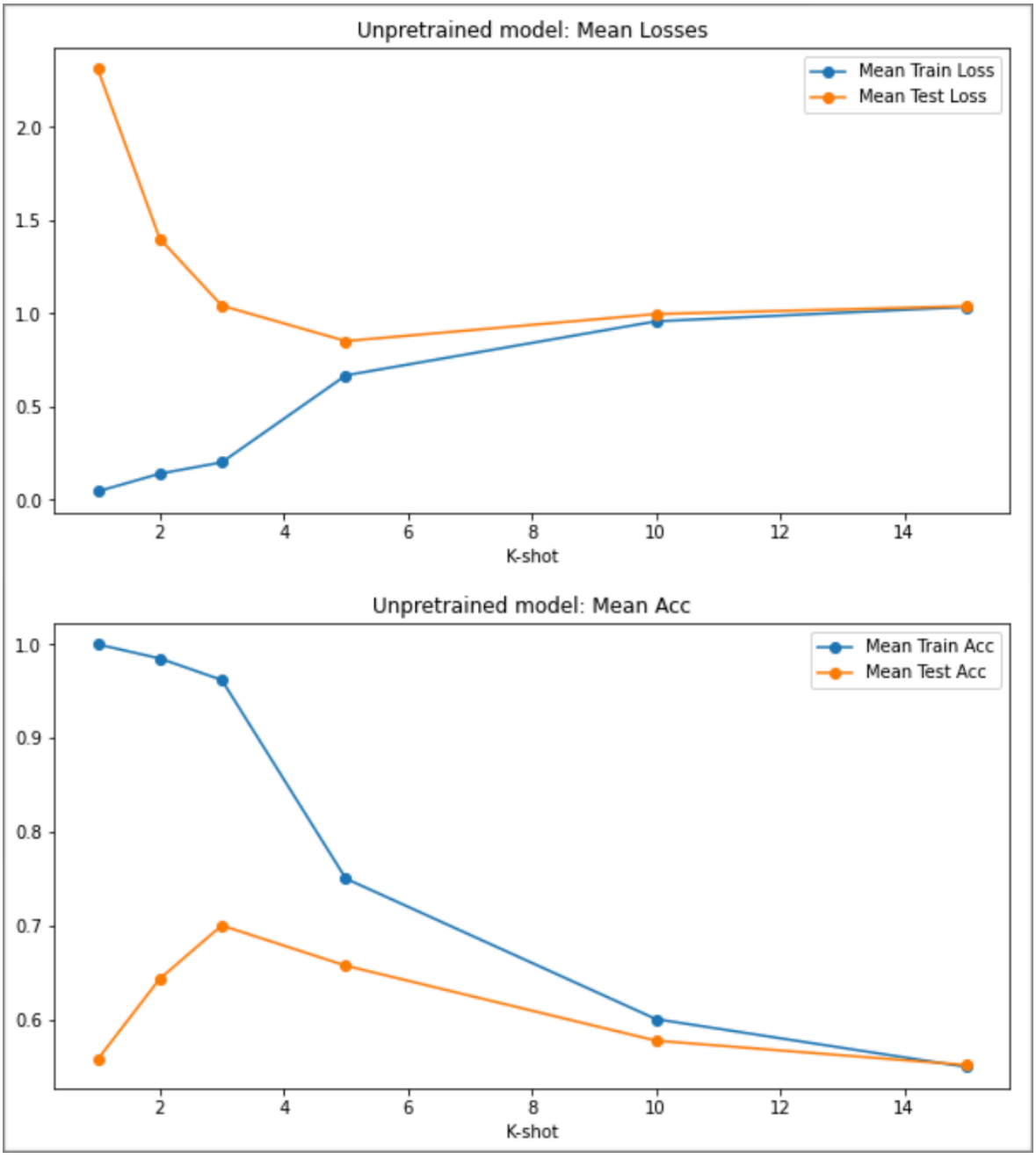
