network for estimating Value Function be $\hat{V}_\theta : S \mapsto [0,1]^{\text{num\_players}}$ which is parameterized by weights θ. For any state s, $\hat{V}_\theta(s)$ gives a vector of value for each player. But if we give reward as 1 for winning and 0 otherwise, the value can also be interpreted as the probability of winning. To meet the constraint that the outputs must sum to 1, we use a Softmax activation on the final layer. We also maintain an eligibility trace vector z_i for each player i. The shape of this vector is same as shape of parameter vector θ. Training

Algorithm 2 TD(λ) with state features

```
Initialise weights $\theta$ arbitrarily
for each episode do
    Set all $z_i \rightarrow \mathbf{0}$ //Eligibility Trace Vector
    Assume Agent born in state $s$
    for each step in episode do
        Take action $a$, get reward vector $r$, go to state $s'$
        for each player $i$ do
            if $s'$ is a terminal state then
                $\delta_i \leftarrow r_i + \gamma \hat{V}_\theta(s')_i - \hat{V}_\theta(s)_i$
            else
                $\delta_i \leftarrow r_i - \hat{V}_\theta(s)_i$
            end if
            $z_i \leftarrow \gamma \delta_i z_i + \nabla_\theta \hat{V}_\theta(s)_i$
            $\theta \leftarrow \theta + \alpha \delta_i z_i$
        end for
        $s \leftarrow s'$
    end for
end for
```

for 100,000 episodes gives a very good model in practice and weight converge nicely. Once the model is trained, we can perform inference by expanding the game tree till some depth, evaluating all the leaf nodes, and then performing an ExpectiMiniMax search.

1.4.3 Difficulty Setting and Sub-Optimal Play

For making easier to play against agents, we can utilize the information provided by neural network. During the game tree search the can be programmed to pick sub-optimal moves with some probability that is tune-able and depends on difficulty level.

1.5 Results

	Win Rate against Random	**Win Rate against Expert**
Random	50%	23%
Expert	77%	50%
TD(λ) (depth 1)	75%	67%
TD(λ) (depth 2)	76%	75%
TD(λ) (depth 3)	80%	80%

Table 1.1: 2 Player Ludo Results

	Win Rate against Random	**Win Rate against Expert**
Random	50%	25%
Expert	75%	50%
TD(λ) (depth 1)	90%	75%
TD(λ) (depth 2)	92%	78%
TD(λ) (depth 3)	93%	80%

Table 1.2: Snakes and Ladders Results

Depth \Game Time)	**120 secs**	**360 secs**	**600 secs**
1	72%	69%	64%
2	74%	81%	77%
3	76%	82%	81%

Table 1.3: Ludo 2P against Expert

The agents learns to play very quickly. We asked human volunteers to playing against the agent and they were surprised to see the agent's long term decision making ability. The jump in accuracy with depth suggests that deeper game tree search is very beneficial for the agent.

Bibliography

[1] *Sutton, Richard S. and Barto, Andrew G.* Reinforcement Learning: An Introduction
`https://mitpress.mit.edu/books/reinforcement-learning`

[2] *Faisal Alvi et al.* Complexity Analysis and Playing Strategies for Ludo and its Variant Race Games
`https://ieeexplore.ieee.org/document/6031999`

[3] *D. Richie* GAME-PLAYING AND GAME-LEARNING AUTOMATA
`https://www.sciencedirect.com/science/article/pii/B9780080113562500112`

[4] *Shivaram Kalyanakrishnan* Foundations of Intelligent and Learning Agents, IITB
`https://www.cse.iitb.ac.in/~shivaram/teaching/old/cs747-a2020`

Chapter 2

Introduction : Few-shot Translation of Satellite Images

2.1 Abstract

There has been immense advancements in image to image translations using deep learning techniques. In phase 1 of my dual degree project, I have spent this time broadening my deep learning knowledge. I started from very basic knowledge of neural networks and moved to CNNs, RNNs, learning about various types of GANs, and various methods for using them for translation of image between different but related domains. The motivation is to use this knowledge and apply it for translating aerial photographs to street view images. Such an algorithm can be utilized for localization of any street view, by just referencing it with already available global satellite image data. I target to achieve this in a few-shot fashion by building upon the great techniques we have seen published in last few years.

2.2 Background

2.2.1 Deep Learning and CNNs

Deep Learning is a powerful data driven tool for approximating functions of extremely high dimensionality (images, music, text, depth maps etc.). The function is represented using any combination of various kinds of neural networks, non-linear activation functions, and some kind of specialized aggregation/masking function (batch-norm, dropout, max-pool, concate-

