A AN IN-DEPTH OVERVIEW OF EXISTING DATASETS

Due to the discussed importance of f_e on GNN performance there is a lot to discuss about prior datasets that exist within the space of GNNs in regards to these functions.

A.1 PYTORCH GEOMETRIC

Pytorch Geometric python library provides a standard interface on top of Pytorch to allow for the development of graph based machine learning. The library also provides a sample of datasets from previous papers published in this field. A brief overview of popular graph-connected datasets, their task and embeddings is available in table 1.

As is clear from the table the current standard embedding for datasets is bag of words. In the cases where bag of words approaches are not used the approach is grounded in classical text representations such as n-grams and word vectors.

The tasks in these popular datasets are node classification where the node data is frequently text. We therefore say that on the node level these are text classification tasks. The only instance of a non-text classification task is Flickr Zeng et al. (2019), though based on the fact that the underlying data is image descriptions this could also be considered a text classification task.

This demonstrates how limited the reach of GNNs currently stand as they are being trained on datasets that behave very similarly where the only difference is the specifics of the available data. We feel that this does not therefore fully test the capabilities of GNNs and puts too much emphasis on bag of words and text classification.

A.2 OPEN GRAPH BENCHMARK

The results in this paper focus on *node property prediction* as the data that unconnected models ordinarily work on is easily transferred to nodes in a graph. So when discussing Open Graph Benchmark (Hu et al., 2020, OGB) the focus is on the node property prediction subset (OGBN).

The goal of OGB is to create a standard set of datasets that can be used to compare different GNN architectures so a discussion as to way we did not use their datasets is warranted. The available datasets *ogbn-products, ogbn-proteins, ogbn-arxiv, ogbn-papers100M* and *ogbn-mag* all use variations on the same text representations used in Appendix A.1. These include Bag of Words (BoW), word2vec and skip-gram. This means the same discussions on these classical text holds here.

We see that the majority of the tasks focus on text classification, excluding *ogbn-protein*, this again draws into question how well these datasets are testing the range of classification tasks. Further to this, focusing mainly on BoW style embeddings raises the question of whether we are building good BoW extractors or graph information extractors.

We do not compare against these datasets as information on how they extracted and washed their data is not readily available and therefore finding matching raw data was not possible. Without the matching raw data we cannot compare a raw version of the dataset to the embedding dataset provided by OGBN.

A.3 FLICKR

The prior Flickr dataset used in *Zeng et al.* Zeng et al. (2019) originated from *McAuley et al.* McAuley & Leskovec (2012) which aimed to utilize network connections and image descriptions rather than the images themselves. The specific embedding function that the paper used is Bag of Words.

This embedding function is a valid representation of images but it is not easily applicable to other image datasets. Thus GNNs trained on this dataset are confined to images with descriptions that have been transformed using the same top 500 words. Noting that this list of top 500 words is not readily available.

A.4 AMAZON

Zeng et al. Zeng et al. (2019) also provide an Amazon dataset (AmazonProducts) covering the entirety of Amazon. Without a known source we instead use available Amazon databases online to download and generate our own dataset. The embedding function used is to tokenise the reviews by 4-grams and take the single value decomposition. This is, as with Flickr, not easily applicable outside of the original dataset.

An alternative Amazon dataset (Amazon) is also available from *Shchur et al.* Shchur et al. (2018) created originally in *McAuley et al.* McAuley et al. (2015). Though the original source of the dataset used a pre-trained Caffe model to embed the product images this dataset did not use these. Instead they created their own embeddings using the bag of words standard with the product reviews as the raw data.

