Ryan Ott
14862565
ryan.ott@student.uva.nl

1 Transfer Learning

1.1 Comparing Models

1.1.a Inference Speed per Image against Accuracy and Number of Parameters

With a Pearson correlation coefficient of 0.85 there seems to be a strong positive linear correlation between the top-1 accuracy and the inference speed. The ViT-B/32 model performs best overall with a higher accuracy for its inference speed than the trend suggests. Shown in Figure 1.

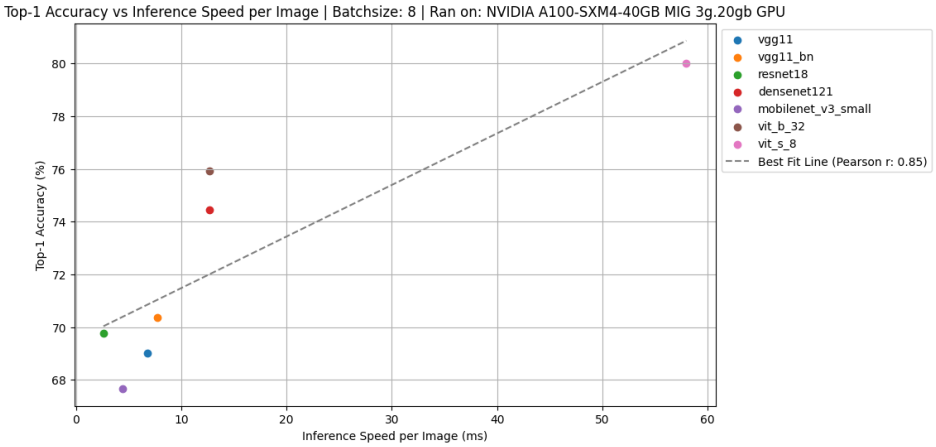

Figure 1: Top-1 Accuracy vs Inference Speed per Image

For inference speed against number of trainable parameters, we see no correlation. The number of trainable parameters should only play a role when the model is being trained, not during inferencing, so this result matches our expectations. Shown in Figure 2.

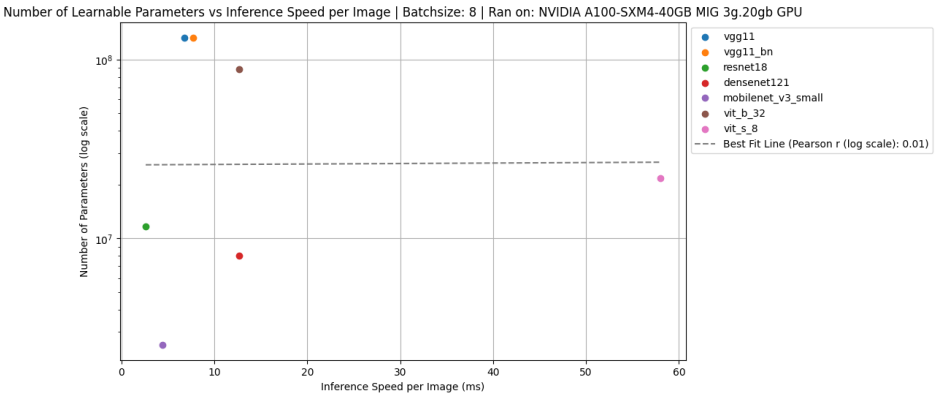

Figure 2: Number of Trainable Parameters vs Inference Speed per Image

1.1.b Inference Speed per Image with and without torch.no_grad()

With torch.no_grad() the inference speed should be faster because gradients are not computed and stored for the backward pass within the context manager's scope, thus requiring less operations and saving compute. This is confirmed by the results, the inference speed is slightly faster with torch.no_grad() across the models. Shown in Figure 3.