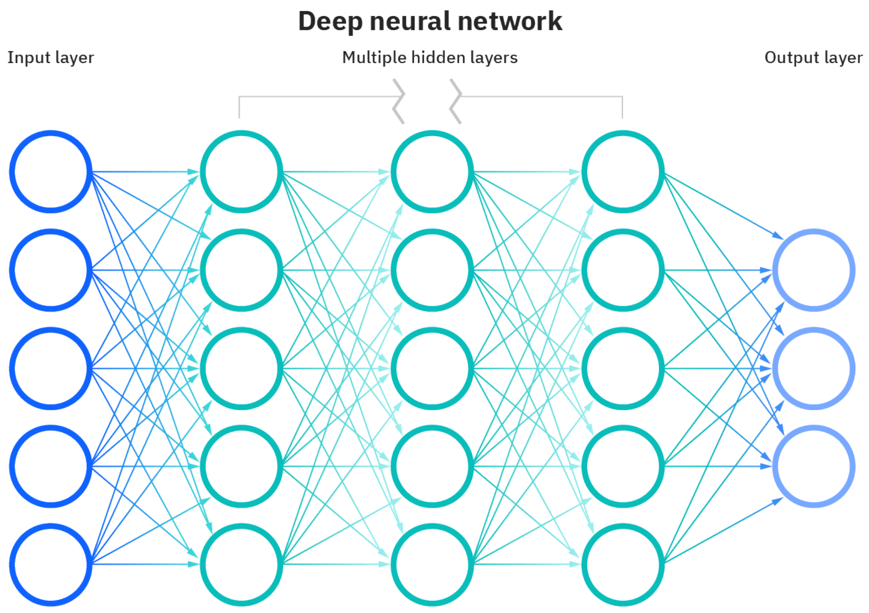


Figure 2.1: A simple Feed Forward Network

nation, etc). These functions are themselves parametrized by upto million and billions of scalar 'weights'. If we have some labelled examples/data $\{x_i, y_i\}$ with inputs x and outputs y, then we can optimize our network f_θ with weights θ such that it minimized a metric called loss function L. So ideally, after training enough, weights converge at

$$\theta^* = \arg\min_\theta L(f_\theta(x), y)$$

The great success of neural networks come majorly from the large number of parameters. In fact, the authors of [10] proved that neural networks can approximate any continuous function to any desired precision.

Convolutional Neural Networks

Using dense networks for images that aren't very low resolution turns out to be extremely infeasible. The number of parameters becomes impractically large very quickly and overfitting becomes unavoidable due the extreme over-parameterization. To work with images, we turn to traditional image processing and look at how convolutions with small filters can be used to extract information from images. These filter coefficients can be learned using optimization methods for any given loss function. Such filters can be stacked into filter banks and

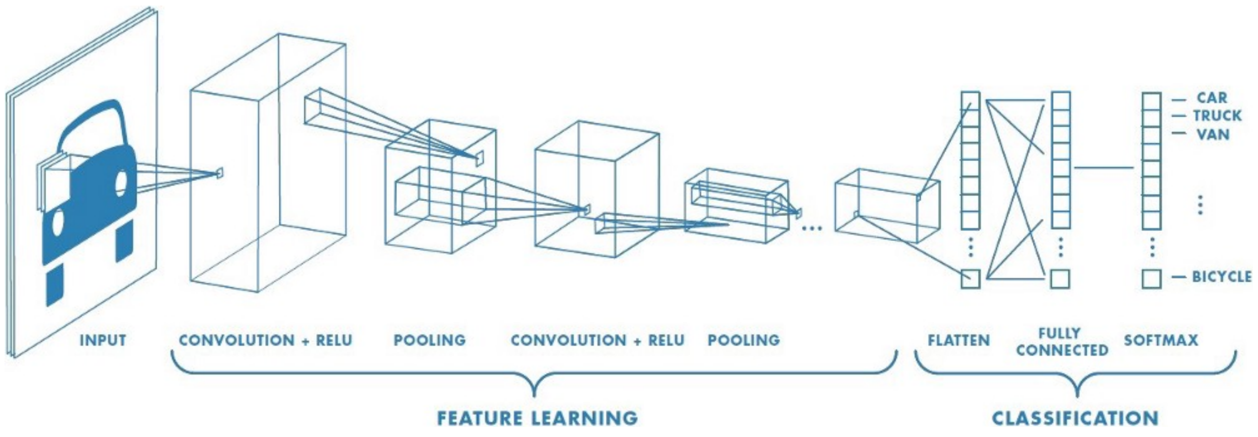


Figure 2.2: A typical CNN architecture

activation functions can be applied between multiple such filter banks. The feature maps obtained can be reduced in size by pooling or strided filtering. The obtained smaller feature map is then flattened and fed int a feed forward network to get required results.

2.2.2 Generative Adversarial Networks

This class of deep learning models were introduced in 2014 by Ian Goodfellow's team [3]. Two networks are made adversaries in a minimax game. These class of models have proven to be very successful in image generation as well as translation.

$$\min_G \max_D \mathbb{E}_{x\sim p_{\text{data}}(x)}[\log(D(x)] + \mathbb{E}_{z\sim p_z(z)}[\log(1 - D(G(z)))]$$

The generator network takes a latent random variable z as input and produces data sample $G_\theta(z)$ sampled from a distribution parametrized by network weights θ. The generator network is trained to model the distribution of real data p_{data}. Here $G_\theta(z) \sim p_g$ which is the generator's distribution. Meanwhile, the discriminator network D is typically a classifier trained to predict if a datapoint is real or fake. $D_\theta(x)$ is the probability that x came from p_{data} rather than p_g.

