Exploring Emerging Properties in Multimodal Neural Networks

Pranav Singh
ps4364@nyu.edu

Akash Mishra
am11533@nyu.edu

Mayank Poddar
mp6021@nyu.edu

1 Objective

There has been a sharp increase in the number of people using social media recently. Because of this, there has been an increase in the amount of content posted online. Manual moderation of content online is extremely difficult. To overcome this barrier, many automated moderation techniques have been designed. These techniques work with a singular modality (like text, audio, images, etc.) but suffer in a multi-modal environment where one modality complements one or more modalities implying subtle hate/aggressive speech. To address this problem, various unimodal, as well as multimodal approaches have been proposed. In general vision, only models perform worse than natural language models; with this study, we aim to address this by using novel, unimodal self-supervised techniques and further studying the scope of their expansion to make them fully multimodal. These multimodal self-supervised techniques are able to match the accuracy of fully supervised multimodally finetuned models.

2 Dataset

The 2020 hateful meme dataset Kiela et al. [2020] has confounding examples, i.e., counter-intuitive examples based on text or visual features alone. This makes learning from a single modality impossible. Furthermore, even within the purview of multimodal approaches, this is exceptionally challenging as vision and language embeddings extracted from two images may be the same. Still, the downstream labels for the two images would be different. Hence Embedding collapse is an issue.

For the hateful meme challenge dataset we have 8500 images in the training set and a further 2000 images in the testing set. We perform an 80/20 split to get the validation set on the training set. Each entry in the dataset has an image, label, and extracted caption.

3 Approach

3.1 Unimodal and Multimodal Learning

For our approach, we pretrained unimodal and multimodal self-supervised techniques on the training set for 100 epochs. Once trained, we then perform end-to-end finetuning on the self-supervised models. We conducted these experiments over five different seed values to account for variability.

3.2 Graph Neural Network

The second approach that we explored was using GNNs (Graph neural networks). Here, we tried to generate subgraphs for each meme using the image and textual features. Since we want to create dependence between two different kinds of data, i.e., image and text, we try to associate text embedding with image features. Once we have generated this initial subgraph, we pass this into our GNN model, namely GCN Kipf and Welling [2016], Sage, and GAT Veličković et al. [2017]. Our intuition is that this graph will learn the dependence between the multimodal data and help us identify whether the meme is hateful.

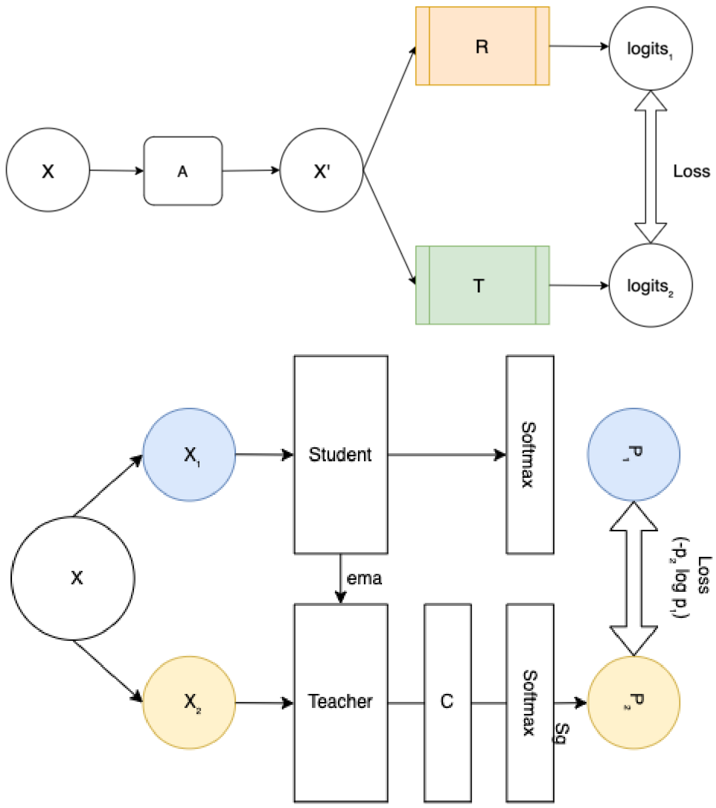

Figure 1: (Top) We show CASS, R represents ResNet-50, a CNN and T in the other box represents the Transformer used (ViT Base/Base-16); X is the input image, which becomes X' after applying augmentations. Note that CASS applies only one set of augmentations to create X'. X' is then passed through both the arms to compute similairy loss. This is different from DINO, that passes different augmentation of the same image through networks with the same architecture but different parameters. The output of the teacher network is centred on a mean computed over a batch. Another key difference is that in CASS, loss is computed over logits meanwhile in DINO it is computed over softmax output.

