Geometric Tree Classification by Markov Chain Modeling

Arvind Rathnashyam
RPI CS and Math
rathna@rpi.edu

Radoslav Ivanov
RPI CS
ivanor@rpi.edu

Malik Magdon-Ismail
RPI CS
magdon@cs.rpi.edu

September 5, 2023

Abstract

Geometric Tree Classification is an important problem in data mining and computer vision, with applications in object detection and medical imaging. In this paper, we first show standard CNNs do *not* learn the *structure* of graph images, rather they learn the background objects. Next, we propose a novel method for Geometric Graph Classification based on Markov Chain Modeling. We model the structure of a tree as a Markov Chain in two ways, (i) each bifurcation point in the tree is represented as a node and each branch is represented as an edge, (ii) each branch is represented as a node. We then use the Markov Chain to define a probability distribution over the possible tree structures, which we use to classify the input tree. Our method achieves superior results over several state of the art models in both synthetic and real datasets. Overall, this research provides a promising new direction for geometric tree classification.

1 Introduction

Geometric Trees are commonly used to represent hierarchical structures in many different applications, particularly in biomedical imaging and linguistics. In this work, we will explore geometric tree classification in images which contain tree structures. Our motivating example is a curated tree dataset we have produced. The state of the art methods for object classification in images are deep convolutional neural networks, [6]. Convolutional Networks are translationally invariant, but they are not invariant to rotations in the 3-dimensional space which is then projected back onto to the image plane. Our real and synthetic datasets consist of tree images undergoing various affine transformations and rotations in the 3-dimensional space which are then imaged and can be seen in Figures 1, 2, 3, 4.

To show Convolutional Neural Networks are unable to learn a good representation of the trees, even with a good amount of data, we run a network similar to AlexNet on 5 real trees, each with 120 training images and 30 testing images. The results can be seen in table 1. As we can see in the figures 2 and 4, our synthetic dataset trees have no background. The very poor results of the neural network in this synthetic case supports

Figure 1: Real

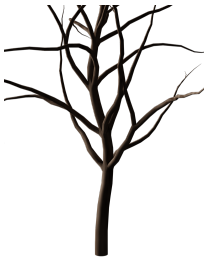


Figure 2: Synthetic

Figure 3: Real

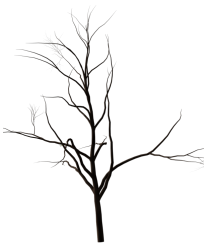


Figure 4: Synthetic

Figure 5: Figures 1 and 3 are the same tree from a different angle and figures 2 and 4 are the same tree from a different angle.

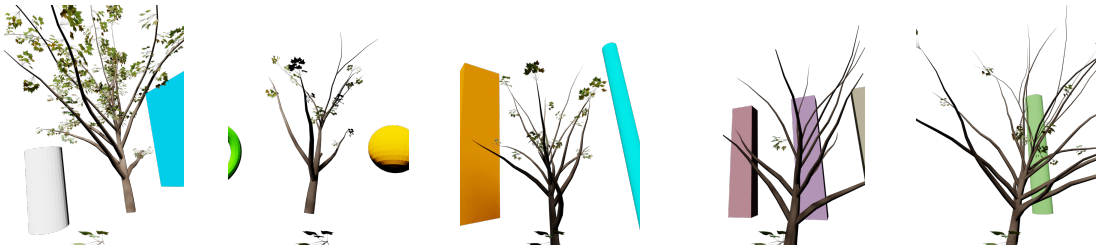


Figure 6: Class 1 Figure 7: Class 2 Figure 8: Class 3 Figure 9: Class 4 Figure 10: Class 5

Figure 11: Tree Images with Random Objects in the Background that are unique to each class of trees. Each class represents the same tree with images taken from different angles and lighting conditions.

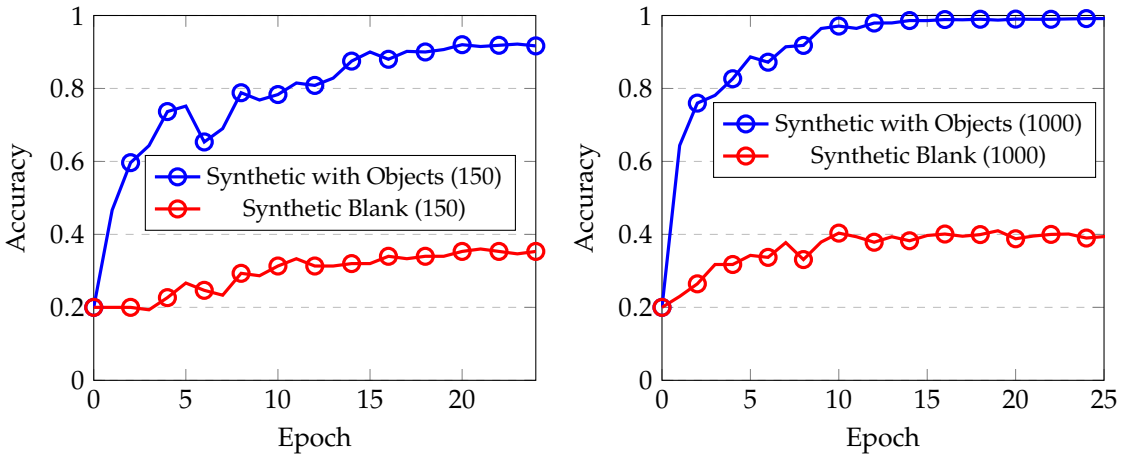


Figure 12: Train Accuracy is given in blue and Test Accuracy is given in red. We train on data with random objects in the background and test on images with no background. This figure shows the increase in data does not overcome the distributional shift of removing background objects.

our informal hypothesis that the CNN is only able to learn an accurate prediction due to it learning features in the background of the real images. To further verify this, we create two datasets of synthetic trees, one where the background is blank, and one where the background has $4-5$ randomly selected and shaped objects. We can see in table 1, the `AlexNet` model learns a significantly better representation when there exists the random objects in the background compared to when the background is blank. Our results can be seen in figure 12

This can be very dangerous in biomedical applications where it is essential to learn a the structure of the graph underlying the image. Our research question is to find a more data-efficient method to learn the tree structure in image classification with higher accuracy.