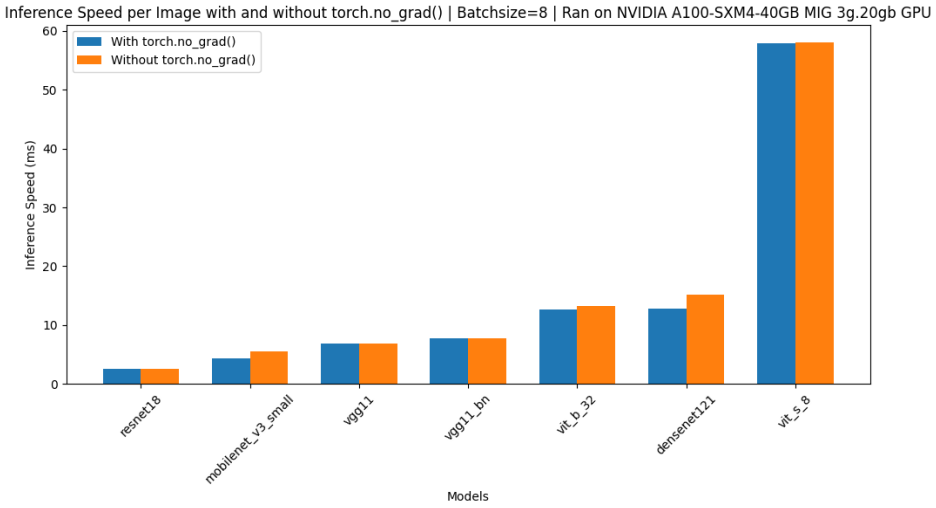

Figure 3: Inference Speed per Image with and without torch.no_grad()

1.1.c vRAM Usage with and without torch.no_grad()

Like with the inference speed, the vRAM usage should be lower with torch.no_grad() because gradients are not stored for the backward pass when performing tensor operations, freeing up memory. Much more considerably than the inference speed, the vRAM usage is lower when using torch.no_grad() as seen in the plot. Shown in Figure 4.

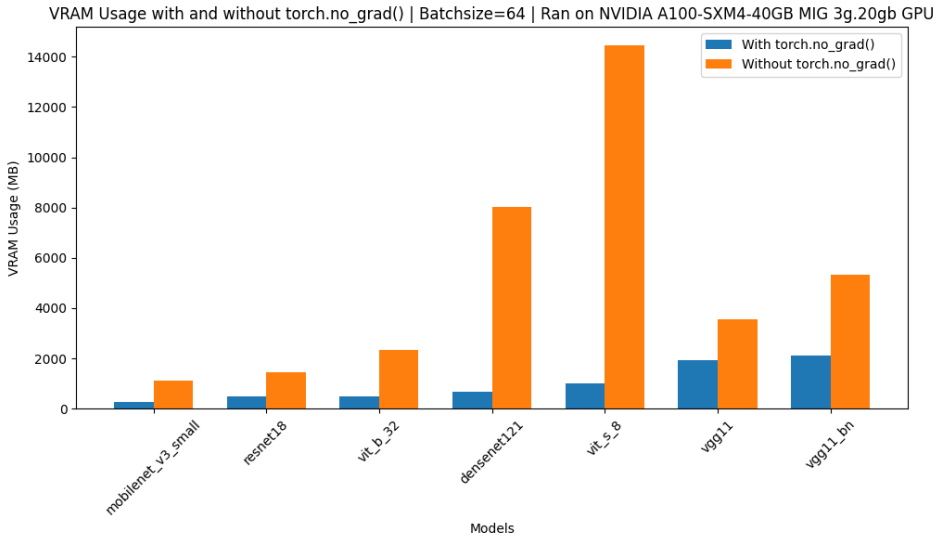

Figure 4: vRAM Usage with and without torch.no_grad()

1.2 Fine-tuning

1.2.a Retraining ResNet-18's Fully Connected Layer

Using the specifications and default hyperparameters from the assignment, the retrained model achieved a test accuracy of 58.55% in CIFAR-100.