4 Implementation

For unimodal pre-training, we used CASS Singh et al. [2022] and DINO Caron et al. [2021]. We trained these techniques only on images. Then, during the fine-tuning process, we concatenated text embedding with the image embedding before finally passing through a classifier to perform downstream classification. We expanded upon this by making CASS fully multimodal during the pre-training task. We use ResNet-50 He et al. [2016] and Vision Transformer Dosovitskiy et al. [2020] for obtaining the image embeddings. Similarly, to obtain the text embeddings we use DistilBERT Sanh et al. [2019].

For GNNs, we will use Image and Text Encoder to get the relevant features. We use ResNet-50 pretrained on COCO Dataset for our image encoder and Distilbert pretrained from hugging face. Finally, we use Mask-RCNN He et al. [2017] along with an Image encoder to generate feature embeddings depicting entities in the image. Similarly, we also pass the text through Tokenizer and then through Distilbert to get textual embedding. A graph is constructed by joining the image feature embeddings to textual embedding and the original image. This subgraph will be passed into larger Graph neural networks to get us co-dependent embedding. This newly generated embedding can then be passed through Linear layers and sigmoid function to get us a classification of the Multimodal data.

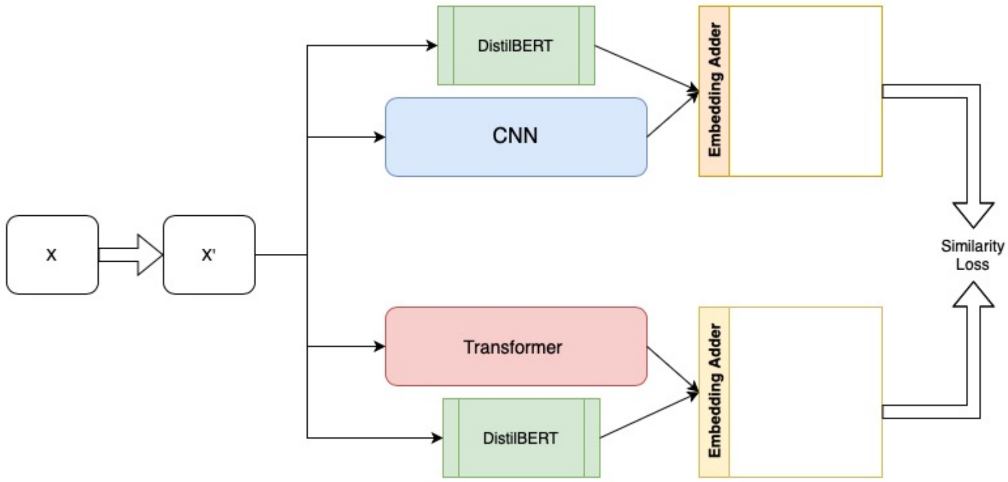

Figure 2: In this figure we show CASS, adopted for multimodal pretraining using DistillBERT and CASS. In this we take an input and apply minimal augmentations for image (resize and channel normalization), and tokenzie the text using DistillBERT tokenizer. We then pass this augmented image and tokenized through a combination of vision and text encoder as shown above. The ouputs of from the two are then combined in an emnedding adder, before calculating counteractive loss during pretraining.

