An Approach to Generalizing Patterns in Graph Based Engineering Drawings

Cole Dewis
University of Alberta
dewis@ualberta.ca

Jasper Eng
University of Alberta
jleng1@ualberta.ca

Chirooth Girigowda
University of Alberta
girigowd@ualberta.ca

Danielle Guloien
University of Alberta
dguloien@ualberta.ca

Jade Pana
University of Alberta
jpana@ualberta.ca

Amogh Panhale
University of Alberta
panhale@ualberta.ca

Aaryan Singh
University of Alberta
ajsingh@ualberta.ca

Agrim Sood
University of Alberta
agrim@ualberta.ca

Kevin Tonkich
University of Alberta
tonkich@ualberta.ca

Abstract

Understanding engineering drawings is essential for efficient project analysis and streamlining the construction workflow. This study explores the identification and abstraction of recurring engineering symbols and their patterns across multiple sets of engineering drawings using machine learning. The goal is to reduce the reliance on brittle, heuristic-based methods by leveraging the use of models which are capable of generalizing across inconsistent formats and a variety of symbol representations. Our proposed solution involves a learning approach that avoids the need for a large, standardized dataset. In turn, this allows us to generalize the aforementioned patterns with minimal labeled data which can be easily fine-tuned on a per-project basis. The results show that several learning-based methods can be successfully applied to this task, while requiring less fine-tuning to handle variations within pattern classes.

1 Introduction

Engineering drawings serve as essential communication tools used by engineers and construction teams to communicate complex building plans. A common use case involves looking through all project-related drawings to identify the occurrence of objects of interest, such as pipes or valves. We refer to these objects in the drawings as *patterns*. A pattern in an engineering drawing consists of two or more pieces of text that are spatially related; and when viewed together, they convey a meaning that cannot be expressed by a single text element. In large-scale projects, hundreds of such drawings may exist, resulting in thousands of patterns. Manually reviewing all of these pages is tedious and time-consuming. Furthermore, these drawings may come from various engineering firms for different projects. There is no standardized format for drawing these patterns. This causes inter-class variations in patterns, making pattern identification more challenging.

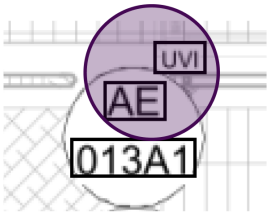


Fig. 1: Example of Pattern Highlighted in Purple

Current proprietary methods designed by PCL for automating pattern extraction rely on heuristic rules to isolate patterns within a drawing.

Preprint. Under review.

However, these heuristics must be fine tuned by a human for each pattern. This makes the approach brittle, time-consuming and difficult to scale. We aim to improve upon existing heuristics by leveraging machine learning algorithms that are able to generalize more effectively.

However, learning-based approaches face a major obstacle in this domain: there is no comprehensive labeled dataset that contains all patterns of interest. Furthermore, each new project in which these algorithms will be deployed may contain new patterns that were unseen during training. As a result, we cannot rely on traditional supervised learning techniques that train a model just once on a comprehensive labeled dataset. Instead, we employ lightweight models and a small amount of human-labeled data. This approach enables us to retrain the models for new projects while still making generalization to unlabeled data possible. Using a small labeled data set is critical as a new dataset may need to be created for each project, and the cost of engineer labeled data is high.

1.1 Data

The dataset consists of text elements with their X and Y coordinates and it's source file. In total, it contains 6,914 data points extracted from 20 engineering drawings. This paper focuses exclusively on patterns consisting of two text elements. However, many of the approaches discussed in this paper could be extended to patterns with three or more elements.

As previously mentioned, we have only a limited amount of human-labeled data for training. However, for evaluation, we use a ground truth data set of 13963 labeled patterns, each consisting of two text elements. The ground truth contains three classes: *Instrument*, *Annotation*, and *Junk*. *Instrument* and *Annotation* are the patterns of interest, while *Junk* encompasses any pair of elements that do not constitute a pattern of interest. Each pattern has exactly one true class. Of the 13963 labels, 11809 (84.6%) are class *Junk*, 1765 (12.6%) are *Instrument*, and 389 (2.8%) are *Annotation*. Any pattern not present in the ground truth set is also considered as *Junk*. Therefore, all the true instances of the *Instrument* and *Annotation* classes are assumed to be included in the ground truth dataset. Furthermore, the ordering of text elements within the pattern does not matter, as the only concern is whether they are present in the pattern or not.

2 Background and Related Works

The task of classifying patterns in engineering drawings based on the spatial relationships between text elements is underexplored in the literature. Therefore, we focus on foundational algorithms that have been applied to similar tasks.

2.1 Data Representations

Since we are working with text elements on a page, a simple representation of the elements is through a feature vector. Assuming that text elements can be grouped into candidate patterns, the classification of one pattern remains unaffected by other nearby patterns. Under this assumption, useful representations of the text elements as a feature vector can be obtained, enabling us to use linear neural network architectures for pattern classification. An alternative and more complex representation of our data is as nodes on a computational graph. Graphs are a natural representation for our data, as the graph structure captures valuable spatial information between pieces of text. These graphs allow us to use models such as graph neural networks.