Our approaches utilize the graph structure of the image, which we have annotated in Figures 15, 16, 17, and 18.

Model	Parameters	Dataset	Classes	Samples	Test Accuracy
`AlexNet`	46,767,493	`Synthetic Blank`	5	150	$85.6_{(4.80)}$
`AlexNet`	46,767,493	`Synthetic with Objects`	5	150	$91.7_{(2.63)}$
`AlexNet`	46,767,493	`Synthetic Blank`	5	1000	$86.8_{(8.07)}$
`AlexNet`	46,767,493	`Synthetic with Objects`	5	1000	$98.9_{(1.01)}$

Table 1: Statistics of the Synthetic and Real Dataset

Many approaches have been made for Tree Graph Classification. Such include the use of Geometric Tree

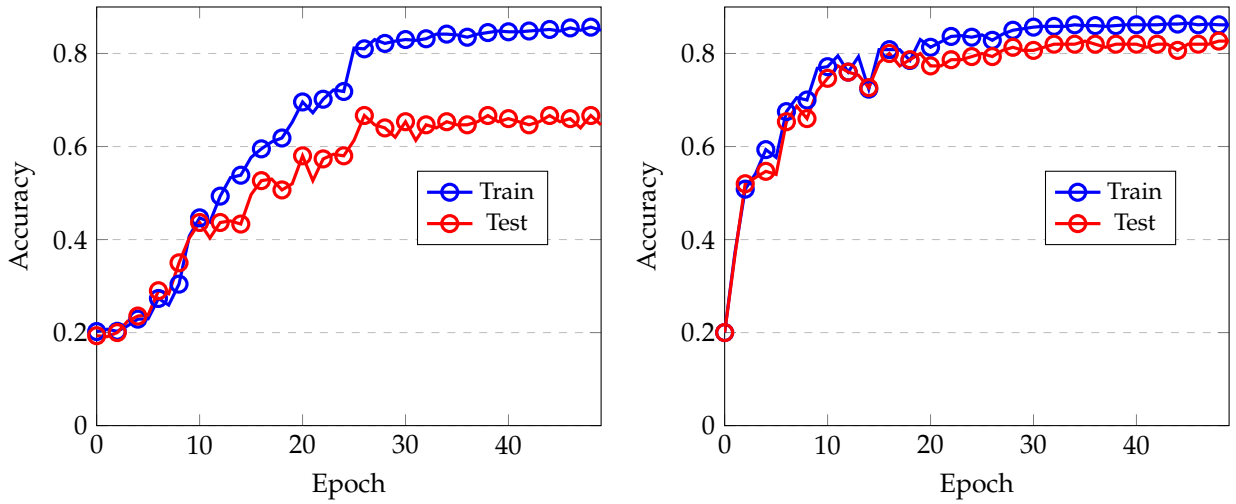


Figure 13: Training and testing results on 750 synthetic images (5 classes, 120 train, 30 test) from synthetically developed trees without background are displayed after training on `AlexNet`. Training and testing results on 750 synthetic images (5 classes, 120 train, 30 test) from synthetically developed trees with random items in the background unique to each class are displayed after training on `AlexNet`.

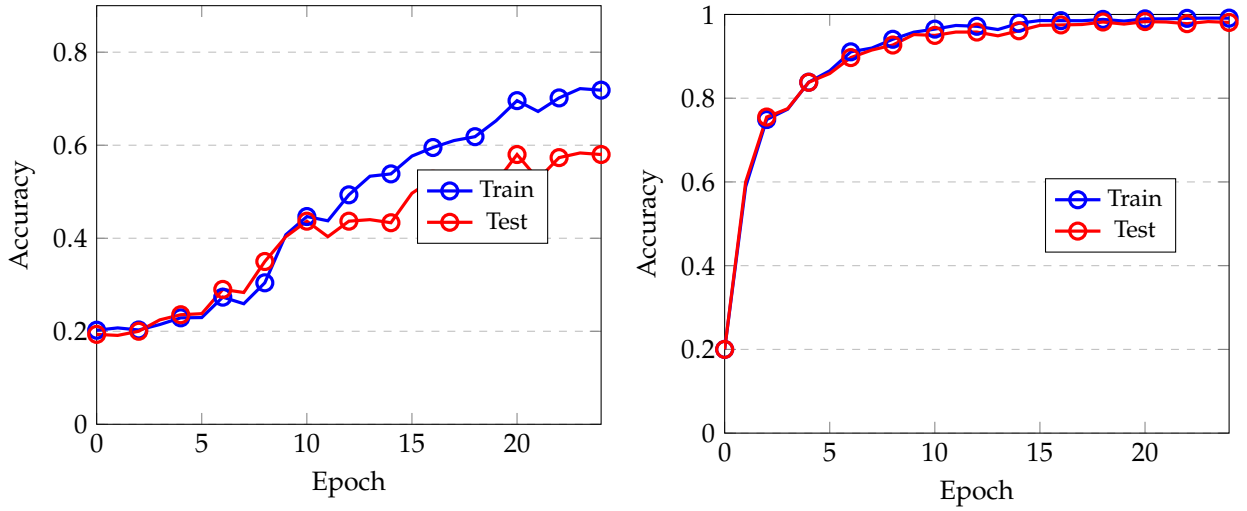
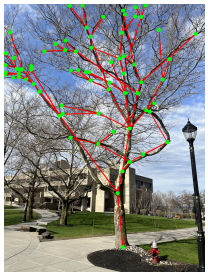


Figure 14: Training and testing results on 1000 synthetic images (5 classes, 800 train, 200 test) from synthetically developed trees without background are displayed after training on `AlexNet`. Training and testing results on 5000 synthetic images (5 classes, 800 train, 200 test) from synthetically developed trees with random items in the background unique to each class are displayed after training on `AlexNet`. As we see in the Background to Background case, more data helps with the training.



