Assignment 2

Question 1.1

a. What is the expression of $f_{1,1}$ according to the above Figure 1? What do you think is missing and how can we mitigate this problem?

$f_{1,1}$ can be constructed by centering the convolution $h_{0,0}$ over the pixel $g_{1,1}$ in image g. The equation for obtaining $f_{1,1}$ is thus,

$$\begin{aligned} f_{1,1} = {} & g_{2,2}h_{-,-} + g_{2,1}h_{-,0} + g_{2,0}h_{-,+} + g_{1,2}h_{0,-} \\ & + g_{1,1}h_{0,0} \\ & + g_{1,0}h_{0,+} + g_{0,2}h_{+,-} + g_{0,1}h_{+,0} + g_{0,0}h_{+,+} \end{aligned}$$

However, some of these pixel values are out of bound for the image g, specifically the pixels $g_{2,0}, g_{1,0}, g_{0,2}, g_{0,1}, g_{0,0}$. These pixels are necessary to calculate the convolution $g \circledast h$ centered at $g_{1,1}$. Typical method to mitigate this problem is using padding of the image g. A padding method like zero padding simple assign 0 to these padding pixels.

b. i) Run the given code with net_type='Net1', and vary different conv_type from 'valid', 'replicate', 'reflect', 'circular', 'sconv' and 'fconv', and report the validation and test scores in the form of a table.

Type of convolution : valid
Type of network : Net1

Results for validation dataset {0: 100.0, 1: 100.0, 2: 100.0, 3: 100.0, 4: 100.0, 5: 100.0, 6: 100.0, 7: 100.0, 8: 100.0, 9: 100.0}
mean: 100.0000 std: 0.0000 for validation
Results for test dataset {0: 0.3, 1: 0.1, 2: 0.0, 3: 0.1, 4: 0.0, 5: 0.0, 6: 0.1, 7: 0.3, 8: 0.0, 9: 0.0}
mean: 0.0900 std: 0.1136 for test

Type of convolution : replicate
Type of network : Net1

Results for validation dataset {0: 98.6, 1: 98.1, 2: 98.8, 3: 98.5, 4: 97.1, 5: 98.8, 6: 98.2, 7: 98.0, 8: 98.5, 9: 98.5}

mean: 98.3100 std: 0.4784 for validation
Results for test dataset {0: 92.9, 1: 92.8, 2: 88.8, 3: 94.7, 4: 95.3, 5: 91.7, 6: 96.3, 7: 94.7, 8: 95.8, 9: 95.0}
mean: 93.8000 std: 2.1629 for test

Type of convolution : reflect
Type of network : Net1

Results for validation dataset {0: 100.0, 1: 100.0, 2: 100.0, 3: 100.0, 4: 100.0, 5: 100.0, 6: 100.0, 7: 100.0, 8: 100.0, 9: 100.0}
mean: 100.0000 std: 0.0000 for validation
Results for test dataset {0: 0.0, 1: 0.0, 2: 0.0, 3: 0.0, 4: 0.0, 5: 0.0, 6: 0.0, 7: 0.0, 8: 0.0, 9: 0.0}
mean: 0.0000 std: 0.0000 for test

Type of convolution : circular
Type of network : Net1

Results for validation dataset {0: 99.9, 1: 99.9, 2: 99.7, 3: 99.9, 4: 98.9, 5: 99.7, 6: 99.7, 7: 99.5, 8: 99.8, 9: 99.4}
mean: 99.6400 std: 0.2939 for validation
Results for test dataset {0: 81.5, 1: 82.7, 2: 77.4, 3: 86.9, 4: 83.2, 5: 82.6, 6: 86.5, 7: 79.8, 8: 81.1, 9: 84.4}
mean: 82.6100 std: 2.7504 for test

Type of convolution : sconv
Type of network : Net1

Results for validation dataset {0: 98.8, 1: 99.0, 2: 98.8, 3: 99.1, 4: 99.0, 5: 99.3, 6: 98.7, 7: 98.8, 8: 98.9, 9: 98.5}
mean: 98.8900 std: 0.2119 for validation
Results for test dataset {0: 6.5, 1: 6.9, 2: 6.5, 3: 6.1, 4: 6.2, 5: 6.5, 6: 6.2, 7: 6.1, 8: 6.4, 9: 6.3}
mean: 6.3700 std: 0.2326 for test

Type of convolution : fconv
Type of network : Net1

Results for validation dataset {0: 88.4, 1: 87.6, 2: 88.6, 3: 89.9, 4: 88.1, 5: 89.9, 6: 85.8, 7: 87.1, 8: 88.2, 9: 87.1}
mean: 88.0700 std: 1.1984 for validation
Results for test dataset {0: 88.4, 1: 88.6, 2: 88.6, 3: 89.9, 4: 89.1, 5: 89.9, 6: 88.9, 7: 88.4, 8: 88.9, 9: 87.8}
mean: 88.8500 std: 0.6249 for test

convolution type	validation accuracy (mean)	validation accuracy (std)	test accuracy (mean)	test accuracy (std)
valid	100.0000	0.0000	0.0900	0.1136
replicate	98.3100	0.4784	93.8000	2.1629
reflect	100.0000	0.0000	0.0000	0.0000
circular	99.6400	0.2939	82.6100	2.7504
sconv	98.8900	0.2119	6.3700	0.2326
fconv	88.0700	1.1984	88.8500	0.6249

b. ii) Look at the data samples from the train and test sets provided in the README.md. Based on the structure of the images, guess the pattern of class label 0 and class label 1. How do the samples in the train set differ from the ones in the test set?