2.2 Graph Neural Networks

If we choose to represent our data as graphs, we can use Graph Neural Networks, a concept introduced by Scarselli et al [15]. Predicting relationships between nodes on a graph can be formulated as edge classification, where each edge's label indicates the type of relationship between its endpoints [1]. A standard approach is to use Graph Convolutional Networks (GCNs) to learn node embeddings. GCNs rely on a form of *neighborhood aggregation*, often called graph convolutions, which aggregate information from each node, it's neighbors, and the connecting edges. This has the following general form [6]:

$$\mathbf{x}_i' = \gamma_\theta(\mathbf{x}_i, \bigoplus_{j \in N(i)} \phi_\theta(\mathbf{x}_i, \mathbf{x}_j, \mathbf{e}_{ij})) \tag{1}$$

where **x** denotes a node feature vector, **e** an edge feature vector, $N(i)$ the neighbors of node i, $\oplus$ an aggregation such as mean, min, or max. The γ_θ and ϕ_θ are differentiable functions, typically neural networks.

2.3 Bayesian Neural Networks and Active Learning

If we instead represent the data as simple feature vectors, we can use Bayesian Neural Networks (BNNs) as shown by Neal et al [13]. These extend Multi Layer Perceptrons (MLPs) by learning probability distributions over model weights instead of fixed point estimates. By learning a distribution over the weights, BNNs can provide more reliable uncertainty estimates when making predictions, as shown by Jospin et al [8]. BNNs are also less prone to overfitting, as shown by Blundell et al. [3]. BNN training uses the Bayes by Backpropagation method [3], which approximates the true underlying distribution of the data $\boldsymbol{p}$ with a simple distribution $\boldsymbol{q}$ by minimizing the Kullback-Leibler (KL) divergence between them:

$$q_\theta(w \mid \mathcal{D}) \approx p(w \mid \mathcal{D}) \tag{2}$$

$$\theta_{\text{opt}} = \arg\min_\theta \text{KL}\left[q_\theta(w \mid \mathcal{D}) \,\|\, p(w \mid \mathcal{D})\right] \tag{3}$$

This approach uses the reparameterization trick introduced by Kingma and Diedrick [9] to ensure differentiability. An important variant of BNNs is the Last-N-Layer-BNNs [8], in which only the last N layers of the network are Bayesian layers. Zeng et el [21] show that this retains uncertainty estimation benefits while reducing computation compared to a full BNN.

Extending this, active learning has emerged as a method that may improve model performance when labeled data is limited. It assumes a large unlabeled dataset and the ability to query an engineer for labels, which is useful as there is limited access to labeled data. Cohn et al. [5] developed a method called *selective sampling*, which tries to ensure that each query requests a label for a data point that provides information to the model that is not available to it given the current training dataset. In this way, we maximize the information gained from every query and continually increase the coverage of our dataset.

2.4 Class Imbalance

Due to the significant class imbalance in our ground truth, the *junk* class contributes disproportionately to the overall loss. Weighted cross entropy, first introduced by Song et al. [16], is one method to mitigate the effects of such imbalance. This technique, an extension of the standard Cross Entropy (CE) loss, re-weights the loss based on each class's frequency by assigning each class c a weight α_c. A method that further improves on weighted CE loss is focal loss presented by Lin et al. [10]:

$$\text{Focal Loss}(p_t, c) = -\alpha_c(1 - p_t)^\gamma \log(p_t) \tag{4}$$

Here, p_t is the predicted probability for the true class, and γ is the focusing parameter. This formulation emphasizes misclassified examples, as when p_t is low, $(1 - p_t)$ becomes high.

3 Methods

Our performance system for this task is broken down in Figure 2. We first extract a data representation through pairwise clustering, discussed in section 2.1, which can be seen as a preprocessing step for our learning algorithms. A classifier can then be trained on a small amount of engineer-labeled data, formatted using our chosen data representation. This is the focus of our learning system and, once obtained, is the final stage of our performance system.

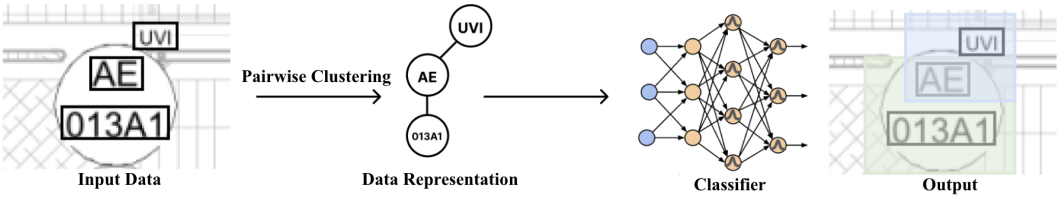


Fig. 2: Diagram of End-to-End Performance System.

3.1 Pairwise Clustering & Graph Creation

We use a simple pairwise clustering approach to generate candidate patterns that are fed into a classifier. This approach selects the top-**n** nearest candidates based on Euclidean distance within a fixed radius **r**. The result is an adjacency list in which each row corresponds to a text element and each column contains 0 to **n** candidate neighbors. From this list, we extract unique sets of pairwise clusters so that each pairwise relationship occurs only once. Note that a single text element can belong to multiple unique pairs. When both pairs representing the same relation between the nodes **(a, b)** and **(b, a)** occur, the first occurrence, **(a, b)**, is stored as a cluster and the second pair is omitted to avoid duplicates. As mentioned above, this does not affect the classification task, since the order of elements in a pattern does not change its class.

Once the final clusters are generated, they are converted into specific formats for classification. The first approach involves forming a linear feature vector. Features can include spatial information such as distances and angles between nodes, as well as features based on the text from each node in the cluster. Alternatively, another approach involves creating a graph representation, in which each pairwise cluster forms an edge, and nodes are the text elements. We then assign feature vectors to nodes based on the text they contain, and edge feature vectors can contain spatial information as above.

