Minor - 2

Question 1

▼ Methodology

▼ 1. Student Teacher training

The student-teacher network is a type of neural network that uses a pre-trained larger teacher network to guide the training of a smaller student network. The teacher network provides supervision to the student network by providing soft targets or labels. The methodology to train a student-teacher network can be broken down into the following steps:

1. Pre-train the teacher network: Train a larger and more complex neural network on a large dataset to achieve a high level of performance on a specific task.
2. Prepare the training data: Prepare the training dataset, which consists of input-output pairs, to be used to train the student network.
3. Train the student network: Train the smaller student network on the training data using a combination of hard targets and soft targets generated by the teacher network. Soft targets are the probabilities assigned to each class by the teacher network, while hard targets are one-hot vectors representing the ground truth labels. The soft targets are used to regularize the student network during training, forcing it to learn more robust representations.
4. Optimize the loss function: The loss function used to optimize the student network should combine both hard and soft targets, with appropriate weighting of the two terms. The optimization algorithm used should be chosen to ensure convergence to the global minimum.

5. Fine-tune the student network: Once the student network has been trained, it can be fine-tuned using the original task-specific dataset to further improve its performance.
6. Evaluate the student network: Finally, evaluate the performance of the student network on the task-specific dataset and compare it with the performance of the teacher network. This evaluation can help to assess the effectiveness of the student-teacher training approach.

Overall, the methodology to train a student-teacher network involves using a larger pre-trained teacher network to guide the training of a smaller student network on a specific task, with the aim of achieving improved performance and generalization.

▼ 2. student teacher training along with EMA

1. For the purposes of EMA, the student and teacher networks have the same architecture during student-teacher training, furthermore,
2. After the weights of the student model have been updated with gradient descent, the teacher model weights are updated as an exponential moving average of the student weights.
3. Both model outputs are used for prediction, but at the end of the training, the teacher network accuracy is better

▼ Network Architecture

▼ Without EMA

Student net

```
student_CNN(
(conv1): Conv2d(3, 6, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
(conv2): Conv2d(6, 36, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
(conv3): Conv2d(36, 64, kernel_size=(5, 5), stride=(1, 1), padding=(1, 1))
(pool): AvgPool2d(kernel_size=2, stride=2, padding=0)
(fc1): Linear(in_features=14400, out_features=256, bias=True)
(fc2): Linear(in_features=256, out_features=10, bias=True)
)
```

Teacher net

```
CNN(
(conv1): Conv2d(3, 6, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
(conv2): Conv2d(6, 12, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
(conv3): Conv2d(12, 24, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
(conv4): Conv2d(24, 48, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
(conv5): Conv2d(48, 96, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
(conv6): Conv2d(96, 192, kernel_size=(5, 5), stride=(1, 1), padding=(1, 1))
(conv7): Conv2d(192, 384, kernel_size=(5, 5), stride=(1, 1), padding=(1, 1))
(pool): AvgPool2d(kernel_size=2, stride=2, padding=0)
(fc1): Linear(in_features=75264, out_features=256, bias=True)
(fc2): Linear(in_features=256, out_features=10, bias=True)
)
```

▼ With EMA

For EMA, both student and teacher have the same architecture else EMA wont be possible

```
CNN(
(conv1): Conv2d(3, 6, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
(conv2): Conv2d(6, 12, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
(conv3): Conv2d(12, 24, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
(conv4): Conv2d(24, 48, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
(conv5): Conv2d(48, 96, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
(conv6): Conv2d(96, 192, kernel_size=(5, 5), stride=(1, 1), padding=(1, 1))
(conv7): Conv2d(192, 384, kernel_size=(5, 5), stride=(1, 1), padding=(1, 1))
(pool): AvgPool2d(kernel_size=2, stride=2, padding=0)
(fc1): Linear(in_features=75264, out_features=256, bias=True)
(fc2): Linear(in_features=256, out_features=10, bias=True)
)
```