The pattern for class label 0 appears to have a red small rectangle on the left of the green small rectangle. The pattern for class label 1 is having the green small rectangle to the left of the red small rectangle.

The test set distribution appears to differ from the train set distribution in that the spatial positions of the rectangles are restricted to different regions of the images. In the training data for label 0, the small rectangles appear in the upper regions of the image, but for the same label 0 in the test set the rectangles appear in the lower region of the image. The opposite is true for the images labeled label 1.

c. i) From the network architecture of ‘Net1’, infer the variables that affect the conv_type. What is the difference between conv_type ‘valid’, ‘sconv’ and ‘fconv’?

For Net1, the variables that affect the type of convolutions used in the network architecture are the padding type and padding size used for the convolutional layers. The conv_type 'valid', 'sconv', and 'fconv' all use zero padding for the convolutions, but increases in the padding size. 'valid' uses zero padding but pad for a size of 0, which equates to not padding. 'sconv' uses zero padding for a pad size of 1, and 'fconv' uses zero padding for a pad size of 2.

c. ii) Why do the test accuracies of conv_type='valid', 'sconv' and 'fconv' (i.e., acc_valid, acc_sconv and acc_fconv) follow the order – acc_valid < acc_sconv < acc_fconv?

The model architecture of Net1 is composed of 4 convolution layers, an adaptive max pooling layer, and then linear layer that maps the maxpooled channel values to a dimension of 2 for the 2 target classes. The order of the test accuracies for the different conv types with different padding size relates to the image size, and the size of the output after the 4 convolution layers. We can calculate the size of the outputs to an convolution layer with the following formula,

$$\text{output size} = \frac{\text{input size} + 2 * \text{padding size} - \text{kernel size}}{\text{stride size}} + 1$$

For conv_type = 'valid', this means that with the input image size of (32, 32) per channel, the output size after the 4 convolution is only (2, 2) per channel. This means that spatial information is largely lost after the convolutions, and the max pool layer select only the strongest signal from the (2, 2) for each channel. As established in (b.ii), the pattern that distinguishes the classes arise from the a combination of spatial and channel information: red rectangles on the left for label 0 and green rectangles on the left for label 1. With spatial information nearly all lost, conv_type of 'valid' results in a very poor performance. It can be observed that as the padding size increase to 1 for sconv and 2 for fconv, more spatial information are preserved in the final output. (4, 4) for sconv and (6, 6) for fconv. With fconv we then observe that enough spatial information are preserved such that a higher test accuracy of 88.8 can be achieved.

c. iii) Why is the test accuracy of conv_type='reflect' less than 'fconv'?

With conv type of reflect, the padding type used for all 4 convolution layers of the network is 'reflect', which pads the image with pixel values "reflected" from the edges from the image. For example with a pad size of 1 using reflect padding,

$$\begin{matrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{matrix} \quad \Rightarrow \quad \begin{matrix} 5 & 4 & 5 & 6 & 5 \\ 2 & 1 & 2 & 3 & 2 \\ 5 & 4 & 5 & 6 & 5 \\ 8 & 7 & 8 & 9 & 8 \\ 5 & 4 & 5 & 6 & 5 \end{matrix}$$

We can observe that the model trained with reflect achieves a perfect accuracy on the validation data where label 0 have rectangles red, green (in that order) in the upper region of the image and label 0 have rectangles green, red (in that order) in the lower region of the image. Intuitively, by reflecting the pixels for a pad size of 2, the signals of the small rectangles on their respective sides are reflected to the opposite sides. However, this reflection forms a pattern in the signals that the model overfits on during training. This is observed by its very poor generalization when the same red-green order of label 0 is translated to the lower portion of the image. It led to a complete misclassification and a test set accuracy of 0.

c. iv) Why is the test accuracy of conv_type='replicate' more than 'fconv'?

With replicate the edge pixels are replicated on the padding pixels, unlike reflect, replicate does not change the ordering of the signals, but rather magnify them. This means that the signals of red-left green-right in the label 0 of the training set translates well to the similar signals of the label 0 data in the test set, despite the translation of the signals to the lower portion of the image. Compared to fconv that only pads with 0, which does not help improve the signals after convolutions, replicate leads to a better performance.