3.2 Unsupervised Classification

To begin, we consider unsupervised classification methods. This approach is motivated by the fact that projecting data to higher-dimensional spaces can enhance its separability. Thus, by doing so, we can apply simple clustering algorithms on the latent space to classify the data. We then need to label a small number of data points per cluster to solve the problem.

As a first unsupervised approach, we consider radial basis functions [14]. For this, we randomly sample a set of p centers, $\mathbf{c}$, from our dataset, and then compute a new feature vector $\boldsymbol{\phi}$ as:

$$\boldsymbol{\phi}_j = \exp(-\frac{1}{2\sigma^2}||\mathbf{x} - \mathbf{c}_j||_2^2) \tag{5}$$

where σ is the width of a Gaussian around center $\mathbf{c}_j$. This changes the data to a vector of size p.

Autoencoders offer another approach for increasing dimensionality [11]. We train an autoencoder on our feature vector representation, where the goal is to learn an encoder g to produce $\boldsymbol{\phi}_i = g(\mathbf{x}_i)$ and a decoder f to reproduce $\mathbf{x}_i = f(\boldsymbol{\phi}_i)$. This is done by minimizing the reconstruction error (Appendix A.2) between $f(g(\mathbf{x}))$ and $\mathbf{x}$. Once we have learned g, we get new feature vectors as $\boldsymbol{\phi}_{new} = g(\mathbf{x}_{new})$.

3.3 Feature Vector Classification

Using the candidate pairs of text extracted in section 3.1, we now discuss methods to classify these pairs. We begin with simple approaches to act as baselines before moving on to more complex approaches using Bayesian neural networks and graph neural networks.

3.3.1 Baseline Approaches

To facilitate a better analysis of the results of our more complex models, we consider two baseline methods that use the feature vector representation. The First being a K-Nearest-Neighbors approach [17]. Given a set of labeled training pairs, a new pair is classified as the most common label in its K-nearest-neighbors in the training data. These are determined using a KD-Tree data structure [2].

The second baseline is a two-layer Multi Layer Perceptron (MLP) [12] learned from a training set. This acts as a baseline method for other neural network approaches. To account for the class imbalance in the training dataset, we use a Focal Loss, as discussed in section 2.

3.3.2 Neighborhood Based Matching

The first attempt was to use a simple rule-based approach to identify neighborhoods of spatially related text elements. Each neighborhood is comprised of a set of feature vectors whose similarity exceeds a confidence threshold of 0.85. This approach uses the clusters that were discussed in

section 3.1. It starts with an initial pair that contains a single feature vector randomly sampled from the dataset. Next, the model iterates over every pair in the dataset and calculates its similarity to the centroid of every existing neighborhood. A new neighborhood is created for this data point if this similarity falls below the chosen threshold. However, if the similarity is above the threshold the data point is added to the neighborhood that is most similar to. The centroid for this neighborhood is then recalculated.

After identifying all relevant neighborhoods, a subset of feature vectors is then randomly selected from each one to be labeled by the engineer running the program. The label for each subset is then propagated to its respective source neighborhood, with ties broken by favoring instrument, then annotation. Inference for new vectors is done by assigning the label of the nearest centroid.

3.3.3 Bayesian Neural Networks

Our Bayesian network implementation begins with a multi-layer perceptron that projects the input into a higher dimension, followed by a series of Bayesian layers, using the last-N-layer paradigm discussed in section 2. We use flipout layers for the Bayesian network from Wen et al [20]. These layers allow us to reparameterize the prior distribution of each weight vector in our network independently, reducing variance in the gradient compared to the standard reparameterization trick. These layers use a Gaussian prior, allowing the computation of a KL divergence for Bayes by Backpropagation by comparing the learned and prior distributions. The KL terms are summed during a forward pass and used to compute the Evidence Lower Bound (ELBO) loss during training, formulated as:

$$\text{ELBO} = \int_{\theta} q_{\phi}(\theta) \log \left(\frac{p(\theta, D)}{q_{\phi}(\theta)} \right) d\theta = \log(p(D)) - D_{KL}(q_{\phi}||p) \tag{6}$$

For inference, we take multiple samples from the posterior of the model through Monte Carlo sampling and average their outputs [8]:

$$\hat{y} = \frac{1}{|\Theta|} \sum_{\theta_i \in \Theta} \Phi_{\theta_i}(x) \tag{7}$$

after which we can take the prediction as the class with highest probability.

3.3.4 Active Learning

As mentioned earlier, active learning provides a more intelligent way to decide what data is labeled. This may allow us to be more efficient when training on a limited dataset. We implement a simplified heuristic version of selective sampling [5], where we leverage the fact that BNNs measure the confidence of predictions. We do this by first training the model on a small initial random sample of labeled data. After this training step we evaluate the performance of the model on the entire unlabeled dataset. This gives us a measure of the model's prediction confidence. We select the data points that the model is least confident in predicting, as we believe these provide the most information to the model. We then repeat this process, but now starting with a larger labeled dataset until the number of samples we have queried reaches the maximum number of allowed queries.