1.2.b Increasing Model Performance using Data Augmentation

By applying `torchvision.transforms.RandomHorizontalFlip(p=0.5)` around half of the images in the training set are flipped horizontally. This should increase the model's performance because it is being trained on more varied data, thus making it more robust to different inputs. The model achieved a test accuracy of 59.27%, a small improvement.

1.2.c Last vs First Convolutional Layer

The first convolutional layers tend to capture lower-level features like edges, corners or textures, while the last layers capture higher-level features like shapes and objects in a larger receptive field. As such, better performance should be achievable by fine-tuning the last layers (along with the classifier layers) because they are more specialized to the downstream task than the first layers.

2 Visual Prompting

2.1 CLIP Baseline

2.1.a Top-1 Accuracy on CIFAR-10 and CIFAR-100 with CLIP ViT-B/32 Backbone

Dataset	Train Accuracy (%)	Test Accuracy (%)
CIFAR-10	88.73	88.94
CIFAR-100	63.56	63.15

Table 1: Zero-shot CLIP Top-1 Accuracy on CIFAR-10 and CIFAR-100

2.1.b Prompting CLIP for a New Classification Task

Prompt: *"This is an image mostly coloured ___"*
Labels: *"red", "green", "blue"*

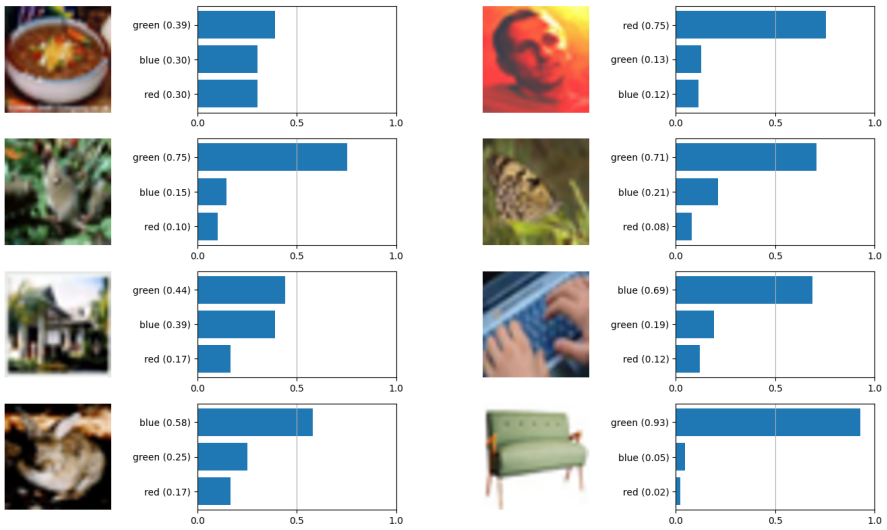

Figure 5: Primary Colour Identification

We see in Figure 5 that the model is able to identify the primary colour of the image with a high degree of accuracy. For images with a distinct primary colour, the model is able to identify it with high confidence, and for images with no clear distinct primary colour, it returns failry spread out probabilities. The contrastive learning approach of CLIP allows it to generalize well to new tasks using prompting.

Prompt: *"This image displays an object that is ___"*
Labels: *"human-made", "from-nature"*
Here too the model performs quite well, considering the task is slightly more abstract than identifying just primary colours. For clearly human-made objects like a computer keyboard or a couch, the model is able to identify it with high confidence. It does however mislabel some examples, like the image of the bunny being labelled as human-made. The more even spread of probabilities for the bunny image suggests that the model is not very confident in its prediction. See Figure 6.