Despite the promising results, this version of GAN is extremely difficult to train. These models suffer from *mode collapse* where output distribution just collapses in a single degenerate distribution. To address such problems there have been many developments on this GAN. Some of the most used are the following.

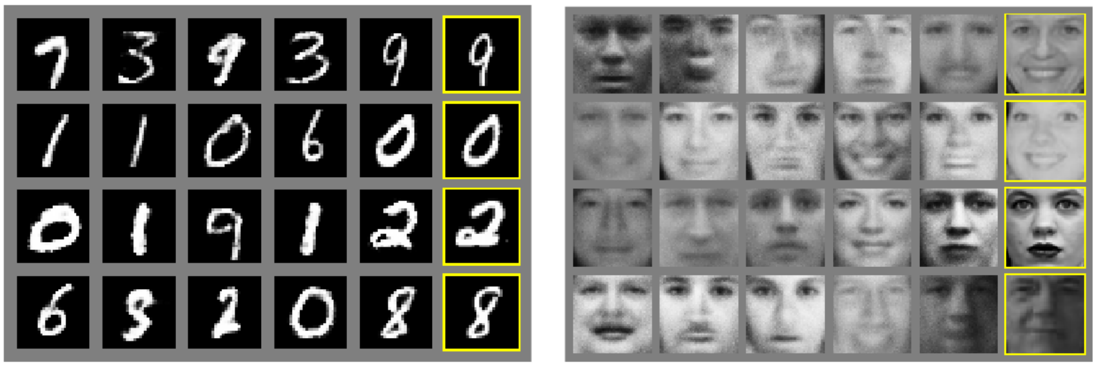


Figure 2.3: Some images generated by MLP based GANs. Rightmost column is the closest sample from the training set, which shows that outputs are new.

DCGAN

Deep Convolutional Generative Adversarial Networks [4] uses CNNs as the generator and the discriminator networks. The paper also introduced a set of best practices for model architecture and training methods for stable training of GANs.

Wasserstein GANs

This paper introduces WGANs [8], an alternative to traditional GAN training. The authors show that we can improve the stability of learning, get rid of problems like mode collapse, and provide meaningful learning curves useful for debugging and hyper-parameter searches. Furthermore, authors show that the original GAN algorithm minimizes the Kullback Leibler Divergence between modelled and target distribution.

Conditional GANs

Instead of sampling a latent random variable z and generating a high dimensional datapoint $G_\theta(z)$ from it, GANs can be extended to a conditional model if both the generator and discriminator are conditioned on some extra information y. y could be any kind of auxiliary information, such as class labels or data from other modalities. We can perform the conditioning by feeding y into the both the discriminator and generator as additional input layer. The new objective for minimax game is

$$\min_G \max_D \mathbb{E}_{x \sim p_{\text{data}}(x)}[\log(D(x|y)] + \mathbb{E}_{z \sim p_z(z)}[\log(1 - D(G(z|y)))]$$