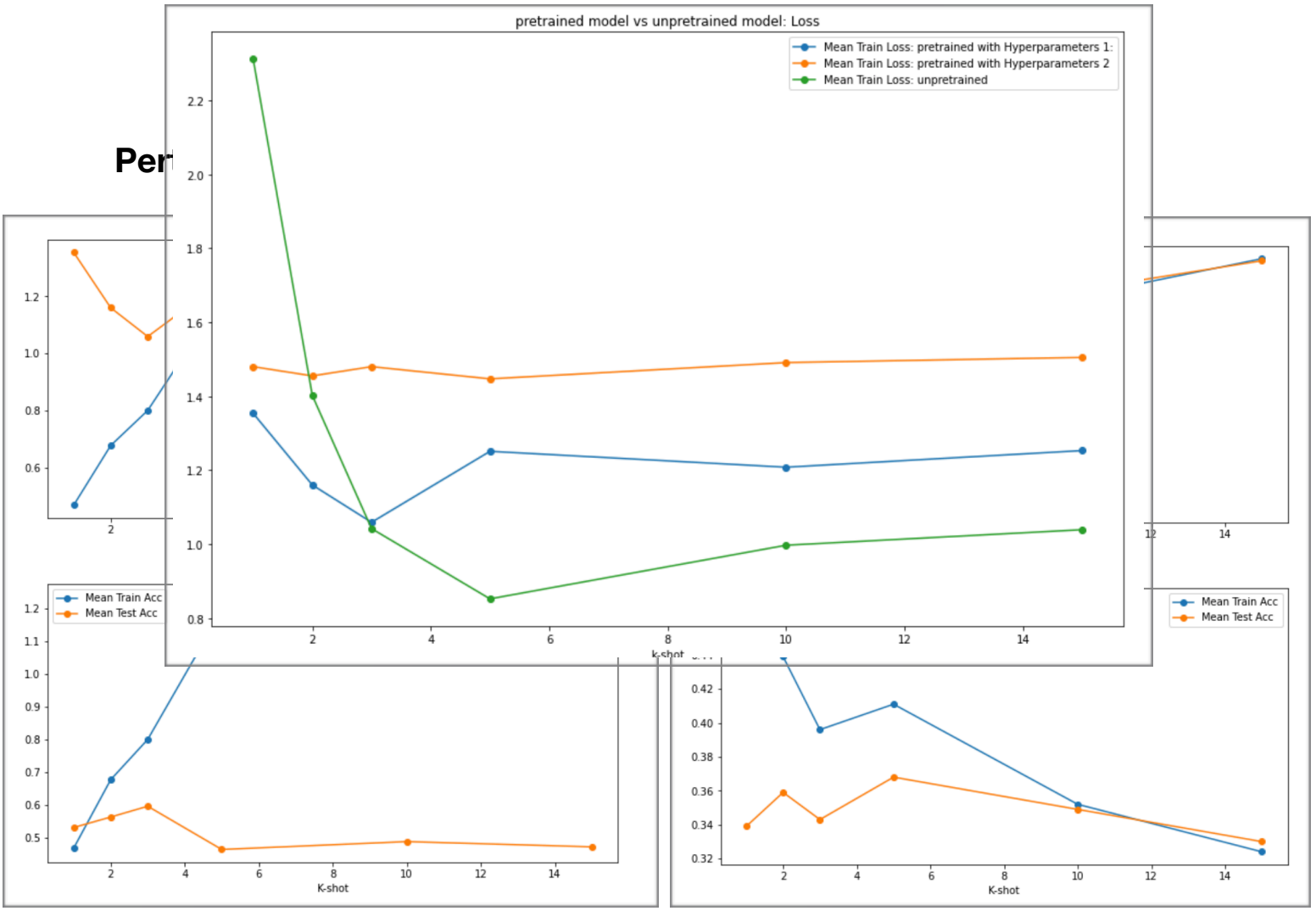
Assignment 2: Low Resources (CS326)