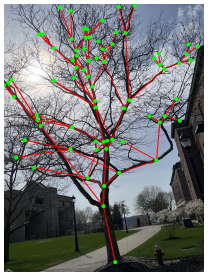
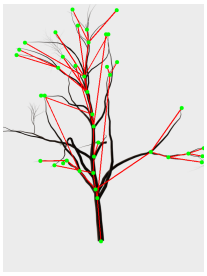



Figure 15: Real Figure 16: Synthetic Figure 17: Real Figure 18: Synthetic

Figure 19: Geometric Tree Graphs overlaying the Tree Images

Distance ([3]), feature-matching approaches ([7]), geometric graph matching ([9],[1]).

Contributions

In this paper, we assume there is a method to obtain the tree structure underlying an image and we propose multiple novel approaches for geometric tree classification by modelling the tree as a Markov Chain. We give two methods that use only the topological features of the graph for classification, and we propose one method that uses both the topological and geometric features of the graph for classification. Our methods perform better than the State of the Art in our geometric tree classification datasets.

2 Related Work

In this section we will describe relevant works with regards to learning geometric graphs and trees.

A family of Geometric Tree Kernels is presented in [4] for the classification of Geometric Trees. The best performing kernels were root-path kernels, which are formulated as the following

$$K_r(T_1, T_2) = \sum_{v_i \in V_1, v_j \in V_2} k_p(p_{ir}, p_{jr}) \tag{1}$$

where $k_p(x_{ij}, x'_{kl}) = \begin{cases} \langle x_{ij}, x'_{kl} \rangle & \text{(linear)} \\ e^{-\lambda \|x_{ij} - x'_{kl}\|_2^2} & \text{(Gaussian)} \end{cases}$

Here the researchers model the difference between root-paths by landmarking a set of points on each path and comparing the difference, either with a linear dot product or with the Gaussian. They then use kernel methods to solve the classification task.

Graph Neural Networks have gained considerable attention in the past few years. However, only a subset of such Graph Neural Networks are capable of Graph Classification in tasks where the Graphs can have different number of nodes and edges.

The Graph Convolutional Network [5] is a semi-supervise approach for Graph Classification. It is based on the principles of the popular Convolutional Neural Networks used typically in images. The layer wise propagation rule is defined as:

$$H^{(l+1)} = \sigma\left(\tilde{D}^{-\frac{1}{2}}\tilde{A}\tilde{D}^{-\frac{1}{2}}H^{(l)}W^{(l)}\right) \tag{2}$$

The convolutional structure is similar to convolutional neural networks for images where in lower layers local features are learned and in later layers more hollistic features about the graph are learned.

Graph Isomorphism Networks,[10] are a powerful GNN which has provably maximal discrimative power among GNNs. In each iteration the GIN updates node representations with the following equation:

$$h_v^{(k)} = \text{MLP}^{(k)}\left(\left(1 + \epsilon^{(k)}\right) \cdot h_v^{(k-1)} + \sum_{u \in \mathcal{N}(v)} h_u^{(k-1)}\right) \tag{3}$$

3 Markov Chain Modeling

In this section we will discuss the background for Markov Chain Modeling and then present our algorithms.

3.1 Markov Chain Preliminaries

A Markov Chain can be represented by a 3-tuple $(\mathcal{S}, \mathcal{P}, s_0)$, [8]. $\mathcal{S}$ represents the state space of the Markov Chain, $\{s_0, s_1, \ldots, s_n\}$. $\mathcal{P}$ represents the probabilistic transition matrix. Each entry $\mathcal{P}_{ij}$ represents the probability of moving to s_j from s_i. s_0 represents the start state of the Markov Chain. The Markov Property states $\mathbb{P}[s_n|s_{n-1}, s_{n-2}, \ldots, s_0] = \mathbb{P}[s_{n-1}]$. All states in the Markov Chain must follow this property.

3.2 Topological Features

Throughout this section, we will represent $\phi(X) : \mathcal{T} \to \mathbb{R}^d$ represent the feature mapping from Tree to vector, $\psi(i) : V \to \mathbb{Z}$ represent the distance in edges from node i to the root, $\varphi : V \to \mathbb{N}_0$ represent the number of outgoing edges from node i. The state space $\mathcal{S}$ represents all two-tuple points of the form (a, b). Given a vertex, v, in a tree T, we determine the state of v as $(\psi(v), \phi(v))$. In practice, we set a maximum on a and b, in our datasets $a \leq 10$ and $b \leq 10$, so our feature vector is not too sparse and to limit the effect of unseen vertices in the testing set.

3.2.1 Markov Chain Likelihood Approach

In our first proposed method, we use only topological features of the tree for the graph. This means we do not utilize the node locations in $\mathbb{R}^2$ for our training or prediction. We will first define the Markov Chain $(\mathcal{S}, \mathcal{P}, s_0)$.

We determine the classification problem as follows:

$$c = \arg\max_{c \in C} \mathbb{P}[C = c|T] \tag{4}$$

under the condition

$$\sum_{c=1}^{m} \mathbb{P}[C = c|T] = 1 \ \ \forall T \in \mathcal{T} \tag{5}$$

This represents the conditional probability a geometric tree T, is in class c of the m total classes. Similar to [4], we identify a tree by it's root-paths. This means a tree T is made by a set of root-paths, and on each path is a set of nodes from the root to a leaf. Thus, the number of root paths in a tree equals the number of leaves. We want to calculate the probability a Tree T is from class c. We first start with Bayes' Theorem.