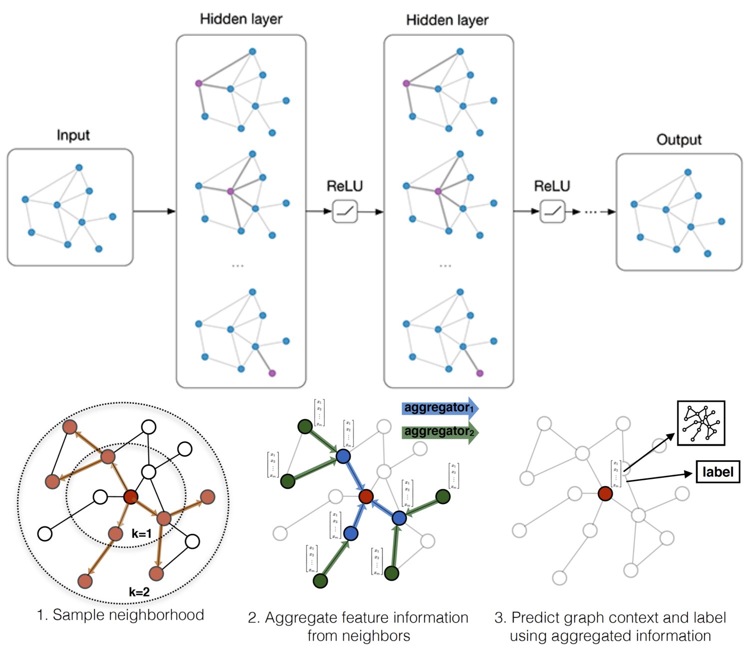

Figure 3: (Top) GCN is based on an efficient variant of convolutional neural networks which operate directly on graphs. Idea comes from getting a localized first-order approximation of spectral graph convolutions. (Bottom) Unlike other GNN, GraphSAGE Hamilton et al. [2017] provides inductive framework to efficiently generate node embeddings for unseen data.

5 Challenges

A major challenge with the Hateful meme dataset is the confounding samples, i.e two memes can have the same image but with different captions, and thereby different labels. Similarly, two memes

can have different images with the same text and hence have different downstream labels. This makes mapping the samples in a latent space extremely difficult. To mitigate this problem we developed multimodal solutions i.e instead of using only one modality we included embedding inputs from the two involved modalities i.e text and image. Furthermore, we have used experimented with graph neural networks (GNN), as they have amazing scene and relationship representation capabilities. We present our results in Table Another limiting factor is the relatively poor performance of vision-only methods as compared to text-only methods.
Lastly, in the case of Graph Neural network, we are facing trouble building a global heterogenous GNN which would better take into account the multimodal nature of our challenge.

Also, since each meme within the dataset is transformed into a subgraph, the scale of data explodes tremendously which limits our batch size and in turn our training speed.

6 Experimentation Results

In Figure 2 we describe and compare the architecture of CASS and DINO.

Modality	Model	AUROC	Accuracy
Unimodal	ResNet-50 224 (Supervised)	0.5157±0.081	0.5175±0.044
	ResNet-50 224 (CASS)	0.5398±0.091	0.5455±0.031
	ResNet-50 224 (DINO)	0.5129±0.009	0.5267±0.055
	ResNet-50 384 (Supervised)	0.5116±0.076	0.511±0.016
	ResNet-50 384 (CASS)	**0.5405±0.045**	0.5445±0.015
	ResNet-50 384 (DINO)	0.532±0.033	0.543±0.056
Multimodal	ResNet-50 - 384 (CASS)	0.501±0.078	**0.551±0.023**

Table 1: In this table we compare the performance of unimodally and Multimodally trained CASS and unimodally trained DINO and supervised ResNet-50. We observed that CASS trained Resnet-50 for input size 384, performs the best on the primary metric of comparison, but overall multimodally trained CASS ResNet-50.

Modality	Model	AUROC	Accuracy
Unimodal	VitB16 - 224 (Supervised)	0.51293±0.043	0.5235±0.065
	VitB16 - 224 (DINO)	0.513±0.077	0.5195±0.014
	VitB16 - 224 (CASS)	0.5196±0.051	0.5545±0.01
	VitB16 - 384 (Supervised)	0.5±0.022	0.545±0.087
	VitB16 - 384 (DINO)	0.5002±0.011	0.624±0.071
	VitB16 - 384 (CASS)	**0.53386±0.009**	0.59±0.017
Multimodal	VitB16 - 384 (CASS)	0.5±0.078	**0.625±0.032**

Table 2: In this table we compare the performance of unimodally and Multimodally trained CASS and unimodally trained DINO and supervised ViT/Base16. We observed a sinmilar trend as in Table 1, results for input image 384 were considerably better than that for 224. CASS trained models perform generally better than other models show considerable increase in performance over DINO and supervised training.