It therefore seems that CLIP is able to generalize well to new tasks using prompting, but it is not perfect and can make mistakes when the task is more abstract. This approach to zero-shot learning is very powerful

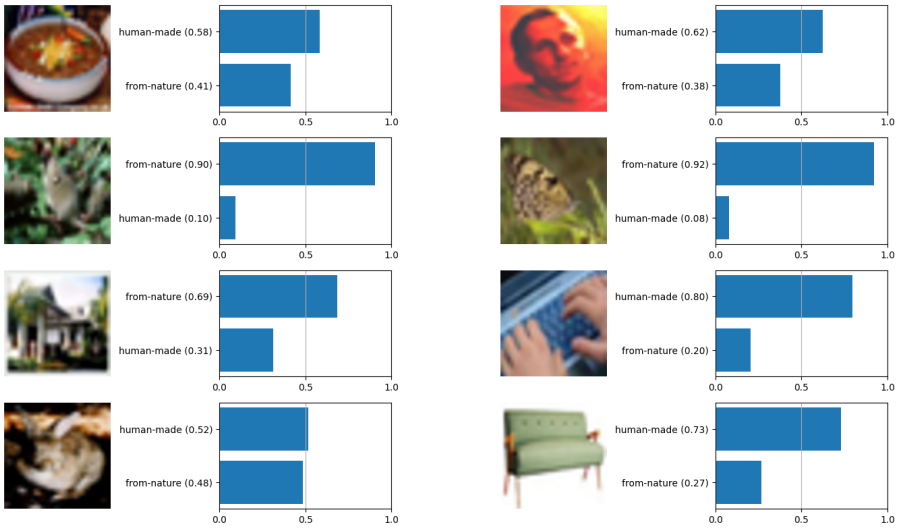

Figure 6: Distinguishing Human-made and Natural Objects

because it allows us to perform a wide range of tasks without having to train a new model for each task. This is especially useful when the task is not very complex and does not require a lot of training data, because training a new model for each task would be very inefficient.

2.2 Effect of Prompt Types on Classification Accuracy

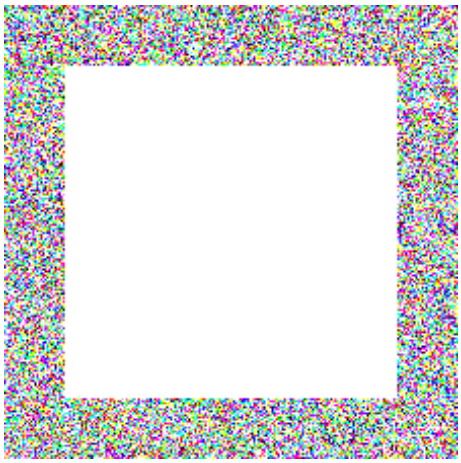

Figure 7: Fixed Patch Visual Prompt (Single Pixel Top-Left)

Figure 8: Padding Visual Prompt

Dataset	Fixed Patch Accuracy (%)	Padding Accuracy (%)	Baseline Accuracy (%)
CIFAR-10	89.57	92.74	88.94
CIFAR-100	64.49	71.39	63.15

Table 2: CLIP Top-1 Accuracy on CIFAR-10 and CIFAR-100's Test with Different Prompt Types Compared to Baseline

Comparing these results obtained using visual prompts on the CIFAR-10 and 100 test set to the baseline results in seen in Table 4, we see that the accuracy is higher for both prompt types, with the padding prompt type performing significantly better than the fixed patch prompt type. This is likely because the padding prompt type has a lot more trainable parameters (69840 for padding size 30) than the fixed patch prompt type (3 for patch size 1, one per channel), and thus is able adapt better to the task. Espeically for the CIFAR-100 dataset this improvement is noticeable. likely due to the fact that the CIFAR-100 dataset is more complex than the CIFAR-10 dataset, and thus requires more trainable parameters to perform well.