$$\mathbb{P}[C = c|T] = \frac{\mathbb{P}[T|C = c]\,\mathbb{P}[C = c]}{\mathbb{P}[T]} \tag{6}$$

$$= \frac{\mathbb{P}[T|C = c]\,\mathbb{P}[C = c]}{\mathbb{P}[T|C = c]\,\mathbb{P}[C = c] + \mathbb{P}[T|C \neq c]\,\mathbb{P}[C \neq c]} \tag{7}$$

$$= \frac{\mathbb{P}[T|C = c]\,\mathbb{P}[C = c]}{\mathbb{P}[T|C = c]\,\mathbb{P}[C = c] + \sum_{j=1}^{m} \mathbb{1}_{j \neq c} \mathbb{P}[T|C = j]\,\mathbb{P}[C = j]} \tag{8}$$

$$= \frac{\mathbb{P}[T|C = c]\,\mathbb{P}[C = c]}{\sum_{j=1}^{m} \mathbb{P}[T|C = j]\,\mathbb{P}[C = j]} \tag{9}$$

let $\mathcal{B}(T)$ represent set of branches in tree T

$$= \frac{\left(\prod_{b \in \mathcal{B}(T)} \mathbb{P}[b|C = c]\right) \mathbb{P}[C = c]}{\sum_{j=1}^{m} \left(\left(\prod_{b \in \mathcal{B}(T)} \mathbb{P}[b|C = j]\right) \mathbb{P}[C = j]\right)} \tag{10}$$

Let us here note that a branch is a sequence of states from the root to the branch of the markov chain

$$= \frac{\left(\prod_{b\in\mathcal{B}(T)}\prod_{s_i\in b}\mathcal{P}^{(c)}_{s_{i-1},s_i}\right)\mathbb{P}\left[C=c\right]}{\sum_{j=1}^{m}\left(\left(\prod_{b\in\mathcal{B}(T)}\prod_{s_i\in b}\mathcal{P}^{(c)}_{s_{i-1},s_i}\right)\mathbb{P}\left[C=j\right]\right)} \tag{11}$$

$\mathcal{P}^{(c)}_{s_{i-1},s_i}$ represents the transition probability from state s_{i-1} to s_i in the Markov Chain for the trees in class (c).For numerical stability, we take the log of the product so we are able to convert this calculation into a series of sums.

$$= \frac{\left(\sum_{b\in\mathcal{B}(T)}\sum_{s_i\in b} - \log\left(\mathcal{P}^{(c)}_{s_{i-1},s_i}\right)\right)\mathbb{P}\left[C=c\right]}{\sum_{j=1}^{m}\left(\left(\sum_{b\in\mathcal{B}(T)}\sum_{s_i\in b} - \log\left(\mathcal{P}^{(c)}_{s_{i-1},s_i}\right)\right)\mathbb{P}\left[C=j\right]\right)} \tag{12}$$

When we run inference, we choose the $c \in C$ that minimizes the negative log likelihood in equation 12.

3.2.2 Calculating $\mathcal{P}$

Let $\mathcal{T}^{(k)}$ represent the set of trees in the training data set from class k, and let $N^{(k)}(\cdot)$ represent the number of occurences of reaching $\cdot$ within $\mathcal{T}^{(k)}$.

$$\mathcal{P}^{(k)}_{s_{i-1},s_i} = \frac{N^{(k)}(s_{i-1}\rightarrow s_i)}{N^{(k)}(s_{i-1})} \tag{13}$$

3.2.3 Stationary Distribution Approach

We will now discuss our second approach. This approach also utilizes only the topological features of the Geometric Tree. The stationary distribution $\boldsymbol{\pi}$, is a vector that satisfies the following conditions

$$\sum_{i=1}^{|\mathcal{S}|}\pi_i = 1 \tag{14}$$

$$\boldsymbol{\pi}\mathcal{P} = \boldsymbol{\pi} \tag{15}$$

Let us note the probabilistic transition matrix of a geometric tree is always irreducible, however, this is a periodic Markov Chain. For example, if we are to start at the root, we can only come back to the root on steps $2, 4, 6, \ldots$. This is because for all states we have self-transition probability equal to 0. To justify this mathematically, let $\mathcal{L}$ represent the eigenvalues such that $|\lambda_1| \geq |\lambda_2| \geq \ldots |\lambda_n|$. Since $\mathcal{P}$ has periodicity of 2, we end up with $|\lambda_1| \approx |\lambda_2|$. It thus follows in the power method:

$$\boldsymbol{x}\mathcal{P}^k = \alpha_1\lambda_1^k\left(\boldsymbol{v}_1 + \frac{\alpha_2}{\alpha_1}\left(\frac{\lambda_2}{\lambda_1}\right)^k\boldsymbol{v}_2 + \cdots + \frac{\alpha_m}{\alpha_1}\left(\frac{\lambda_m}{\lambda_1}\right)^k\boldsymbol{v}_m\right) \tag{16}$$

However since, we have $|\lambda_1| \approx |\lambda_2|$ this simplifies to:

$$\boldsymbol{x}\mathcal{P}^k = \alpha_1\lambda_1\boldsymbol{v}_1 + \alpha_2\lambda_2\boldsymbol{v}_2 \tag{17}$$

We thus need to create an aperiodic Probabilistic Matrix. To create an aperiodic Probabilistic Matrix, we do the following adjustment:

$$\hat{\mathcal{P}} = (1-\varepsilon)\mathcal{P} + \varepsilon\mathbf{R} \text{ where } \mathbf{R} = \begin{pmatrix} 1 & 0 & \ldots & 0 \\ 1 & 0 & \ldots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 1 & 0 & \ldots & 0 \end{pmatrix} \text{ and } \varepsilon \in [0,1] \tag{18}$$