Name: Yasmeen Alsaedyy

Unpretrained model:



Per



Hyperparameters 1:

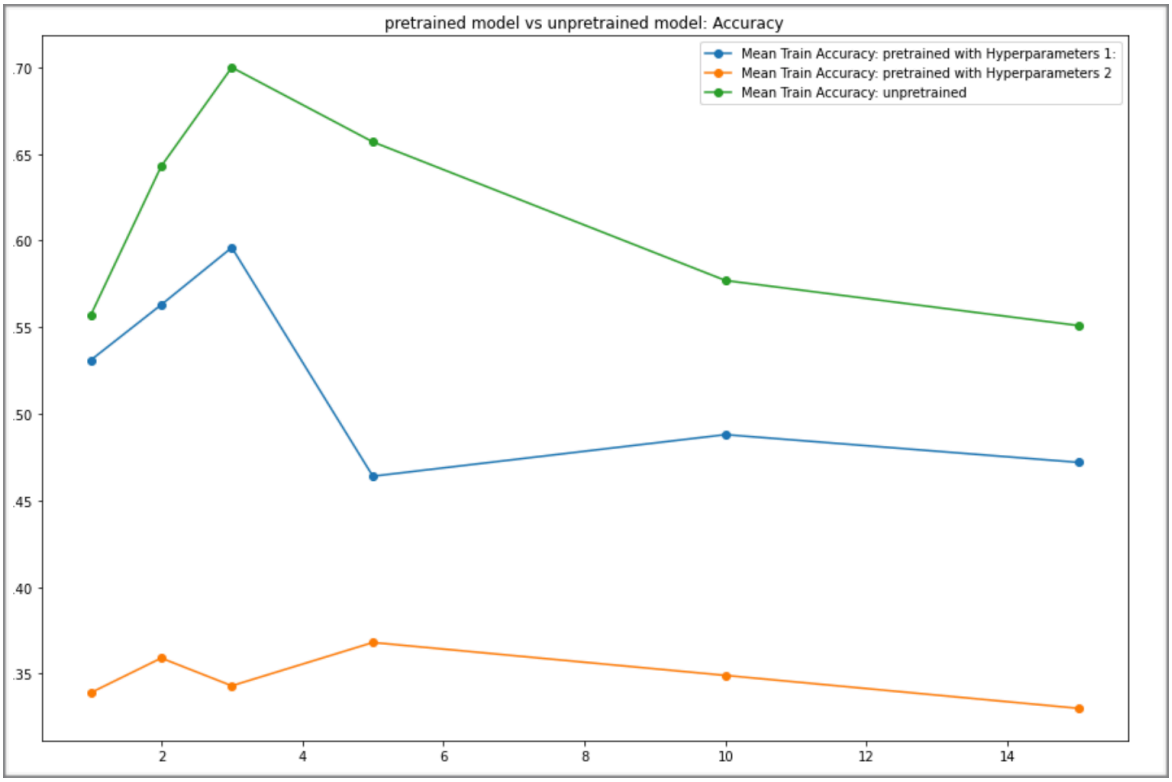
1. batch_size: 20
2. num_train_steps_per_episode: 50
3. num_train_episodes: 4
4. optim_kwargs: {'lr': 0.0001}

Hyperparameters 2:

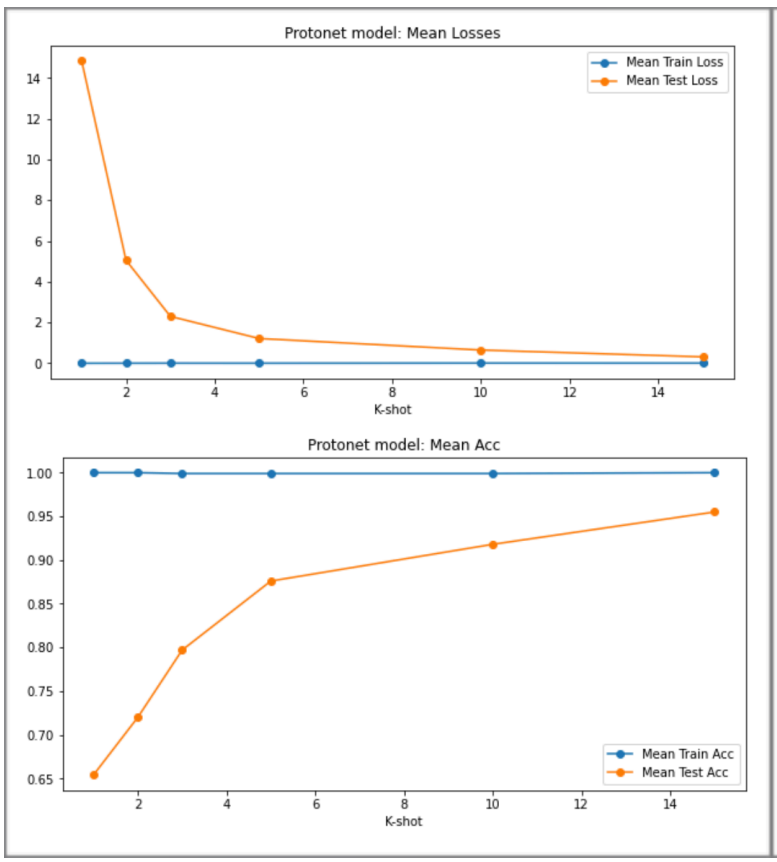
1. batch_size: 40
2. num_train_steps_per_episode: 50
3. num_train_episodes: 8
4. optim_kwargs: {'lr': 0.0001}