GNNs	Encoder	AUROC	Accuracy
GCN	Resnet50 + Distilbert	0.514	0.547
SAGE	Resnet50 + Distilbert	0.481	0.625
GAT	Resnet50 + Distilbert	0.541	0.499

Table 3: In this table we compare the performance of various GNN namely GCN, GraphSage and GAT. With the limited scope in GNN experiment, we are not seeing any significant performance improvement here.

7 High Performance Optimization

For High Performance Optimization, we took our best performing model on multimodality - Multimodal Cross Architecture Self-Supervised Learning model and tried to profile and optimize it.

Initially, we looked at profiling our model on CUDA using torch's autograd profiler to get a better understanding of the computational complexity - FLOPs and identifying the bottleneck operations. Here's the output of the profiler:

Two key points to notice here on the operations sorted by cuda time:

Name	Self CPU %	Self CPU	CPU total %	CPU total	CPU time avg	Self CUDA	Self CUDA %	CUDA total	CUDA time avg	CPU Mem	Self CPU Mem	CUDA Mem	Self CUDA Mem	# of Calls	Total MFLOPs
model inference	2.77%	69.930ms	99.13%	2.499s	2.499s	56.192ms	2.23%	2.520s	2.520s	-4 b	-420 b	0 b	-5.33 Gb	1	--
aten::conv2d	0.03%	828.000us	79.73%	2.010s	37.226ms	647.000us	0.03%	2.018s	37.372ms	0 b	0 b	509.99 Mb	0 b	54	98807.316
aten::convolution	0.06%	1.404ms	79.68%	2.009s	37.205ms	951.000us	0.04%	2.017s	37.360ms	0 b	0 b	509.99 Mb	0 b	54	--
aten:: convolution	0.04%	1.017ms	79.62%	2.007s	37.174ms	730.000us	0.03%	2.017s	37.343ms	0 b	0 b	509.99 Mb	0 b	54	--
aten::cudnn convolution	79.10%	1.994s	79.56%	2.006s	37.148ms	2.013s	79.90%	2.016s	37.328ms	0 b	0 b	509.99 Mb	345.22 Mb	54	--
aten::batch norm	0.19%	4.780ms	4.31%	108.695ms	2.051ms	4.626ms	0.18%	107.699ms	2.032ms	0 b	0 b	502.96 Mb	0 b	53	--
aten::linear	0.48%	12.084ms	2.94%	74.185ms	570.654us	10.461ms	0.42%	106.604ms	820.031us	0 b	0 b	860.09 Mb	0 b	130	--
aten:: batch norm impl index	0.03%	775.000us	4.11%	103.553ms	1.954ms	583.000us	0.02%	103.073ms	1.945ms	0 b	0 b	502.96 Mb	0 b	53	--
aten::cudnn batch norm	3.36%	84.828ms	4.06%	102.435ms	1.933ms	88.300ms	3.50%	102.490ms	1.934ms	0 b	0 b	502.96 Mb	0 b	53	--
aten::matmul	0.22%	5.573ms	1.94%	48.821ms	290.601us	3.794ms	0.15%	85.812ms	510.786us	0 b	0 b	1.95 Gb	0 b	168	--
aten::relu	2.33%	58.747ms	2.52%	63.662ms	1.299ms	58.486ms	2.32%	63.984ms	1.306ms	0 b	0 b	0 b	0 b	49	--