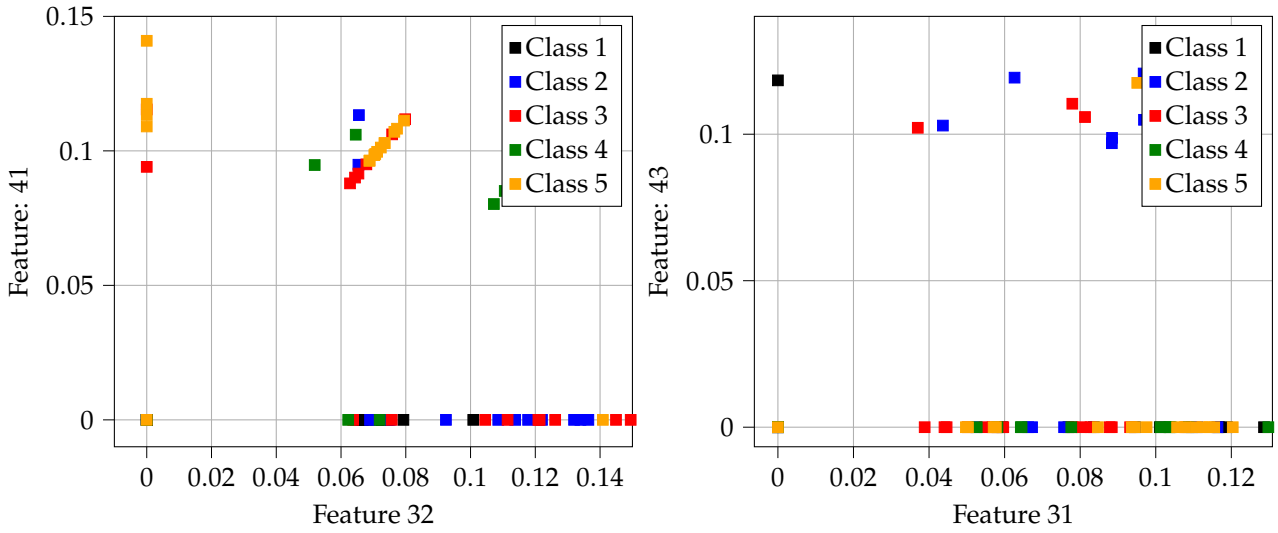


Figure 20: Graphs of 4 Highest Variance Features in Stationary Distribution Method

We call ε the return probability. From any node in the tree, with probability ϵ, we can return back to the root of the tree. With higher ϵ, we will be less likely to explore areas of the tree from the root. Mathematically, this works fine as both $\mathcal{P}$ and $\mathbf{R}$ have only non-negative entries. We can now find the stationary distribution of $\hat{\mathcal{P}}$. We do this by calculating the eigenvector that is associated with the 1 eigenvalue. Let this be $\hat{\boldsymbol{v}}$, we then map this feature to the state space described in the beginning of § 3.2. This is done by a sum-aggregation. In other words, for each vertex, v, in the tree we obtain it's state representation $(\psi(v), \phi(v))$, and the value assigned to the state $(\psi(v), \phi(v))$ is equal to the summation of all probabilities of vertices with the same state representation. We then use this feature as an input to an SVM. Our results can be seen in Table 3 and 4.

To impose a better inductive bias on the task of classification of the geomtric tree, we introduce the idea of a return probability. Let ε determine the return probability. At each state in the graph, we have a probability of ε chance of going back to the root. We want to see if this has any effect on the accuracy and if it is useful for the classification. In figure 21, we see the at low return probabilities, the accuracy is maximized. As ε is increased, the nodes with greatest distance from the node are explored less and less, thus leading the stationary vector to be less descriptive of the tree graph.

The stationary vectors are high dimensional vectors, so it is not possible to visualise all dimensions. To see the effectiveness of this feature, we graph the entries in the vector with the highest variance. The graphs can be seen in Figure 20. We can see the data is not seperable with just the few features, it requires many more features.

3.3 Geometric Features

In our third proposed method, we utilize the locations of the nodes in $\mathbb{R}^2$ in conjunction with the Markov Chains for classification.

We will define a new Markov Chain $(\mathcal{S}, \mathcal{P}, s_0)$, where $s \in \mathcal{S}$ represents the branches of the tree.

Definition 1. *The **Hitting Distance** of a Geometric Markov Chain is defined as:*

$$D_A = \inf\{n \in \mathbb{R}^+ : \sum_{i=1}^{m} d(s_{i-1}, s_i) = n\} \tag{19}$$

where d represents a metric in the $\mathbb{R}^2$ space.

Definition 2. *The **Expected Hitting Distance** of a Geometric Markov Chain defined by the tuple $(\mathcal{S}, \mathcal{P}, s_0)$ is defined as follows:*

$$\mathbb{E}[D_{iA}] = \mathbb{E}[D_A | s_0 = i] \tag{20}$$

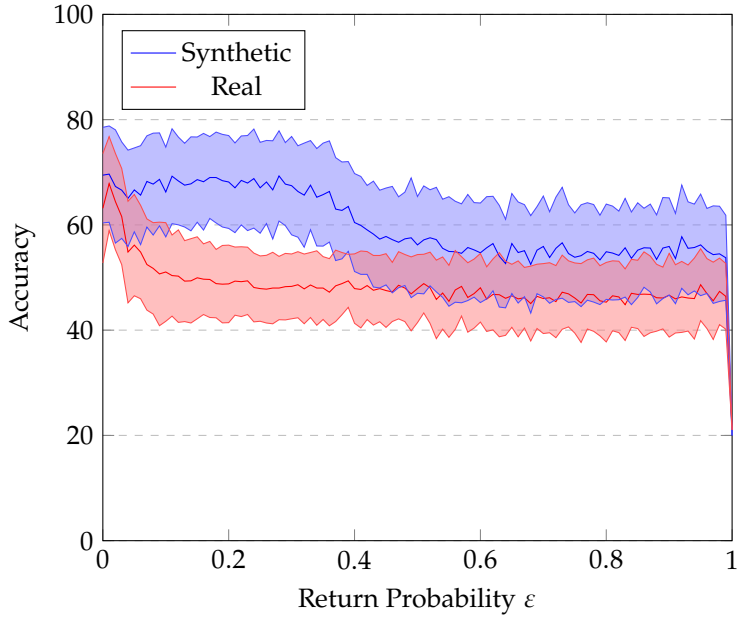


Figure 21: Effect of Return Probability on the accuracy

The Hitting Distance defined in equation 19 can be solved with the following set of equations which are based on the law of total probability.