Algorithm 1 Active Learning Algorithm

Given an unlabeled dataset, D, a labeled dataset X, and a maximum number of allowed labeled data points, X_{max}
Initialize a Bayesian Classifier Φ
Initialize number of labels per query N
While $|X| < X_{max}$:
    Train Φ on X as a train set
    Get predictions, and confidences: $\hat{Y}, C \leftarrow$ Evaluate(Φ, D)
    Get N least confident predictions: L = LeastConfident(D, C, N)
    Query engineer: q = Query(L)
    $X \leftarrow X \cup q$
return Φ

3.4 Graph Based Classification

Our graph neural network approach performs edge classification with a three-stage neural network. First, we use a residual GCN with edge-conditioned message passing and dropout to compute an embedding for each node. Our GCN uses NNConv layers [7], which aggregate information from neighboring nodes and edges:

$$\mathbf{x}_i' = \mathbf{\Theta}\mathbf{x}_i + \sum_{j \in N(i)} \mathbf{x}_j \cdot \phi_\Theta(\mathbf{e}_{ij}) \tag{8}$$

We chose this layer as it allows edge feature vectors of any size, which is important as spatial information is well suited for edge features.

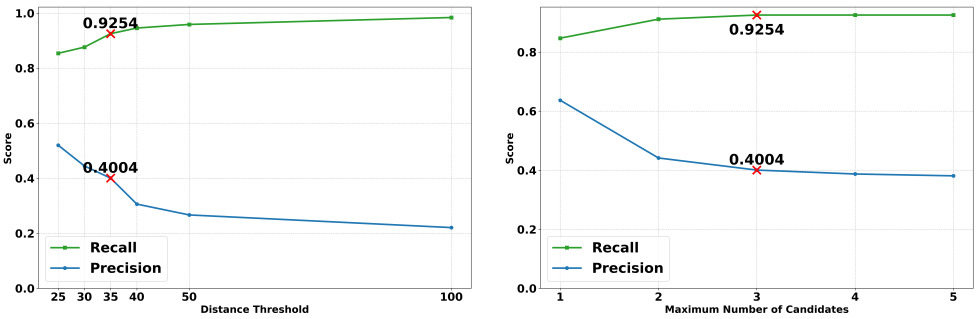
To classify an edge between two nodes, we use the following feature vector. First we use a concatenation of the two node embeddings. We then add a vector from an attention-based global pooling module similar to Touvron et al. [18], which computes a weighted average of node embeddings with learned attention scores. Finally, we add the absolute difference of the pair of node features, and their element-wise product to add initial context for classification. This feature vector is passed through a simple MLP to classify the edge. The full process is summarized in algorithm 2. As with our other classifiers, we utilize focal loss to handle class imbalance.

Algorithm 2 GNN Forward Pass (Single Graph)

Input: Nodes X, Edges E, Edge Attributes A, GCN f, Edge MLP c, Global Pooling Module g
Compute node embeddings: $H \leftarrow f(X, E, A)$
Compute global context: $G \leftarrow g(H)$
For each edge $e = (i, j)$:
    Form edge features: $f_e \leftarrow \text{concat}\Big(H[i],\ H[j],\ |H[i] - H[j]|,\ H[i] \odot H[j],\ G\Big)$
Compute logits: $z \leftarrow c(f_e)$

4 Results

4.1 Pairwise Clustering



(a) Precision and Recall vs Distance Threshold

(b) Precision and Recall vs Max Candidates (Distance=35)

Fig. 3: Pairwise Clustering Performance. Points marked with an X indicate the values for the parameters used for downstream classification, **r**=35 and **n**=3.

In our discussion of the results of our pairwise clustering, we make a distinction between how precision and recall are calculated compared to the classifiers. All calculations are made using only the two classes of interest (*Instrument*, and *Annotation*). Precision is calculated with true positives as the number of pairwise connections that are found and are also present in the ground truth dataset, and false positives as any pairwise connections that are found in the ground truth dataset which are not found.

It is important to recognize that the recall of this step will act as an upper bound for the recall of the classification step. This is because any instrument or annotation pairs in the ground truth that are not generated by this step cannot be classified in later stages, and thus cannot be correctly classified.

The precision at this step is also important, as it relates to the class imbalance of the pairs we have to classify. As precision decreases, the amount of junk labels in our dataset increases, which may harm the training of our classifiers. With this said, the distance threshold should favor high recall while attempting to minimize large class imbalance due to low precision.

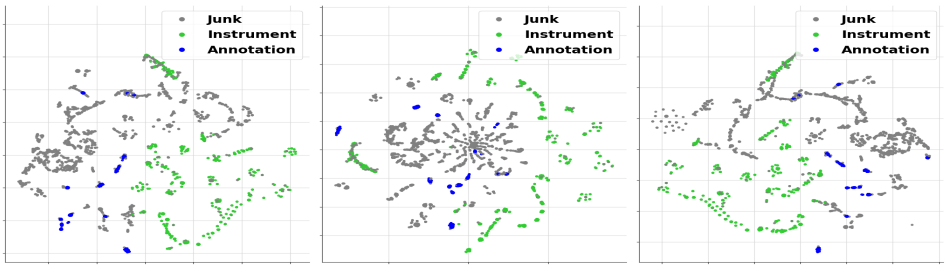
Figure 3 shows the precision and recall from pairwise clustering with varying distance thresholds and max candidates. For the rest of our classification results, we use a distance **r**=35 and max candidates **n**=3, because these values provide a significant improvement to our recall, before reaching the point where precision dramatically decreases.