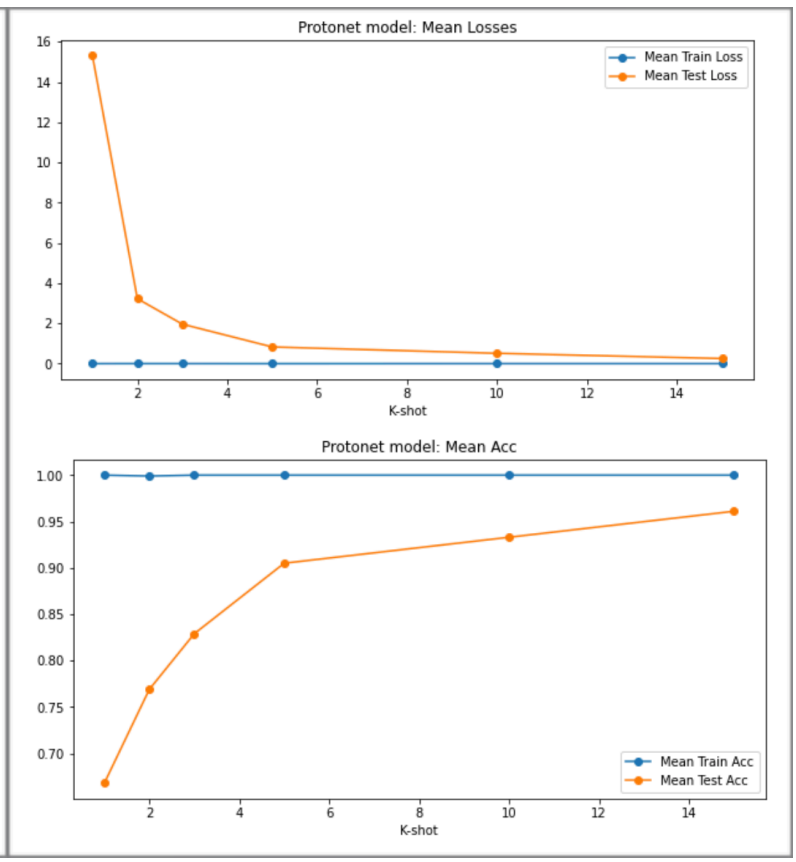
1. **Does it work better than the unpretrained baseline? If not, why?**

   As shown below, the pertained model with a different setup performs worse than the unretained model. However, it seems increasing the number of episodes in the pertained affect the model and made it perform better. In my opinion, the pertained perform worse because it has been fed during the training was less than the unpretrained.



Protonet Model:





Hyperparameters 1:

1. batch_size: 20
2. num_train_steps_per_episode: 50
3. num_train_episodes: 4
4. optim_kwargs: {'lr': 0.001}

