Unveiling Bacterial Microcolony features through Graph Attention Representation Learning

by

Sushma Dhamodharan

A research report
presented to the University of Waterloo
in fulfillment of the
research requirement for the degree of
Masters in Mathematics
in
Computational Mathematics

Waterloo, Ontario, Canada, 2024

Author's Declaration

I hereby declare that I am the sole author of this report. This is a true copy of the report, including any required final revisions, as accepted by my supervisor.

I understand that my work may be made electronically available to the public.

Abstract

Calibration of an agent-based model (ABM) in a biological environment to simulate experimental observation is often a challenging task. The difficulty arises while selecting meaningful features that are both representative and suitable for constructing a robust goodness-of-fit measure for the spatio-temporal biophysics engine.

In this research, we focus on the development of a simple Graph Attention Network (GAT), for representing the high-dimensional simulation of *Escherichia coli* (E. coli) microcolonies, that grows dynamically through space and time. Our goal is to learn a representation embedding from a high-dimensional experimental data. To tackle this problem, an initial first step is to learn a representation from the simulated data that effectively captures the distinctive characteristics of different colonies through a triplet loss training strategy. The triplet loss ensures that microcolonies with similar growth dynamics have a similar feature embedding, compared to the dissimilar growth parameters.

To ensure the generalizability of the trained encoder, we evaluate the performance of the model using unseen parameter combinations. In future works the resulting feature vectors are passed through a regressor to get calibration parameters that align experimental and simulated data. Furthermore, we assess the influence of embedding dimensions and epochs on the accuracy and consistency of the encoder during calibration.

This work provides a novel approach for calibrating a biological spatio-temporal agent-based model, whilst offering insights about the data which an otherwise not be measured directly.

Acknowledgements

I would like to thank Dr. Brian Ingalls, Atiyeh Ahmedi, Zaheer Mohideen and Dhruva Abhijit Rajwade for offering their time, thoughts, and support throughout this project and report.

Table of Contents

List of Figures

Chapter 1

Introduction

Microbes are the foundation of our planet [3], the tree of life. Life without other beings is possible, but life without microbes is not possible. They influence all life forms and the earth's physical and chemical structure. They are very crucial for the survival of humanity and provide essential materials for medicine and industry. Microbial communities are naturally found in various environments and habitats. Studying them is crucial as they could be engineered to benefit humanity.

They can be engineered to produce vaccines and antibiotics, they could also be engineered to extract metals (biomining) [17], and their potential extends beyond wastewater treatment or producing biodegradable plastics. Powerful new technologies such as genomics, nanotechnology, rapid DNA sequencing have shaped how the world views microbes, increasing the need to be educated about microbes. Although the threat of a biowar is increasing, microbes are still beneficial for our health it's usage in vitamins, lactic acid and vitamin K regulates inflammatory responses [24]. Thus, the study of these microbes is very crucial.

Microbes are microscopic organisms like bacteria, viruses and fungi and our human body is home to millions of them. Although there isn't a very clear definition of what constitutes a microbe, there are several articles that says that they are not completely invisible to the naked eye, [11] explicitly states that although fungi are part of a microbial community and it can range from being macroscopic to microscopic in size.

Microbes are found in various environments, from soil and water to the human body. The study of microbes is known as microbiology, and it has led to many important discoveries and applications. They are capable of complex social interactions like higher organisms, and can coordinate their group behavior communicate to each other by quorum sensing

[18]. The network interactions determines species abundance, and their spatial arrangement [45].These communities form elaborate structures that mimic human cities and are called biofilms. Their natural habitats such as the mammalian gut or soil, often makes it difficult to study them due to their spatial complexity. To develop a fundamental understanding of how these microbial communities colonize and grow, we attribute to using biofilms for maximizing their biotechnological potential. Under the microscope the biofilms look like cities, and have their own channels of water, waste and food. Due to the proximity of cells in a biofilm it allows for easier cell-to-cell interactions, like gene exchange by conjugation. Hence biofilms are an excellent way to monitor the interactions.

Mathematical and computational models [30] help us understand processes involved in complex systems. It allows us to decipher the what, how and why of any given system. Mathematical models refer to the theoretical approaches and computational models refers to the computational approaches used to solve these mathematical equations, they are often used together and are called models. The purpose of a mathematical model is to allow us to answer unanswered questions and test different scenarios that explain the current behavior of the system. However, in many cases studying these dynamic systems often involve in creating a virtual system using mathematical models. A complex system like a biofilm [12] which exhibits spatio-temporal behavior requires mathematical modeling. Although no model is perfect and may not be able to make precise predictions, nevertheless they give new insights into the mechanisms involved and stimulate further investigations.

Microbes exhibit spatio-temporal behavior and involve a highly complex biophysics engine, the usage of numerical methods tend to oversimplify the process involved. Simulation as presented by Wanner in [43] of these systems takes accountability of the variability involved in the process. Hence, simulating these system help us understand the interactions better. Mathematical modeling in particular spatially explicit agent-based models [27] can be used to simulate these complex intra and inter cellular processes by incorporating the spatial and temporal dynamics in a real world setting.

Although these models are an excellent framework, they are highly parametrized and stochastic, requiring precise calibration for predictive performance, which is a computationally expensive procedure [27]. Although state-of-the-art computational algorithms enable the simulation of systems containing up to 10^7 cells, the simulated population size is still several order smaller magnitude than many biological systems such as the human microbiome or wastewater treatment facilities [10]. Previous work [45] [2] shows that this process can be done informally for finding the right parameter set for calibration. However, considering the complex environment of our microbial communities which grows both spatially and temporally, whether we can find the features that describes these dynamics and provide a goodness of fit measure is also of question.

In this report we explore the use of graph attention network an encoder pipeline architecture to get a one dimensional feature embedding for a bacterial population observed from a single-cell time-lapse simulated data. We will extend this work on how different feature embedding sizes of the E. coli micro colony will perform in distinguishing these micro colonies with different parameter setting.

1.1 Agent Based Modeling

The dynamics of the microbial community greatly depends on the cell-cell interactions, competition for nutrients, metabolite exchange, toxin production, antibiotic activation and quorum sensing. The cell-cell interactions affect the spatial arrangement and determine the abundance of the micorbe's in a biofilm. This spatial structure influences the cooperative and competitive interactions as stated in[45]. Environment's colonization history also affects the community composition. The dependencies makes precision manipulation difficult, and hence mathematical models can be used to understand the colony behavior.

There are 3 main approaches that can be used to model these systems, namely Ordinary differential equations, partial differential equations and agent-based models. The differences in these models arise from the spatial resolution. Each model offers certain benefits and disadvantages. The first offers low computational costs, but doesn't take into account the spatial dependency. The second model accounts for the spatial dynamics but produces local averages, rather than individuals. The third model, captures the spatial dynamics and heterogeneity among cells, but has a very high computational complexity and the order of magnitude of simulated data is still several orders less than the experimental data [27]. Using the models in conjunction helps our predictive power to describe intra, inter and extra-cellular processes. To make accurate predictions these models needs to validated against experimental data. Since the observations are made at single-resoltuion the observations are to be measured at the same resolution with the experimental data as well.

ABM allow us to study emergent properties that arise from local interactions [39], among multiple independent components. The model gives flexibility to control individual components and thereby allowing us to study the causation at macro-level. This is referred to as 'downward causation' or 'bottom-up' approach. Agent-based models involve two main components namely agents and environment. Agents are computational representation of individual actors, capable of interacting passing messages and have their own set of actions based based on the interactions. Each agent is an autonomous entity. Further this population can include heterogeneous agents, where different agents have their own roles,

actions and knowledge. Agents typically have 4 main characteristics as defined by Abdou, Hamill, and Gilbert, 2012 [14], and listed below.

- Perception: Agents can perceive their environment, including other agents in their vicinity.
- Performance: They have a set of behaviours that they are capable of performing such as moving, communicating with other agents, and interacting with the environment.
- Memory: Agents have a memory in which they record their previous states and actions.
- Policy: They have a set of rules, heuristics or strategies that determine, given their present situation and their history, what they should do next.

The environment is the virtual world the agents interact and present. The environment can be neutral with no impact on the agents or can also be a medium where the environment stimulates certain physical features. In the later case, the models are called *spatially explicit*, and the agents have coordinates to indicate their positions.

In summary, Agent-based models can be used as a platform for running and observing simulations. It can be thought of as a simulation of a laboratory where the environment and the behavior of agents along with their individual attributes can be altered and the outcomes can be observed over multiple simulation runs. By enabling the ability to simulate individual action of many agents and observing the resulting system over time in multiple simulation runs means that agent-based models can be useful tools for studying processes that operate at multiple scales.

The spatially explicit agent-based model that we have will help us interpret the bacterial microcolonies generated from a single-cell microbe, by understanding the individual cells actions, behavior and environmental attributes grown over time. The variability in the model, captures the variability in a real world setting for the same parameters. This process is much needed as each cell reacts in a different way and that's one among the reasons as to why we have a different colony structure every single time we repeat the experiment. Given the stochastic nature of the agent-based model, it will generate a different simulation for the same parameter, which is a desirable simulation we thrive to generate. The main challenges in using this modeling structure, is the computationally intensive calibration process.