2.3 Deep Prompts

From Figure 3 in the assignment PDF we see how we can inject a learnable deep prompt into the CLIP model. In the diagram it is appended to the beginning of the sequence that is fed into a Transformer block. I initially accidentally appended it to the end of the sequence, but then chose to compare the two approaches which was quite interesting to see. I also investigated the effect of different numbers of deep prompts (1, 4 & 8) as well as the injection layer. The results are shown in Table 3, as well as a comparison between the best visual prompt accuracies and baseline readings in Table 4.

Dataset	Injection Layer	Prompts	Front Acc (%)	Back Acc (%)
CIFAR-10	0	1	95.32	**95.48**
		4	**96.31**	94.97
		8	**96.59**	94.67
	6	1	94.63	**94.82**
		4	**95.82**	94.46
		8	**96.00**	94.23
	11	1	91.17	**92.44**
		4	91.18	**92.35**
		8	89.25	**92.19**
CIFAR-100	0	1	75.90	**76.27**
		4	**78.43**	75.78
		8	**79.98**	74.86
	6	1	73.72	**74.29**
		4	**76.95**	74.01
		8	**78.11**	73.56
	11	1	**69.62**	68.39
		4	**69.61**	68.00
		8	**69.64**	67.67

Table 3: Image Classification Performance with CLIP using Different Numbers of Prompts, Injected at Different Layers and Appended to the Front or Back of the Sequence

Injecting prompts at early layers yields higher classification performance across the board than at later layers. This could be because it allows the model to learn more specialized lower-level features that are more useful for the classification task at hand, although this could differ with dataset too. The difference between injecting at layer 0 and 6 is also less pronounced than between layer 6 and 11, which might hint to the fact that deep prompts are most useful for lower-level to mid-level features.

Increasing the number of prompt generally correlates well with an increase in classification accuracy too, having the largest effect when injecting at early to middle layers on the CIFAR-100 dataset. This could be because the CIFAR-100 dataset is more complex than the CIFAR-10 dataset and the model benefits from having more guidance from the prompts.

Appending the prompts to the front of the sequence does seem to yield slightly better results than appending them to the back of the sequence. Although the attention mechanism of the Transformer should allow the model to learn to attend to the prompts regardless of their position in the sequence, appending our style of prompt (”This is a photo of a {}”) perhaps is better suited to being at the beginning, whereas more of a question or summary style prompt could be better suited to being at the end of the sequence. This is just specularitory and could be interesting for futher investigation.

We see that the best performing deep prompt model outperforms the visual prompt model, resulting in a considerable improvement over the baseline model.

Dataset	Deep Prompt Acc (%)	Visual Prompt Acc (%)	Baseline Acc (%)
CIFAR-10	96.59	92.74	88.94
CIFAR-100	79.98	71.39	63.15

Table 4: CLIP Top-1 Accuracy on CIFAR-10 and CIFAR-100’s Test Set with Different Prompt Types Compared to Baseline

The reason why deep prompts might outperform the visual prompts is that they can interact with the latent representation of the image in the model, allowing more complex interactions between the prompt and the image. This can guide the network better than the visual prompts which only can augment the raw input data.

2.4 Robustness to Noise

I investigated how robust CLIP and Resnet-18 are to distributional shifts on CIFAR-100 by adding Gaussian noise to the test set. The previously trained CLIP model using a visual padding prompt of size 30 on the CIFAR-100 dataset was used for this, while for Resnet-18 the fine-tunned model was used where only the fully connected layer was retrained - but not the one being shown augmented data during training as to not bias it towards having seen noisy data before.
Previously, in this configuration CLIP achieved a test accuracy of 71.39% and Resnet-18 achieved a test accuracy of 58.55%. The results of adding Gaussian noise to the test set are shown in Table 5.