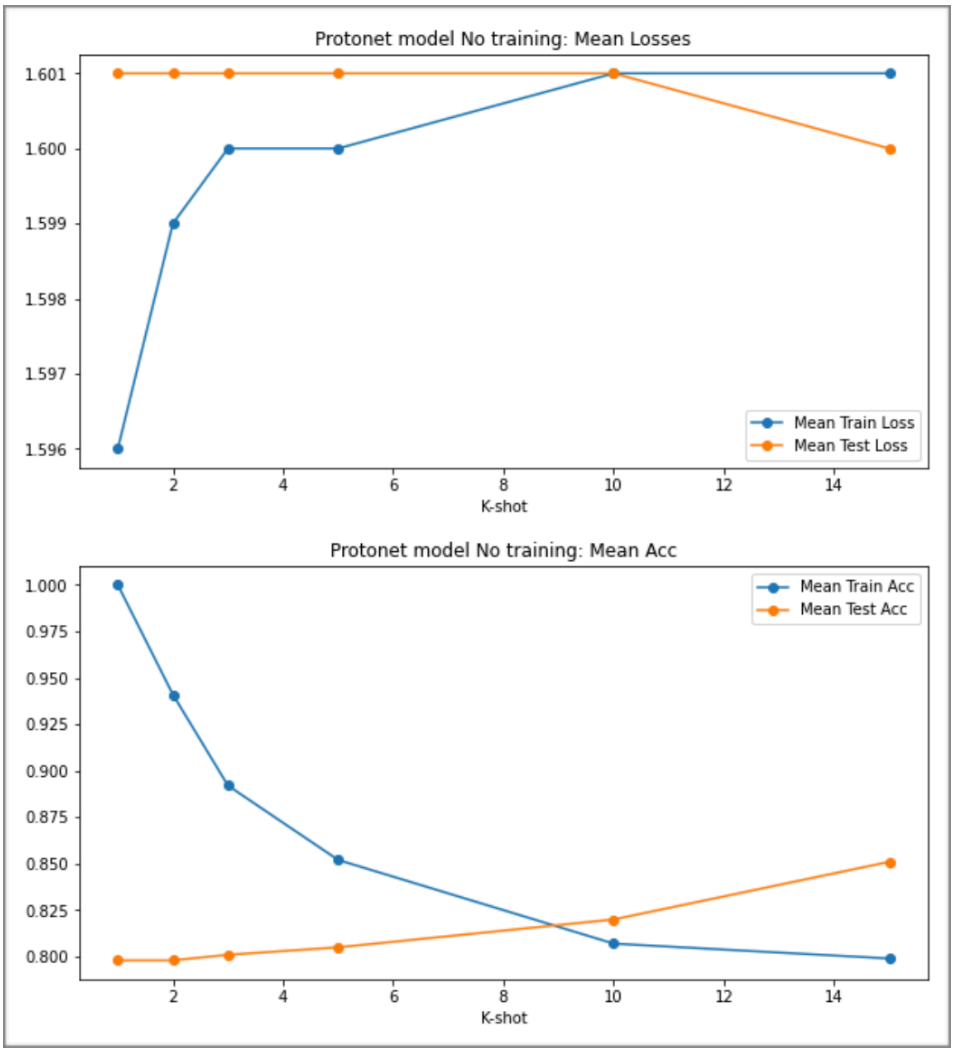
Hyperparameters 2:

1. batch_size: 40
2. num_train_steps_per_episode: 50
3. num_train_episodes: 8
4. optim_kwargs: {'lr': 0.001}

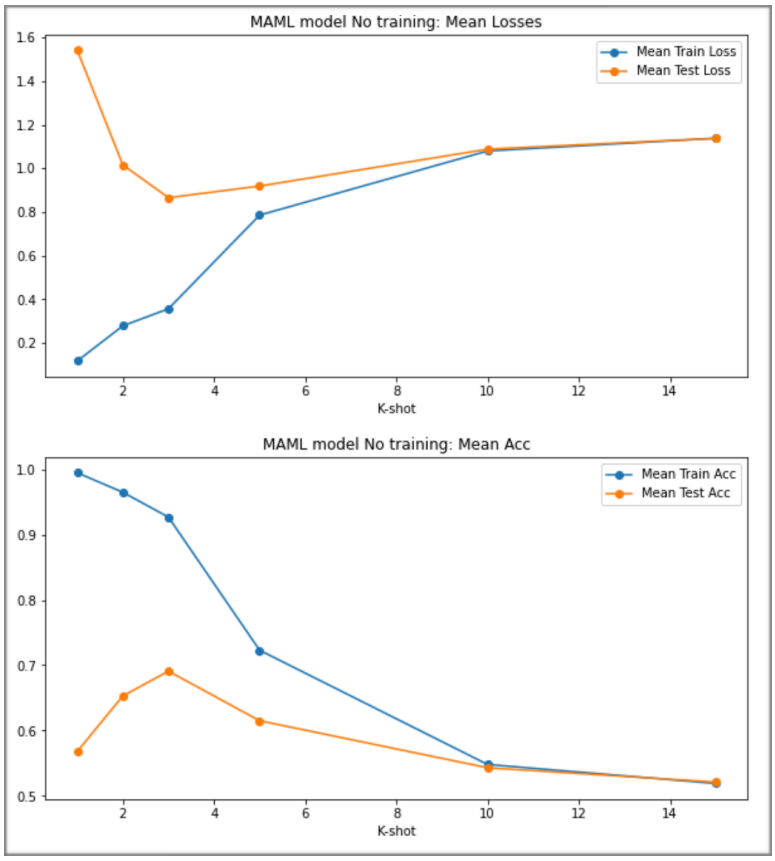
- **What are the advantages of the model? What are the disadvantages? How is it better/worse compared to other methods in terms of the quantitative performance? In terms of the implementation simplicity? In terms of training speed? In terms of inference speed? In terms of memory consumption?**

