Minor 2

Question 1:

Parameter selection:

DOB: DD/MM/YYYY —--> 03/06/1999 Last three roll number: ABC —-> 061

M.Tech dataset: Tiny Imagenet

MM 06 even

- Weight initialization: xavier
- Pool: Avgpool

Model details

- Teacher model
  - ABC - 061 odd
  - 12 conv 1 pool
  - 12 filters in first layer
- Student model
  - DD odd
  - 4 conv 1pool
- FCs
  - ABC sum is 7 odd
  - 1FC with 512 nodes

- Import required libraries
- Write device agnostic code
- Helper function for image display
- Get Tiny image dataset from online source
- Make dataset and create data loader from it
- Build model for teacher and student with xavier weight initialization
- Take cross entropy as loss function
- Write train and test loop
- Write function for loss and accuracy plot
- Now train teacher and student model independently

Train student with teacher help using EMA

- In below image shows the EMA in teacher and student

Train student with teacher help without EMA

- change in tain loop of teacher student without EMA

```
# 1. Forward
   y_train_teacher = teacher(x_train)
   y_pred = model(x_train)
# 2. Loss
```

```
    loss = loss_fn(y_pred, y_train_teacher)
```

```
Algorithm Image for student teacher with EMA
```

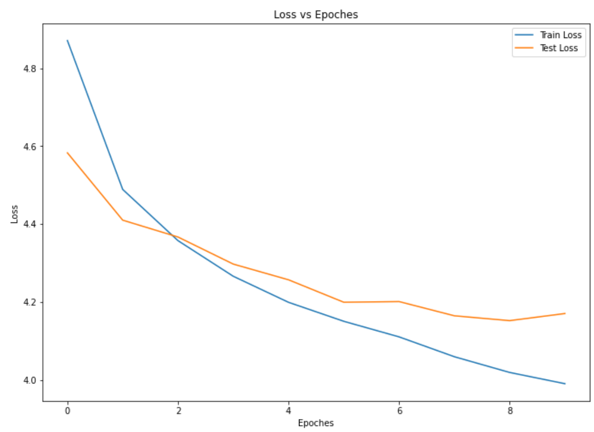
Data: train set $(\mathcal{X}, \mathcal{Y})$, Unlabel data$(\mathcal{Z})$
Hyper parameters: r, α, $p1$, $p2$, $\rho1$, $\rho2$, *epochs*;
Create Model : *student*(θ), *teacher*(θ') ;
Train *teacher* for 1 epoch;
while *epochs* **do**
    while *steps* **do**
        1: Insert noise with probability as per strategies ($p1$, $p2$, $\rho1$, $\rho2$) in $\mathcal{X}$ i.e. $\mathcal{X}_\eta$;
        2: *student*$(\mathcal{X}_\eta) = \mathcal{Y}_\eta$;
        3: Classification cost $(C(\theta))$ =Binary Cross Entropy$(\mathcal{Y},\mathcal{Y}_\eta)$;
        4: Again create different noise data as mentioned in step 1 i.e. $\mathcal{X}_{\eta'}$;
        5: *teacher*$(\mathcal{X}_{\eta'}) = \mathcal{Y}_{\eta'}$;
        6: Calculate Consistency cost $J(\theta)$=Mean Squared Error$(\mathcal{Y}_\eta,\mathcal{Y}_{\eta'})$;
        7: Calculate Overall cost $O(\theta) = \lambda C(\theta) + (1-\lambda)J(\theta)$;
        8: Calculate *gradients*, $O(\theta)$ w.r.t θ ;
        9: Apply *gradients* to θ;
        10: Update Exponential Moving average of θ to θ' i.e. $\theta'_t = \alpha\theta'_{t-1} + (1-\alpha)\theta_t$;
    end
end