$$D_{iA} = \begin{cases} \sum_{j \in \mathcal{S}} \mathcal{P}_{ij} D_{jA} & \text{if } i \neq A \\ 0 & \text{if } i = A \end{cases} \tag{21}$$

Similarly the Expected Hitting Distance defined in equation 20 can be solved with the following set of equations:

$$\mathbb{E}\left[D_{iA}\right] = \begin{cases} \sum_{j \in \mathcal{S}} \mathcal{P}_{ij} \left(d(i,j) + \mathbb{E}\left[D_{jA}\right]\right) & \text{if } i \neq A \\ 0 & \text{if } i = A \end{cases} \tag{22}$$

However, this system of equations grows exponentially with respect to the number of nodes. Thus we propose a novel approach to find the expected hitting distance of each node from the root.

To solve for the expected distance from the root to vertex A. Let Q denote all branches that can be reached from the root without having to go through A, and let R denote all branches that can only be reached from the root by going through A. Let us reorganize the probabilistic transition matrix into the following.

$$\mathcal{P}_{\mathcal{B}} = \begin{pmatrix} \mathbf{Q} & \mathbf{R} \\ \mathbf{0} & \mathbf{I}_r \end{pmatrix} \tag{23}$$

To calculate the Expected Hitting Distance, we want to know the expected distance before we reach an absorptive state. Note by the formulation of the transient and absorptive states, the first absorptive state we reach is A. Thus we solve for the fundamental matrix:

$$N = \sum_{k=0}^{\infty} \mathbf{Q}^k = (\mathbf{I} - \mathbf{Q})^{-1} \tag{24}$$

Then the expected distance becomes:

$$\mathbf{D_A} = (\mathbf{I} - \mathbf{Q})^{-1}\mathbf{D} \quad \text{where} \quad \mathbf{D} = \begin{pmatrix} 0 & \ell(b_1) & \ell(b_2) & \dots & \ell(b_n) \end{pmatrix} \tag{25}$$

where $\ell(b)$ represents the length of branch b.

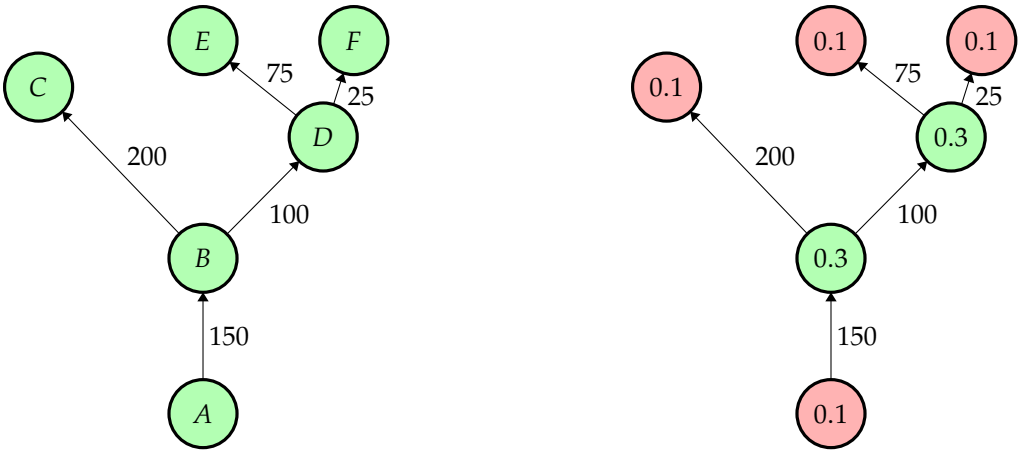


Figure 22: Stationary Distribution of Toy Example

3.3.1 Calculating Expected Hitting Distance to all nodes

Our proposed method which uses Geometric Features calculates the Expected Hitting Distance from the Root to all other Nodes in the Geometric Tree Graph. Since we want a normalized way to compare Geometric Trees, for each $s \in \mathcal{S}$ we average over the expected distances in the original tree.

We will now define the new Markov Chain $(\mathcal{S}_{\mathcal{B}}, \mathcal{P}_{\mathcal{B}}, s_0)$. The s_0 remains the same as in the original. All $b = (s_i, s_{i+1}) \in \mathcal{S}_{\mathcal{B}}$ such that $(s_i, s_{i+1}) \in E$, where E represents the edges of the orginal graph T.

The matrices for the toy example in figure 22 are as follows:

$$\begin{array}{c} \\ A \\ B \\ C \\ D \\ E \\ F \end{array}\!\!\begin{array}{c} \begin{array}{cccccc} A & B & C & D & E & F \end{array} \\ \begin{pmatrix} 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \end{pmatrix} \end{array} \rightarrow \begin{array}{c} \\ A \\ B \\ C \\ D \\ E \\ F \end{array}\!\!\begin{array}{c} \begin{array}{cccccc} A & B & C & D & E & F \end{array} \\ \begin{pmatrix} 0 & 1 & 0 & 0 & 0 & 0 \\ 1/3 & 0 & 1/3 & 1/3 & 0 & 0 \\ 0 & 1/3 & 0 & 0 & 0 & 0 \\ 0 & 1/3 & 0 & 0 & 1/3 & 1/3 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \end{pmatrix} \end{array} \rightarrow \begin{array}{c} \\ A \\ B \\ C \\ D \\ E \\ F \end{array}\!\!\begin{array}{c} \\ \begin{pmatrix} 0.1 \\ 0.3 \\ 0.1 \\ 0.3 \\ 0.1 \\ 0.1 \end{pmatrix} \end{array}$$