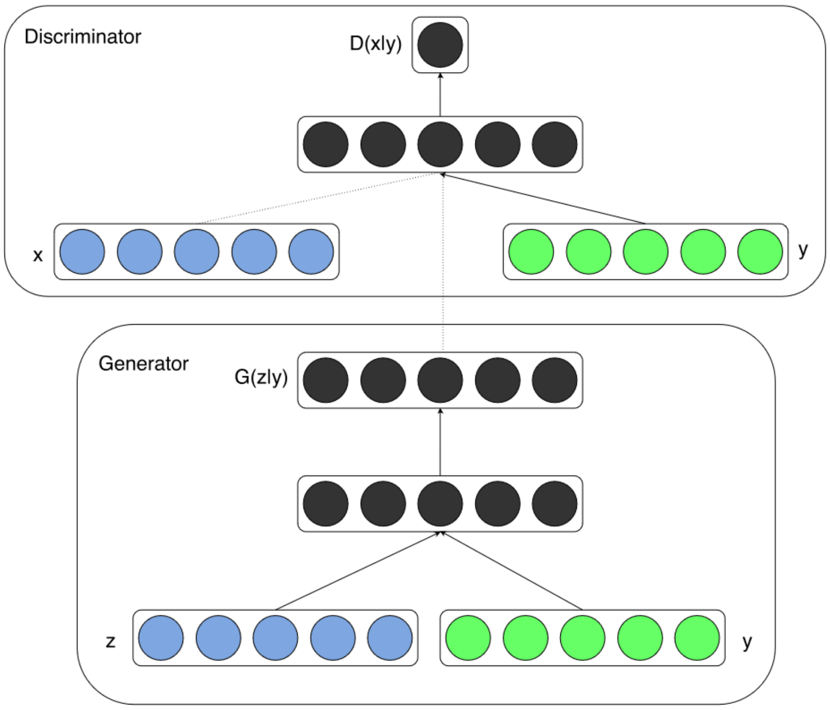


Figure 2.4: CGAN algorithm illustration

2.3 Literature Review

2.3.1 Image to Image Translation

The Conditional Generative Adversarial Network framework can be used for image to image translation when paired data is available. CGANs achieve state of the art results in applications as diverse as sketch to photo conversion, image colorization, image inpainting, and style transfer. The general idea is to use a convolutional neural network to embed an image into a smaller feature map and then progressively upscale it using either multiple interpolation or multiple transpose convolution layers with an activation in between. In a lot of papers that I read, authors have used a U-Net based architecture for this task. Although it was originally designed for Semantic Segmentation[11] of biomedical images, it works very well as an image to image translation model. It benefits greatly from the skip connections between downward and upward blocks which enables residual learning and smooth gradient flow.

"*Instead, for our final models, we provide noise only in the form of dropout, applied on several layers of our generator at both training and test time.*" - [2]

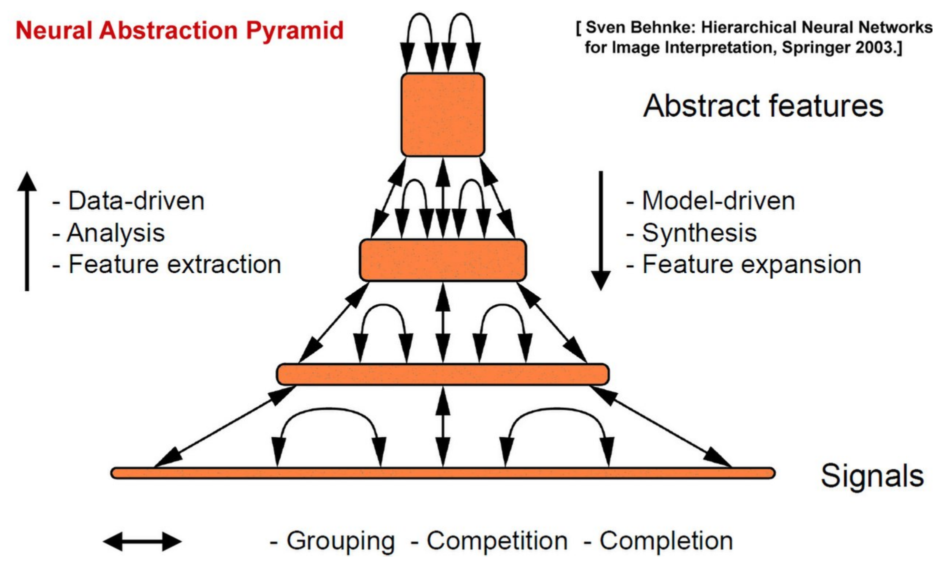

Neural Abstraction Pyramid
[Sven Behnke: Hierarchical Neural Networks for Image Interpretation, Springer 2003.]
Abstract features
- Data-driven
- Analysis
- Feature extraction
- Model-driven
- Synthesis
- Feature expansion
Signals
- Grouping - Competition - Completion

Figure 2.5: The neural abstraction pyramid which explains motivation behind encode-decoder like architectures