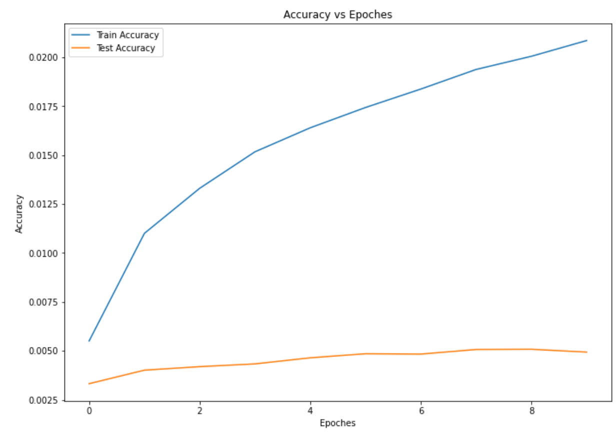
Result:

Only teacher:

```
Epoch: 1  Train Loss: 4.8713 / Test Loss: 4.5830 -/- Train Accuracy: 
0.0055 / Test Accuracy: 0.0033
Epoch: 2  Train Loss: 4.4891 / Test Loss: 4.4101 -/- Train Accuracy: 
0.0110 / Test Accuracy: 0.0040
```

Epoch: 3 Train Loss: 4.3575 / Test Loss: 4.3668 -/- Train Accuracy: 0.0133 / Test Accuracy: 0.0042
Epoch: 4 Train Loss: 4.2660 / Test Loss: 4.2973 -/- Train Accuracy: 0.0152 / Test Accuracy: 0.0043
Epoch: 5 Train Loss: 4.1991 / Test Loss: 4.2569 -/- Train Accuracy: 0.0164 / Test Accuracy: 0.0046
Epoch: 6 Train Loss: 4.1504 / Test Loss: 4.1993 -/- Train Accuracy: 0.0174 / Test Accuracy: 0.0049
Epoch: 7 Train Loss: 4.1104 / Test Loss: 4.2010 -/- Train Accuracy: 0.0184 / Test Accuracy: 0.0048
Epoch: 8 Train Loss: 4.0594 / Test Loss: 4.1645 -/- Train Accuracy: 0.0194 / Test Accuracy: 0.0051
Epoch: 9 Train Loss: 4.0190 / Test Loss: 4.1520 -/- Train Accuracy: 0.0200 / Test Accuracy: 0.0051
Epoch: 10 Train Loss: 3.9902 / Test Loss: 4.1703 -/- Train Accuracy: 0.0208 / Test Accuracy: 0.0049

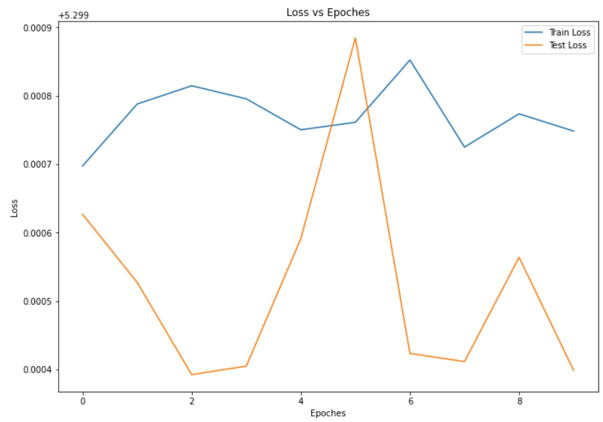


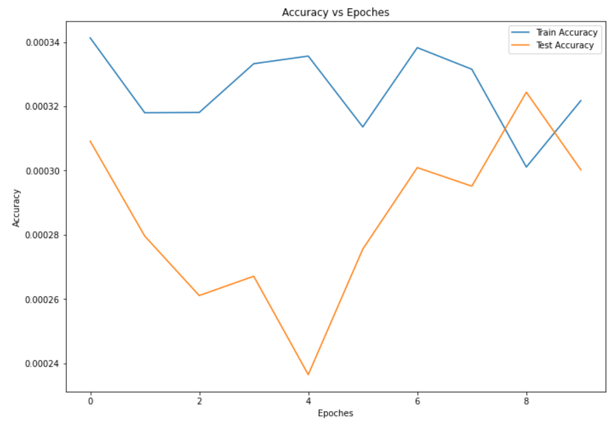


Only Student:

Epoch: 1 Train Loss: 5.2997 / Test Loss: 5.2996 -/- Train Accuracy: 0.0003 / Test Accuracy: 0.0003
Epoch: 2 Train Loss: 5.2998 / Test Loss: 5.2995 -/- Train Accuracy: 0.0003 / Test Accuracy: 0.0003
Epoch: 3 Train Loss: 5.2998 / Test Loss: 5.2994 -/- Train Accuracy: 0.0003 / Test Accuracy: 0.0003
Epoch: 4 Train Loss: 5.2998 / Test Loss: 5.2994 -/- Train Accuracy: 0.0003 / Test Accuracy: 0.0003
Epoch: 5 Train Loss: 5.2998 / Test Loss: 5.2996 -/- Train Accuracy: 0.0003 / Test Accuracy: 0.0002
Epoch: 6 Train Loss: 5.2998 / Test Loss: 5.2999 -/- Train Accuracy: 0.0003 / Test Accuracy: 0.0003
Epoch: 7 Train Loss: 5.2999 / Test Loss: 5.2994 -/- Train Accuracy: 0.0003 / Test Accuracy: 0.0003
Epoch: 8 Train Loss: 5.2997 / Test Loss: 5.2994 -/- Train Accuracy: 0.0003 / Test Accuracy: 0.0003

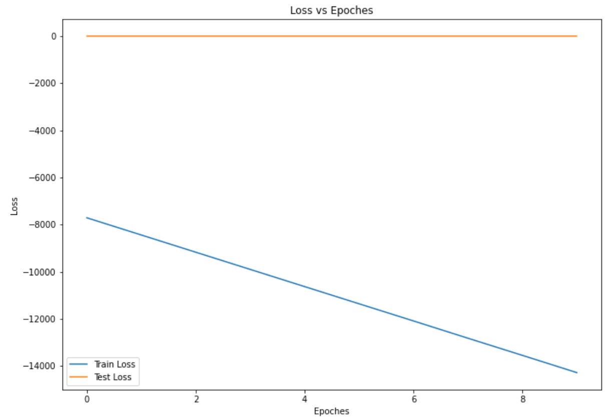
Epoch: 9 Train Loss: 5.2998 / Test Loss: 5.2996 -/- Train Accuracy: 0.0003 / Test Accuracy: 0.0003
Epoch: 10 Train Loss: 5.2997 / Test Loss: 5.2994 -/- Train Accuracy: 0.0003 / Test Accuracy: 0.0003

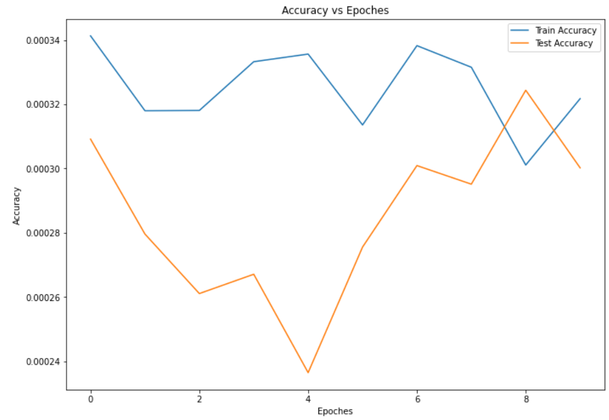




Teacher Student With EMA:

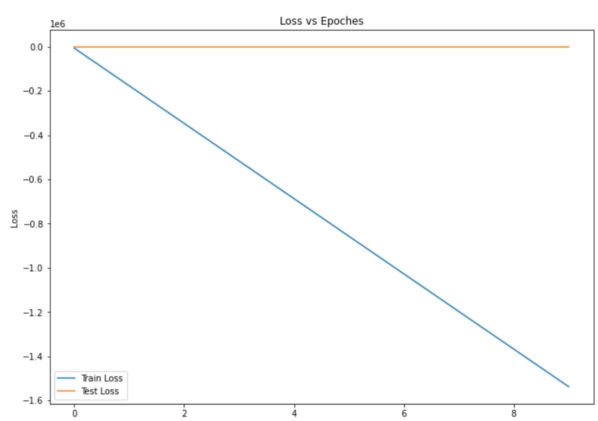
Epoch: 1 Train Loss: -7714.5342 / Test Loss: 5.2996 -/- Train Accuracy: 0.0003 / Test Accuracy: 0.0003
Epoch: 2 Train Loss: -8444.3789 / Test Loss: 5.2995 -/- Train Accuracy: 0.0003 / Test Accuracy: 0.0003
Epoch: 3 Train Loss: -9174.6748 / Test Loss: 5.2994 -/- Train Accuracy: 0.0003 / Test Accuracy: 0.0003
Epoch: 4 Train Loss: -9904.1396 / Test Loss: 5.2994 -/- Train Accuracy: 0.0003 / Test Accuracy: 0.0003
Epoch: 5 Train Loss: -10634.1113 / Test Loss: 5.2996 -/- Train Accuracy: 0.0003 / Test Accuracy: 0.0002
Epoch: 6 Train Loss: -11363.6172 / Test Loss: 5.2999 -/- Train Accuracy: 0.0003 / Test Accuracy: 0.0003
Epoch: 7 Train Loss: -12091.8252 / Test Loss: 5.2994 -/- Train Accuracy: 0.0003 / Test Accuracy: 0.0003
Epoch: 8 Train Loss: -12824.4014 / Test Loss: 5.2994 -/- Train Accuracy: 0.0003 / Test Accuracy: 0.0003
Epoch: 9 Train Loss: -13551.9561 / Test Loss: 5.2996 -/- Train Accuracy: 0.0003 / Test Accuracy: 0.0003
Epoch: 10 Train Loss: -14284.1543 / Test Loss: 5.2994 -/- Train Accuracy: 0.0003 / Test Accuracy: 0.0003

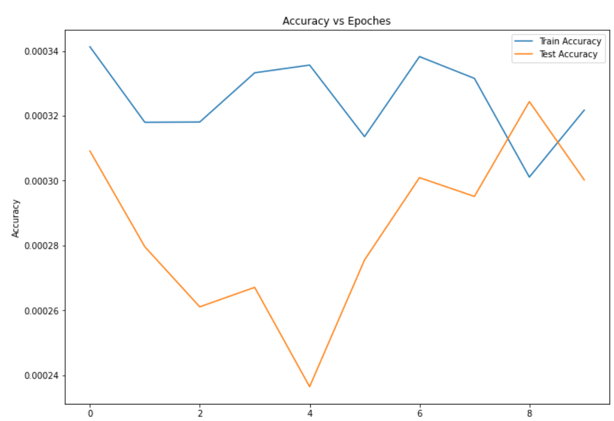




Teacher Student without EMA:

Epoch: 1 Train Loss: -6641.7715 / Test Loss: 5.2996 -/- Train Accuracy: 0.0003 / Test Accuracy: 0.0003
Epoch: 2 Train Loss: -176750.9219 / Test Loss: 5.2995 -/- Train Accuracy: 0.0003 / Test Accuracy: 0.0003
Epoch: 3 Train Loss: -346859.1875 / Test Loss: 5.2994 -/- Train Accuracy: 0.0003 / Test Accuracy: 0.0003
Epoch: 4 Train Loss: -516963.8438 / Test Loss: 5.2994 -/- Train Accuracy: 0.0003 / Test Accuracy: 0.0003
Epoch: 5 Train Loss: -687073.2500 / Test Loss: 5.2996 -/- Train Accuracy: 0.0003 / Test Accuracy: 0.0002
Epoch: 6 Train Loss: -857177.6250 / Test Loss: 5.2999 -/- Train Accuracy: 0.0003 / Test Accuracy: 0.0003
Epoch: 7 Train Loss: -1027256.8125 / Test Loss: 5.2994 -/- Train Accuracy: 0.0003 / Test Accuracy: 0.0003
Epoch: 8 Train Loss: -1197280.2500 / Test Loss: 5.2994 -/- Train Accuracy: 0.0003 / Test Accuracy: 0.0003
Epoch: 9 Train Loss: -1367235.7500 / Test Loss: 5.2996 -/- Train Accuracy: 0.0003 / Test Accuracy: 0.0003
Epoch: 10 Train Loss: -1537199.3750 / Test Loss: 5.2994 -/- Train Accuracy: 0.0003 / Test Accuracy: 0.0003





Question 2:

Q1: Can we derive mathematical equations for numbers of hidden layers and hidden nodes in neural networks?

ANSWER:

GPT:

Q: how to decide number of hidden layer and hidden node in neural network

ANS: As an AI language model, I can tell you that there is no fixed formula to determine the number of hidden layers and nodes in a neural network. However, here are some guidelines that you can follow:

1.
2. Start with one hidden layer: For simple problems, one hidden layer may be sufficient. You can start with one hidden layer and gradually increase the number of layers if necessary.
3. Use cross-validation: Use cross-validation techniques to estimate the performance of different network architectures. This will help you determine the optimal number of hidden layers and nodes.
4. Consider the size of your dataset: The size of your dataset will influence the number of hidden layers and nodes required. If you have a small dataset, a simpler network with fewer hidden layers and nodes may be better to avoid overfitting. If you have a large dataset, you may need a more complex network with more hidden layers and nodes to capture the underlying patterns in the data.
5. Consider the complexity of the problem: More complex problems may require deeper networks with more hidden layers and nodes. If the problem is relatively simple, a smaller network may be sufficient.
6. Use regularization techniques: Regularization techniques such as dropout and L1/L2 regularization can help prevent overfitting and reduce the number of hidden layers and nodes required.

In general, the number of hidden layers and nodes should be determined by balancing model complexity with the ability to generalize to new data. It's important to keep in mind that the optimal architecture will depend on the specific problem you are trying to solve.

Q: But it is possible we try multiple datasets on different architectures and get some kind of mathematical equation for the number of hidden layers and hidden nodes.

ANS:

While it is possible to try multiple datasets and architectures to determine the optimal number of hidden layers and nodes, it is unlikely that a mathematical equation can be derived to give the exact number of hidden layers and nodes required for a specific problem.

The number of hidden layers and nodes depends on the complexity of the problem, the amount of data available, and the specific features of the dataset. Therefore, a more empirical approach such as cross-validation or experimentation is often used to determine the optimal architecture.

However, there are some rules of thumb that can be used as a starting point, such as the "universal approximation theorem" which suggests that a neural network with one hidden layer can approximate any continuous function to a desired level of accuracy. This suggests that a single hidden layer may be sufficient for simple problems. For more complex problems, additional hidden layers may be required.

Ultimately, the best approach is to experiment with different architectures and evaluate their performance on the specific problem at hand.

Q: Someone try this in previous years
ANS:

Yes, researchers have experimented with different architectures to determine the optimal number of hidden layers and nodes for neural networks. For example, in 1998, a study was conducted by a group of researchers at Microsoft Research which found that a neural network with one hidden layer performed better than networks with more than one hidden layer for certain tasks.

However, more recent studies have shown that deeper networks with multiple hidden layers can outperform shallower networks with one or two hidden layers for more complex problems, especially when large amounts of data are available.