d. i) Run the given code with net_type='Net2', and vary different conv_type from 'valid', 'replicate', 'reflect', 'circular', 'sconv' and 'fconv', and report the validation and test scores in the form of a table.

```
Type of convolution : valid
Type of network : Net2
```

```
Results for validation dataset {0: 100.0, 1: 100.0, 2: 100.0, 3: 100.0, 4: 100.0, 5: 100.0, 6: 100.0, 7: 100.0,
8: 100.0, 9: 100.0}
mean: 100.0000 std: 0.0000 for validation
Results for test dataset {0: 0.0, 1: 0.0, 2: 0.0, 3: 0.0, 4: 0.0, 5: 0.0, 6: 0.0, 7: 0.0, 8: 0.0, 9: 0.0}
mean: 0.0000 std: 0.0000 for test
```

```
Type of convolution : replicate
Type of network : Net2
```

```
Results for validation dataset {0: 100.0, 1: 100.0, 2: 100.0, 3: 100.0, 4: 100.0, 5: 100.0, 6: 100.0, 7: 100.0,
8: 100.0, 9: 100.0}
mean: 100.0000 std: 0.0000 for validation
Results for test dataset {0: 0.0, 1: 0.0, 2: 0.0, 3: 0.0, 4: 0.0, 5: 0.0, 6: 0.0, 7: 0.0, 8: 0.0, 9: 0.0}
mean: 0.0000 std: 0.0000 for test
```

```
Type of convolution : reflect
Type of network : Net2
```

```
Results for validation dataset {0: 100.0, 1: 100.0, 2: 100.0, 3: 100.0, 4: 100.0, 5: 100.0, 6: 100.0, 7: 100.0,
8: 100.0, 9: 100.0}
mean: 100.0000 std: 0.0000 for validation
Results for test dataset {0: 0.0, 1: 0.0, 2: 0.0, 3: 0.0, 4: 0.0, 5: 0.0, 6: 0.0, 7: 0.0, 8: 0.0, 9: 0.0}
mean: 0.0000 std: 0.0000 for test
```

```
Type of convolution : circular
Type of network : Net2
```

```
Results for validation dataset {0: 100.0, 1: 100.0, 2: 100.0, 3: 100.0, 4: 100.0, 5: 100.0, 6: 100.0, 7: 100.0,
8: 100.0, 9: 100.0}
mean: 100.0000 std: 0.0000 for validation
Results for test dataset {0: 0.0, 1: 0.0, 2: 0.0, 3: 0.0, 4: 0.0, 5: 0.0, 6: 0.0, 7: 0.0, 8: 0.0, 9: 0.0}
mean: 0.0000 std: 0.0000 for test
```

```
Type of convolution : sconv
Type of network : Net2
```

```
Results for validation dataset {0: 100.0, 1: 100.0, 2: 100.0, 3: 100.0, 4: 100.0, 5: 100.0, 6: 100.0, 7: 100.0,
8: 100.0, 9: 100.0}
mean: 100.0000 std: 0.0000 for validation
Results for test dataset {0: 0.0, 1: 0.0, 2: 0.0, 3: 0.0, 4: 0.0, 5: 0.0, 6: 0.0, 7: 0.0, 8: 0.0, 9: 0.0}
mean: 0.0000 std: 0.0000 for test
```

```
Type of convolution : fconv
Type of network : Net2
```

```
Results for validation dataset {0: 100.0, 1: 100.0, 2: 100.0, 3: 100.0, 4: 100.0, 5: 100.0, 6: 100.0, 7: 100.0,
8: 100.0, 9: 100.0}
mean: 100.0000 std: 0.0000 for validation
Results for test dataset {0: 0.0, 1: 0.0, 2: 0.0, 3: 0.0, 4: 0.0, 5: 0.0, 6: 0.0, 7: 0.0, 8: 0.0, 9: 0.0}
mean: 0.0000 std: 0.0000 for test
```

convolution type	validation accuracy (mean)	validation accuracy (std)	test accuracy (mean)	test accuracy (std)
valid	100.0000	0.0000	0.0000	0.0000
replicate	100.0000	0.0000	0.0000	0.0000
reflect	100.0000	0.0000	0.0000	0.0000
circular	100.0000	0.0000	0.0000	0.0000
sconv	100.0000	0.0000	0.0000	0.0000
fconv	100.0000	0.0000	0.0000	0.0000

d. ii) Do the test accuracies for each of the conv_types in net_type='Net2' increase or decrease w.r.t their corresponding conv_type counterparts in 'Net1'?

The test accuracies for all of the different conv_types decrease in Net2 when compared to Net1. In fact, all test accuracies for Net2 are 0, and their respective validation accuracies are 100.

d. iii) State the reason behind this change in test accuracies

In Net2 after the convolution layers, instead of applying adaptive max pooling which extracts the strongest signal per channel, a flattening operation flattens the tensor to a single dimension. For example, with "conv_types=fconv", the dimensions after the convolution layers is $[64, 6, 6]$, as previously discussed in (c. ii), but the flattening operation flattens the tensor to give a dimension of $[2304]$. By doing this, Net2 has destroyed the spatial information extracted by the convolutions, which as mentioned in the previous sub-questions, is crucial for generalizing on the test dataset. More specifically, instead of extracting the signals regardless of location on the image like a max pooling layer would, the flattening operation restructures the data to a single dimension. Because the signals occur at different regions of the image for the train and test sets, the trained model cannot generalize to the test data where the signals of the red and green rectangles occur in a different section of the flattened data.

Question 1.2

a) Plot the accuracy of the model during inference respect to the angle of rotation of test images and include the figure in your answers. What can

you conclude about rotational invariance of CNNs from the plot? Explain why are there larger accuracy at certain degrees.

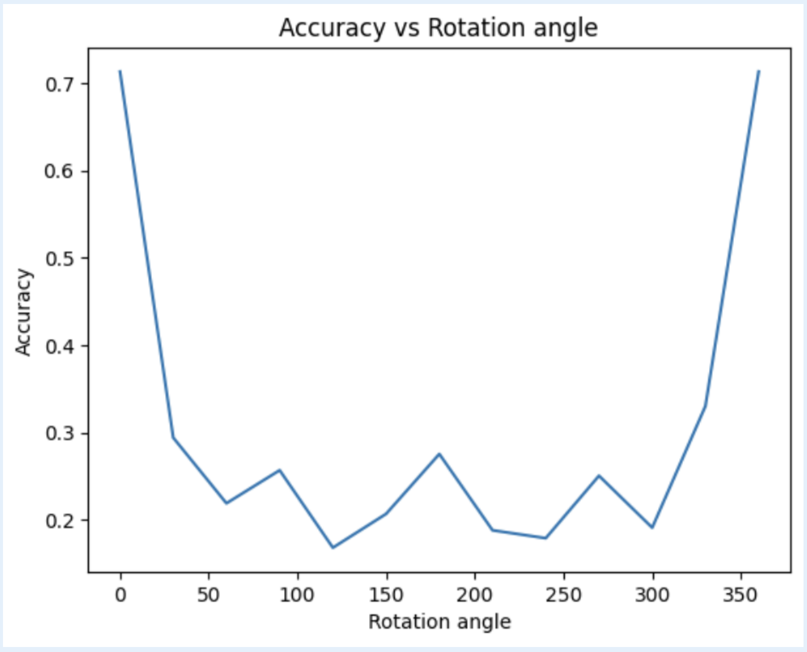

Figure 1: Accuracy of model during inference respect to angle of rotation of test images. Model not trained on rotated images.