4.2 Classification Evaluation

Our models are evaluated on a held out test set. We do a file level split of our 20 graphs, which preserves graph structure for graph based approaches: 14 for training, 3 for validation and 3 for testing. As the different files vary in the number of each class they contain, the data splits are created attempting to keep the proportion of each label as close as possible between files. This resulted in a train set with 10326 labeled patterns, validation with 1818 labeled patterns, and test with 1819 labeled patterns. When training models, labels are randomly sampled from the training set, to simulate the process of a human labeling a random set of data.

We use the F2 score as an evaluation metric. The F2 score is a weighted version of an F1 score that assigns higher importance to recall than precision [4]. This was chosen as we prefer to create too many patterns rather than not enough; that is, we prefer a false positive to a false negative. Specifically, our evaluation uses a Micro (pooled) F2 score over the instrument and annotation classes. We do not consider the junk class as we are not concerned about the model's performance on it.

4.3 Unsupervised Classification



(a) T-SNE Plot of Original Vectors

(b) T-SNE Plot of Radial Basis Function Latent Vectors

(c) T-SNE Plot of Autoencoder Latent Vectors

Fig. 4: T-SNE Visualizations of Unsupervised Latent Spaces. From these plots, it can be seen that the unsupervised approaches do not produce linearly separable latent spaces.

Figure 4 shows T-SNE [19] visualizations of our original vector latent space, as well as the latent spaces from radial basis function and autoencoder approaches. From these plots it is clear that while the unsupervised approaches can generate different latent spaces, none of them are immediately linearly separable into three groups for classification. As a result, unsupervised methods alone are not as a sufficient standalone solution to the problem, and shift to focus instead on supervised approaches to learn more separable latent spaces.

4.4 Supervised Classification

We now discuss our supervised methods. We consider 7 models: BNN, MLP, Active BNN, KNN, GCN, GCN Baseline, and Pattern based. Each of these methods are an implementation of the algorithms outlined in section 3. GCN Baseline is identical to the GCN, but with a simplified feature vector, of only concatenated node embeddings $H[i]$ and $H[j]$. This allows us to analyze the effect of including higher order features. For the GCN methods, node features vectors are $[$*text length*, *no. digits*, *has hyphen*$]$ and edge features are $[D(x_1, x_2), D(y_1, y_2),$ angle$]$. For all other models, our input feature vector contains $[D(x_1, x_c), D(y_1, y_c), D(x_2, x_c), D(y_2, y_c),$ angle$]$, where $D(a, b)$ is a distance, and (x_c, y_c) is the centroid of the two pieces of text.

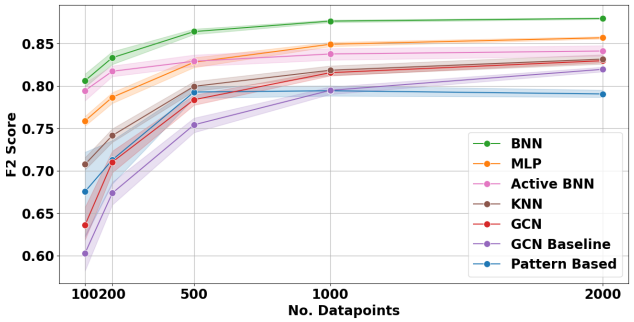


Fig. 5: Model F2 scores vs number of training samples.

Model	Micro F2 Score
BNN	0.864 ± 0.00338
MLP	0.828 ± 0.00536
Active BNN	0.829 ± 0.00669
KNN	0.800 ± 0.00647
GCN	0.784 ± 0.00604
GCN Baseline	0.754 ± 0.00880
Pattern Based	0.793 ± 0.00704

Table 1: Mean F2 Scores over 100 data points with 99.5% Confidence Intervals for Models Trained with 500 Labels

Table 1 compares the results of our models when trained with 500 random samples from the training set. Using these, we run statistical tests to compare the results. After a Shapiro-Wilk test for normality ($p = 0.05$) was rejected, we used a non-parametric Kruskal-Wallis ($p = 0.005$) test to determine if there was a difference in the mean F2 scores across classifiers. After confirming that the difference between classifiers was statistically significant we performed a post hoc Dunn's Multiple Comparison Test ($p = 0.005$) to determine exactly which algorithms were different from each other. From the matrix of Dunn's test, found in Appendix A.1, we order model performance as follows with models that were not significantly differentiable being grouped together: BNN $>$ {MLP, Active BNN} $>$ {KNN, GCN, Pattern Based} $>$ GCN Baseline.

We also consider the performance of the models with different numbers of training samples. This provides valuable insight for the real-world application of this project, where the amount of labeling that is considered acceptable can vary. It is therefore important that we provide insight on which models will perform best with a range of different training samples. Figure 6 shows the mean performance of the models over 100 runs as the number of labels randomly sampled from the training set varies, with a 99.5% confidence interval error bar. The plot indicates that the BNN is consistently the leading model. Its active learning counterpart lags behind, suggesting that the model may be overfitting to the smaller datasets that are given to the model during early stages of training, and is unable to learn properly with more data. Additionally, we can see that the GCN algorithm initially under performs, but catches up as more data becomes available. This may be due to the two-stage nature increasing the amount of data required for the training process. The GCN also outperforms the baseline GCN, which suggests our additional features and attention pooling assist the edge classification.