If we then consider the state space with $\mathcal{S}$ with maximum outgoing degree 2 and maximum distance from root 3. The feature vector for the toy example in figure 22 is then:

$$\phi(X) = \Big(\underbrace{0 \quad 0.1 \quad 0}_{\psi(i)=0} \quad \underbrace{0 \quad 0 \quad 0.3}_{\psi(i)=1} \quad \underbrace{0.1 \quad 0 \quad 0.3}_{\psi(i)=2} \quad \underbrace{0.2 \quad 0 \quad 0}_{\psi(i)=3} \Big)$$

$$\mathbf{B} = \begin{array}{c} \\ A \\ AB \\ BA \\ BC \\ CB \\ BD \\ DB \\ DE \\ ED \\ DF \\ FD \end{array}\!\!\begin{array}{c} \begin{array}{cccccccccccc} A & AB & BA & BC & CB & BD & DB & DE & ED & DF & FD \end{array} \\ \begin{pmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 0 \end{pmatrix} \end{array}$$

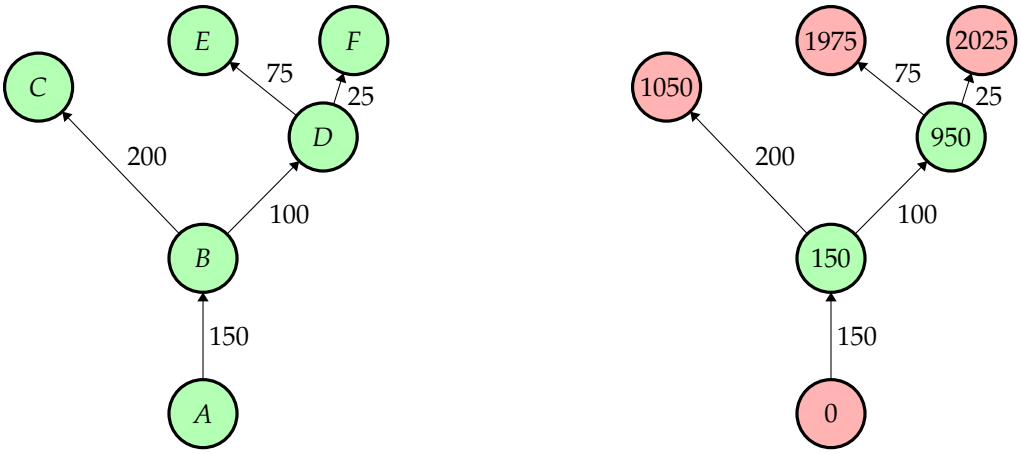


Figure 23: Expected Distance Representation of Toy Example

If we then consider the state space with $\mathcal{S}$ with maximum outgoing degree 2 and maximum distance from root 3. The feature vector for the toy example in figure 22 is then:

$$\phi(X) = \big(\underbrace{0 \quad 0 \quad 0}_{\psi(i)=0} \quad \underbrace{0 \quad 0 \quad 150}_{\psi(i)=1} \quad \underbrace{1050 \quad 0 \quad 950}_{\psi(i)=2} \quad \underbrace{4000 \quad 0 \quad 0}_{\psi(i)=3} \big)$$

We then use $\phi(X)$ as input to a RBF kernel SVM.

4 Empirical Evaluation

Here we test our methods against two different datasets. In table 2, we list different statistics of each dataset.

Dataset	Number of Graphs	Classes	Avg. Nodes	Avg. Edges
`Synthetic`	100	5	38.29	39.29
`Real`	100	5	55.84	56.84

Table 2: Statistics of the Synthetic and Real Dataset

4.1 Synthetic Data

In this section we test our methods against the current state of the art methods in Geometric Tree Classification with our synthetically created dataset.

4.2 Real Data

In this section we test our methods against the current state of the art methods in Geometric Tree Classification with our dataset formulated from real world data.

4.3 Experimental Details

Here we will describe the details of the baseline methods and current state of the art for Graph Classification.

With the limited data, the neural networks, despite having a large amount of parameters, are unable to find good representations of the data that generalize well to the testing set.

Method	Accuracy
Markov Chain Maximum Likelihood	**70.33**$_{(\mathbf{9.304})}$
Stationary ('rbf', $p = 0.01$)	$\underline{69.65}_{(9.145)}$
Geometric Expected Hitting Distance ('rbf')	$56.00_{(7.461)}$
Roothpath, Gaussian [4]	$15.5_{(0.212)}$
Graph Convolutional Network [5]	$29.00_{(8.699)}$
Graph Isomorphism Network [10]	$24.67_{(8.055)}$
Vertex Histogram Kernel	$42.67_{(10.71)}$
Spectral Baseline Method [2]	$20.07_{(3.392)}$

Table 3: Accuracy and Computational Time for our Synthetic Dataset

Method	Accuracy
Markov Chain Maximum Likelihood	$58.77_{(1.026)}$
Stationary ('rbf')	**68.84**$_{(\mathbf{9.667})}$
Geometric Expected Hitting Distance ('rbf')	$\underline{62.81}_{(8.26)}$
Roothpath, Gaussian [4]	$13.68_{(5.367)}$
Graph Convolutional Network [5]	$34.39_{(10.677)}$
Graph Isomorphism Network [10]	$26.49_{(9.064)}$
Vertex Histogram Kernel	$35.96_{(10.141)}$
Spectral Baseline Method [2]	$21.05_{(4.611)}$

Table 4: Accuracy and Computational Time for our Real Dataset

5 Discussion and Further Research

We find our 3 Markov Chain Modeling Techniques as superior to other state of the art methods in graph classification. Typically, in images the size of the tree will not be very large, so we believe our methods can be used for a wide variety of applications in biomedical imaging. In general, it the stationary distribution method is very fast to run as for each tree it only needs to calculate the dominant eigenvector. The Expected Distance approach, although a strong approach, requires a calculation of the fundamental matrix for each subtree in the tree, so in the case of large trees it will not scale as well as the Stationary Distribution Approach.