This experiment shows that the trained CNN model could only classify objects in the image (with high accuracy) that are in their original orientation, either not rotated or rotated 360 degrees. This implies that CNNs are not rotationally invariant, because the performance of the trained classifier varied for images rotated for different angles. When the images are rotated to a different angle, the trained model could no longer classify the objects in the image. Since the model is trained on non-rotated images, it could only classify objects that are not rotated, hence the higher accuracies closer to 0 or 360 degrees of rotation.

b) Train a new model like the previous one but adding rotational augmentation to its datasets, and plot its accuracy during inference with respect to the angle of rotation of the test images. Include the figure in your answers. Describe the differences observed compared to the plot in Question 1.2(a) and explain the reasons for these differences.

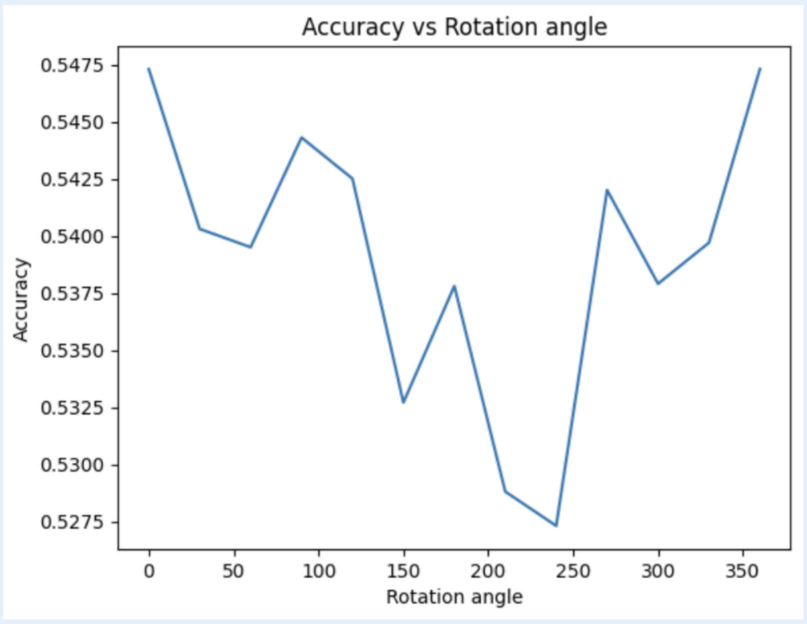

Figure 2: Accuracy of model during inference respect to angle of rotation of test images. Model trained with random rotation transformation of up to 360 degrees.

By retraining the model with rotation augmented images, the CNN model is capable of performing better on rotated images at test time. We see an improved test time accuracy for most angles between 0 and 360. This shows that by providing the CNN model with images that are rotated during training, the model has learned to classify rotated objects, in addition to their non-rotated versions. However, we also observe that the accuracies at the extreme angles decreased, from 0.7 down to 0.48. This shows that there is a trade-off by introducing rotation to the training images, as the model now aims to generalize to a broader range of rotations rather than optimizing for just the unrotated images.

Question 2.1

a) Discuss the computational challenges that arise in Transformer models when handling long input sequences. Your response should: (a) Briefly describe why long sequences are challenging for Transformers. (b) Suggest a method to address this challenge.

Transformers rely on the attention mechanism in encoding the pairwise interactions between all tokens of the input sequence. Capturing this pairwise interaction in the input sequence leads to an exponential computational complexity $O(n^2)$. More specifically, the matrix multiplication for calculating the Query-Key similarity ($QK^\top$) that measures how much each token should attend to others tokens produces a $n \times n$ matrix, and for each element in the matrix, the dot product involves d_k multiplications and additions. The total number of operations for computing $QK^\top$ which has shapes $n \times d_k$ and $d_k \times n$ respectively, is thus $O(n^2 d_k)$. As n grows to a larger sequence length, the computation grows quadratically, which becomes computationally intensive to calculate.

A method for addressing the quadratic growing computation complexity of the attention mechanism is the use of sparse attention. Methods like local attention limits the attention of a token to only a fixed window size of w where $w \ll n$, each token attends to only w surrounding tokens, and the computational complexity reduces to $O(nw)$.

b) Discuss the receptive field of Transformer and Convolutional Neural Networks (CNNs), and their capability to capture long-range dependencies.

Receptive field refers to the region of the input data that a particular neuron or feature in the network is sensitive to, which means the receptive field determines that amount of contextual information from the input a neuron can utilize.

Convolutional Neural Network have a local receptive field that is determined by the size of the convolutional kernels as each neuron of a convolutional layer is connected to a local region of its input through the kernels. This means that CNNs struggle with capturing long range dependancies, since usual kernel of convolutions are restricted to a small size of 3×3 or 5×5. Although the receptive field of CNNs grows with the number of layers, each layer of the network only adds a small fixed amount to the total receptive field due to the usually small kernel size and small stride size.