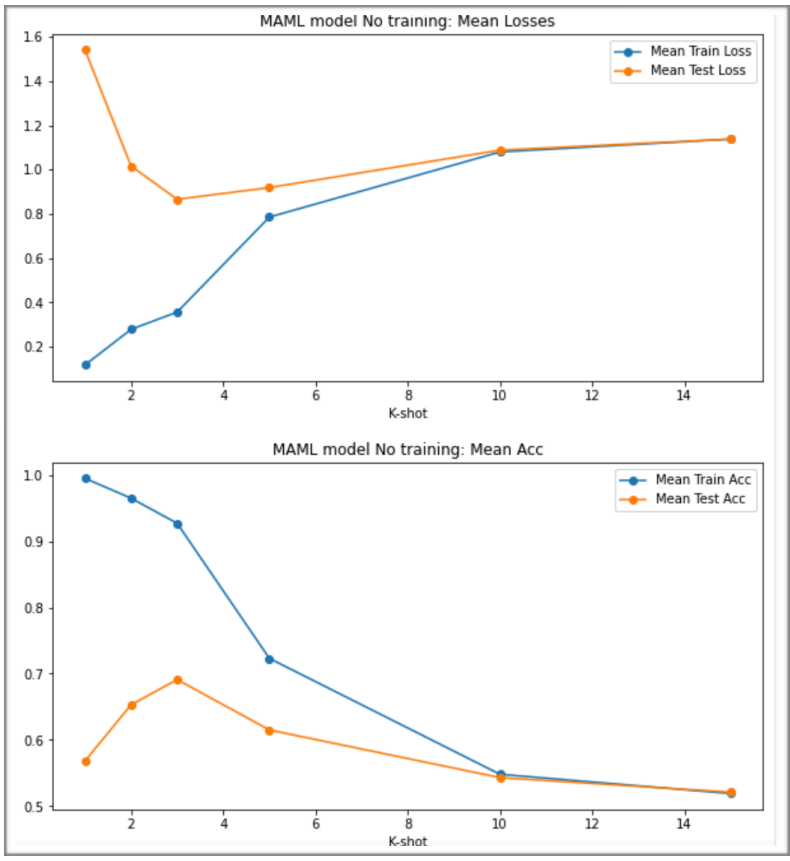
  The main advantage of Protonet is its simplicity. Protonet is simple to implement, highly extensible, and fast at Test-Time. However, it consumes a lot of computational resources.

- **How well does it perform, when we do not train the model at all? Why?**
  It will perform like k-means clustering. I setup the lr =0 and num_train_episodes = 0. It seems that it performs very badly. When the K-shot is small the model severs from the overfitting. However, this issue disappeared when K-shot increasing. In my opinion, this is due to the number of episodes when it sets to zero means the model doesn't have a class sample to compute the centers and ties to compute them randomly.



MAML Model:





Hyperparameters 1:

1. inner_loop_lr : 30
2. num_inner_steps: 50
3. ft_optim_kwargs: 'lr': 0.001
4. Seeds: 42

Hyperparameters 2:

1. inner_loop_lr : 40
2. num_inner_steps: 60
3. ft_optim_kwargs: 'lr': 0.001
4. Seeds: 1