aten::softmax	0.01%	283.000us	2.28%	57.507ms	2.396ms	214.000us	0.01%	61.045ms	2.544ms	0 b	0 b	751.16 Mb	0 b	24	--
aten:: softmax	2.26%	56.880ms	2.26%	57.095ms	2.379ms	60.831ms	2.41%	60.831ms	2.535ms	0 b	0 b	751.16 Mb	751.16 Mb	24	--
aten::mm	0.33%	8.299ms	0.42%	10.582ms	88.183us	59.535ms	2.36%	59.535ms	496.125us	0 b	0 b	860.02 Mb	860.02 Mb	120	451852.370
aten::addmm	1.01%	25.511ms	1.04%	26.155ms	2.615ms	25.686ms	1.02%	25.854ms	2.585ms	0 b	0 b	76.00 Kb	76.00 Kb	10	24.232
aten::empty	0.35%	8.873ms	0.89%	22.440ms	37.214us	18.534ms	0.74%	18.534ms	30.736us	420 b	420 b	1.26 Gb	1.26 Gb	603	--
aten::add	0.47%	11.817ms	0.53%	13.317ms	70.089us	15.744ms	0.62%	15.744ms	82.863us	0 b	0 b	0 b	0 b	190	--
aten::gelu	0.57%	14.410ms	0.60%	15.015ms	625.625us	15.494ms	0.61%	15.494ms	645.583us	0 b	0 b	376.95 Mb	376.95 Mb	24	--
aten::empty like	0.07%	1.770ms	0.78%	19.730ms	122.547us	1.678ms	0.07%	14.478ms	89.925us	0 b	0 b	910.73 Mb	0 b	161	--
aten::bmm	0.12%	3.100ms	0.21%	5.413ms	112.771us	12.238ms	0.49%	12.238ms	254.958us	0 b	0 b	846.61 Mb	846.61 Mb	48	50234.278
aten::max pool2d	0.00%	122.000us	0.47%	11.923ms	11.923ms	109.000us	0.00%	11.976ms	11.976ms	0 b	0 b	27.00 Mb	0 b	1	--
aten::max pool2d with indices	0.43%	10.891ms	0.47%	11.772ms	11.772ms	11.867ms	0.47%	11.867ms	11.867ms	0 b	0 b	27.00 Mb	27.00 Mb	1	--
aten::eq	0.46%	11.622ms	0.47%	11.769ms	980.750us	11.776ms	0.47%	11.776ms	981.333us	0 b	0 b	6.00 Kb	6.00 Kb	12	--
aten::reshape	0.06%	1.636ms	0.54%	13.684ms	109.472us	1.384ms	0.05%	8.125ms	65.000us	0 b	0 b	374.33 Mb	0 b	125	--
aten::adaptive avg pool2d	0.00%	54.000us	0.31%	7.841ms	7.841ms	56.000us	0.00%	7.838ms	7.838ms	0 b	0 b	32.00 Kb	0 b	1	--
aten::clone	0.07%	1.753ms	0.46%	11.502ms	106.500us	1.590ms	0.06%	7.788ms	72.111us	0 b	0 b	407.98 Mb	0 b	108	--
aten::mean	0.31%	7.737ms	0.31%	7.776ms	7.776ms	7.782ms	0.31%	7.782ms	7.782ms	0 b	0 b	32.00 Kb	32.00 Kb	1	--
aten::masked fill	0.01%	277.000us	0.26%	6.480ms	540.000us	336.000us	0.01%	6.473ms	539.417us	0 b	0 b	19.34 Mb	0 b	12	--
aten::clamp min	0.02%	540.000us	0.19%	4.683ms	95.571us	480.000us	0.02%	5.498ms	112.204us	0 b	0 b	0 b	0 b	49	--
aten::clamp min	0.15%	3.699ms	0.18%	4.576ms	86.340us	5.363ms	0.21%	5.477ms	103.340us	0 b	0 b	16.00 Kb	0 b	53	--