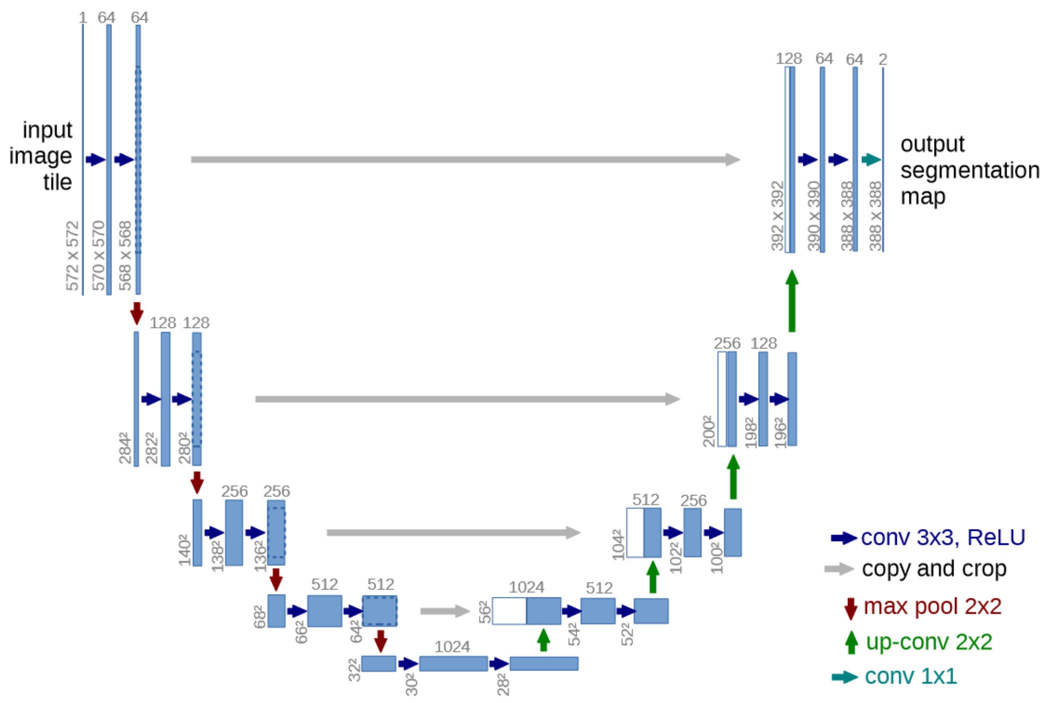


Figure 2.6: Original U-Net architecture with skip connection

However, here the GAN is conditioned on the input image and the randomness does not come from a latent variable that we sample but rather the randomness is provided by the dropout layers used [2].

In [1], the authors propose a novel image retrieval pipeline where a network R_E extracts the local features from a street-view image which is then aggregated into global descriptors using a spatial attention module SA. The feature descriptor is then matched against feature descriptors from the encoder module of the GAN's generator G_E. This pipline is trained in a supervised fashion using a novel retrieval loss L_R modelled as a soft margin ranking loss.

$$L_R(G_E, R_E, SA) = \mathbb{E}_{I_{ps},I_g}\mathbb{E}_{I_g \neq \tilde{I}_g}[\log(1 + e^{\alpha d(I_g,I_{ps}) - \alpha d(\tilde{I}_g,I_{ps})})]$$

where I_g is ground truth, I_{ps} is a satellite view image after polar transformation and the distance metric d is

$$d(I_g, I_{ps}) = \|SA(R_E(I_g)) - SA(G_E(I_{ps}))\|_2^2$$

The polar transformation and rest of the architecture is explaied in next section.

2.3.2 Non-Supervised Learning

Cycle Consistent Adversarial Networks

One of the seminal papers that deal with image translation in unpaired data is [5] known as CycleGAN. The authors used pairs of GANs to translate image to and from different domains. The novelty comes from the fact that both of these networks are trained in synergy.

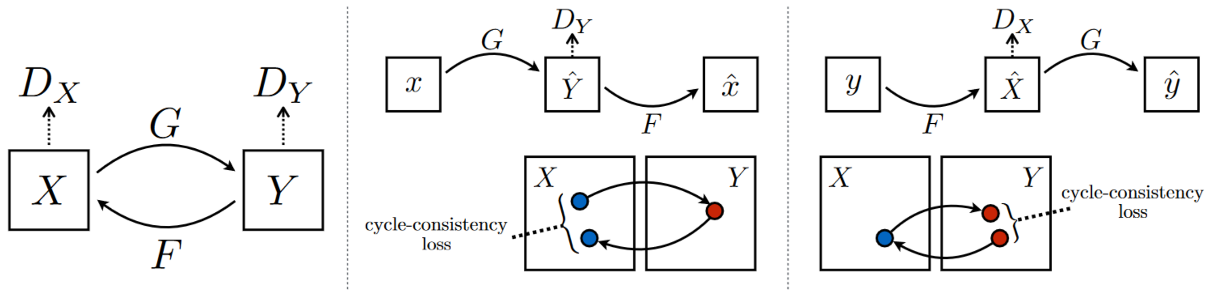


Figure 2.7: The Cycle Consistency introduced in CycleGAN

Let the 2 image domains be X, Y and the GANs for translation be (G_X, D_Y), (G_Y, D_X). Then the loss function used in training can be written as.

$$L(G_X, G_Y, D_X, D_Y) = L_{\text{GAN}}(G_X, D_Y, X, Y) + L_{\text{GAN}}(G_Y, D_X, Y, X) + L_{\text{cycle}}(G_X, G_Y)$$

where L_{GAN} is the normal adversarial loss and

$$L_{\text{cycle}} = \mathbb{E}_{x \sim p_{\text{data}}(x)}[\|G_X(G_Y(x)) - X\|_1] + \mathbb{E}_{y \sim p_{\text{data}}(y)}[\|G_Y(G_X(y)) - Y\|_1]$$

This novel cycle consistency loss incentivizes the networks to generate outputs which can again be translated back to the same image.

FUNIT: Few-Shot Unsupervised Translation

Most translation methods even if they work with unpaired data require a high volume of data. However, humans can pick up the essence of an object from looking at a few examples and generalizing from it. The FUNIT framework [6] successfully achieves this few-shot generation capability by using adversarial training and a novel network design.