Test Dataset	CLIP Accuracy (%)	Resnet-18 Accuracy (%)
CIFAR-100	71.39	58.55
CIFAR-100 + Gaussian Noise	61.89	25.15

Table 5: CLIP and Resnet-18 Top-1 Accuracy on CIFAR-100's Test Set with and without Gaussian Noise

Where we can see a clear drop in accuracy for both models, although CLIP seems much more robust to the noise than Resnet-18, CLIP lost around 10% accuracy while Resnet-18's more than halved.
The reason this might occur could come down to the learned visual prompts. CLIP's visual prompts perhaps allow it to learn more robust features that are more invariant to noise, with the padding prompt type of size 30 being able to attend to a large part of the image and thus being able to learn less sensitive features. Resnet-18 on the other hand only had its final fully connected layer retrained, which might not help it to be more robust to noise, if anything it might make it more inclined to overfit to the training data it saw during fine-tuning.

2.5 Visual Prompts Effectiveness on CIFAR-110

For investigating the effectiveness of learned visual prompts the best performing visual prompt models, one trained on CIFAR-10 and one trained on CIFAR-100. When comparing the performance of these on the 110-way classification task, we see that the model trained on CIFAR-100 performs better than the model trained on CIFAR-10, achieving a slightly higher test top-1 accuracy (see Table 6).

Dataset	CIFAR-10 Prompt Acc (%)	CIFAR-100 Prompt Acc (%)
CIFAR-110	67.705	70.525

Table 6: CLIP Top-1 Accuracy on CIFAR-110's Test Set with Different Prompt Types

This is the expected result because the model trained on CIFAR-100 has already seen most of the classes in CIFAR-110, and thus is able to perform better on it than the model trained on CIFAR-10. The fact that having only seen $< 10\%$ of the classes in CIFAR-110 the model trained on CIFAR-10 is able to achieve a test accuracy of 67.705% is quite impressive, and shows the power of CLIP's zero-shot learning capabilities and the learned visual prompts.

2.6 Transfer Learning vs Prompting

2.6.a What are some advantages of prompting over linear probing in terms of robustness against distributional shifts?

- Parameter Optimization Efficiency
- Less Compute for Input Manipulation

2.6.b What are some advantages of prompting over linear probing concerning computational resources?

- Prompt Dataset Overfitting Prevention

2.6.c What represents some drawback(s) of prompting compared to linear probing concerning memory requirements?

- Increased Memory Load for Prompting

2.6.d What represents a disadvantage of prompting compared to full fine-tuning and linear probing in terms of adaptability to downstream tasks?

- Limited Data Space Manipulation

3 Graph Neural Networks

3.1 Adjacency Matrices

3.1.a

First graph:
[[0, 1, 1, 1, 1, 1, 1], [1, 0, 0, 0, 0, 0, 0], [1, 0, 0, 0, 0, 0, 0], [1, 0, 0, 0, 0, 0, 0], [1, 0, 0, 0, 0, 0, 0], [1, 0, 0, 0, 0, 0, 0], [1, 0, 0, 0, 0, 0, 0]]

Second graph:
[[0, 1, 1, 1, 1, 1, 1], [1, 0, 1, 0, 0, 0, 1], [1, 1, 0, 1, 0, 0, 0], [1, 0, 1, 0, 1, 0, 0], [1, 0, 0, 1, 0, 1, 0], [1, 0, 0, 0, 1, 0, 1], [1, 1, 0, 0, 0, 1, 0]]

Third graph:
[[0, 1, 0, 0, 0, 1], [1, 0, 1, 0, 0, 0], [0, 1, 0, 1, 0, 0], [0, 0, 1, 0, 1, 0], [0, 0, 0, 1, 0, 1], [1, 0, 0, 0, 1, 0]]

3.1.b

[[2 0 1 0 1 0], [0 2 0 1 0 1], [1 0 2 0 1 0], [0 1 0 2 0 1], [1 0 1 0 2 0], [0 1 0 1 0 2]]

3.1.c

$(A^2)_{uv}$ represents the number of paths of length 2 from node u to node v in the graph. This extends to $(A^n)_{uv}$ representing the number of paths of length n from node u to node v in the graph.