Overall, the optimal architecture for a neural network depends on many factors, including the complexity of the problem, the size of the dataset, and the specific features of the data. Therefore, a more empirical approach such as experimentation or cross-validation is often used to determine the optimal number of hidden layers and nodes.

Q: summary of above Microsoft result and conclusion
ANS:

The Microsoft Research study conducted in 1998 found that neural networks with a single hidden layer outperformed networks with multiple hidden layers for certain tasks. Specifically, the study found that for binary classification problems, a neural network with a single hidden layer and a number of nodes equal to the number of inputs performed better than networks with more than one hidden layer.

However, it's important to note that this was a relatively small study conducted on a limited number of datasets, and the results may not be generalizable to all problems. Additionally, more recent studies have found that deeper networks with multiple hidden layers can outperform shallower networks for more complex problems, especially when large amounts of data are available.

Overall, the optimal architecture for a neural network depends on many factors, and a more empirical approach such as experimentation or cross-validation is often used to determine the best number of hidden layers and nodes for a specific problem.

Q: but we can take all parameter in consideration like dataset size, its complexity, learning rate, optimisation function, other hyper parameter and then we can come with mathematics formula to decide number of hidden layers and hidden nodes
ANS:

While it's true that you can take all of these parameters into consideration, it's unlikely that you can come up with a single mathematical formula that will give you the optimal number of hidden layers and nodes for any problem.

The reason for this is that the optimal architecture depends on many factors, and these factors interact in complex ways. Therefore, it's difficult to derive a single equation that will accurately capture all of these interactions.

Instead, a more empirical approach such as experimentation or cross-validation is often used to determine the best architecture for a specific problem. This involves trying different architectures and hyperparameters, and evaluating their performance on the specific problem at hand.

While there are some rules of thumb that can be used as a starting point, such as the "universal approximation theorem" which suggests that a neural network with one hidden layer can approximate any continuous function to a desired level of accuracy, it's important to keep in mind that the optimal architecture depends on the specific problem you are trying to solve.

MY ANS:

I tried some hidden layer and node combinations on Gurnum dataset result are as below.

Hidden Layer Number	Hidden nodes Number	Total Node	Train Accuray %	Test Accuracy %
13	3	39	52	16
13	2	26	95	24
13	4	52	42	15
12	3	48	43	18
14	3	42	80	20

Yes we can come up with mathematical formulas(equation) for the number of hidden layers and numbers of hidden nodes but for that we need to build multiple models on multiple datasets, different optimizers with different loss function, multiple hyperparameters etc.
https://colab.research.google.com/drive/1zV93RhDCJn00TbgaVl7WGSrFvriA1H_o#scrollTo=tKVwDx1oQ_Yy

At cost of too much time and too many model building and training 🙂

Q2:

A) let take 256 x 256 pass it in one convolution layer with 5 kernel size, 2 stride, 1 padding now apply max pool with kernel 3 and stride 2 now pass output to average pool with kernel 5 and stride 2 give the output size
ANS:

Given:

- Input image size: 256 x 256
- Convolution layer with 5 kernel size, 2 stride, 1 padding
- Max pooling layer with kernel size 3 and stride 2
- Average pooling layer with kernel size 5 and stride 2

Output size after the convolution layer can be calculated as:

- Output size = ((input_size - kernel_size + 2*padding) / stride) + 1

- Output size = ((256 - 5 + 2*1) / 2) + 1
- Output size = 127

Output size after the max pooling layer can be calculated as:

- Output size = ((input_size - kernel_size) / stride) + 1
- Output size = ((127 - 3) / 2) + 1
- Output size = 63

Output size after the average pooling layer can be calculated as:

- Output size = ((input_size - kernel_size) / stride) + 1
- Output size = ((63 - 5) / 2) + 1
- Output size = 30

Therefore, the output size after applying the above operations on a 256 x 256 input image would be 30 x 30.