B OUR DATASETS

Table 6: The number of nodes and classes with the shape of the data x_i for the three new datasets with each of the different embedding functions used. Note that Flickr_v2 also has the raw images represented by -

Dataset	Embedding	# Nodes	# Classes	Input Shape
Flickr_v2	-	76659	7	(3, 224, 224)
	ResNet50	76659	7	(1024)
	ResNet18	76659	7	(512)
	VGG16	76659	7	(25088)
AmazonElectronics	Bag of Words	92048	6	(500)
	RoBERTa Byte Pair	92048	6	(512)
	RoBERTa Encoded	92048	6	(1024)
	RoBERTa	92048	6	(1024)
AmazonInstruments	Bag of Words	21127	7	(500)
	RoBERTa Byte Pair	21127	7	(512)
	RoBERTa Encoded	21127	7	(1024)
	RoBERTa	21127	7	(1024)

In this paper we look at the Flickr (Hamilton et al., 2017) and Amazon (Hamilton et al., 2017; Shchur et al., 2018) datasets (specifically looking at the Electronics and Instruments subsection of Amazon). An overview of the provided datasets and embeddings is present in Table 6.

As demonstrated in Section 2 the process of building a graph dataset is currently signified by choosing an embedding function f_e to apply to the underlying dataset $\boldsymbol{X}$. We provide three cases of the underlying dataset $\boldsymbol{X}$ and provide a selection of embeddings from sensible embedding functions. We also provide a *raw* graph-connected image dataset, by which we mean the underlying image dataset $\boldsymbol{X}$ is not reduced to a matrix of 1D feature vectors but remains a multi-dimensional tensor of multi-dimensional image tensors.

As discussed in Section 2, versions of these datasets (Zeng et al., 2019; Shchur et al., 2018) utilize classical word embeddings. We propose using large pre-trained models as embedding functions as these can extract richer features ($\boldsymbol{X}_e$) that may better represent the underlying data ($\boldsymbol{X}$) of the dataset.

We will host our datasets in a website and make both these baselines and datsets accessible to the community (website link will be added after acceptance).

B.1 FLICKR_V2

The prior Flickr (Zeng et al., 2019) dataset does not contain the raw image data nor the flickr identifiers only the embeddings created in *McAuley et al.* McAuley & Leskovec (2012) but SNAP provides an adjacency matrix and flickr identifiers. We follow how *Zeng et al.* Zeng et al. (2019) generated

their dataset by downloading all the images still available, but we are unable to match the exact images they sourced or the labels. This does mean we cannot directly compare results but we find that the benchmark results we have achieved are similar.

As shown in Table 1 Flickr uses BoW but as discussed in Section 5.1.1 we deem this inappropriate for the image dataset. A potential solution would be to process the images, as discussed, to create text descriptions that can be converted into BoW embeddings. However, this would either require a human, and thus the benefits of using a neural network (such as time efficiency of classification) is lost, or a neural network. In the case of a neural network we claim that the provided networks for image classification would perform better. This is an area for further investigation especially as we found BoW works better in the case of Amazon Instruments.

As the dataset is created from four separate image classification challenges (Everingham et al., 2015; Chua et al., July 8-10, 2009; Huiskes & Lew, 2008) the existing labels and tags for these datasets are not well aligned. We therefore only select the NUS-Wide (Chua et al., July 8-10, 2009) subsection. To create labels for our dataset we use GloVe to generate vector representations of the NUS-Wide tags and find the closest vector representation of 7 hand-picked labels.

The hand-picked labels: ["People", "Buildings", "Places", "Plants", "Animals", "Vehicles", "Scenery"]

This new dataset we call Flickr_v2.

B.2 AMAZONELECTRONICS AND AMAZONINSTRUMENTS

We use the review text as our raw input features, this means a number of Amazon items are unsuitable as they do not have any reviews. The connections between nodes are based on Amazons three similarity metric "Co-viewed", "Co-bought" and "Similar Items". We only consider direct connections between nodes in a specific Amazon category (in our case Electronics and instruments) making sure removing any nodes that are not directly connected to another node.

Amazon products do not belong to a single category and therefore a products categories are closer to tags than labels. Thus, similar to Flickr_v2, we use GloVe word vectors to label our dataset based on the set of categories that each product belongs to against a hand-picked selection of categories.