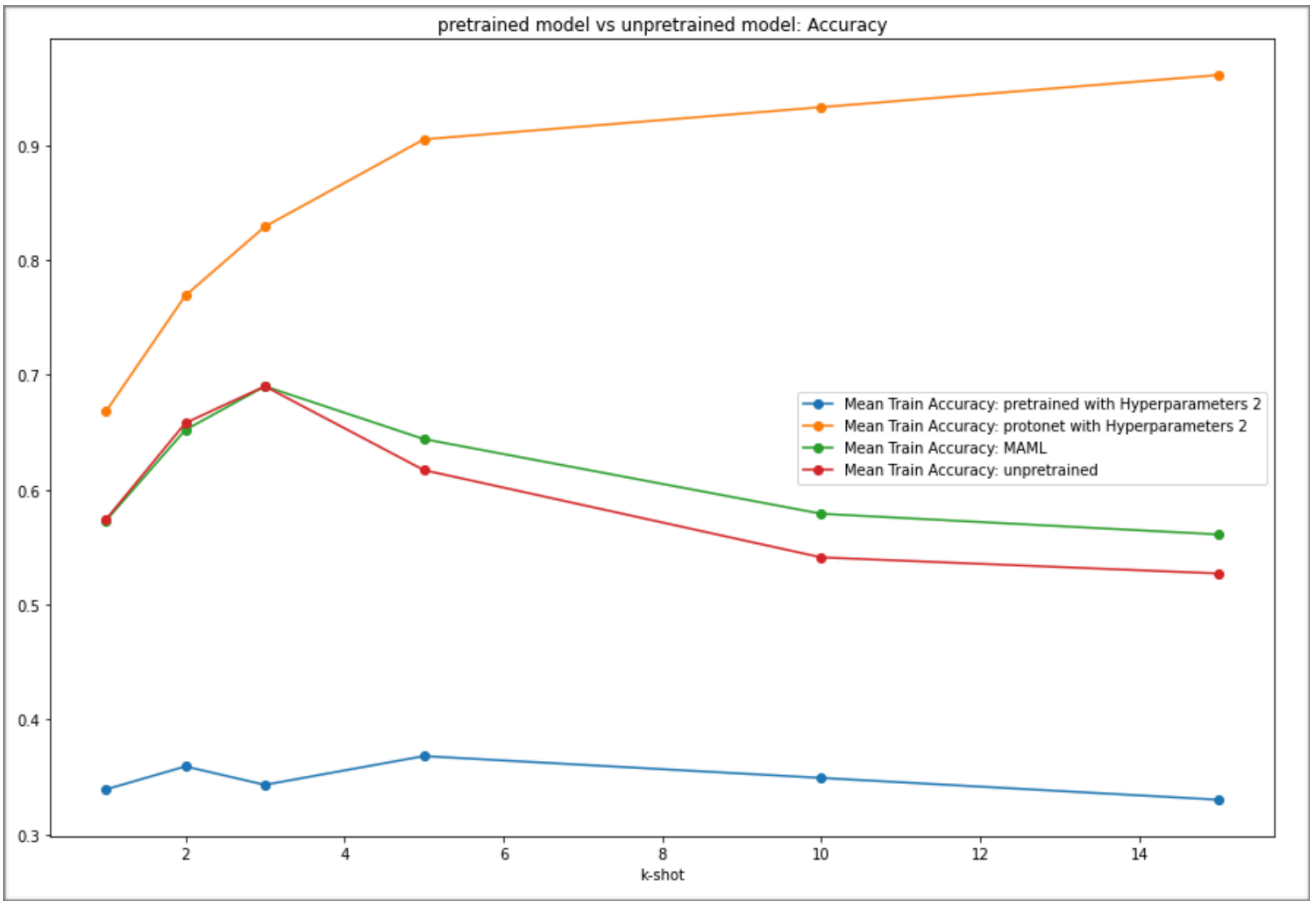
Finding: When the k-shot increases the model seems to preform worse. In my opinion, this due to the gradient degradation (vanishing and exploding gradients). However, I don't figure out why the model is retrieving the same result regardless changing the **Hyperparameters. Below answers based on my reading since the model is not working appropriately.**

2. **What are the advantages/disadvantages of MAML? How does it compare to other methods in terms of performance/training speed/simplicity/ease of implementation?**
   MAML compare to the above models is more complex. Also, it takes longer Test-Time, and it consumes less computational resources than ProtoNet. Although it requires a large amount of GPU memory space to perform gradient updates.

   1. **Why MAML is unstable? How can we alleviate this?**
      MAML is unstable and extremely sensitive to hyperparameter and architecture changes. The instability in MAML is due to the gradient degradation (vanishing and exploding gradients). Based on my reading, to solve this problem we can add Adding skip-connections that connect the iterations or by using explicit gradients. Explicit gradients can be computed by computing getting the target set loss after every inner loop update, and then computing a weighted average of the per-step losses to be the optimization loss.

3. **If you implemented the model in pytorch, explain why we couldn't use a traditional nn.Module for MAML implementation.**

   This due to Pytorch impelamtion of nn.Module. Pytorch can't differentiate through Module parameters updates.



ProtoNet outperforms all the 3 models. MAML and unpretrained model preforms nearly the same (I think this due to a logical error).