B) let take 256 x 512 pass it in one convolution layer with 5 x 6 kernel size, 2 x 3 stride, 1 padding now apply max pool with kernel 3 x 4 and stride 2 x 3 now pass output to average pool with kernel 5 x 4 and stride 2 x 1 give the output size
ANS:

Given:

- Input image size: 256 x 512
- Convolution layer with kernel size 5 x 6, stride 2 x 3, and 1 padding
- Max pooling layer with kernel size 3 x 4 and stride 2 x 3
- Average pooling layer with kernel size 5 x 4 and stride 2 x 1

Output size after the convolution layer can be calculated as:

- Output size = ((input_size - kernel_size + 2*padding) / stride) + 1
- Output size = ((256 - 5 + 2*1) / 2 + 1, (512 - 6 + 2*1) / 3 + 1)
- Output size = (126, 169)

Output size after the max pooling layer can be calculated as:

- Output size = ((input_size - kernel_size) / stride) + 1
- Output size = ((126 - 3) / 2 + 1, (169 - 4) / 3 + 1)
- Output size = (62, 56)

Output size after the average pooling layer can be calculated as:

- Output size = ((input_size - kernel_size) / stride) + 1
- Output size = ((62 - 5) / 2 + 1, (56 - 4) / 1 + 1)
- Output size = (30, 53)

Therefore, the output size after applying the above operations on a 256 x 512 input image would be 30 x 53.

MY ANS:
Both answers are right with proper procedure.

Q3: Write train function for Teacher Student with EMA and Without EMA.
ANS:
GPT:
With EMA:
```
def train_student(student, teacher, ema, train_loader, optimizer,
criterion, device):
    student.train()
    teacher.eval()
    ema.apply_shadow()
    for i, (inputs, targets) in enumerate(train_loader):
        inputs, targets = inputs.to(device), targets.to(device)
        optimizer.zero_grad()
        outputs = student(inputs)
        with torch.no_grad():
            teacher_outputs = teacher(inputs)
              soft_targets = nn.functional.softmax(teacher_outputs / 5.0,
dim=1)
        loss = criterion(outputs, soft_targets)
        loss.backward()
        optimizer.step()
        ema.update()
    ema.restore()
```

Without EMA:
```
# Train loop
for epoch in range(num_epochs):
    student.train()
    for i, (inputs, targets) in enumerate(train_loader):
        optimizer.zero_grad()
        outputs_student = student(inputs)
        outputs_teacher = teacher(inputs).detach()
```

```
            loss = criterion(F.log_softmax(outputs_student / temperature,
dim=1),
                             F.softmax(outputs_teacher / temperature, dim=1))
        loss.backward()
        optimizer.step()
        running_loss += loss.item()
        if i % log_interval == 0:
                 print('Epoch {} Batch {} Loss: {:.6f}'.format(epoch, i,
running_loss / log_interval))
            running_loss = 0.0
```

My ANS:
Both perform same with EMA only code writing is different
https://medium.com/analytics-vidhya/knowledge-distillation-in-a-deep-neural-network-c9dd59aff89b
https://towardsdatascience.com/model-distillation-and-compression-for-recommender-systems-in-pytorch-5d81c0f2c0ec
https://abdulkaderhelwan.medium.com/implementation-of-teacher-student-model-in-pytorch-a1195734dd08

REFERENCES:

https://medium.com/analytics-vidhya/knowledge-distillation-in-a-deep-neural-network-c9dd59aff89b
https://towardsdatascience.com/model-distillation-and-compression-for-recommender-systems-in-pytorch-5d81c0f2c0ec
https://abdulkaderhelwan.medium.com/implementation-of-teacher-student-model-in-pytorch-a1195734dd08
https://pytorch.org/docs/master/optim.html
https://towardsdatascience.com/pytorch-ignite-classifying-tiny-imagenet-with-efficientnet-e5b1768e5e8f

NOTE: no need of README file