In this paper, we introduce Markov Chain Modeling of Geometric Tree Graphs. We test our method against various baselines and state of the art methods. Thus, we believe modeling the geometric tree as a Markov Chain imposes a good inductive bias in the Geometric Tree Classification Problem and hope this work encourages researchers to see Markov Chain Modeling as a promising new direction in the problem of Geometric Tree Classification.

Our two datasets are small scale, since they have on average only 20 images per class it makes this a very difficult dataset for traditional learning algorithms. As with the limited amount of data, the Markov Chain Model is able to find a statistically significant good representation of the graph.

References

[1] Ayser Armiti and Michael Gertz. Geometric graph matching and similarity: A probabilistic approach. *ACM International Conference Proceeding Series*, 06 2014.

[2] Nathan de Lara and Edouard Pineau. A simple baseline algorithm for graph classification, 2018.

[3] Aasa Feragen. Complexity of computing distances between geometric trees. In Georgy Gimel'farb, Edwin Hancock, Atsushi Imiya, Arjan Kuijper, Mineichi Kudo, Shinichiro Omachi, Terry Windeatt, and Keiji Yamada, editors, *Structural, Syntactic, and Statistical Pattern Recognition*, pages 89–97, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg.

[4] Aasa Feragen, Jens Petersen, Dominik Grimm, Asger Dirksen, Jesper Holst Pedersen, Karsten Borgwardt, and Marleen de Bruijne. Geometric tree kernels: Classification of copd from airway tree geometry. In James C. Gee, Sarang Joshi, Kilian M. Pohl, William M. Wells, and Lilla Zöllei, editors, *Information Processing in Medical Imaging*, pages 171–183, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg.

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

[6] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C.J. Burges, L. Bottou, and K.Q. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 25. Curran Associates, Inc., 2012.

[7] Uddipan Mukherjee and Meenakshisundaram Gopi. Finding Feature Similarities Between Geometric Trees. In John Keyser, Young J. Kim, and Peter Wonka, editors, *Pacific Graphics Short Papers*. The Eurographics Association, 2014.

[8] J. R. Norris. *Markov Chains*. Cambridge Series in Statistical and Probabilistic Mathematics. Cambridge University Press, 1997.

[9] Miguel Amável Pinheiro, Jan Kybic, and Pascal Fua. Geometric graph matching using monte carlo tree search. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 39(11):2171–2185, 2017.

[10] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How powerful are graph neural networks? In *International Conference on Learning Representations*, 2019.

6 Appendix

6.1 Convolutional Neural Network Architectures

Here we provide the architecture used for image classification for the 5 classes in section 1. This is a slightly smaller version of the `AlexNet` model. The reason for reducing the size is because our problem contains significantly less classes than the ImageNet Dataset. We also resize all images from $600 \times 800 \times 3$ to $224 \times 224 \times 3$. Trained with ADAM optimizer, $\alpha = 0.001$, batch size is 16, and trained for 20 epochs.

Table 5: Convolutional Neural Network

Layer	**Filter**	**Depth**	**Stride**	**Padding**	**Activation**
Convolution	11×11	96	$(4, 4)$	$(1, 1)$	ReLU
Max Pooling	3×3	-	2	0	-
Convolution	5×5	256	$(1, 1)$	$(2, 2)$	ReLU
Max Pooling	3×3	-	2	0	-
Convolution	3×3	384	$(1, 1)$	$(1, 1)$	ReLU
Convolution	3×3	384	$(1, 1)$	$(1, 1)$	ReLU
Convolution	3×3	256	$(1, 1)$	$(1, 1)$	ReLU
Dropout	$p = 0.5$	-	-	-	-
Linear	-	2048	-	-	ReLU
Dropout	$p = 0.5$	-	-	-	-
Linear	-	2048	-	-	ReLU
Dropout	$p = 0.5$	-	-	-	-
Linear	-	2048	-	-	ReLU
Softmax	-	5	-	-	-

6.2 Graph Neural Network Architectures

Here we provide the architectures used for the Graph Convolutional Network and the Graph Isomorphism Network for the experiments in sections 4.1 and 4.2.

Table 6: Graph Neural Network Architectures

(a) GCN Network

Layer	**Dim**	**Act**
GCNConv	2×250	ReLU
GCNConv	250×250	ReLU
GCNConv	250×250	ReLU
Global Mean Pool	-	-
Dropout	$p = 0.5$	-
Linear	250×5	-
Log Softmax	5×1	-

(b) GIN Network

Layer	**Dim**	**Act**
Linear	2×250	ReLU
Batch Normalization	-	-
Linear	250×250	ReLU
GINConv	-	-
Global Add Pool	-	-
Linear	250×250	-
Batch Normalization	-	ReLU
Linear	250×250	ReLU
GINConv	-	-
Global Add Pool	-	-
Linear	250×250	-
Batch Normalization	-	ReLU
Linear	250×250	ReLU
GINConv	-	-
Global Add Pool	-	-
Linear	750×750	ReLU
Dropout	$p = 0.5$	-
Linear	750×5	-
Log Softmax	5×1	-

6.3 Geometric Tree Generation Interface

We also provide details on our software implementation of a Geometric Tree Generation Interface. We provide details on the general user cycle.

1. Upload folder of tree images to the software.
2. Select a folder to save graph annotations to.
3. Annotate the Graph
   There are two modes the user can be in while annotating the graph. Either in *parent* mode or not in *parent* mode. If the user is in *parent* mode, then either the user can place a new node which will be a *parent*, or if the user clicks within 5 pixels of another node, then that node will become a parent. After you click on parent mode, you will be taken out of *parent* mode. When you are not in parent mode, any node you click will have an edge from the *parent* node.
   If you want to delete any nodes, click middle-button on the mouse and it will delete the whole subtree of the node within 5 pixels of your click point.
4. Save Graph
   The graph is saved in a pickle file to the save directory the user already chose as an adjacency list. The keys in the adjacency list are the pixel locations of the nodes, and the list is the pixel locations of the neighbor nodes.

The software is free to download at `https://github.com/caped-doshi/tree_identification`