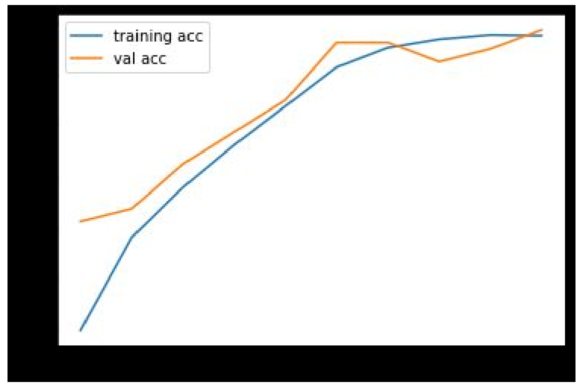
▼ Results and discussions

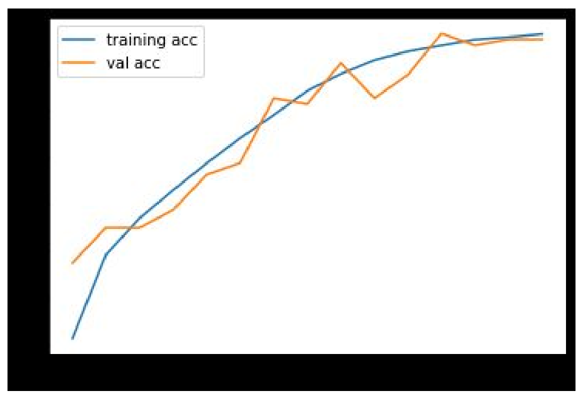
▼ Tuning Hyperparameter epoch in teacher network:

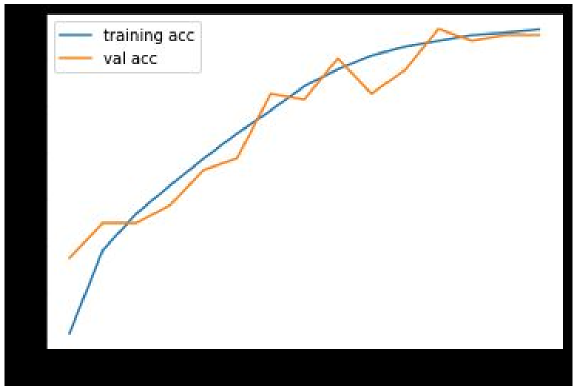
epoch	Accuracy
6	57.9%
10	70.77%
15	65.8%

LOSS CURVE

ACCURACY CURVE

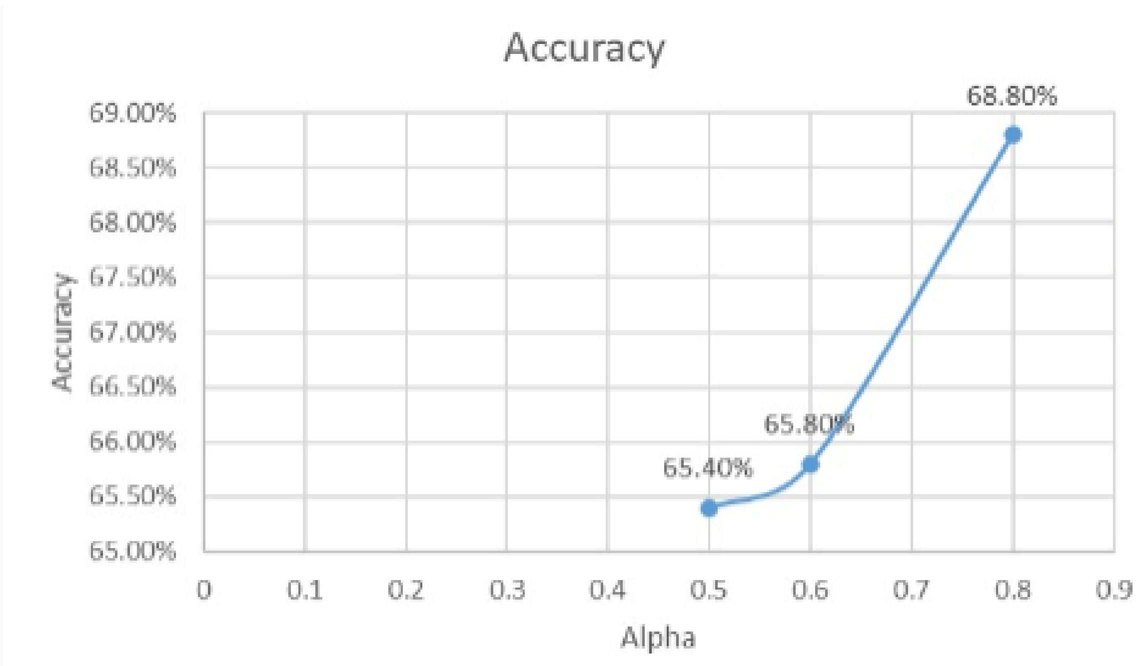






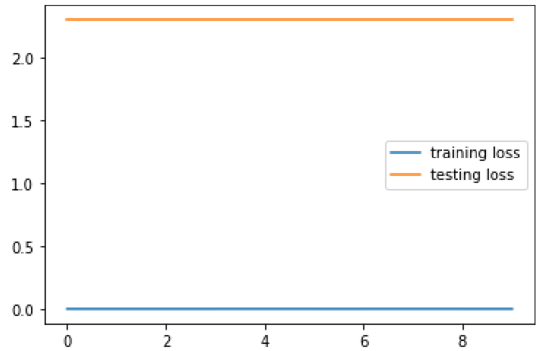
▼ Tuning Hyperparameter alpha in student network:

Alpha	Accuracy
0.5	65.4%
0.6	65.8%
0.8	68.8%



▼ Comparing Networks with and without Student-teacher training

Without



10.5% accuracy

With Student teacher

68.5% accuracy

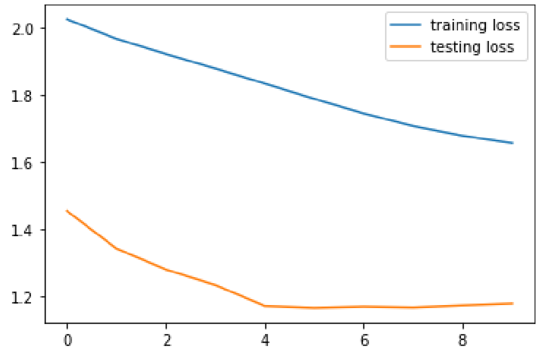
As we can observe, student-teacher training drastically improves performance

▼ Comparing Student-teacher Networks with and without EMA

Student(EMA)	72.65%		
Student(without EMA)	68.5%		
teacher(EMA)	74.60%		
teacher(without EMA)	70.77%		

👉 we can notice That EMA increases accuracies for both student and teacher network, with teacher network(EMA) having the highest accuracies

👉 we can also observe below EMA has also reduced overfitting , and significantly decreased that testing loss



(student with EMA)

(student without EMA)

Question 2

▼ 1. Theoretical question

We learned about Transformers in our Deep learning course. A vision transformer works on the same transformer architecture but requires partitioning the input image into patches of the same shape. So, instead of tokens getting vectorized, here patches of images are being vectorized. You wish to use a vision transformer for a downstream task and a single input is in the form of documents with multiple pages. However, the vanilla vision transformer only accepts a single image as in inputs, but your input is of multiple images. You cannot merge these images into a single image due to the degradation of quality. How will you plan to solve this issue?

▼ ChatGPT

One way to address the issue of using a vanilla vision transformer for documents with multiple pages is to **treat each page as a separate image and apply the vision transformer to each page individually.** The output representations for each page can then be concatenated and fed into a downstream task.

To do this, you could use a sliding window approach to partition each page into patches of the same shape, and then apply the vision transformer to each patch individually. The resulting representations for each patch can be aggregated using a pooling operation, such as mean or max pooling, to obtain a single representation for each page. The representations for all the pages can then be concatenated along a new dimension, resulting in a tensor of shape (num_pages, sequence_length, hidden_size), where sequence_length represents the number of patches used to represent each page.

This concatenated tensor can then be fed into a downstream task, such as classification or object detection. **Alternatively, you could use another transformer block to process this concatenated tensor and obtain a final output for the downstream task.**

Overall, the key idea is to apply the vision transformer to each page individually and concatenate the resulting representations to obtain a final representation for the entire document. This approach should allow you to leverage the power of vision transformers for multi-page documents without sacrificing the quality of the input images.

▼ Shortcomings of ChatGPT's answer

The answer suggests treating each individual page of a document as a single image and then concatenating them. While it is technically possible to do so it isn't a good approach. This is because when each individual image is treated like a separate image, each image would have its own positional encoding/embedding, unrelated to each other. however, the order of pages does definitely matter in a document, and adjacent pages are related to each other

A better way to approach this would be to alter the positional encodings/embeddings of every image such that they are related to each other. For example, lets say our model divided an image into 4 patches with positional encodings/embeddings as 1,2,3,4 for page number 1, we can change the positional encoding/embeddings of page 2 from 1,2,3,4 to 5,6,7,8. By doing this patches of page 1 and patches of page 2 are made relevant to each other. This relevancy was missing in the ChatGPTs answer

However, ChatGPT's answer vaguely mentions to "**use another transformer block to process this concatenated tensor and obtain a final output for the downstream task".**

even in this, since the tensors have already been concatenated without positional encodings being modified, we have lost the relevancy information

▼ 2. Theoretical question

For NLP tasks a vanilla transformer uses sinusoidal positional encoding, but since images are more “structured” than simple sentences, suggest a way to incorporate the structured information

▼ ChatGPT

While sinusoidal positional encoding works well for sequential data like text, it may not be optimal for structured data like images. In the case of images, we can use spatial positional encoding techniques that take into account the spatial relationships between the pixels in the image.