1.2 Model Calibration

Model calibration is the process of assigning values to model parameters to reproduce experimental data. The degree of confidence in model predictions reduces due to the uncertainty involved. The simplest approach as stated in [45] is to characterize each component separately. However for complex systems such as a microbial community often involves parameters that cannot be directly inferred or observed. In such cases a goodness of fit function is used to minimize the error under a maximum likelihood function, however in a non-linear dynamic model it offers less support as it only provides an approximation and not an confident estimate for uncertainty analysis. To account for the uncertainty, bayesian computing [7] is used to refine parameters by comparing against the experimental observations.

The model parameters were chosen by **systematic calibration against spatial summary statistics** method, which directly observes the desired parameters through image processing, cell segmentation and tracking [34]. For our research purposes 32 calibration parameters were measured in our previous works [45].

Statistic	**Goal of Statistic**
Microcolony shape	Quantify eccentricity of developing colony
Dyad structure	Characterize structure of 2-cell 'colony' immediately before the second division
Biofilm base circularity	Characterize shape of the biofilm base
Microcolony density	Quantify packedness of cells within developing colony
Order parameter	Quantify anisotropy within developing colony
Correlation length of scalar order parameter	Characterize 'patchiness': spatial scale over which orientation of neighbouring cells is aligned
Micropatch area	Quantify 'patchiness': similar to correlation length of scalar order parameter
Topological defect density	Characterize 'patchiness': density of topological defects
Defect velocity	Characterizes the evolution of a microcolony's internal structure
Age distribution of cell poles within the developing microcolony	Characterize degree of mixing during colony development
Orientation of cells at the microcolony boundary	Characterize tendency of boundary cells to align with the colony boundary

Statistic	**Goal of Statistic**
Gradient of cell velocity normal to microcolony boundary	Characterize growth inhibition due to pressure gradients
Cell-cell distance	Characterize cell spacing
Vertical and radial alignment	Characterize 3D structure; identify transition from monolayer to multilayer growth
Single-strain population counts	Captures population abundances
Shannon species diversity index	Species biodiversity metric
Shannon entropy	Quantify randomness of pixel identities in an image
Intensity correlation quotient	Characterize colocalization or exclusion of pairs of species in space
Contagion index	Quantify dispersion and intermixing of different populations
Probability matrix for nearest cell-cell adjacencies	Global quantification for likelihood of nearest neighbour to cell centroid
Neighbour index	Characterize interspecific adjacencies relative to the initial adjacencies
Proportion of conspecific neighbours	Quantify interspecies mixing from a probability distribution
Structure factor	Quantifies characteristic length scales of spatial patterning
Segregation index	Normalized metric to quantify population segregation
Colony edge roughness	Increased in communities with antagonistic interactions
Fractal dimension	"Jaggedness" of species boundaries
Intermixing index	Estimate spatial mixing between multiple species
Single-strain sector size	Indirect measurement of spatial mixing between multiple species
Order parameter; phase transitions	Quantify synchrony in gene expression between populations

Table 1.1: ABM Parameters, referred from [45]

The simulations were developed with the *CellModeller*[34] package, and for calibration two parameters were of primary importance to characterize the biophysics engine, the

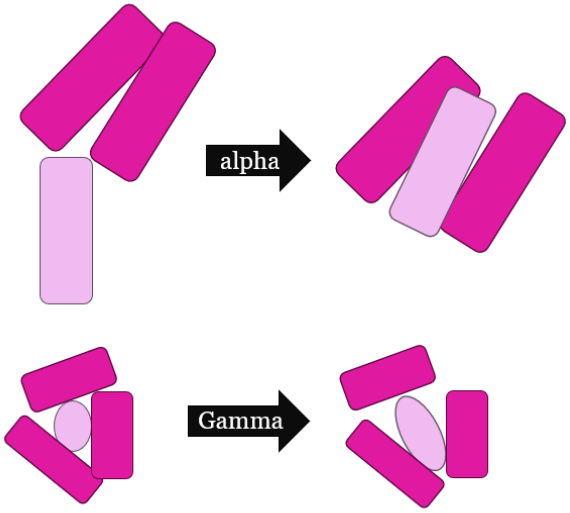


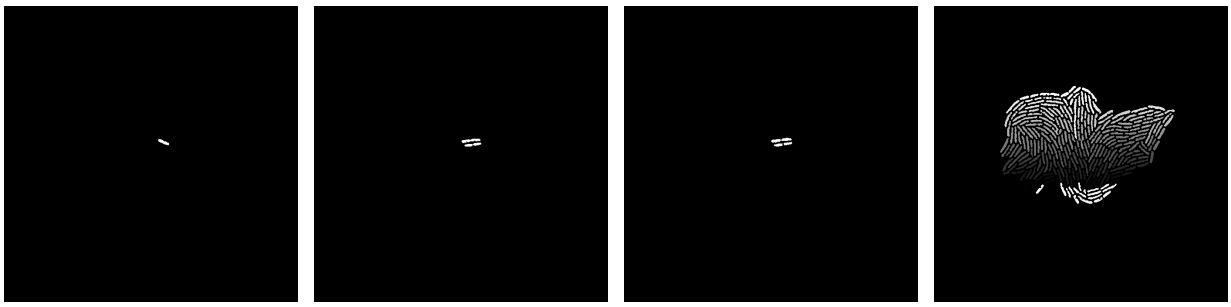
Figure 1.1: ABM parameters α and γ, the left side indicates when α and γ is low, the right side shows how the displacement and growth affects the cells and its neighbors as the value increases.

parameter α and γ where α captures the displacement of cells and γ captures the growth of cells. When α is high it means that the cells are not rigid enough that when another cell pushes through it leads to an unresolved overlap but when α is low it means that the cells are rigid and do not allow for any displacement, α tries to penalize such displacement. When γ is high it means, although a cell is tightly surrounded by other cells the growth of that cell pushes beyond those restrictions. And when γ is low the growth gets inhibited because of the neighboring cells. Even if they are different parameters, they two are inversely proportional to each other, but are calibrated separately to allow for wider range of behaviors. Ideally, overall the entire structure of the micro colony is defined by these pushes. The *Figure* 1.1 explains the characteristics visually.

1.3 Cell Tracking, Measurement and Relationships

Simulated data was generated for every time-lapse mimicing the experimental time-lapse observations using a custom Python package mentioned in the article[2]. These simulated data, already accounts for the parent-daughter relationship. Using these generated cell

attributes for every cell in every time lapse, we generate the cell-cell relationships using a custom Python module based on the algorithm 1.



(a) $t = 0$. (b) $t = 3$. (c) $t = 23$. (d) $t = 85$.

Figure 1.2: Samples of Time lapse images from first frame to last frame

Relationships can be defined as two types, Lineage (Parent-Daughter relationship) and Contact (Cell - Neighbor cell). The simulated data from [2] records the parent-daughter relationship from step (i) to step (i + 1) in every iteration. This information, along with the attributes of every cell and the contact edges generated using the algorithm 1 is crucial to learn a feature embedding from the micro colony structure. The $cell_i$, $cell_j$ refers to the cell position 1.1 from the list of cells present in the time stamp.

Algorithm 1 Identify Neighboring Cells in Timelapse Data

```
for each stepnum in timelapse do
    for each cell_i in cells of stepnum do
        for each cell_j in cells of stepnum do
            if distance(cell_i, cell_j) < threshold then
                Record that cell_i is a neighbor of cell_j at stepnum.
            end if
        end for
    end for
end for
```

1.4 Related Work

Calibration of agent-based models is a very well studied domain, a popular approach is to use the Bayesian calibration [19], which accounts for the uncertainty analysis. To further

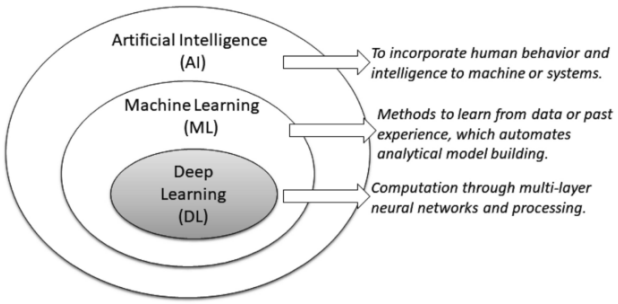


Figure 1.3: An illustration of the position of deep learning with respective to machine learning and artificial intelligence [1]

improve on the uncertainty and sensitivity analysis the POM (pattern-oriented modeling) was implemented as a pipeline for calibration [2]. With ABM, we introduced similarity through stochasticity [2] to learn an embedding from the simulated data, and pass the embedding through a regressor to learn the ABM (agent-based model) parameters [32]. However, this model had difficulties in data augmentation as every time step had different sequence lengths. To tackle this problem, the data modeling was altered to be represented as graph structures to train an encoder architecture as graph attention network to learn feature embeddings.

1.5 Deep Learning

Before understanding the graph attention network we first have to understand the background concepts involved. Nowadays, artificial intelligence (AI), machine learning (ML) and deep learning (DL) are three popularly used terms for intelligent systems. The figure 1.3 illustrates the positions of DL comparing with AI and ML. According to the figure DL 1.3 is part of ML and AI.