3.2 Graph Convolution

3.2.a Where in the Equation does the Structural Information on the Graph Appear?

The structural information on the graph appears in the equation through the term

$$\sum_{u \in N(v)} \frac{h_u^{(l)}}{|N(v)|}$$

This sum represents the aggregation of the embeddings of the neighbours of node v, weighted by the inverse degree of each neighbour node. This is then multiplied by the weight matrix $W^{(l)}$ to combine the embeddings of the neighbours into a single embedding for node v after adding the self-connection term $B^{(l)}h_v^{(l)}$ and applying the nonlinearity σ.

3.2.b How is the graph convolution related to message passing neural networks?

A graph convolution can be seen as a subclass of a message passing scheme, where in a graph convolution we are limited to the immediate neighbours of a node and the message and update functions are parametrized by linear transformations. In a message passing scheme we can define the message and update functions more flexibly, and we can propagate messages to nodes that are not immediate neighbours of a node.

3.3 Graph Convolutions Continued

3.3.a Rewriting the Graph Convolution in Matrix Form

Using the Adjacency matrix A and the Degree matrix D, we can rewrite the graph convolution to matrix form as follows:

$$H^{(l+1)} = \sigma\left(W^{(l)}D^{-1}AH^{(l)} + B^{(l)}H^{(l)}\right)$$

Where $D^{-1}AH^{(l)}$ represents the normalized sum of the embeddings of the neighbours.

3.3.b Which function does this graph convolutional network use to aggregate over neighboring nodes?

The graph convolutional network uses the mean function to aggregate over neighbouring nodes. Its the sum of all neighbouring embeddings divided by the number of neighbours.

3.3.c Is Rewriting to Matrix Form Possible for All Aggregation Functions?

No, rewriting to matrix form is not possible for all aggregation functions. The aggregation function must be applicable uniformly across all nodes and their neighbours. Max pooling for example is not applicable because it would require a different number of neighbours for each node, and thus a different number of terms in the sum for each node. This coupled with its nonlinearity makes it impossible to rewrite to matrix form.

3.4 Connections to Standard Neural Network Architectures

Firstly, scaling to larger graphs would be difficult because the adjacency matrix would grow quadratically with the number of nodes in the graph. This would make the input to the feed-forward neural network very large and thus computationally expensive to train and evaluate. For social networks for example with millions of people, this would be infeasible.

Secondly, the loss of structural information would be problematic. The adjacency matrix would be flattened and concatenated with the node embeddings, and thus the structural information would be lost. This would make it difficult to learn from the graph structure, which is the main advantage of graph neural networks. For example, in chemistry the structure of the molecule is very important for its properties, and thus this approach would not be suitable.

Thirdly, the approach lacks invariance to node permutations. The same graph can be represented by multiple adjacency matrices, depending on the ordering of nodes. When concatenating a flattened adjacency matrix with node embeddings, different permutations of the same graph would result in different input vectors to the feed-forward neural network. This lack of permutation invariance of a FFNN could lead to inconsistent model performance, as the network might interpret these permutations as distinct graphs. This could be an issue in for example molecular structure classification, where different representations of the same graph should yield the different predictions.

3.5 Relation to Transformers

3.5.a How can a Transformer Model be Seen as a Graph Neural Network?

The nodes in this case would be the words in the sentence, and the edges would be the attention weights between the nodes. The graph structure would be a fully connected graph, where each node is connected to every other node, and the attention scores between words represent the edge weights.

3.5.b How Can the Transformer Model Process Inputs for Which the Order Matters While Staying Permutation Invariant in its Architecture and Computation?

Transformers encode the order of the input sequence in positional embeddings, which are added to the input embeddings before being fed into en/decoder. This allows the model to retain information and learn from the order of the input sequence. The self-attention mechanism which is used to compute the attention scores is permutation invariant, and thus the model is permutation invariant in its architecture and computation.