Hand-picked labels for Electronics: ["Camera Photo and Lighting", "Audio and Video", "Bags, Cases and Covers", "Batteries and Chargers", "Peripherals, Keyboards and Mice", "Storage and Networking"]

Hand-picked labels for Instruments: ["Electric Guitar", "Acoustic Guitar", "Percussion", "Live Sound & Stage", "Studio Recording Equipment", "Microphones & Cable", "Amplifiers & Effects"]

These new datasets we call AmazonElectronics and AmazonInstruments.

The statistics for these datasets is available in Table 6

C GRAPH ATTENTION NETWORKS

The original paper *Velickovic et al.* Veličković et al. (2018) proposed a new approach to graph representation learning incorporating the popular attention mechanisms of sequence-based tasks.

The basic concept is to provide and attention mechanism $\boldsymbol{a}$ which calculates the attention coefficients between a node v_i and its neighbours $\mathbb{N}_i$, employing self-attention. This attention coefficient between node v_i and a neighbour v_j. These coefficients are then normalised to compare across nodes using the *softmax* transform resulting in the final coefficient for a given neighbour α_{ij}

$$\alpha_{ij} = \frac{exp(\text{LeakyReLU}(\boldsymbol{a}^T[\boldsymbol{\theta h}_i || \boldsymbol{\theta h}_j]))}{\sum_{k \in \mathbb{N}_i} exp(\text{LeakyReLU}(\boldsymbol{a}^T[\boldsymbol{\theta h}_i || \boldsymbol{\theta h}_j]))} \tag{7}$$

This attention mechanism occurs for each head in a layer allowing for multi-headed attention. The attention coefficient and weight matrices for each head are denoted with a superscript k where $k \in$

$[K]$ and K is the total number of heads. We can thus relate GAT to the basic MESSAGE-PASSING network equation 1 as such

$$\boldsymbol{h}_i^l = \text{AGGREGATE}_k^K(\sigma(\sum_{j \in \mathbb{N}_i} \alpha_{ij}^k \boldsymbol{\theta}^k \boldsymbol{h}_j^{l-1})) \tag{8}$$

where AGGREGATE is some aggregation function, the original paper used MEAN and CONCATENATE.

A later paper ***Brody et al.*** Brody et al. (2021) questioned the attentiveness of the proposed method and provided the following attention mechanism instead of equation 7

$$\alpha_{ij} = \frac{exp(\boldsymbol{a}^T \text{LeakyReLU}(\boldsymbol{\theta}[\boldsymbol{h}_i || \boldsymbol{h}_j]))}{\sum_{k \in \mathbb{N}_i} exp(\boldsymbol{a}^T \text{LeakyReLU}(\boldsymbol{\theta}[\boldsymbol{h}_i || \boldsymbol{h}_j]))} \tag{9}$$

This graph connection approach allows us to train the attention mechanism $\boldsymbol{a}$ across the graph $\mathcal{G}$ without having to train the weight matrix $\boldsymbol{\theta}$. So if we are provided with suitable weights we can greatly reduce the training cost.

D Detailed Results

Tables 7 to 10 are the same results as those present in Section 6 but with added standard deviation from the mean.

Table 7: Test accuracy on Flickr_v2 with different embeddings, compared against the corresponding unconnected vision model. Included is the difference Δ of each model to the unconnected model and the standard deviation of each result. The details of the embedding styles are available in Appendix B.