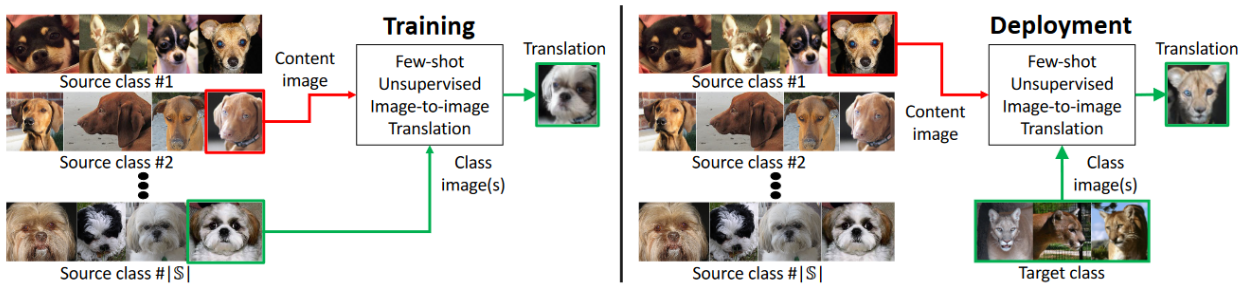


Figure 2.8: **Training** We train the network to translate between source classes. **Inference** The network is shown a few samples from target class.

Unsupervised translation is learning a translation function from a source class to a target class without pairwise supervision. This problem is to recover a joint distribution from samples of marginal distributions. There exist infinite viable solution to this problem. CycleGAN deals with this problem by using a cycle consistency constraint. FUNIT method assumes that both distributions have a partially shared latent space. It also addresses the problem of resuing a trained network to translate to a different but similar target class without any re-training.

The generator G takes in a source image x and k images from target class $\{y_1, ..., y_k\}$

and outputs a translated image $\bar{x} = G(x, \{y_1, ..., y_K\})$. For training the network, this is the expression which is optimized

$$\min_D \max_G L_{\text{GAN}}(G, D) + \lambda_{\text{R}} L_{\text{R}}(G) + \lambda_{\text{FM}} L_{\text{FM}}(G)$$

Here L_{R} is the reconstruction loss which ensures that when target class image is same as input, output should be identical.

$$L_{\text{R}}(G) = \mathbb{E}_x[\|x - G(x, x)\|_1]$$

And $L_{\text{FM}}(G) = \mathbb{E}_{x,\{y_1,...,y_k\}}$ is the feature-matching loss, where features are calculated by using the discriminator D as a feature extractor D_f by removing last prediction layer. We extract features from output image $\bar{x}$ and the target images and ensure they are close. This acts as a regularization term.

$$L_{\text{FM}}(G) = \mathbb{E}_{x,\{y_1,...,y_k\}}[\|D_f(\bar{x}) - \sum_{i=1}^{k} \frac{D_f(y_i)}{k}\|_1]$$

Details about inference and model architecture are in the next section.

2.4 Methodology

2.4.1 Datasets

I am using the CVACT and the CVUSA datasets, both of which contain paired satellite and street view images.

	CVACT	CVUSA
Ground View FOV	360	360
Street view resolution	1664x832	1232x224
GPS-tag	Yes	No
Satellite resolution	1200x1200	750x750
Training samples	35532	35532
Testing samples	92802	8884

Table 2.1: Comparison between the major available datasets

Since the CVACT dataset isn't publicly available, I had to reach out to the authors of

[7] for access. I have also explored the option of creating our own dataset by acquiring the required images from the Google Maps API, but this would need a paid access and thus this will be explored in later stage of the project.

2.4.2 Polar Transformation

The polar tranformation introduced by the authors of [1] exploits the geometric relationship between street view and top down view. If the top down view were to unwrapped around the center and stretched out, we get a very structurally similar image to the street view. This similarity can be used to our advantage by conditioning the GAN on this transformed image instead.

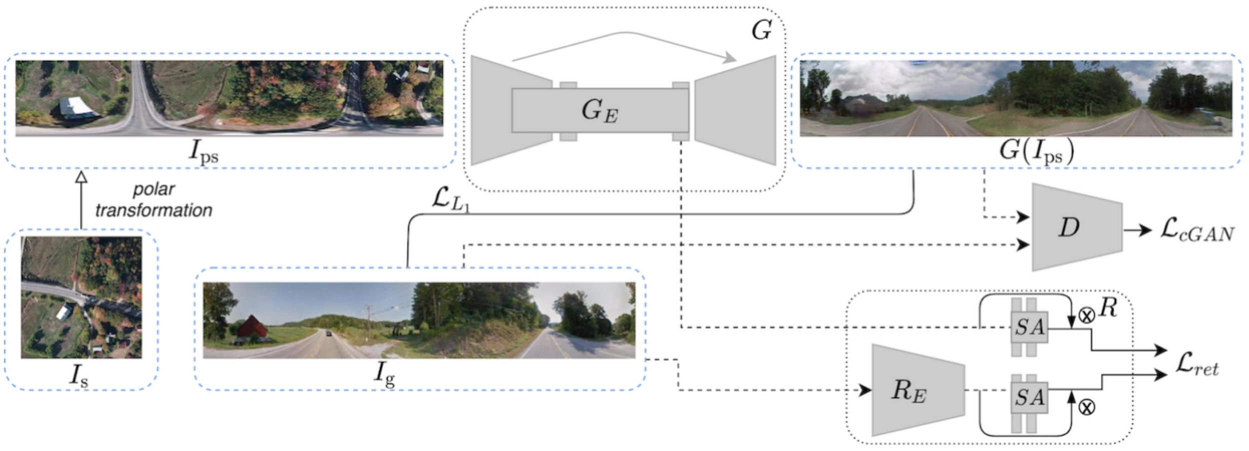


Figure 2.9: Model architecture from [1]