Transformers are capable of capturing long range dependencies as the attention mechanism models a global receptive field where each token attends to every other tokens in the input sequence. In other words, each feature of the network is sensitive to, and depends on, all other regions of the input sequence. This allows transformers to utilize all contextual information in the input sequence.

c) Explain why the scaling factor $\sqrt{d_k}$ is used in the self-attention mechanism (refer to Eq. (4)). Describe its effect on the computation of attention scores.

Without scaling by $\sqrt{d_k}$ in the equation $\text{Attention}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{softmax}\left(\frac{\mathbf{Q}\mathbf{K}^\top}{\sqrt{d_k}}\right)\mathbf{V}$, the variance of the dot product in the nominator grows with dimensionality of the query and key vectors d_k, which leads to larger values as d_k increases. This scaling prevents too large values from entering the softmax, which can lead to peaked softmax outputs that destabilize the training through vanishing gradients or can prevent a more balanced attention weights that attends to a range of tokens (due to the peaked output).

We can show that scaling by $\sqrt{d_k}$ is effective by first assuming that components of the vectors Q_i and K_j are independent and identically distributed random variables. This means $\mathbb{E}[Q_{il}] = \mathbb{E}[K_{jl}] = 0$ and $Var(Q_{il}) = Var(K_{jl}) = \sigma^2$ for all l in $[1 : d_k]$. The dot product between Q_i and K_j is,

$$Q_i \cdot K_j = \sum_{l=1}^{d_k} Q_{il} K_{jl}$$

Since the sum of the random variables has a variance equal to the sum of their variance when the variables are independent,

$$Var(Q_i \cdot K_j) = \sum_{l=1}^{d_k} Var(Q_{il} K_{jl})$$

The variance of the product of the two independent random variables can be calculated with,

$$Var(Q_{il}K_{jl}) = \underbrace{\mathbb{E}[Q_{il}^2 K_{jl}^2]}_{=\mathbb{E}[Q_{il}^2]\mathbb{E}[K_{jl}^2]} - (\underbrace{\mathbb{E}[Q_{il}K_{jl}]}_{=\mathbb{E}[Q_{il}]\mathbb{E}[K_{jl}]=0})^2$$

And since that the two variables have both have a mean of zero, the expectation of their square is their variance,

$$Var(Q_{il}) = \mathbb{E}[Q_{il}^2] - (\mathbb{E}[Q_{il}])^2 = \sigma^2$$
$$\Rightarrow \mathbb{E}[Q_{il}^2] = \sigma^2$$

Therefore the sum of variance over l is,

$$Var(Q_i \cdot K_j) = \sum_{l=1}^{d_k} Var(Q_{il}K_{jl}) = \sum_{l=1}^{d_k} \sigma^4 = d_k \sigma^4$$

Variance of the dot product in the nominator is then normalized with division by $\sqrt{d_k}$:

$$Var\left(\frac{Q_i \cdot K_j}{\sqrt{d_k}}\right) = \frac{Var(Q_i \cdot K_j)}{d_k} = \frac{d_k \sigma^4}{d_k} = \sigma^4$$

d) Explain the advantages of using multiple attention heads compared to a single attention mechanism with the same total computational cost.

The advantage of dividing the total attention of the model amongst multiple heads, where d_{model} is the total hidden dimension of the model and each head thus have a hidden dimension of d_{model}/h for h number of heads, is that the total computational cost can be kept the same while allowing different attention head to learn different feature representations and dependencies in the input sequence. In a single attention head, the expressiveness of the attention is limited by a linear combination of the value matrix V, with attention weights determined by the softmax output (softmax $\left(\frac{\mathbf{QK}^\top}{\sqrt{d_k}}\right)$). This implies that the rank of the attention output matrix is determined by the minimum of the ranks of the softmax output or V. If the softmax output is low rank, ie, that the attention focus on certain tokens, the attention output will be low rank and the expressiveness of the single attention head is limited. By using multiple attention heads, each with a lower dimension of d_{model}/h for h different heads, the total rank in the output matrix when the multiple heads are combined can be higher, and more complex dependencies in the input sequence can be learned.

Question 2.3

c) Why is the MLP wide and not deep?

The choice of a wide but not deep MLP block in the Transformer block is meant to maintain computational efficiency while allowing the MLP to model complex feature representations of the individual tokens from attention output. In theory, the Universal Approximation Theorem should allow a single hidden layer to approximate any continuous function given the layer is sufficiently wide. The choice of a wide but shallow MLP block means that it should be able to extract the desired rich features while avoiding the common issues with deeper networks, such as vanishing gradients, training instability, etc. Additionally, Transformer models often contain multiple decoder blocks stacked ontop of one another, allowing them to capture hierarchical representations while keeping the individual block computationally efficient, as wider MLPs are more amenable to parallelization compared to deeper sequential layers.

Question 2.4

a) What happens if we do not make use of any positional embeddings?

Without positional embedding, the token embeddings that enter the transformer block do not carry any information about the token's position or its relationship to other tokens based on order. Because the self-attention mechanism computes the attention scores using the dot product of queries and keys derived from the token embeddings, without positional information, the mechanism cannot capture any positional information. This will lead to the phrases "John sat on the couch" and "Couch on the sat John" being indistinguishable to the model, ie, similar to a bag-of-words treatment. With loss ability to model syntactic structure and contextual information, the transformer model without positional embeddings would perform very poorly on any task that requires sequential, ordered information.

b) Discuss the limitations of absolute position embeddings in large language models (LLMs) and the advantages of using relative position embeddings in Transformer models.