aten::masked fill	0.18%	4.517ms	0.19%	4.862ms	405.167us	4.687ms	0.19%	4.872ms	406.000us	0 b	0 b	0 b	0 b	12	--
aten::layer norm	0.02%	491.000us	0.27%	6.689ms	131.157us	380.000us	0.02%	4.764ms	93.412us	0 b	0 b	210.00 Mb	0 b	51	--
aten::native layer norm	0.10%	2.532ms	0.24%	5.978ms	117.216us	3.027ms	0.12%	4.384ms	85.961us	0 b	0 b	210.00 Mb	0 b	51	--
aten::copy	0.06%	1.395ms	0.09%	2.300ms	21.296us	3.543ms	0.14%	3.543ms	32.806us	0 b	0 b	0 b	0 b	108	--
aten::mul	0.02%	522.000us	0.13%	3.243ms	270.250us	3.312ms	0.13%	3.312ms	276.000us	0 b	0 b	731.54 Mb	731.54 Mb	12	191.767
aten::transpose	0.09%	2.361ms	0.17%	4.412ms	20.521us	2.346ms	0.09%	3.164ms	14.716us	0 b	0 b	0 b	0 b	215	--
aten::t	0.07%	1.860ms	0.19%	4.768ms	36.677us	1.520ms	0.06%	3.155ms	24.269us	0 b	0 b	0 b	0 b	130	--
aten::view	0.12%	3.004ms	0.16%	4.131ms	6.990us	2.311ms	0.09%	2.311ms	3.910us	0 b	0 b	0 b	0 b	591	--
aten:: unsafe view	0.11%	2.851ms	0.58%	14.745ms	58.512us	1.533ms	0.06%	2.011ms	7.980us	0 b	0 b	0 b	0 b	252	--
aten::add	0.05%	1.323ms	0.07%	1.722ms	33.765us	1.820ms	0.07%	1.820ms	35.686us	0 b	0 b	208.59 Mb	208.59 Mb	51	51.342
aten::expand	0.06%	1.424ms	0.14%	3.471ms	29.168us	1.306ms	0.05%	1.724ms	14.487us	0 b	0 b	0 b	0 b	119	--
aten::as strided	0.06%	1.619ms	0.09%	2.283ms	5.794us	1.357ms	0.05%	1.357ms	3.444us	0 b	0 b	0 b	0 b	394	--
aten::contiguous	0.00%	114.000us	0.04%	1.093ms	91.083us	139.000us	0.01%	1.048ms	87.333us	0 b	0 b	14.31 Mb	0 b	12	--
aten::embedding	0.01%	195.000us	0.04%	999.000us	249.750us	270.000us	0.01%	985.000us	246.250us	0 b	0 b	4.52 Mb	0 b	4	--
aten::div	0.02%	432.000us	0.04%	951.000us	79.250us	937.000us	0.04%	937.000us	78.083us	0 b	0 b	17.64 Mb	17.64 Mb	12	--
aten::index select	0.01%	189.000us	0.02%	548.000us	137.000us	341.000us	0.01%	545.000us	136.250us	0 b	0 b	4.52 Mb	0 b	4	--
aten::relu	0.00%	85.000us	0.03%	727.000us	363.500us	83.000us	0.00%	466.000us	233.000us	0 b	0 b	8.00 Kb	0 b	2	--
aten::expand as	0.00%	111.000us	0.02%	406.000us	33.833us	144.000us	0.01%	390.000us	32.500us	0 b	0 b	0 b	0 b	12	--
aten::cat	0.00%	60.000us	0.02%	518.000us	172.667us	40.000us	0.00%	332.000us	110.667us	0 b	0 b	7.14 Mb	0 b	3	--
aten::slice	0.01%	175.000us	0.01%	312.000us	34.667us	232.000us	0.01%	293.000us	32.556us	0 b	0 b	0 b	0 b	9	--
aten:: cat	0.01%	279.000us	0.02%	446.000us	148.667us	229.000us	0.01%	292.000us	97.333us	0 b	0 b	7.14 Mb	0 b	3	--