5 Conclusion & Future Work

This paper explored several approaches for extracting and classifying spatial pattens from engineering drawings. Our results show that a variety of supervised classifiers can succeed at this task, with Bayesian networks giving the best results. If deployed in the real world our method would lead to reduced overhead for engineers who manually search through diagrams, and offers a much more generalizable solution compared to current heuristic approaches. Additionally, because of the low number of samples required for each of our approaches, our methods require less human time and effort compared to current heuristic approaches. While we only considered patterns of size two, our results show the potential of the approach, and an important next step is to generalize to patterns of different sizes. As this will increase the complexity of the problem, we believe that both Bayesian and graph models should still be considered as pattern size changes.

For other future work, a natural next step is to explore whether more sophisticated feature engineering, such as additional geometric features, can improve model performance. Additionally, extracted graphic elements of the drawings could be included to provide additional context to the model. Further analysis of features may be able to better determine which information is most beneficial to our classifiers. More complex active learning methods should be explored, as they may be better able to query the human for new points and improve upon our method. Finally, although our method achieved a high recall, the initial clustering step of this problem leaves room for much further work, potentially being considered a learning problem on its own.

Acknowledgements

We would like to extend our sincere thanks to our domain expert Brian Gue, as well as Orion Sehn, Spencer Olson, and Christoph Sydora for providing us with data and invaluable insights throughout the course of this project. Additionally, we would also like to thank Professor Russ Greiner, and our teaching assistant Sam Scholnick-Hughes for their continuous support and encouragement as we navigated CMPUT 469.

References

[1] Charu Aggarwal, Gewen He, and Peixiang Zhao. Edge classification in networks. In *2016 IEEE 32nd International Conference on Data Engineering (ICDE)*, pages 1038–1049, 2016.

[2] Jon Louis Bentley. Multidimensional binary search trees used for associative searching. *Commun. ACM*, 18(9):509–517, September 1975.

[3] Charles Blundell, Julien Cornebise, Koray Kavukcuoglu, and Daan Wierstra. Weight uncertainty in neural network. In *International conference on machine learning*, pages 1613–1622. PMLR, 2015.

[4] Peter Christen, David J. Hand, and Nishadi Kirielle. A review of the f-measure: Its history, properties, criticism, and alternatives. *ACM Comput. Surv.*, 56(3), October 2023.

[5] David Cohn, Les Atlas, and Richard Ladner. Improving generalization with active learning. *Machine Learning*, 15(2):201–221, May 1994.

[6] Matthias Fey and Jan Eric Lenssen. Fast graph representation learning with pytorch geometric, 2019.

[7] Justin Gilmer, Samuel S. Schoenholz, Patrick F. Riley, Oriol Vinyals, and George E. Dahl. Neural message passing for quantum chemistry, 2017.

[8] Laurent Valentin Jospin, Hamid Laga, Farid Boussaid, Wray Buntine, and Mohammed Bennamoun. Hands-on bayesian neural networks—a tutorial for deep learning users. *IEEE Computational Intelligence Magazine*, 17(2):29–48, 2022.

[9] Diederik P Kingma, Max Welling, et al. Auto-encoding variational bayes, 2013.

[10] Tsung-Yi Lin, Priya Goyal, Ross Girshick, Kaiming He, and Piotr Dollar. Focal loss for dense object detection. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, Oct 2017.

[11] Umberto Michelucci. An introduction to autoencoders, 2022.

[12] Fionn Murtagh. Multilayer perceptrons for classification and regression. *Neurocomputing*, 2(5):183–197, 1991.

[13] Radford M Neal. *Bayesian learning for neural networks*, volume 118. Springer Science & Business Media, 2012.

[14] Carl Edward Rasmussen and Christopher K. I. Williams. *Gaussian Processes for Machine Learning*. The MIT Press, 2006.

[15] Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, and Gabriele Monfardini. The graph neural network model. *IEEE Transactions on Neural Networks*, 20(1):61–80, 2009.

[16] Zhenyu Song, Zhanling Shi, Xuemei Yan, Bin Zhang, Shuangbao Song, and Cheng Tang. An improved weighted cross-entropy-based convolutional neural network for auxiliary diagnosis of pneumonia. *Electronics*, 13(15), 2024.

[17] Kashvi Taunk, Sanjukta De, Srishti Verma, and Aleena Swetapadma. A brief review of nearest neighbor algorithm for learning and classification. In *2019 International Conference on Intelligent Computing and Control Systems (ICCS)*, pages 1255–1260, 2019.

[18] Hugo Touvron, Matthieu Cord, Alaaeldin El-Nouby, Piotr Bojanowski, Armand Joulin, Gabriel Synnaeve, and Hervé Jégou. Augmenting convolutional networks with attention-based aggregation, 2021.

[19] Laurens van der Maaten and Geoffrey Hinton. Visualizing data using t-sne. *Journal of Machine Learning Research*, 9(86):2579–2605, 2008.

[20] Yeming Wen, Paul Vicol, Jimmy Ba, Dustin Tran, and Roger Grosse. Flipout: Efficient pseudo-independent weight perturbations on mini-batches. *arXiv preprint arXiv:1803.04386*, 2018.

[21] Jiaming Zeng, Adam Lesnikowski, and Jose M. Alvarez. The relevance of bayesian layer positioning to model uncertainty in deep bayesian active learning, 2018.