Limitations of absolute position embeddings

1. Fixed length leads to poor generalization: since absolution position embeddings are typically implemented as a fixed-sized embedding matrix, a model trained with absolution position embeddings are inflexible to varied input length, and may overfit to the positions seen during training. Positions unseen during training or are undertrained during training will lead to degraded performance at test time.
2. Lack of inductive bias for sequential data: sequential data like natural language often have inherent information encoded in the relative positions of the tokens. With absolution position embeddings, the model do not capture how tokens relate to one another positionally. The model might learn to encode the absolute positions of tokens in a sequence, but not the relative positions of the tokens to each other.

Advantages of relative position embeddings

1. Improved generalization to varied sequence length: as relative positional embeddings encode the distance between tokens rather than absolute positions of the tokens, relative positional embeddings allow the model to generalize to sequences of varying length.
2. Modeling of relative relationships: encoding relative distance between tokens allow the method to help the model learn important dependency information like semantics and syntax in natural languages.

3. Memory efficiency: instead of storing a fixed sized embedding matrix that may take up significant memory for long sequences, relative positions can be calculated on-the-fly, reducing memory consumption.

Question 2.8

b) Devise specific prompts that test the accumulated knowledge of the model using generate.py. You are free to tweak various generation parameters not related to training, such as temperature, p threshold, and the prompt.

With 5 epochs of training on the Fairy Tale dataset, we can observe that the model learned to repeat patterns, sentences, and writing style of the fairy tales, but fails to demonstrate semantic or syntactic awareness.

Prompt 1: "Once upon a time, in a faraway kingdom, there lived a humble woodcutter and his wife, who had two children named ", with temperature: 0.7 and top_p: 0.9
Example output:

```
Once upon a time, in a faraway kingdom, there lived a humble woodcutter and
his wife, who had two children named their way, and said:

 ‘Tell me, glass, tell me true!
  Of all the ladies in
-------------------------------------------------------------
Once upon a time, in a faraway kingdom, there lived a humble woodcutter and
his wife, who had two children named that had eaten the honey: and so the
dwarf knew
which was the youngest. Thus
```

Prompt 2: "As soon as she was alone that dwarf came in, and said, ‘What will you give me to spin ", with temperature: 1.0, top_p: 0.9
Example output:

```
As soon as she was alone that dwarf came in, and said, ‘What will you give
me to spin gold out as my
head for it.’ ‘And the silver?’ ‘Oh! I worked hard for that se
-------------------------------------------------------------
As soon as she was alone that dwarf came in, and said, ‘What will you give
me to spin gold for you this three-legged
horse. When the king returned to his palace, h
```

Prompt 3: "The dark forest was filled with ancient trees whose twisted branches reached out like ", with temperature: 0.75, top_p: 0.95
Example output:

```
The dark forest was filled with ancient trees whose twisted branches reached
out like a green for her, but they
all did not food the trade upon it; the butler fini
-------------------------------------------------------------
The dark forest was filled with ancient trees whose twisted branches reached
out like a dog
tied to a rope, and did not know what to do, for he never goes
out, and
```

Prompt 4: "In the first chapter, the youngest brother received an enchanted cloak. Now, as he faced the dragon, he remembered ", with temperature: 0.65, top_p: 0.85
Example output:

```
In the first chapter, the youngest brother received an enchanted cloak. Now,
as he faced the dragon, he remembered and fetch her again.

And as they came to the wood where the fox first met th
-------------------------------------------------------------
In the first chapter, the youngest brother received an enchanted cloak. Now,
as he faced the dragon, he remembered the greatest fear, and begged
their pardon. And now as they were saying they
```

We can observe from the above output from the model that most of the output does not make semantic or syntactic sense, but the model does output tokens in a manner that resembles phrases from the training dataset. In the first prompt, we see that instead of the continuing the prompt, the phrase "Tell me, glass, tell me true" from the training dataset was repeated. Similarly in prompt 2, which is a direct quote from the training dataset, the model correctly continues the phrase "spin gold". Adjusting temperature or top_p did not significant difference in the type of output obtained.

Question 2.9

b) Discuss the benefits of using FlashAttention and what they stem from.

Computationally, the attention mechanism is not bottlenecked by computation, but rather by read and writes of the large NxN matrices in the softmax step. FlashAttention offers improvements in memory usage and computational speed by computing attention with tiling (loading matrix block by block to SRAM) and making use of recomputation on the backwardpass (instead of storing and loading from memory).

Question 3.1

For the sub-questions a, b and c provide your answers as a matrix in the notation of a 2D python list. For example: 1,2,3], [4,5,6], [7,8,9

a) Write down the adjacency matrices for the graphs shown in Fig. 6 Please respect the indicated node ordering

Left graph in adjacency matrix:
[[0, 1, 1, 0, 1], [1, 0, 1, 0, 0], [1, 1, 0, 1, 0], [0, 0, 1, 0, 0], [1, 0, 0, 0, 0]]

$$\begin{bmatrix} 0 & 1 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Right graph in adjacency matrix:
[[0, 1, 0, 0, 0, 0], [1, 0, 1, 0, 1, 0], [0, 1, 0, 1, 0, 0], [0, 0, 1, 0, 1, 1], [0, 1, 0, 1, 0, 1], [0, 0, 0, 1, 1, 0]]

$$\begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 & 0 \end{bmatrix}$$

b) For the first graph (Fig. 6, left), also write down the square of the adjacency matrix A^2. How can you interpret the entries of the squared adjacency matrix A^2? How can you interpret the entries of the adjacency matrix to the power of *n*, $(A^n)_{uv}$?