Deep learning was first introduced as a concept of artificial neural network in [15], and when properly trained produced significant success in a variety of classification and regression tasks [36]. A typical neural network is composed of neuron, each of the neuron generates a series of real valued activations for the target outcome. In deep learning, we have multiple layers 1.5 of these neurons and non linear transformations. A neuron typically 1.4 contains multiple components an input X, weight W, bias b, an activation function f and corresponding output signal y. When multiple layers (Multi-layer perceptron - MLP) are involved, the layers in between are referred to as hidden layers. A typical MLP

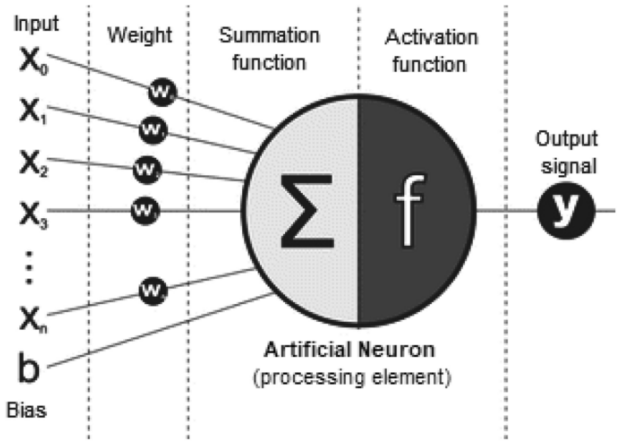


Figure 1.4: Schematic representation of a neuron, highlighting the important components [29]

consists of an input layer that receives the input, an output layer that makes decision or prediction, with one or more hidden layers. The output is determined by activation functions like tanh, ReLU, sigmoid and softmax. To train the model, it goes through a process called as Backpropagation and optimization algorithms such Adaptive Moment estimation (Adam), stochastic gradient descent (SGD) are used.

Although these models can be applied in a variety of domains and tasks, building an appropriate model is a challenging task, due to the dynamic nature in a real-world setting. These models are also considered as black-box models as they cannot be easily interpreted.

1.6 Convolution Neural Networks

Convolution neural network (CNN) also called as ConvNet was first introduced by [22] in 1989, has a deep feed-forward neural network architecture, and has shown to have a greater generalizability compared to a fully connected network [28]. It has the ability to learn highly abstract features. The architecture proposed was called LeNet and has convolution layers followed by subsampling layers implemented to learn features from pixel data. CNN has consistently been able to provide desirable results than many other classical models due to the following reasons [4] [38] [40],

- The concept of Weight sharing, has considerable reduced the number of parameters that needs to be trained, and does not suffer overfitting.

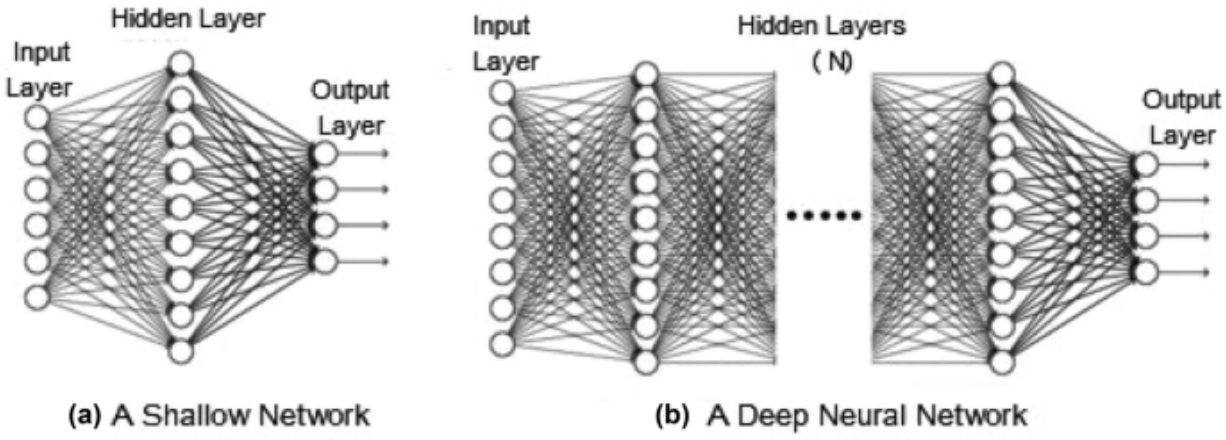


Figure 1.5: Example of a shallow and deep neural network [26]

- Classification is incorporated with feature extraction in the learning process.
- It is difficult to implement large networks compared to CNN

They are widely used in various domains like image classification, object detection, face detection and vehicle recognition.

1.6.1 General Model

A typical model of ANN consists of an input layer, an output layer and a weight vector. The input vector denoted as X, produces output Y by performing some function F on it 1.1.

$$f(X, W) = Y \tag{1.1}$$

The vector W, represents the strength of interconnection between two neurons of adjacent layers. The obtained weight vector will aid us in our task of image classification for instance.

Generally CNN has 4 components, namely a convolution layer, pooling layer, activation function and a fully connected layer. The figure 1.6 depicts the overall structure.

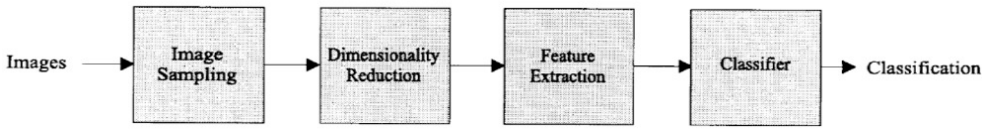


Figure 1.6: Elementary components of CNN [16]

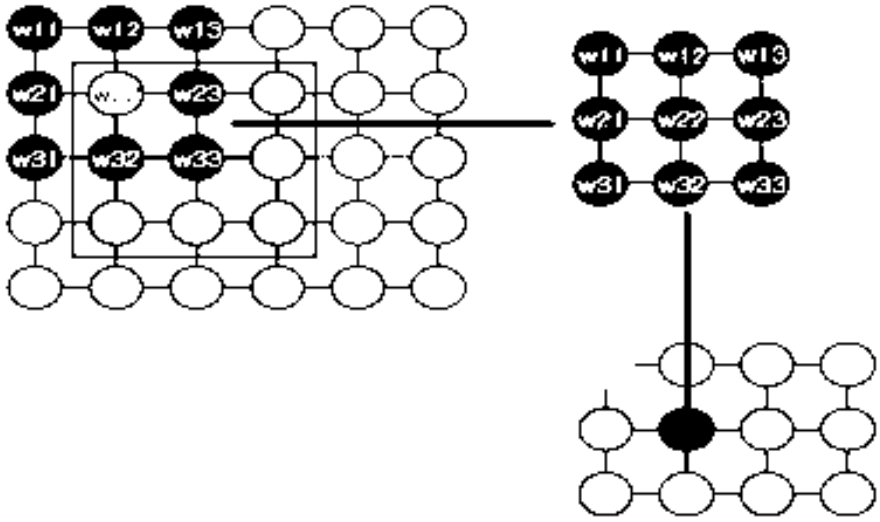


Figure 1.7: Receptive field of particular neuron in the next layer [8]

1.6.2 Convolution Layer

In the task of image classification, an input image is passed to the input layer, and the output is predicted using the features extracted from the image. An individual neuron is connected to othe neurons in the next adjacent layer, this local correlation is termed as receptive field. The features from the image are extracted using a receptive field. The receptive field from a previous layer forms a weight vector equal at all points on the neurons on the next layer [16]. As the neurons share the same weight 1.7, they can detect similar features at different locations in the input data.

The weight vector knows as filter or kernel slides over the input data to collect similar features called as the feature map. This method of horizontal or vertical sliding is called the convolution operation. If we N filters, we will be having N feature maps. Due to receptive field, number of training parameters is significantly reduced.