The transformation of coordinates $(x_s, y_s) \rightarrow (x_{ps}, y_{ps})$ can be written as

$$x_s = \frac{W_s}{2} + \frac{W_s}{2}\frac{y_{ps}}{H_{ps}}\sin\left(\frac{2\pi}{W_{ps}}x_{ps}\right) \qquad y_s = \frac{H_s}{2} - \frac{H_s}{2}\frac{y_{ps}}{H_{ps}}\cos\left(\frac{2\pi}{W_{ps}}x_{ps}\right)$$

Here $(W_s, H_s), (W_{ps}, H_{ps})$ are the image dimensions. This transformation helps us to bridge the domain gap a little. Then we can easily train the GAN as well as the retrieval pipeline as described in previous section.

2.4.3 Model Architecture

I am going to use a similar architecture to the 2.10 for the FUNIT method of few shot learning. From the literature review, I have learned that training a good model would

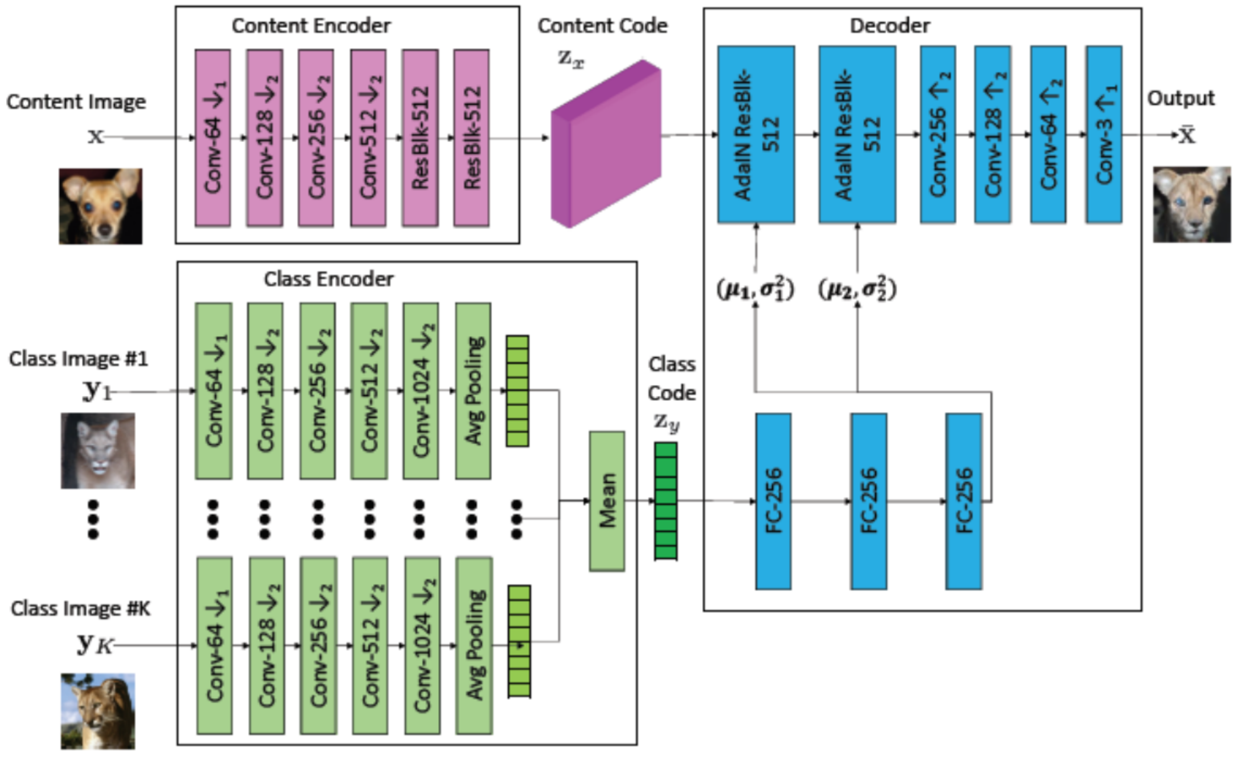


Figure 2.10: Model architecture for FUNIT method

require adversarial training loss as well as a good reconstruction loss and an end-to-end retrieval ranking loss for efficient retrieval of images. I also learned that it's feasible to perform retrieval using feature maps themselves rather than the output images.