[[3, 1, 1, 1, 0], [1, 2, 1, 1, 1], [1, 1, 3, 0, 1], [1, 1, 0, 1, 0], [0, 1, 1, 0, 1]]

$$A^2 = \begin{bmatrix} 0 & 1 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \end{bmatrix} \cdot \begin{bmatrix} 0 & 1 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \end{bmatrix} = \begin{bmatrix} 3 & 1 & 1 & 1 & 0 \\ 1 & 2 & 1 & 1 & 1 \\ 1 & 1 & 3 & 0 & 1 \\ 1 & 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 & 1 \end{bmatrix}$$

The square of the adjacency matrix A^2 represents the number of paths with length 2 that connects u and v for $(A^2)_{uv}$. That is, $(A^2)_{11}$ shows that there are 3 paths with length of 2 that starts at node 1 and ends at node 1. $(A^2)_{31}$ shows that there are 1 path with length of 2 that starts at node 3 and ends at node 1 (the path that goes through node 2). This can be generalized for adjacency matrix of power n, where $(A^n)_{uv}$ represents the number of n-length paths between u and v.

c) The Laplacian matrix captures the relationship between connected nodes, thus influencing how feature values propagate through the network. For the second graph (Fig. 6, right), write down the Laplacian matrix. Then, without performing calculations, reason which node (or nodes) changes the most in feature values when the Laplacian is applied to an input matrix X (F = LX), and explain why.

Laplacian matrix is calculated with $L = D - A$ where D is the degree matrix and A is the adjacency matrix. The graph on the right has the degree matrix,

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 3 & 0 & 0 & 0 & 0 \\ 0 & 0 & 2 & 0 & 0 & 0 \\ 0 & 0 & 0 & 3 & 0 & 0 \\ 0 & 0 & 0 & 0 & 3 & 0 \\ 0 & 0 & 0 & 0 & 0 & 2 \end{bmatrix}$$

The Laplacian matrix is thus,

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 3 & 0 & 0 & 0 & 0 \\ 0 & 0 & 2 & 0 & 0 & 0 \\ 0 & 0 & 0 & 3 & 0 & 0 \\ 0 & 0 & 0 & 0 & 3 & 0 \\ 0 & 0 & 0 & 0 & 0 & 2 \end{bmatrix} - \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 & 0 \end{bmatrix} = \begin{bmatrix} 1 & -1 & 0 & 0 & 0 & 0 \\ -1 & 3 & -1 & 0 & -1 & 0 \\ 0 & -1 & 2 & -1 & 0 & 0 \\ 0 & 0 & -1 & 3 & -1 & -1 \\ 0 & -1 & 0 & -1 & 3 & -1 \\ 0 & 0 & 0 & -1 & -1 & 2 \end{bmatrix}$$

[[1, -1, 0, 0, 0, 0], [-1, 3, -1, 0, -1, 0], [0, -1, 2, -1, 0, 0], [0, 0, -1, 3, -1, -1], [0, -1, 0, -1, 3, -1], [0, 0, 0, -1, -1, 2]]

The nodes that change the most are the nodes that have a

1. high degree, which have larger diagonal entries in L that contributes more to the change in their feature value.
2. larger number of connections to different nodes (indicated in the off-diagonal -1 terms), since a node connected to more nodes with different feature values should change more.
   Based on these factors, nodes 2, 3 and 5 should change most with the highest degree (3) and most connections. Conversely, node 1 should change the least.

Question 3.2

For an undirected graph with N nodes, each node v is associated with a d-dimensional embedding h_v. Let us consider the following graph convolution that propagates the embeddings for all nodes from layer l to the next layer $l+1$:

$$h_v^{(l+1)} = \sigma\left(W^{(l)} \sum_{u \in N(v)} \frac{h_u^{(l)}}{|N(v)|} + B^{(l)} h_v^{(l)}\right)$$

The nonlinearity, e.g., ReLU, is denoted by σ. The matrices $B^{(l)}, W^{(l)} \in \mathbb{R}^{d^{(l+1)} \times d^{(l)}}$, are parameterizing the self-connection and the combination of neighboring node activations respectively. The neighborhood of a node v is denoted by $N(v)$ and consists of all nodes connected to v with an edge. Hence, $|N(v)|$ stands for the number of neighboring nodes.

How does the presence of the bias term $B^{(l)} h_v^{(l)}$ influence the final embedding of node v compared to a scenario without this term?

a) How does the presence of the bias term $B^{(l)} h_v^{(l)}$ influence the final embedding of node v compared to a scenario without this term?

Eq. (6) is independent of $h_v^{(l)}$, and only depends the embeddings of neighboring nodes $h_u^{(l)}$, if the bias term $B^{(l)} h_v^{(l)}$ is removed. Therefore this bias term is necessary for the update to incorporate its own previous embedding into the computation of the next layer of embedding. The bias terms allows the node to retain more of its own information and prevents oversmoothing of the different nodes' representations.

b)

In message-passing neural networks (Gilmer et al. 2017), after every layer, the information contained in the embeddings of a node is propagated to the neighbor nodes in so-called messages. These are summed up to update the node embedding:

$$m_v^{l+1} = \sum_{u \in N(v)} \text{message}(h_v^l, h_u^l, e_{uv})$$
$$h_v^{l+1} = \text{update}(h_v^l, m_v^{l+1})$$

Here, 'message' and 'update' are nonlinear functions that can be parametrized by artificial neural networks. How does the graph convolution presented in the ReLU update equation relate to the message-passing approach described here?