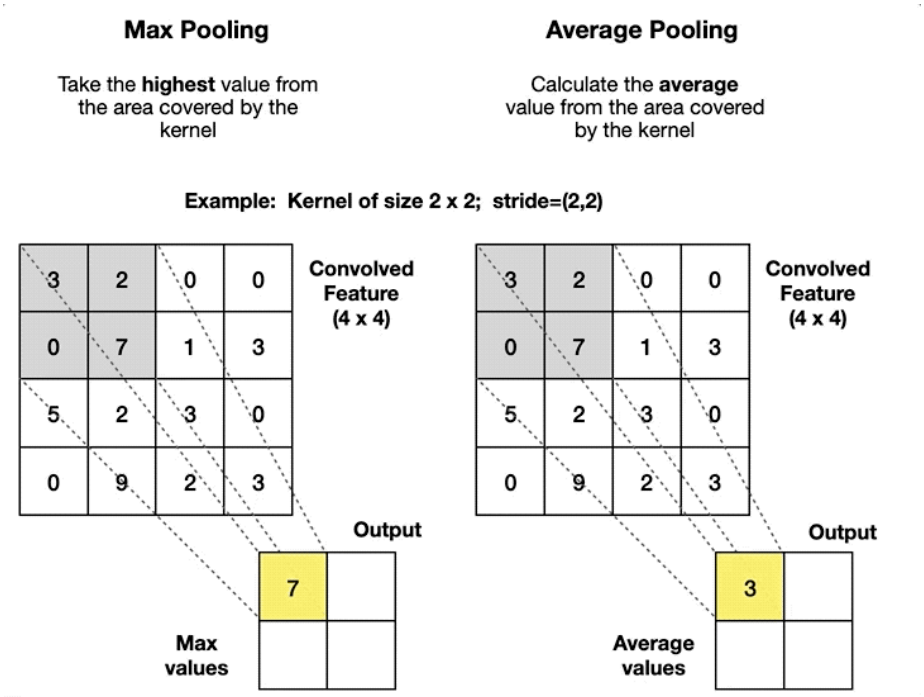


Figure 1.8: Pooling operation performed by a 2x2 window left indicating max pooling and the right indicating average pooling. [35]

1.6.3 Pooling Layer

The location of a feature becomes less significant after the convolution, the output of the convolution is then passed to a pooling layer. The pooling layer reduces the trainable parameters and introduces translation invariance. To perform the operation the pooling window is collected and is passed as a window over the output received from previous layer. There are variations to the pooling methodologies, the most common being max-pooling and average pooling.

1.6.4 Fully connected Layer

The output of the previous layers is passed to a fully connected layer to train. The input data is passed as batches compared to that of full dataset in one epoch. The model achieves global minima, but the computational time increases as the data size increases.

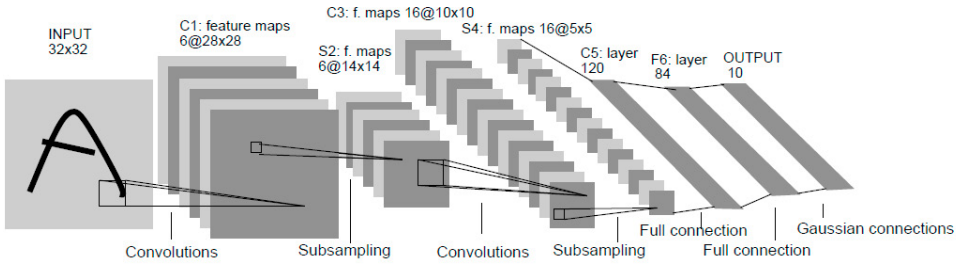


Figure 1.9: Architecture of LeNet5, a CNN where each box represents a different feature map [22]

1.6.5 Activation Function

Activation functions like sigmoid and Rectified Linear Unit (ReLU) are popularly used. But using RelU has considerable advantages compared to the former due to its differentiable nature and the training time to one factor of the sigmoid [46].

1.6.6 Model Architecture

There are several architectures that have been proposed over the years. For illustration purposes, LeNet the first introduced architecture is explained 1.9.

The LeNet architecture [22] consists of 5 convolution layers and 3 fully connected layers. Every unit in convolution layers has 25 kernels with a window size of 5x5 each, which slides over to learn the feature map. Various feature maps are generated and to reduce the precision on position sub-sampling is performed, and the size remains the same after the convolution. In this case sub-sampling is performed in 2x2 window and the average is computed over the 4 inputs received. This is passed to the next fully connected layer where its multiplied with trainable weights and added with bias and passed to a sigmoid function. The learning is accomplished by back propagation.

1.7 Recurrent Neural Network

Recurrent Neural Network (RNN) are a class of deep learning models that are used for handling sequential data [25]. Unlike a traditional feed forward network, RNNs have the ability to maintain memory from the previous input stages. This makes these models

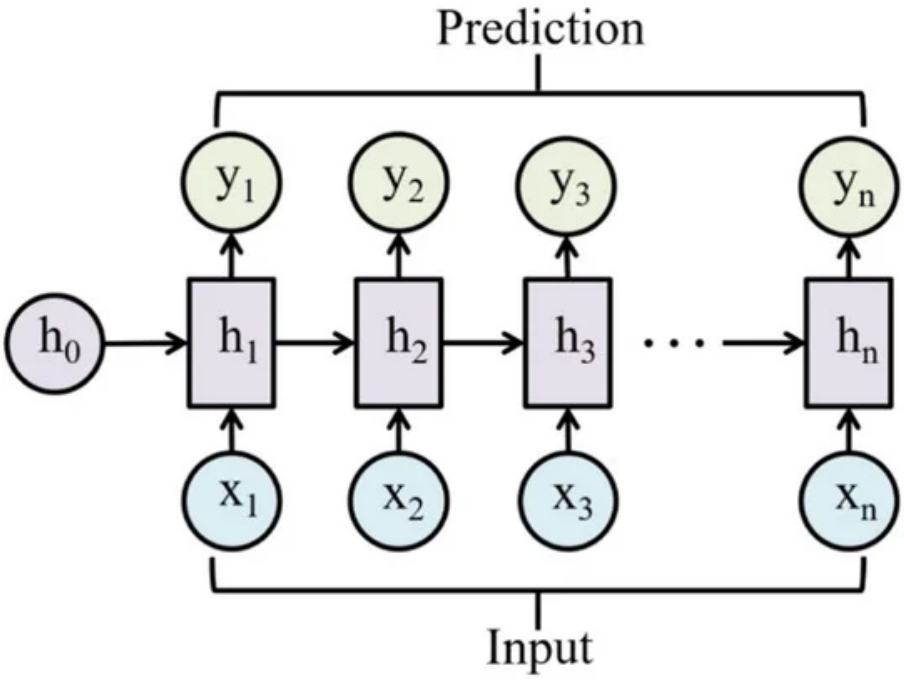


Figure 1.10: Architecture of RNN [25]

well suited for natural language processing, speech recognition, and time series forecasting, where context and the order of data points are crucial.

Architecture of RNN

The *figure* 1.10 illustrates the general structure of an RNN. This model captures contextual information by maintaining a hidden state that captures previous inputs to the current state. The basic architecture consists of an input layer, a hidden layer, and an output layer. Unlike feed forward neural networks, RNNs have recurrent connections allowing information to cycle within the networks. At each time step, t, the RNN takes an input vector and updates its hidden state using the following steps:

$$h_t = \sigma(W_{hh}h_{t-1} + W_{xh}x_t + b_h) \tag{1.2}$$

$$o_t = \mathrm{g}(W_o h_t + b_o) \tag{1.3}$$

The problem with this model, is its prone to vanishing/exploding gradient problem during back propagation, making it difficult to learn the long term dependencies.

To avoid this problem new methods such as GRU (Gated Recurrent Units) and LSTM (Long Short Term Memory) methods were introduced. To capture the past information effectively Echo state network, and Long Delays where introduced [31] [33].

1.8 Attention

The attention mechanism was first introduced in [5] to improve the performance of the encoder-decoder model for machine translation. It tries to universally represent a single concept of interest. The basic idea is the mechanism tries to focus on important factors when processing data. Attention is a fancy name for weighted average.

1.8.1 Transformer

Sequence-Sequence model led to the invention of Sequence-Sequence with Attention, which was the building block of the popular model Transformers. The transformer architecture tries to train the data based on the attention captured for the input at a given time. It was first introduced in the paper "Attention is All you need" [41]. It is an encoder-decoder architecture that can process sequential data in parallel and process the contextual information with long dependencies in the form of attention.

1.8.2 Self-Attention Mechanism

This mechanism 1.11 will allow us to model each word or token in the input sequence to consider the relevance of every other token, rather than encoding a sequence in a fixed, linear fashion as in RNN.

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V \tag{1.4}$$

To calculate the attention of a target word with respect to the input word, we use the Query (Q) of the target and the Key (K) of the input and then calculate a matching score. These matching scores act as the weights of the Value (V) vectors.

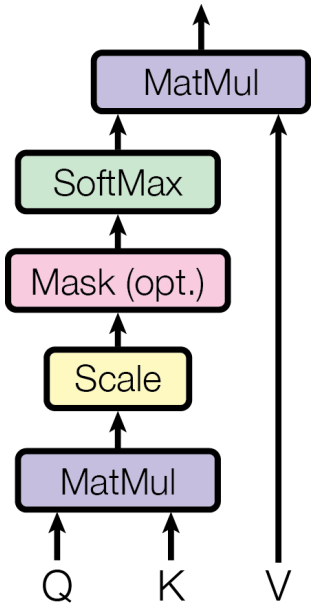


Figure 1.11: Self-Attention Mechanism [41]

1.8.3 Parallelization

Due to the resilliance on attention mechanism, the model can process the tokens in parallel and allow for faster processing. This leads to significant improvements in training speed, particularly on GPUs.

1.8.4 Positional Encoding

The model introduces positional encodings which are added to the input embeddings to give the model information about the sequence order.

1.8.5 Multi-Head Attention

The model uses multi-head attention 1.12, instead of a single attention where each attention tries to capture different variants of information. This information can be learned in parallel. This allows the model to learn different aspects of the relationships in the data at once and capture a broader range of interactions.