Figure 4: Torch Autograd Profiler on Multimodal CASS

1. Convolution Operations (highlighted in red) take the majority of the time spent on the forward operation: **80.43%**
2. The arithmetic complexity / total number of FLOPs is higher for linear layers (highlighted in blue, likely arising from transformer's attention mechanisms) and FLOPs for Convolution Operation only accounts for **16.44%** of the total number of floating point operations.

Below are the results of different runs of MultiModal CASS Model on different configurations of NYU HPC.

Num GPUs	Type GPU	DataParallel	cuDNN Benchmarking	GPU Core Utilization	GPU Mem Utilization	Per Epoch Time (After Warmup)
1	RTX8000	No	No	97%	63%	8 min 51 secs
1	RTX8000	Yes	Yes	83%	54%	8 min 50 secs
4	RTX8000	Yes	Yes	44.31%	15.64%	5 min 4.8 secs

Table 4: nvidia-smi profile and time counters for model on varying number of GPUs

We deploy strong scaling on our problem statement and notice speedup and scaling efficiency as follows:

$$\text{speedup} = t_{serial}/t_{parallel} = 1.74$$
$$\text{scaling-efficiency} = t_{serial}/(t_{parallel} * p) = 43.47\%$$

To address the low GPU Core Utilization and Memory Utilization, we employed large batch_size (32 as opposed to 4 before) and increased the num_workers (14 as opposed to 4 before) to reduce the idle time for GPU and bump up the memory utilization. The speedup increased to **3.05** and the scaling efficiency bumped to **76.32%** between the serial and parallel implementations.

Num GPUs	Type GPU	DataParallel	cuDNN Benchmarking	GPU Core Utilization	GPU Mem Utilization	Per Epoch Time (After Warmup)
1	RTX8000	Yes	Yes	95.01%	58.31%	6 min 46 secs
4	RTX8000	Yes	Yes	76.43%	42.13%	2 min 13 secs

Table 5: Performance Improvement with larger batch_size and increased num_workers

Another small experiment we did is to compare the Tesla V100 GPU node vs Quadro RTX 8000 on single GPU performance.

	Quadro RTX8000	Tesla V100
CUDA Cores	4608	5120
Tensor Cores	576	640
Memory	48GB	32GB
GPU Core Utilization	95.01%	95.42%
GPU Mem Utilization	58.31%	46.29%
Per Epoch Time (after Warmup)	6 min 46 secs	5 min 37 secs

Table 6: Performance contrast between RTX8000 and V100 1-GPU implementations

	Training Time
Graph Convolution Network	5 hour 48min
Graph Attention Network	6 hour 29min
GraphSage	5 hour 46min

Table 7: Total time comparison for Graph Networks training on RTX8000 with Core Utilization at **87.76%** and Memory Utilization at **62.93%**

Surprisingly, even though RTX8000 has 48GB memory vs V100 has 32GB memory, the memory utilization in case of Tesla V100 is lesser. This can be seen in all our experiments.

8 Conclusions

In table 8, we summarize our best performing models. We observed that the use of self-supervised pretraining allows models to learn helpful language and image based priors that then helps them to perform better on downstream classification tasks. Furthermore, we also compared the performance of traditional architectures with GNNs.

Modality	Model	AUROC	Accuracy
Unimodal	ResNet-50 (CASS)	0.5405±0.045	0.5445±0.015
	Vit/Base-16 (CASS)	0.53386±0.009	0.59±0.017
Multimodal	ResNet-50 (CASS)	0.501±0.078	0.551±0.023
	Vit/Base-16 (CASS)	0.5±0.078	0.625±0.032
Multimodal GNN	GCN	0.514	0.547
	GAT	0.564	0.499
	SAGE	0.481	0.625

Table 8: In the above table we summarize our best performaing models, based on performance metrics described by the challenge on the test set.

References

Douwe Kiela, Hamed Firooz, Aravind Mohan, Vedanuj Goswami, Amanpreet Singh, Pratik Ringshia, and Davide Testuggine. The hateful memes challenge: Detecting hate speech in multimodal memes. *Advances in Neural Information Processing Systems*, 33:2611–2624, 2020.

Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks, 2016. URL `https://arxiv.org/abs/1609.02907`.

Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph attention networks, 2017. URL `https://arxiv.org/abs/1710.10903`.

William L. Hamilton, Rex Ying, and Jure Leskovec. Inductive representation learning on large graphs, 2017. URL `https://arxiv.org/abs/1706.02216`.

Pranav Singh, Elena Sizikova, and Jacopo Cirrone. Cass: Cross architectural self-supervision for medical image analysis. *arXiv preprint arXiv:2206.04170*, 2022.

Mathilde Caron, Hugo Touvron, Ishan Misra, Hervé Jégou, Julien Mairal, Piotr Bojanowski, and Armand Joulin. Emerging properties in self-supervised vision transformers. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 9650–9660, 2021.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, et al. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929*, 2020.

Victor Sanh, Lysandre Debut, Julien Chaumond, and Thomas Wolf. Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter. *arXiv preprint arXiv:1910.01108*, 2019.

Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. Mask r-cnn, 2017. URL `https://arxiv.org/abs/1703.06870`.