One way to incorporate structured information for images is to use grid positional encoding, which encodes the spatial position of each pixel in the image. This can be done by adding a set of learned 2D coordinates to the input embeddings of the image, with each coordinate representing the x and y position of the pixel in the image.

Another approach is to use attention-based positional encoding, where attention is applied to the spatial locations of the pixels in the image. This can be achieved by using multi-head attention to learn different attention patterns for different spatial positions in the image.

▼ My answer

Along with chat GPT’s Answer, we can also use positional embeddings instead of positional encodings

▼ 2. Numerical Question

Consider a transformer model which is given the input as :
"The sun sets in the west every evening". (for simplicity consider one word to a token). This transformer model generates 100-dimensional embeddings. For one self-attention block let the number of time a positional encoding is added "a", and let the dimensions of query, key, and value linear layers be "b" X "c", "d" X "e", "f" X "g". Let the dimension of the score matrix be "h" X "i". Assume that 4 heads are used for multi-head attention blocks, considering this let the dimension of the last linear layer (in multi-head attention blocks that comes

after concatenation) be "j" X "k". Calculate the value of "a" , "b" ,"c" , "d" ,"e" ,"f" ,"g", "h" , "i" , "j" ,"k". Show the steps involved

▼ ChatGPT answer

The given input "The sun sets in the west every evening" contains 8 tokens. So, the number of tokens is 8.

For the self-attention block, the inputs are queries, keys, and values, which are multiplied with their respective weights and then transformed into a score matrix. The score matrix is then multiplied with the values to get the attention output. This output is passed through a linear layer to get the final output. During this process, a positional encoding is added before the self-attention operation.

Let's consider the dimensions of the transformer model as follows:

- Embedding dimension: 100
- Number of heads: 4

To calculate the values of a, b, c, d, e, f, g, h, i, j, k, we need more information about the architecture of the transformer model, such as the number of layers and the hidden dimension of the feed-forward layer.

Without this information, we can make some assumptions based on typical values used in transformer models:

- The hidden dimension of the feed-forward layer is usually 4 times the embedding dimension. So, let's assume it to be 400.
- The dimension of the score matrix (h x i) is usually the same as the dimension of the queries/keys/values (b x c, d x e, f x g). So, let's assume h = b = d = f and i = c = e = g.
- The dimension of the last linear layer (after concatenation) is usually equal to the hidden dimension of the feed-forward layer. So, let's assume j = 400 and k = 100.

With these assumptions, we can calculate the values of a, b, c, d, e, f, g, h, i, j, k as follows:

- The input embeddings have a dimension of 8 x 100 (8 tokens with 100-dimensional embeddings).

- The queries, keys, and values are obtained by multiplying the input embeddings with weight matrices of dimensions 100 x b, 100 x d, and 100 x f, respectively. So, b = d = f = 100.
- The score matrix is obtained by multiplying the queries with the transpose of the keys. So, h = b = d = f = 100 and i = c = e = g = 100.
- The output of each attention head is a matrix of dimensions 8 x c (8 tokens with c-dimensional attention output). Since there are 4 heads, the concatenated output has a dimension of 8 x 4c. This is then passed through a linear layer with weight matrix of dimensions 4c x j. So, j = 400 and k = 100.
- Finally, the output of the self-attention block is obtained by passing the concatenated output through a linear layer with weight matrix of dimensions 8 x 400 and then adding a residual connection with the input embeddings. This addition requires that the input embeddings have the same dimension as the output of the linear layer, so a = 8 and b = 400.

Therefore, the values of a, b, c, d, e, f, g, h, i, j, k are:

a = 8
b = 400
c = 100
d = 100
e = 100
f = 100
g = 100
h = 100
i = 100
j = 400
k = 100

▼ ChatGPT answer’s shortcomings

ChatGPT’s answers were wrong for the value of b, it made incorrect assumptions, so b,c,d,e,f=100

furthermore, the score matrix would be 8X8 since there are 8 tokens therefore it failed as calculating i and j as well

Also, since 4 attention heads are used, the concatenated layer would be of dimension 8X(100*4) therefore the dense layer has to be of dimension 400X100 so that the final output can be 8X100 like original so j and k are 400 and 100