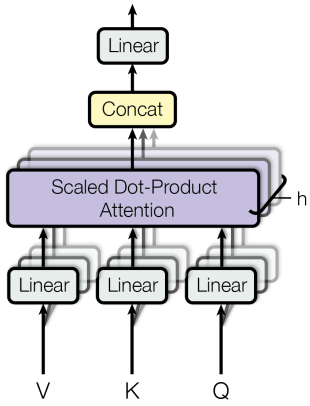


Figure 1.12: Multi Head Attention [41]

1.8.6 Encoder-Decoder Structure

Like other sequence-sequence models, the Transformer has an encoder decoder architecture 1.13. The encoder processes the input sequence and creates a representation, which is then used by the decoder to generate the output sequence for this representation. Both encoder and decoder use stacks of self-attention layers to capture dependencies at multiple levels of abstraction.

1.8.7 Feed-Forward Neural Networks

The structure of the feed forward network (FFN) is as follows, the Linear Layer which is the first layer, followed by ReLU activation and then another linear layer. While the attention mechanism captures global information between words and hence aggregates across columns, the feed forward neural network aggregates across rows and takes a more broader look at each word independently. Despite being processed individually, all positions share the same weights in this network.

1.9 Graph Neural Network

Graph neural networks are a class of deep learning models [44], capable of learning inference from graphs. Extending the concepts of convolution over a certain location in an image,

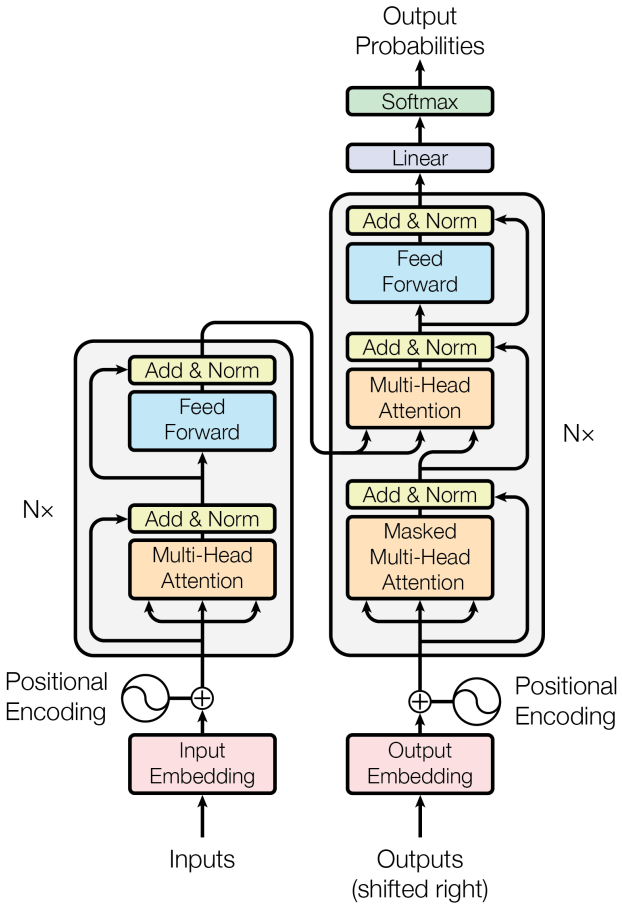


Figure 1.13: Transformer Model Architecture [41]

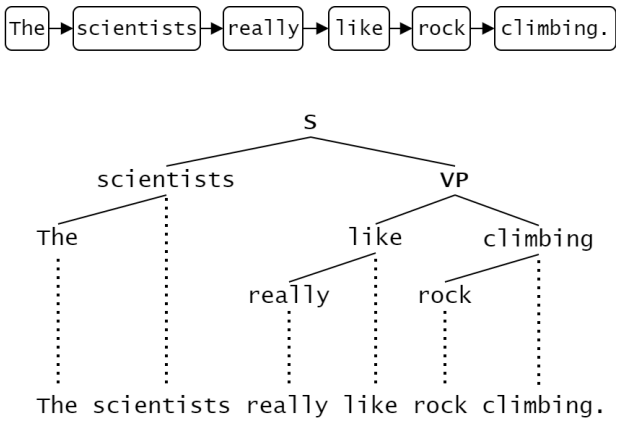


Figure 1.14: Text as graph structure; A vector representation and a Reed– Kellogg diagram (rendered according to modern tree conventions) of the same sentence. The graph structure encodes dependencies and constituencies. [44]

we apply convolution similarly on graph nodes [6]. Graphs are structured data, many real world data like social networks, world wide web and molecules can be expressed as graphs. In social network, each user can be represented as a node, their friendships can be represented as edges. Images and texts can also b represented as Graphs 1.14 1.15.

Hence, CNN can be viewed as a specialized form of Graph Neural Networks (GNNs) applied to grid-like structures. Each pixel in an image represents a node connected to its neighbors, with convolutions (Graph Convolution Networks, GCN) aggregating features across the whole grid.

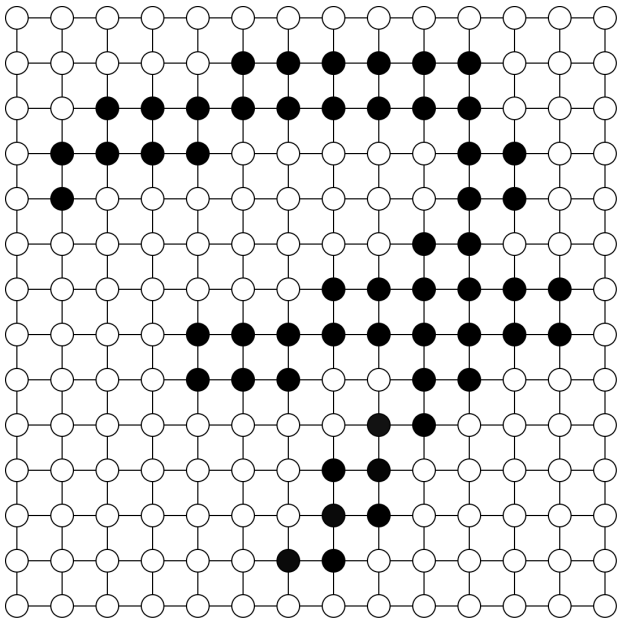


Figure 1.15: Image as graph; A graph representation of a 14 × 14 pixel image of the digit '7'. Pixels are represented by vertices and their direct [44]

Chapter 2

Spatio-Temporal Graph Attention Network

Graph Neural Networks are recently gaining traction on new innovations in Machine learning. Graph neural networks (GNNs) are basically convolutions on graph, which extends similarly as a transformer architecture used in Large Language models (LLMs) [44]. Unlike transformers operating on text, the GNNs operate on nodes in a graph. As graphs have the ability to represent complex relationships, we can leverage this capability to represent our complex micro colonies as graphs.

GNNs can be used for various tasks such as node classification, link prediction and graph classification [44]. In our case, we have to learn a feature embedding (representation) [2], using an encoder architecture from the graph structure. This generated embedding should have the ability to distinguish one representation from the other if generated from different model parameters.

Graph Attention Networks (GATs) are particularly suited for tasks modeled as graph-structures because they use a mechanism called message passing to propagate information across the network. In GATs, each node aggregates information from its neighboring nodes, but unlike traditional GNNs, they allow the option to assign different weights to each neighbor's message.

2.1 Graph Attention Networks

Graph Attention Networks (GAT) was first introduced in [42], leveraging self-attention layers to address the shortcomings of graph convolutions. In most of the implementations of GCN [9] [13] [21] the learned filters depend on the Laplacian eigenbasis, which depends on the graph structure. Thus, a model trained on a specific structure cannot be directly applied to a another graph structure.

Taking inspiration from [41] the authors of [42] came up with a novel approach of using attention on graph structures instead of convolution.

2.1.1 Graph Attention Layer

This layer computes attention scores between neighboring nodes and updates each node's feature representation. For a node v_i, its hidden state is updated by aggregating information from its neighboring nodes $\mathcal{N}(v_i)$.

$$h'_i = \sigma\left(\sum_{j \in \mathcal{N}(v_i) \cup \{i\}} \alpha_{ij} W h_j\right)$$

Where: - h_i and h_j are the feature vectors of nodes v_i and v_j respectively. - W is a learnable weight matrix that projects node features into a higher-dimensional space. - α_{ij} is the attention coefficient between node v_i and its neighbor v_j, determined using an attention mechanism (explained below). - σ is a non-linear activation function (commonly ReLU).

2.1.2 Attention Coefficients

The attention score α_{ij} quantifies the importance of node v_j to node v_i. It is computed using the self-attention mechanism [41], which involves

Calculating a score e_{ij} using the feature vectors of node v_i and its neighbor v_j:

$$e_{ij} = \text{LeakyReLU}\left(a^T \left[W h_i \parallel W h_j\right]\right)$$

Where a is a learned attention vector, W is the weight matrix, and $\parallel$ denotes concatenation of the feature vectors of h_i and h_j.