A Appendix

A.1 Post Hoc Dunn's Test Matrix

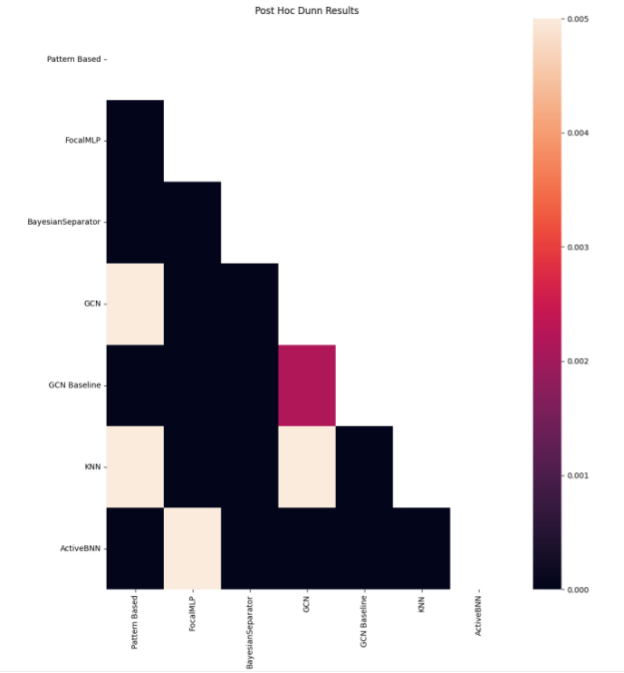


Fig. 6: Model Post Hoc Dunn's Test results.

A.2 Autoencoder Reconstruction Loss

$$\text{argmin}_{f,g} \; ||f(g(\mathbf{x})) - \mathbf{x}||_2^2 \tag{9}$$

A.3 Ground Truth Cleaning

Prior to engaging with any approaches, establishing an accurate ground truth (GT) dataset was essential. The initial GT was provided as a CSV file comprising pairwise clusters of text, each annotated with corresponding x and y coordinates and a designated class label. However, a preliminary inspection revealed significant discrepancies between the text assigned in the GT and the original dataset, particularly when comparing the associated coordinates with those documented in the original text and coordinates file.

To resolve these mismatches, a mapping approach was adopted. Specifically, for each pair in the GT, the given x and y coordinates were employed as keys to retrieve the corresponding text from the original dataset. This cross-referencing ensured that each coordinate pair in the GT was accurately

associated with the correct textual data. Following this alignment process, duplicate entries were identified and systematically removed to enhance the integrity of the dataset.

The preprocessing operation commenced with approximately 43,261 pairwise entries. After mapping and deduplication across all 20 graphs, the dataset was refined to a total of 13,964 unique pairs. The final class distribution within the preprocessed GT was as follows: 1,766 pairs for the 'Instrument' class, 390 pairs for the 'Annotation' class, and 11,808 pairs classified as 'Junk'.

A.4 Human In The Loop Labeling

The dataset provided for training is unlabeled, making the process of building supervised learning models challenging, as all labels are withheld and only available for evaluation purposes. To address this problem, we have two potential options: switching to a fully unsupervised task where GNNs come in, or giving a small subset of input data to a human expert to manually label the data points. The second option here is called Human In The Loop(HITL).

To implement HITL, we first need to run the pairwise clustering algorithm on the dataset. From this, a fixed number of files are chosen, and random pairs are shown to the expert who reviews the data. These pairs are presented over the real engineering drawing for ease of understanding, allowing experts to classify them. The expert chooses whether the shown pair is a valid pair or not, and the choices are recorded in an Excel file, which will serve as labeled data for subsequent training.

For the scope of this project, we assume that the classifications provided by the expert are accurate and serve as the ground truth for subsequent training steps. This intermediate layer of human validation ensures that the models have a more reliable starting point for further training and evaluation.

A.5 Additional Details on GNN Implementation

A.5.1 Graph Formulation

Let G be a graph represented by the tuple

$$G = (X, E, A, Y),$$

where:

- $V = \{v_1, v_2, \ldots, v_n\}$ is the set of n nodes.
- $E \subseteq V \times V$ is the set of (undirected) edges.
- $A \in \{0,1\}^{n \times n}$ is the adjacency matrix, where
$$A_{ij} = \begin{cases} 1, & (v_i, v_j) \in E, \\ 0, & \text{otherwise.} \end{cases}$$
- $X = [\mathbf{x}_1; \mathbf{x}_2; \ldots; \mathbf{x}_n] \in \mathbb{R}^{n \times d}$ is the node-feature matrix, with each $\mathbf{x}_i \in \mathbb{R}^d$.
- $Y = [y_1, y_2, \ldots, y_n] \in \mathcal{Y}^n$ is the vector of node labels (or targets), where y_i may be categorical or real-valued depending on the task.

A.5.2 Message Passing with NNConv

Our model uses edge-conditioned message passing via NNConv layers. For layer l, the node update is:

$$\mathbf{h}_i^{(l+1)} = \sigma\left(\sum_{j \in \mathcal{N}(i)} \mathbf{W}_{ij}\, \mathbf{h}_j^{(l)}\right)$$

where:

- $\mathbf{h}_j^{(l)}$ is the embedding of node j at layer l.
- $\mathcal{N}(i)$ is the set of neighbors of node i.
- $\mathbf{W}_{ij}$ is computed as:
$$\mathbf{W}_{ij} = NN(\mathbf{e}_{ij})$$
with NN being a small neural network.
- $\sigma(\cdot)$ is an activation function, e.g., ReLU.

A.5.3 Residual Connections

To preserve original node features and enable deeper networks, we add a residual connection:

$$\mathbf{h}_i^{(l+1)} = \mathbf{h}_i^{(l)} + \Delta\mathbf{h}_i^{(l+1)}$$

where $\Delta\mathbf{h}_i^{(l+1)}$ is the output from the NNConv layer before adding the input. This technique helps maintain stable gradient flow and preserves local node information.