The graph convolution described by Eq. (6) is a specific instantiation of the MPNN described by Eq. (7), where the message(h_v^l, h_u^l, e_{uv}) is a simplification that depends only on normalized neighboring nodes u embeddings:

$$\text{message}(h_u^l) = \frac{h_u^{(l)}}{|N(v)|}$$

Note that the message is independent of the edge features and the node's own embedding. The update step of the graph convolution then sums over the messages and becomes,

$$h_v^{(l+1)} = \sigma\left(W^{(l)} m_v^{l+1} + B^{(l)} h_v^{(l)}\right)$$

This means the 'update' nonlinear function in MPNN terms is a ReLU function parameterized by learnable weight matrices $W^{(l)}$ and $B^{(l)}$.

Question 3.3

The degree matrix D of a graph with N nodes is an $N \times N$ diagonal matrix. For each node v, it counts the nodes in its neighborhood $N(v)$ (nodes connected to node v with an edge):

$$D_{vu} := \begin{cases} |N(v)| & \text{if } v = u \\ 0 & \text{else.} \end{cases}$$

a)

Rewrite the graph convolution (Eq. 6) to matrix form $H^{(l+1)} = f(H^{(l)})$ with embedding matrix $H^{(l)} = [h_1^{(l)}, \dots, h_{|V|^{(l)}}]^T$. Use the adjacency matrix A (Eq. 5) and the degree matrix D (Eq. 8).

$$h_v^{(l+1)} = \sigma\left(W^{(l)} \sum_{u \in N(v)} \frac{h_u^{(l)}}{|N(v)|} + B^{(l)} h_v^{(l)}\right)$$

The embedding matrix $H^{(l)} = [h_1^{(l)}, \dots, h_{|V|^{(l)}}]^T$ has dimensions $N \times d^{(l)}$. The normalizing term in the message $\frac{1}{|N(v)|}$ can be calculated as the matrix product of the inverse of the diagonal matrix D^{-1} with the adjacency matrix A. This gives a row-normalized adjacency matrix. Aggregation over the neighboring embeddings $h_u^{(l)}$ for u in $N(v)$ is then given by the matrix product between $D^{-1}A \in \mathbb{R}^{N \times N}$ and $H^{(l)} \in \mathbb{R}^{N \times d^{(l)}}$.

The weight matrices $W^{(l)}$ and $B^{(l)}$ both of dimensions $d^{(l)} \times d^{(l+1)}$ can then be applied to project the first and second term to the dimension of $H^{(l+1)} \in \mathbb{R}^{N \times d^{(l+1)}}$:

$$D^{-1}AH^{(l)}W^{(l)} + H^{(l)}B^{(l)}$$

Applying sigmoid element-wise then give us the next layer embedding for each node:

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

b) Is the rewriting of the update formula from index to matrix form (as in (a)) applicable when using the median aggregation function? Please explain your answer.

For a node v, the median aggregation function computes the element-wise median of the embeddings of the node's neighbors. The update equation 6 thus becomes,

$$h_v^{(l+1)} = \sigma\left(W^{(l)} \text{median}_{u \in N(v)}\left(h_u^{(l)}\right) + B^{(l)} h_v^{(l)}\right)$$

Computing the median of an element in the embedding of the node v in the embedding matrix $H^{(l)}$ requires accessing and comparing embeddings in different rows, which is not achievable through matrix multiplication. Specifically, the median function is a nonlinear and non-associative operation that cannot be expressed through matrix multiplications.

Question 3.4

a) How does aggregation occur in GATs compared to Graph Convolutional Networks (GCNs)?

- GCNs aggregate neighbor information using fixed weights derived from the graph's adjacency matrix, treating all neighbors equally after in the normalization, as illustrated in the summation term of Eq. (6).
- GATs uses a learnable attention mechanism to assign different importance to each neighbor, allowing for a learnable and feature-dependent aggregation of the different embeddings from the neighbors. Specifically, aggregation in GATs is weighted by a learned attention score α_{uv}: $h_v^{(l+1)} = \sigma\left(\sum_{u \in N(v)} \alpha_{uv} W^{(l)} h_u^{(l)}\right)$.

b) Consider a scenario where certain nodes in the graph are noisy or irrelevant. How would the attention mechanism in GATs handle this compared to GCNs? Explain how this might affect the overall quality of the node representations learned by each model.

With a fixed weighting scheme in the GCNs, which weights the neighboring embeddings through the normalized adjacency matrix, noisy and irrelevant nodes in the neighbors can impact the aggregated embeddings. This will adversely affect the quality of the learned representations in GCNs. GATs on the other hand will be able to selectively reduce the influence of unimportant or noisy nodes in the neighbors during aggregation when the attention mechanism learns to assign lower attention weights to these embeddings, thus allowing it to learn higher quality representations.

c) How can a standard transformer model applied to a natural language sentence be interpreted as a GNN? In this context, what would represent the nodes and edges, and how would the graph structure be defined?

This can be achieved by interpreting the tokens in natural language as nodes in a fully-connected graph, where the edges of the graph are weighted by the self-attention mechanism of the transformer. The attention mechanism in the transformer is then analogous to message passing, ie, the aggregation of weighted neighboring embeddings in this GNN, where every node (token) is connected to every other node (token).

d) Provide two advantages of using GATs over Transformers for tasks involving graph-structured data.

1. For graph structures that is not fully connected, GATs are more computationally efficient as it performs attention only over each node's immediate neighbors. This means the quadratic computational complexity of transformers can be avoided for sparser graphs when applying GATs, where the computational complexity is proportional to the number of edges in the graph.

2. Modeling graph-structured data with transformers may lead to a dilution of the structural information as the attention mechanism in transformer will allow each node attend to every other node. GATs incorporates the graph structural information by forcing nodes to attend to only neighboring nodes. This allows GATs to correctly model the relationship in the graph and learn higher quality node representations.