2.5 Work for Phase 2

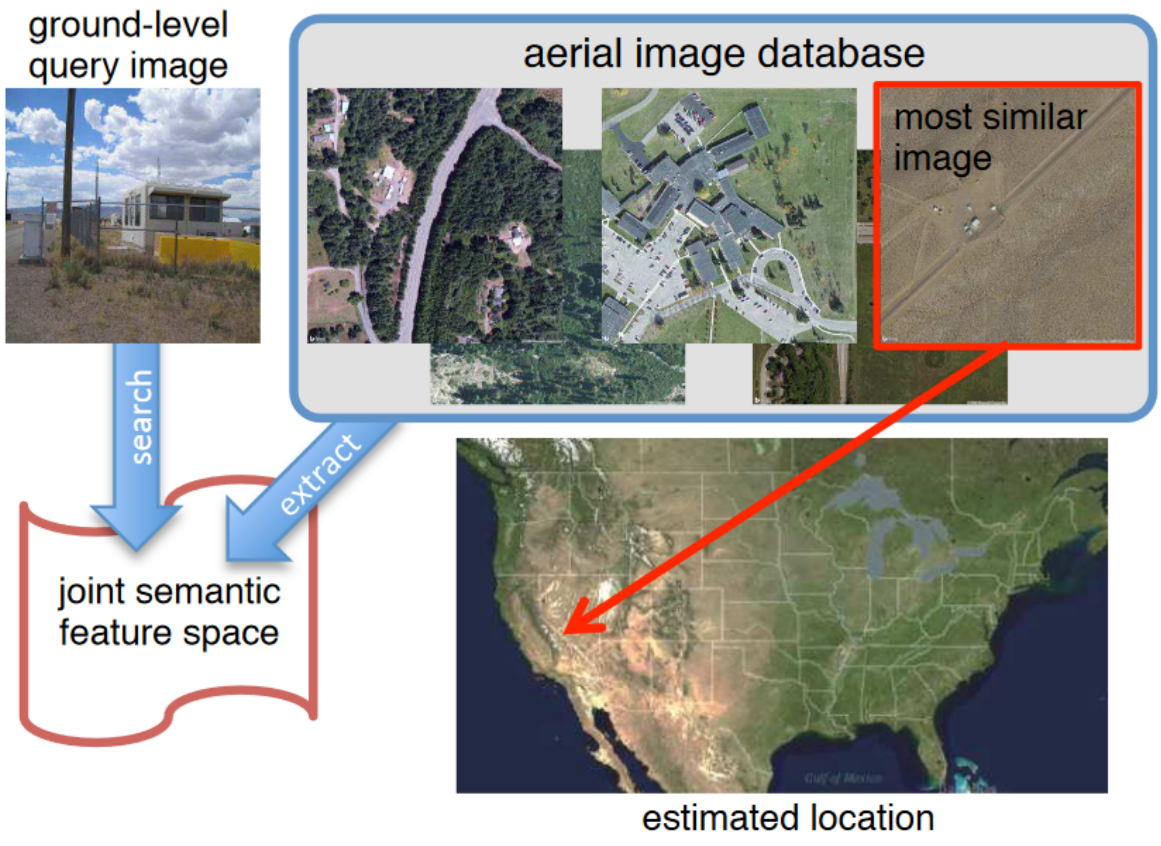


Figure 2.11: The proposed solution

I am training a model based on the FUNIT framework in few-shot fashion, the input is transformed first using the polar transformation described above. This model would be able to generate street views from satellite images and create a database of various places from their satellite images which are widely available. These images or their embeddings could then be matched for accurate localization.

Other than that, FUNIT is quite slow to train, so meanwhile I am planning to see what is the baseline result by training a supervised pix2pix like network on the same task.

Bibliography

[1] *Aysim Toker et al.* Coming Down to Earth: Satellite-to-Street View Synthesis for Geo-Localization
https://arxiv.org/pdf/2103.06818.pdf

[2] *Phillip Isola et al.* Image-to-Image Translation with Conditional Adversarial Networks
https://arxiv.org/abs/1611.07004

[3] *Ian Goodfellow et al.* Generative Adversarial Networks
https://arxiv.org/abs/1406.2661

[4] *Alec Radford et al.* Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks
https://arxiv.org/abs/1511.06434

[5] *Jun-Yan Zhus et al.* Unpaired Image-to-Image Translation using Cycle-Consistent Adversarial Networks
https://arxiv.org/abs/1511.06434

[6] *Ming-Yu Liu et al.* Few-Shot Unsupervised Image-to-Image Translation
https://arxiv.org/abs/1905.01723

[7] *Ming-Yu Liu et al.* Lending Orientation to Neural Networks for Cross-view Geo-localization
https://arxiv.org/abs/1903.12351

[8] *Martin Arjovsky et al.* Wasserstein GAN
https://arxiv.org/abs/1701.07875

[9] *Mehdi Mirza et al.* Conditional Generative Adversarial Nets
https://arxiv.org/abs/1411.1784

[10] *Kurt Hornik et al.* Multilayer Feedforward Networks are Universal Approximators
`https://cognitivemedium.com/magic_paper/assets/Hornik.pdf`

[11] *Olaf Ronneberger et al.* U-Net: Convolutional Networks for Biomedical Image Segmentation
`https://arxiv.org/abs/1505.04597`

U-Net: Convolutional Networks for Biomedical Image Segmentation