The above generated results to be normalized across all neighbors of v_i using a softmax function to ensure that the attention coefficients sum to one

$$\alpha_{ij} = \frac{\exp(e_{ij})}{\sum_{k \in \mathcal{N}(v_i) \cup \{i\}} \exp(e_{ik})}$$

2.1.3 Multi-Head Attention

To increase the model's predictive power, GAT deploys multi-head attention. Multiple attention heads allow the model to learn diverse types of relationships from the graph

$$h'_i = \|_{k=1}^{K} \sigma \left(\sum_{j \in \mathcal{N}(v_i) \cup \{i\}} \alpha_{ij}^{(k)} W^{(k)} h_j \right)$$

Where: - K is the number of attention heads. - $\alpha_{ij}^{(k)}$ and $W^{(k)}$ are the attention scores and weight matrices for the k-th attention head.

The outputs from all attention heads are concatenated (denoted by $\|$) before passing it to the next layer.

2.1.4 Node-Level Aggregation

After the attention coefficients are calculated and the hidden states updated, each node has an updated feature vector h'_i which incorporates information from its neighbors, weighted by attention.

2.1.5 Inductive Learning

GAT models have the ability to generalize to unseen graphs and nodes that were not present in the training phase, which is particularly useful for dynamic or evolving graph data.

The attention mechanism gives GATs flexibility to learn in parallel and also allow for faster computation, and due to inductive learning this can be applied for a wide range f graph structures.

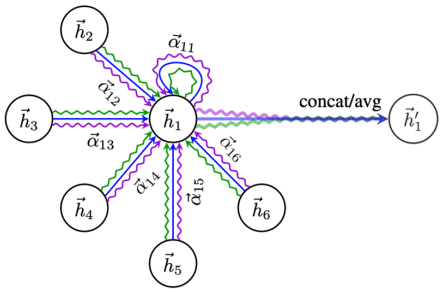


Figure 2.1: Attention on Graph node [20]

2.2 Data Representation

The cells are represented as nodes in the graph, the *CellProfiler*[34] collects parameters such as cell length, orientation, cell position as mentioned in 1.1, they are the node features which is a fixed size representation for every cell. Cell division alters the number of cells at every time step, enabling the graph to grow dynamically. Hence the cells present at every time step are treated separately (they have a new id), whether or not they divide.

The edges of the graph are of two types contact and lineage. Lineage edges are treated as directed, as message is passed from parent to daughter and not vice versa, because the cells divide and don't join back together for the message to pass back. Contact edges are undirected edges as message can pass to and fro from neighbor and the original cell. Incorporating both the edge types in a single graph leads to a multi-graph representation. Combining both of the edges in a real simulation data as shown in *Figure* 2.3

2.3 Simulations

With the aid of the *Cellmodeller* 100 parameter combinations (i.e., choice of alpha and beta values) were chosen and for each parameter 10 simulations were generated, totaling to 1000 simulations.

2.4 Data Sampling and Triplet loss

Our goal was to learn meaningful representations from simulations in a supervised fashion. To learn the message passing effectively we used triplet loss training strategy, which involves

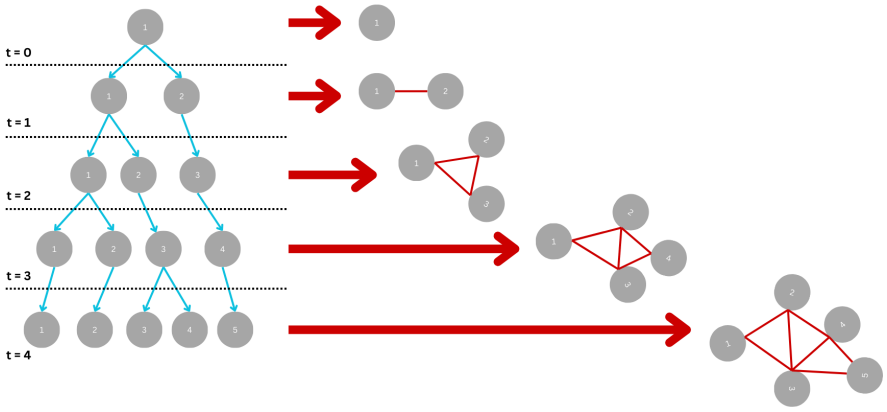


Figure 2.2: Cell Lineage and Neighbor

The left side shows how the cell divide, notice how every cell doesn't divide in each time step. Every time step each node will have different id's. The right shows the layout of the cell in that time step along with contact information of the cells.

constructing triplets. Each parameter set is referred to as a class.

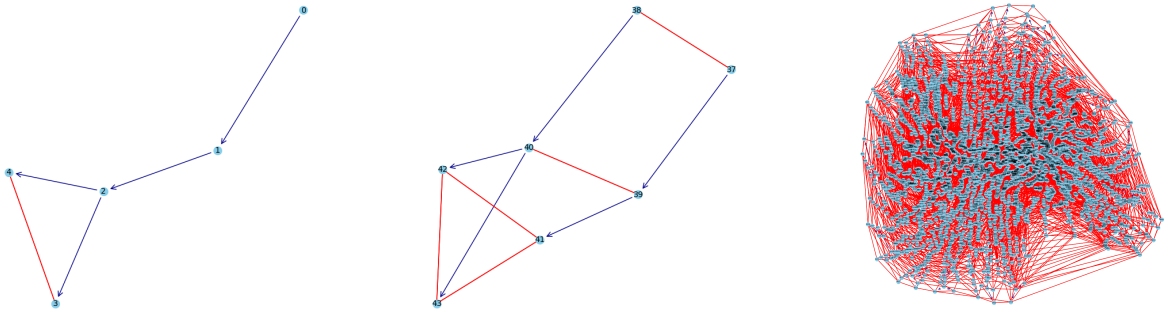
This Algorithm 2 [2] describes the process for constructing triplets for supervised learning with a triplet loss function. The triplets consist of anchor samples, positive samples (same class), and negative samples (different class).

This algorithm creates a dataset of triplets:

- **Anchor ($\mathbf{X_{anc}}$)**: A sample selected from a specific class.
- **Positive ($\mathbf{X_{pos}}$)**: Another sample from the same class as the anchor.
- **Negative ($\mathbf{X_{neg}}$)**: A sample from a different class.

Triplets are sampled until the desired number, N, is reached, as required for training a triplet loss-based neural network. The triplet set is passed to a distance function that identifies the distance of the encoding positive, and negative samples with respective to anchor.

In this report a mixed triplet loss was performed, which combines both the standard triplet loss (Eucledian distance) and cosine-based loss into a single loss function.



(a) (a) t = 0 to t = 3 (b) (b) t = 20 to t = 22 (c) (c) t = 0 to t = 85

Figure 2.3: Graph representation of colony for various intervals. Red lines are contact edges and blue arrows are directed lineage edges

$$\mathcal{L}_{\text{MixedTriplet}} = \mathcal{L}_{\text{Triplet}} + \alpha \cdot \mathcal{L}_{\text{TripletCosine}}$$

- Triplet Loss (Euclidean)

$$\mathcal{L}_{\text{Triplet}} = \frac{1}{N} \sum_{i=1}^{N} \max\left(0, \|\mathbf{a}_i - \mathbf{p}_i\|_2 - \|\mathbf{a}_i - \mathbf{n}_i\|_2 + \text{margin}_1\right)$$

  - $\mathbf{a}_i$: Anchor embedding.
  - $\mathbf{p}_i$: Positive embedding.
  - $\mathbf{n}_i$: Negative embedding.
  - margin_1 : Margin for Euclidean triplet loss.

- Triplet Loss (Cosine)

$$\mathcal{L}_{\text{TripletCosine}} = \frac{1}{N} \sum_{i=1}^{N} \max\left(0, (1 - \cos(\mathbf{a}_i, \mathbf{p}_i)) - (1 - \cos(\mathbf{a}_i, \mathbf{n}_i)) + \text{margin}_2\right)$$

  - $\cos(\mathbf{x}, \mathbf{y}) = \dfrac{\mathbf{x} \cdot \mathbf{y}}{\|\mathbf{x}\|\|\mathbf{y}\|}$: Cosine similarity.
  - margin_2 : Margin for cosine triplet loss.

Algorithm 2 Algorithm to Sample Triplets $X_{\text{pos}}, X_{\text{anc}}, X_{\text{neg}}$ from a Dataset

Require: Dataset D, Number of triplets N

$n \leftarrow 0$
while $n < N$ **do**
  Randomly select class indices i and l from D such that $i \neq l$
  Randomly select an anchor example index j from $D[i]$
  Randomly select a positive example index k such that $k \neq j$ from $D[i]$
  Randomly select a negative example index m from $D[l]$
  $X_{\text{anc}}[n] \leftarrow D[i][j]$
  $X_{\text{pos}}[n] \leftarrow D[i][k]$
  $X_{\text{neg}}[n] \leftarrow D[l][m]$
  $n \leftarrow n + 1$
end while
return $X_{\text{pos}}, X_{\text{anc}}, X_{\text{neg}}$

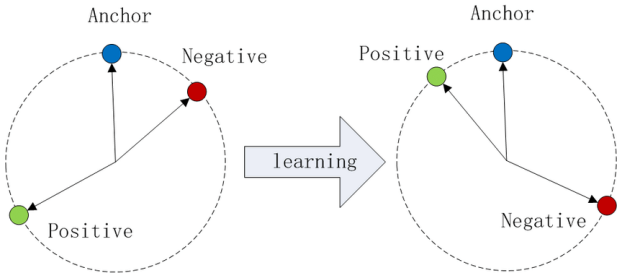


Figure 2.4: Visualization of triplet loss [23]

- Mixed Loss Combination

$$\mathcal{L}_{\text{MixedTriplet}} = \mathcal{L}_{\text{Triplet}} + \alpha \cdot \mathcal{L}_{\text{TripletCosine}}$$

  – α : Weight for the cosine loss component.