A.5.4 Edge Dropout

During training, we apply edge dropout (DropEdge) by randomly omitting edges. For each edge, we sample a binary mask m_{ij} as:

$$m_{ij} \sim \text{Bernoulli}(1-p)$$

Then,

$$\tilde{\mathbf{e}}_{ij} = m_{ij}\,\mathbf{e}_{ij} \quad \text{and} \quad \tilde{E} = \{(i,j) : m_{ij} = 1\}.$$

This regularization prevents over-reliance on specific local structures and improves generalization.

A.5.5 Global Attention Pooling

After computing node embeddings, we extract a global context vector using an attention mechanism:

$$g_i = \sigma(\mathbf{W}_g\,\mathbf{h}_i + b_g)$$

$$\mathbf{g} = \frac{\sum_{i\in V} g_i\,\mathbf{h}_i}{\sum_{i\in V} g_i}$$

where:

- $\mathbf{W}_g$ and b_g are learnable parameters.
- g_i represents the importance of node i.
- $\mathbf{g}$ is the weighted average of node embeddings, emphasizing informative nodes.

A.5.6 Edge Feature Engineering and Classification

For each edge (i,j), we create an enriched feature vector:

$$\mathbf{f}_{ij} = \Big[\,\mathbf{h}_i,\ \mathbf{h}_j,\ \big|\mathbf{h}_i - \mathbf{h}_j\big|,\ \mathbf{h}_i \odot \mathbf{h}_j,\ \mathbf{g}\,\Big]$$

where:

- $\mathbf{h}_i$ and $\mathbf{h}_j$ are the embeddings of the source and target nodes.
- $\big|\mathbf{h}_i - \mathbf{h}_j\big|$ is the elementwise absolute difference.
- $\mathbf{h}_i \odot \mathbf{h}_j$ denotes elementwise multiplication.

This feature vector is then passed to a Multi-Layer Perceptron (MLP) to produce class logits:

$$\mathbf{z}_{ij} = \text{MLP}\big(\mathbf{f}_{ij}\big)$$

A.5.7 Loss Function: Focal Loss with Label Smoothing

To address class imbalance and noisy labels, we use a focal loss with label smoothing. The focal loss is defined as:

$$\mathcal{L}_{\text{FL}} = -\alpha_t(1-p_t)^{\gamma}\log(p_t)$$

where:

- p_t is the predicted probability for the true class.
- α_t is a class balancing factor.
- γ is a focusing parameter that down-weights easy examples.

Label smoothing is applied by replacing the one-hot targets with a smoothed distribution, reducing model overconfidence.

A.5.8 Post-hoc Temperature Scaling

After training, we use temperature scaling to calibrate the softmax probabilities. We scale the logits as follows:

$$\tilde{\mathbf{z}}_{ij} = \frac{\mathbf{z}_{ij}}{T}$$

Then compute the probabilities:

$$p_{ij} = \text{softmax}\left(\tilde{\mathbf{z}}_{ij}\right) = \frac{\exp(\tilde{\mathbf{z}}_{ij})}{\sum_k \exp(\tilde{\mathbf{z}}_{ik})}$$

Here, T is the temperature parameter:

- $T > 1$ produces a softer distribution.
- $T < 1$ produces a sharper distribution.

This calibration ensures that the output probabilities align better with actual model performance.

Algorithm 3 GNN Training Algorithm [why gnns fail 8.1]

Input: Graph dataset $\mathcal{D}$, epochs N, learning rate η, dropout probability p
Initialize: Residual GCN f, edge classifier MLP g, global pooling module, optimizer, scheduler
for epoch $= 1$ to N **do**
  for each graph $G = (X, E, A, Y)$ in $\mathcal{D}$ **do**
    if Training mode **then**
      Apply edge dropout: $(\tilde{E}, \tilde{A}) \leftarrow \text{DropEdge}(E, A, p)$
    else
      $(\tilde{E}, \tilde{A}) \leftarrow (E, A)$
    end if
    Compute node embeddings: $H \leftarrow f(X, \tilde{E}, \tilde{A})$
    Compute global context: $g_c \leftarrow \text{GlobalPool}(H)$
    For each selected edge $e = (i, j)$:
      Form edge features:

$$f_e \leftarrow \text{concat}\Big(H[i],\, H[j],\, |H[i] - H[j]|,\, H[i] \odot H[j],\, g_c\Big)$$

    Compute logits: $z \leftarrow g(f_e)$
    Compute loss: $\mathcal{L} \leftarrow \text{FocalLoss}(z, Y)$
    Backpropagate and update model parameters
  end for
  Update scheduler; Evaluate on validation set
end for
Post-training: Calibrate the model by optimizing the temperature parameter T.

Why GNN's fail in the unsupervised setting?
In our experiments, we used the GNN to generate embeddings for all edges in a graph created using a threshold distance. The resulting embedded space looked random, with no clear structure. When the instrument key and its adjacent text were far apart, the GNN could distinguish those graph structures as a separate set of clusters. However, in cases of stacked drawings, clusters with the adjacent text on top and the instrument key below had similar graph structures and as a result similar embeddings to clusters with the instrument key on top and the adjacent text below. In these cases, the latter might belong to the instrument class while the former always belongs to junk. As a result, the embedded space was filled with mixed junk and instrument embeddings without any distinction between them, and hence we failed to achieve any useful results from this method.