Model	Embedding Styles		
	ResNet18	ResNet50	VGG16
Unconnected Model	45.2% ± 0.1	*46.9%* ± 0.0	**47.0 ± 0.1**
$\Delta \uparrow$	+0.0	*+0.0*	**+0.0**
GCN	41.8% ± 0.4	38.3% ± 0.5	*35.5% ± 0.3*
$\Delta \uparrow$	-3.4	-8.6	*-11.5*
GAT	38.1% ± 0.6	37.1% ± 1.1	27.3% ± 1.2
$\Delta \uparrow$	-7.1	-9.8	-19.7
GAT2	42.1% ± 1.8	41.0% ± 1.5	34.2% ± 0.8
$\Delta \uparrow$	-3.1	-5.9	-12.8
GraphSAGE (Random)	*45.4% ± 0.1*	**47.0% ± 0.0**	35.2% ± 0.2
$\Delta \uparrow$	*+0.2*	**+0.1**	-11.8
GraphSAGE (Neighbour)	**45.8% ± 0.2**	44.5% ± 0.1	34.5% ± 0.2
$\Delta \uparrow$	**+0.6**	-2.4	-12.5

E Hyperparameters

Table 11 details the layers of each model used providing the output hidden features of each layer, the sampler used (the specifics shown in Table 12) and the maximum and minimum learning rates. Where there is a difference in learning rates we use a learning rate scheduler that decreases the learning rate when validation accuracy plateaus. Where two models use the same sampler the parameters of those samplers are identical to keep consistency across the tests.

In the case of GraNet we start with a high learning rate for 20 epochs with the pre-trained model frozen to allow the GNN to train. We then unfreeze the entire model dropping the learning rate

Table 8: Test accuracy on AmazonInstruments with different embeddings compared against a standard unconnected MLP model. Included is the difference Δ of each model to the unconnected MLP and the standard deviation of each result. The embedding styles are explained in Appendix B.

Model	Embedding Styles			
	Bag of Words	Byte Pair	roBERTa Encoded	roBERTa
Unconnected MLP	66.1% ± 0.2	21.0% ± 0.3	43.9% ± 0.4	39.8% ± 0.7
$\Delta \uparrow$	+0.0	+0.0	+0.0	+0.0
GCN	64.0% ± 0.5	20.8% ± 0.3	20.4% ± 0.8	20.4% ± 0.8
$\Delta \uparrow$	-2.1	-0.2	-23.5	-19.4
GAT	*79.3% ± 0.6*	21.6% ± 0.9	47.5% ± 1.9	46.1% ± 4.3
$\Delta \uparrow$	*+13.2*	+0.6	+3.6	+6.3
GAT2	**79.4% ± 0.3**	21.2% ± 0.6	*49.8% ± 5.0*	*47.8% ± 2.8*
$\Delta \uparrow$	**+13.3**	+0.2	*+5.9*	+8.0
GraphSAGE (Random)	67.5% ± 0.3	*23.9% ± 0.6*	45.1% ± 1.2	41.9% ± 0.6
$\Delta \uparrow$	+1.4	*+2.9*	+1.2	+2.1
GraphSAGE (Neighbour)	72.6% ± 0.3	**43.4% ± 0.5**	**62.4% ± 0.5**	**59.9% ± 0.6**
$\Delta \uparrow$	+6.5	**+22.8**	**+18.5**	**+20.1**

Table 9: Test accuracy on AmazonElectronics with different embeddings compared against a standard unconnected MLP model. Included is the difference Δ of each model to the unconnected MLP and the standard deviation of each result. The embedding styles are explained in Appendix B.

Model	Embedding styles			
	Bag of Words	Byte Pair	roBERTa Encoded	roBERTa
Unconnected MLP	71.6% ± 0.3	21.6% ± 0.0	55.8% ± 0.1	51.9% ± 0.2
$\Delta \uparrow$	+0.0	+0.0	+0.0	+0.0
GCN	69.1% ± 0.1	21.7% ± 0.2	22.7% ± 1.1	22.3% ± 1.2
$\Delta \uparrow$	-2.5	+0.1	-33.1	-29.6
GAT	*81.1% ± 0.2*	22.2% ± 0.5	46.1% ± 1.5	40.3% ± 2.9
$\Delta \uparrow$	*+10.5*	+0.6	-9.7	-11.6
GAT2	**81.8% ± 0.3**	22.2% ± 0.6	41.8% ± 5.1	35.7% ± 5.6
$\Delta \uparrow$	**+10.2**	+0.6	-14.0	-16.2
GraphSAGE (Random)	71.3% ± 0.1	*26.3% ± 0.3*	*57.0% ± 0.5*	*53.7% ± 0.5*
$\Delta \uparrow$	-0.3	*+4.7*	*+1.2*	*+1.8*
GraphSAGE (Neighbour)	76.4% ± 0.3	**40.4%** ± 0.4	**67.8% ± 0.4**	**66.4% ± 0.3**
$\Delta \uparrow$	+4.8	**+20.8**	**+12.0**	**+12.5**