The triplet loss training will ensure that simulation with similar parameter distribution (Anchor and positive) will have similar feature embedding compared to the dissimilar one (Anchor and negative). This is done by minimizing the distance between anchor and positive class and maximizing the distance between anchor and negative class as shown in *Figure* 2.4.

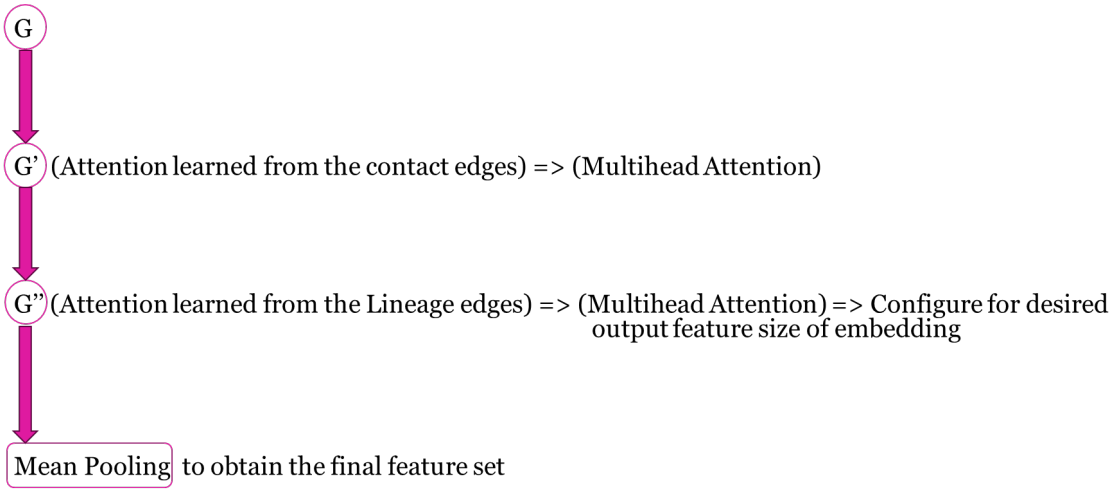


Figure 2.5: Simple GAT

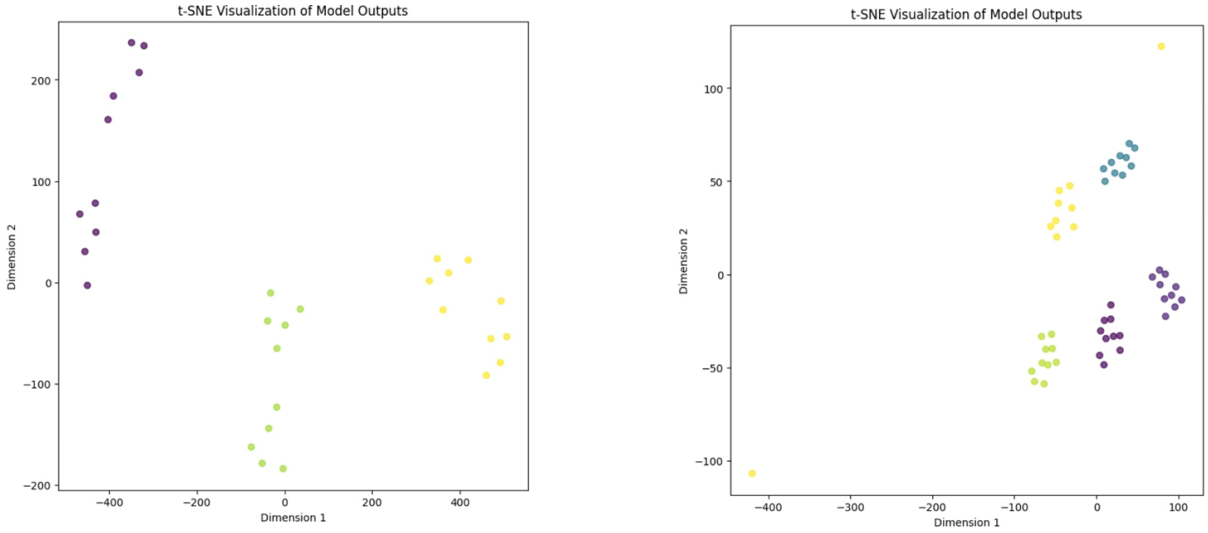
2.5 Model Architecture

The model is an encoder Cells2Vec, that gives us an embedding of desired size, which is made configurable as a hyper parameter. The model first learns the message passing from the contact edges, and then learns the message passing from lineage edges. The order is interchangeable. The message passing is referred to as attention mechanism in graph attention networks. During the message passing the graph goes through multi-head attention, the number of heads are configurable, but currently the performance was tested for 16 heads. The multi head attention is captured for both edges types separately and the resultant is summarized by Global mean pooling.

2.6 Results

Here we present the assessment of the learned representations. We used Python 3 and Pytorch for the implementation. To assess the learned embeddings, the higher dimensional embedding is projected to a 2 dimensional embedding using TSNE and they are visualized to assess the groupings. The encoder was trained using Adam optimizer with the obtained model parameters. And, the model was run for 50 epochs. The *Figure* 2.7 shows the training loss obtained. We generated 4500 triplets, out of which 70% went for training and 15% went for test and 15% for validation. All the three sets had equivalent performance

of 93% accuracy. The model was able to predict well with 1 feature embedding as well as 40 feature embedding. But the amount of information learned seemed to be better with higher feature embedding size as the number of epochs increased. The training loss although slowed down after 30 epochs, didn't reach a saturation or a turning point, it showed the ability to learn even after 50 epochs, but this is yet to be explored in the future. The *Figure* 2.6 shows the combined representation of 3 and 5 parameter groups respectively.



(a) (a) TSNE with 3 parameter groupings

(b) (b) TSNE with 5 parameter groupings

Figure 2.6: Comparison of TSNE visualizations with different parameter groupings.

2.7 Conclusion

In this report we presented a pipeline to learn embeddings from simulations. The encoder architecture uses graph convolutions, more specifically graph attention to capture the message passing between nodes and arrive at a desirable feature set. The nature of these simulation entails complex dynamics of a bacterial micro colony, capturing cell shoving, rigidity and other features etc.,

We learned that this feature embedding, generated using a triplet loss training maximized the distance between the anchor and negative parameter combination and the per-

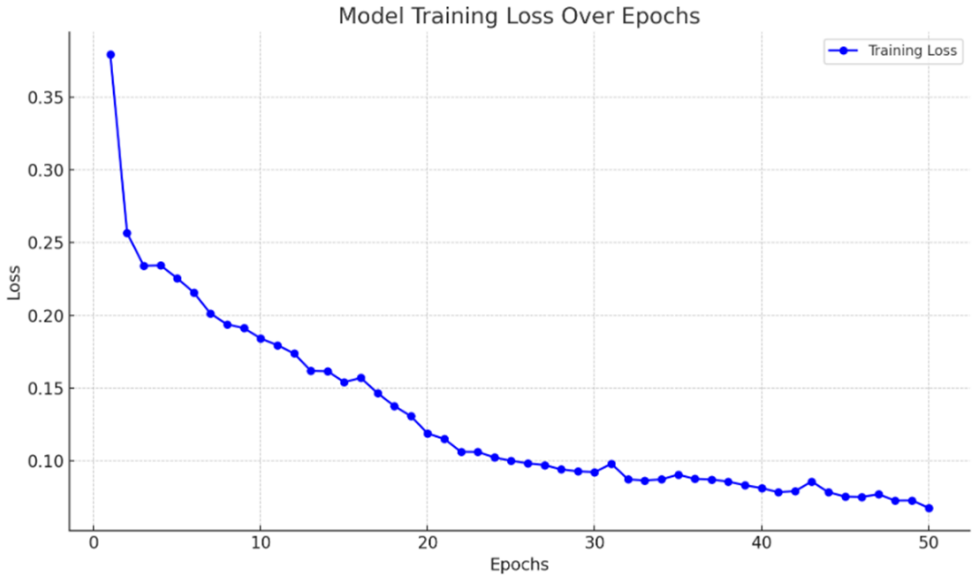


Figure 2.7: Loss over Epochs trained

formance was validated using the training and test data set and the groupings where visualized and they exhibited a satisfactory prformance.

2.8 Future Works

The model shows that representation can be learned effectively using the graph structure in a short duration. However, the hyper parameters are need to be tuned for, the epochs haven't reached a turning point yet, and the number of attention heads haven't yet been explored. And the optimal feature embedding size also needs to be explored for calibration. Currently a 40 feature embedding with 50 epochs and 16 hidden channels take up to 1.5 hours of computation in 6 core GPU. This also puts a need to measure the computational impact.