sharply and train for another 100 epochs. In the first stage no learning rate scheduler is employed, same as for all GNNs, and in the second stage we apply the learning rate scheduler.

For GraphSAINTSampler all setups use a walk length of 2 with 5 steps sampling 100 nodes per node for normalisation calculation.

Table 10: Comparison of GraNet models against the best performing GNNs for a specific embedding. Included is the difference Δ of each model to the unconnected model and the standard deviation of each result. The specifics of the 3 datasets is explained in Appendix B.

Model	Flickr_v2		Electronics	Instruments
	ResNet18	ResNet50	Bag of Words	Bag of Words
Unconnected Model	45.2% ± 0.1	46.9% ± 0.0	71.6% ± 0.3	66.1% ± 0.2
$\Delta \uparrow$	+0.0	+0.0	+0.0	+0.0
GAT2	42.1% ± 1.8	41.0% ± 1.5	*81.8%*	*79.4% ± 0.3*
$\Delta \uparrow$	-3.1	-5.9	*+10.2*	*+13.3*
GraphSAGE (Random)	45.4% ± 0.1	*47.0% ± 0.0*	71.3%	67.5% ± 0.3
$\Delta \uparrow$	+0.2	*+0.1*	-0.3	+1.4
GraphSAGE (Neighbour)	*45.8% ± 0.2*	44.5% ± 0.1	76.4%	72.6% ± 0.3
$\Delta \uparrow$	*+0.4*	-2.4	+4.8	+6.5
GraNet	**46.7% ± 0.1**	**48.7% ± 0.1**	**81.9% ± 0.5**	**79.7% ± 0.9**
$\Delta \uparrow$	**+1.5**	**+1.8**	**+10.3**	**+13.6**

Table 11: Model architecture, sampler and learning rate

Model	Hidden Features	Sampler	Learning Rate	
			Max.	Min.
GCN	256 256	Random Node	1e-2	1e-2
GAT	256 256	GraphSAINT RW	1e-2	1e-2
GAT2	256 256	GraphSAINT RW	1e-2	1e-2
GraphSAGE (Random)	256 256	Random Node	1e-3	1e-3
GraphSAGE (Neighbour)	256 256	Neighbour	1e-3	1e-3
MLP	256 256 128	-	1e-5	5e-7
ResNet18	*as provided*	-	1e-4	5e-6
ResNet50	*as provided*	-	1e-4	5e-6
VGG16	*as provided*	-	1e-4	5e-6
GraNet (MLP + GAT2)	256 256 128	GraphSAINT RW	1e-2	1e-2
GraNet (ResNet18 + GraphSAGE)	*as provided*	Random Node	1e-3	5e-8
GraNet (ResNet50 + GraphSAGE)	*as provided*	Random Node	1e-3	5e-8

Table 12: Sampler parameters

Sampler	Dataset Split	Setup
GraphSAINT RW (Zeng et al., 2019)	Train	roots: 6000
	Validation	roots: 1250
	Test	roots: 2000
Random Node	Train	# partitions:512
	Validation	# partitions:128
	Test	# partitions:256
Neighbour (Hamilton et al., 2017)	Train	# neighbours:[25, 10], batch size:512
	Validation	# neighbours:[25, 10], batch size:128
	Test	# neighbours:[25, 10], batch size:256