References

[1] Arnold Abukari, Jhansi Bharathi Madavarapu, and Edem Bankas. A lightweight algorithm for detecting fake multimedia contents on social media. *Earthline Journal of Mathematical Sciences*, 14:119–132, 11 2023.

[2] Atiyeh Ahmadi, Aaron Yip, Jonathan Chalaturnyk, Marc G. Aucoin, and Brian P. Ingalls. A pipeline for calibrating agent-based models of microbial populations: From image collection to model parameterization. *IFAC-PapersOnLine*, 58(23):1–6, 2024. 10th IFAC Conference on Foundations of Systems Biology in Engineering FOSBE 2024.

[3] American Academy of Microbiology. Microbiology in the 21st century: Where are we and where are we going? Technical report, American Society for Microbiology, Washington, DC, 2004. Report based on a colloquium sponsored by the American Academy of Microbiology held September 5–7, 2003, in Charleston, South Carolina.

[4] Itamar Arel, Derek C Rose, and Thomas P Karnowski. Deep machine learning—a new frontier in artificial intelligence research. *IEEE Computational Intelligence Magazine*, 5(4):13–18, 2010.

[5] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. In *Proceedings of the 3rd International Conference on Learning Representations (ICLR)*, 2014.

[6] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate, 2016.

[7] M. A. Beaumont. Approximate bayesian computation. *Annual Review of Statistics and Its Application*, 6:379–403, 2019.

[8] Anirban Bhowmik and Sunil Karforma. An approach of secret sharing technique based on convolution neural network and dna sequence for data security in wireless communication. *Wireless Personal Communications*, 124:1–32, 06 2022.

[9] Joan Bruna, Wojciech Zaremba, Arthur Szlam, and Yann LeCun. Spectral networks and locally connected networks on graphs. In *International Conference on Learning Representations (ICLR)*, 2014.

[10] Y. Chen, J. K. Kim, A. J. Hirning, K. Josić, and M. R. Bennett. Emergent genetic oscillations in a synthetic microbial consortium. *Science*, 349(6251):986–989, 2015.

[11] Viorica Maria Corbu, Irina Gheorghe-Barbu, Andreea Ștefania Dumbravă, Corneliu Ovidiu Vrâncianu, and Tatiana Eugenia Șesan. Current insights in fungal importance—a comprehensive review. *Microorganisms*, 11(6), 2023.

[12] G. H. Dibdin. Mathematical modeling of biofilms. *Advances in Dental Research*, 11(1):127–132, 1997.

[13] Mikael Henaff, Joan Bruna, and Yann LeCun. Deep convolutional networks on graph-structured data. *arXiv preprint arXiv:1506.05163*, 2015.

[14] Alison J.J. Heppenstall, Andrew T.T. Crooks, Linda M.M. See, and Michael Batty. Agent-based models of geographical systems. In *Agent-Based Models of Geographical Systems*, pages 141–165. Springer Netherlands, 2012.

[15] Geoffrey E. Hinton, Simon Osindero, and Yee Whye Teh. A fast learning algorithm for deep belief nets. *Neural Computation*, 18(7):1527–1554, 2006.

[16] Sakshi Indolia, Anil Kumar Goswami, S. P. Mishra, and Pooja Asopa. Conceptual understanding of convolutional neural network—a deep learning approach. In *International Conference on Computational Intelligence and Data Science (ICCIDS 2018)*. Elsevier, 2018.

[17] D Barrie Johnson. Biomining—biotechnologies for extracting and recovering metals from ores and waste materials. *Current Opinion in Biotechnology*, 30:24–31, 2014. Chemical biotechnology • Pharmaceutical biotechnology.

[18] Mario Juhas. *Microbial Communities*, pages 43–50. Springer International Publishing, Cham, 2023.

[19] Marc C. Kennedy and Anthony O'Hagan. Bayesian calibration of computer models. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 63(3):425–464, 2001.

[20] Duy Khuu and Marian-Andrei Rizoiu. *Polarisation and Influence: Investigating Brexit Opinion Dynamics on Reddit*. PhD thesis, 11 2021.

[21] Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. In *International Conference on Learning Representations (ICLR)*, 2017.

[22] Yann LeCun, Bernhard Boser, John S. Denker, Donnie Henderson, Richard E. Howard, Wayne Hubbard, and Lawrence D. Jackel. Backpropagation applied to handwritten zip code recognition. *Neural Computation*, 1(4):541–551, 1989.

[23] Chao Li, Xiaokong Ma, Bing Jiang, Xiangang Li, Xuewei Zhang, Xiao Liu, Ying Cao, Ajay Kannan, and Zhenyao Zhu. Deep speaker: An end-to-end neural speaker embedding system. In *Proceedings of the International Conference on Neural Information Processing (ICONIP)*, 2017.

[24] Maria L. Marco. Defining how microorganisms benefit human health. *Microbial Biotechnology*, 14(1):35–40, January 2021. Epub 2020 Oct 25.

[25] I. D. Mienye, T. G. Swart, and G. Obaido. Recurrent neural networks: A comprehensive review of architectures, variants, and applications. *Information*, 15(9):517, 2024.

[26] Idongesit D. Mienye and Thomas G. Swart. A comprehensive review of deep learning: Architectures, recent advances, and applications. *Information*, 15:755, 2024.

[27] K. Nagarajan, C. Ni, and T. Lu. Agent-based modeling of microbial communities. *ACS Synthetic Biology*, 11(11):3564–3574, Nov 2022. Epub 2022 Oct 31.

[28] C. Nebauer. Evaluation of convolutional neural networks for visual recognition. *IEEE Transactions on Neural Networks*, 9(4):685–692, 1998.

[29] Clement Nyirenda and Richard Chilipa. Evolving neuro-fuzzy systems for real-time data processing in internet of medical things, 07 2024.

[30] J. Panovska-Griffiths, C.C. Kerr, W. Waites, and R.M. Stuart. Chapter 10 - mathematical modeling as a tool for policy decision making: Applications to the covid-19

pandemic. In Arni S.R. Srinivasa Rao and C.R. Rao, editors, *Data Science: Theory and Applications*, volume 44 of *Handbook of Statistics*, pages 291–326. Elsevier, 2021.

[31] J. Pathak, S. Miven, M. Girvan, I. Oppenheim, and N. Menick. Residual networks as reservoir networks for data assimilation. *IEEE Transactions on Neural Networks and Learning Systems*, 28(9):2772–2786, 2017.

[32] Dhruva Rajwade, Atiyeh Ahmadi, and Brian Paul Ingalls. Cells2vec: Bridging the gap between experiments and simulations using causal representation learning. In *Causal Representation Learning Workshop at NeurIPS 2023*, 2023.

[33] M. Ramsay, B. Zhang, and A. Petroski. Learning to predict dynamics of time-delayed systems with echo state networks. *Scientific Reports*, 12(1):5691, 2023.

[34] Timothy J. Rudge, Paul J. Steiner, Andrew Phillips, and Jim Haseloff. Computational modeling of synthetic microbial biofilms. *ACS Synthetic Biology*, 1(8):345–352, 2012. PMID: 23651288.

[35] Saarthee. Visual magic using ai: How convolutional neural networks decode images. Medium, 2023. Accessed: December 20, 2024.

[36] Iqbal H. Sarker. Deep learning: A comprehensive overview on techniques, taxonomy, applications and research directions. *SN Comput. Sci.*, 2(6):420, 2021.

[37] Alex Sherstinsky. Fundamentals of recurrent neural network (rnn) and long short-term memory (lstm) network. *Physica D: Nonlinear Phenomena*, 404:132306, March 2020.

[38] E. A. Smirnov, D. M. Timoshenko, and S. N. Andrianov. Comparison of regularization methods for imagenet classification with deep convolutional neural networks. *AASRI Procedia*, 6:89–94, 2014.

[39] Timothy Teo. Agent based modelling. In Timothy Teo, editor, *Handbook of Quantitative Methods for Educational Research*, chapter 12, pages 247–265. Sense Publishers, 2013.

[40] Frederick H. C. Tivive and Abdesselam Bouzerdoum. Efficient training algorithms for a class of shunting inhibitory convolutional neural networks. *IEEE Transactions on Neural Networks*, 16(3):541–556, 2005.

[41] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need, 2023.

[42] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph attention networks, 2018.

[43] O. Wanner and P. Reichert. Mathematical modeling of mixed-culture biofilms. *Biotechnology and Bioengineering*, 49(2):172–184, 1996.

[44] Isaac Ronald Ward, Jack Joyner, Casey Lickfold, Yulan Guo, and Mohammed Bennamoun. A practical tutorial on graph neural networks, 2021.

[45] A. Yip, J. Smith-Roberge, S. H. Khorasani, M. G. Aucoin, and B. P. Ingalls. Calibrating spatiotemporal models of microbial communities to microscopy data: A review. *PLoS Computational Biology*, 18(10):e1010533, Oct 2022.

[46] Y. Zhou, H. Wang, F. Xu, and Y. Q. Jin. Polarimetric sar image classification using deep convolutional neural networks. *IEEE Geoscience and Remote Sensing Letters*, 13(12):1935–1939, 2016.