CS330 Autumn 2023 Homework 1
Data Processing and Black-Box Meta-Learning

Due Wednesday October 16, 11:59 PM PST

SUNet ID:

Name:

Collaborators:

By turning in this assignment, I agree by the Stanford honor code and declare that all of this is my own work.

Overview

Goals: In this assignment, we will look at meta-learning for few shot classification. You will:

1. Learn how to process and partition data for meta learning problems, where training is done over a distribution of training tasks $p(\mathcal{T})$.
2. Implement and train memory augmented neural networks, a black-box meta-learner that uses a recurrent neural network [1].
3. Analyze the learning performance for different size problems.
4. Experiment with model parameters and explore how they improve performance.

We have provided you with the starter code, which can be downloaded from the course website. We will be working with Omniglot [2], a dataset with 1623 characters from 50 different languages. Each character has 20 28x28 images. We are interested in training models for K-shot, N-way classification, i.e. training a classifier to distinguish between N previously unseen characters, given only K labeled examples of each character.

PDF Submission: To submit your homework, submit one PDF report to Gradescope containing written answers and Tensorboard graphs (screenshots are fine) to the questions below. The PDF should also include your name and any students you talked to or collaborated with. **Any written responses or plots to the questions below must appear in your PDF submission.**

Code Submission. You should only modify the following two files:

- `submission/load_data.py`
- `submission/mann.py`

However, when submitting to Gradescope, you should **include all the files** inside your `submission` folder, which includes those `.npy` files. **Please only submit the files but not the** `submission` **folder.**

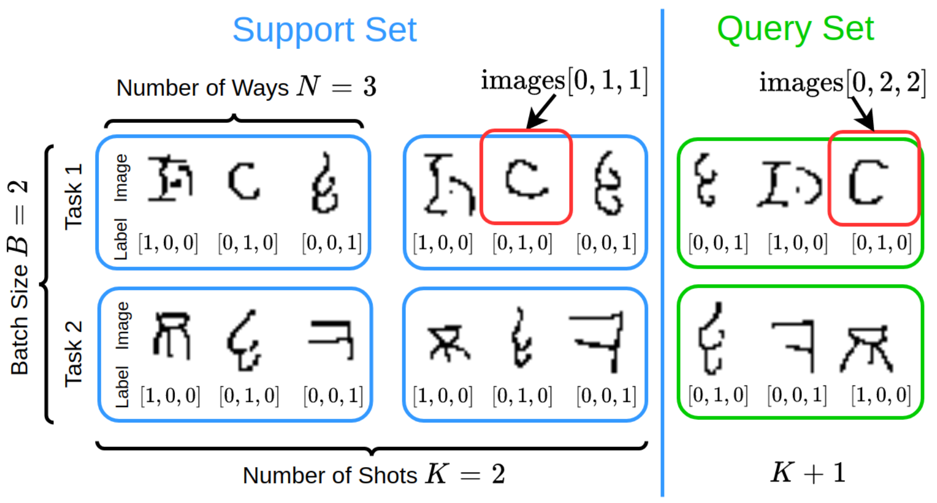


Figure 1: Example data batch from the Data Generator. The first K sets of images form the support set and are passed in the *same order*. The final set of images forms the query set and must be shuffled.

Optional. Autograding Your Code. In this homework, we include autograding functionalities in the released code to facilitate you to debug and develop your code. To run the autograder, simply do:

```
python grader.py
```

The maximum points you can get when running the autograder is **15 / 15 points**. We also have **27 points** from hidden test cases that show up when you submit your code to Gradescope. This makes the total of **42 points** for all the Coding parts. Note that some of the test cases require you to have the `mann_results_*.npy` files ready, which are auto-generated when you finished all the four runs required by **Problem 3**.

[15 points (Coding)]
Problem 1: Data Processing for Few-Shot Classification

Before training any models, you must write code to sample batches for training. Fill in the `_sample` function in the `DataGenerator` class in `load_data.py`. The class already has variables defined for batch size `batch_size` (B), number of classes `num_classes` (N), and number of samples per class `num_samples_per_class` ($K + 1$). Your code should:

1. Sample N different characters from either the specified train, test, or validation folder.

2. Load $K + 1$ images per character and collect the associated labels, using K images per class for the support set and 1 image per class for the query set.

3. Format the data and return two tensors, one of flattened images with shape $[K + 1, N, 784]$ and one of one-hot labels $[K + 1, N, N]$.

Note that your code only needs to return one single (image, label) tuple.

You **do not need to handle batching in function** since we can basically batch the inputs using an instance of torch.utils.data.DataLoader for multiple tasks, and the final shape input images to your MANN model is thus $[B, K+1, N, 784]$, and that of the input labels is $[B, K+1, N, N]$, where B is the batch size. Check the declaration of `meta_train_iterable` and `meta_train_iterable` in `main.py` on how we use your implemented `DataGenerator`.

Figure 1 illustrates the data organization. In this example, we have: (1) images from $N = 3$ different classes; (2) we are provided $K = 2$ sets of labeled images in the support set and (3) our batch consists of only two tasks, i.e. $B = 2$.

1. We will sample both the support and query sets as a single batch, hence one batch element should obtain image and label tensors of shapes $[K+1, N, 784]$ and $[K+1, N, N]$ respectively. In the example of Fig. 1, `images[0, 1, 1]` would be the image of the letter "C" in the support set with corresponding class label $[0, 1, 0]$ and `images[0, 2, 2]` would be the the letter "C" in the query set (with the same label).

2. We must shuffle the order of examples in the **query set**. In principle, you should be able to shuffle the order of data in the support set as well; however, this makes the model optimization much harder. **You should feed the support set examples in the same, fixed order**. In the example above, the support set examples are always in the same order. Note here that images and labels should be shuffled in the same order, otherwise, the one-to-one mapping between images and labels may get messed up. Hint: we encourage you to use np.random.shuffle here.

We provide helper functions to (1) take a list of folders and provide paths to image files/labels, and (2) to take an image file path and return a flattened numpy matrix. The functions `np.random.shuffle` and `np.eye` will also be helpful. **Be careful about output shapes and data types!**

[15 points (Coding)]
Problem 2: Memory Augmented Neural Networks (MANN) [1, 3]

We will now be implementing few-shot classification using memory augmented neural networks (MANNs). The main idea of MANN is that the network should learn how to encode the first K examples of each class into memory such that it can be used to accurately classify the $K + 1$th example. See Figure 2 for a graphical representation of this process.

Data processing will be done as in SNAIL [3]. Each set of labels and images are concatenated together, and the $N * K$ support set examples are sequentially passed through the network as shown in Fig. 2. Then the query example of each class is fed through the network, **concatenated with 0 instead of the true label**. The loss is computed between the query set predictions and the ground truth labels, which is then backpropagated through the network. **Note**: The loss is *only* computed on the set of N query images, which comprise of the last examples from each character class.

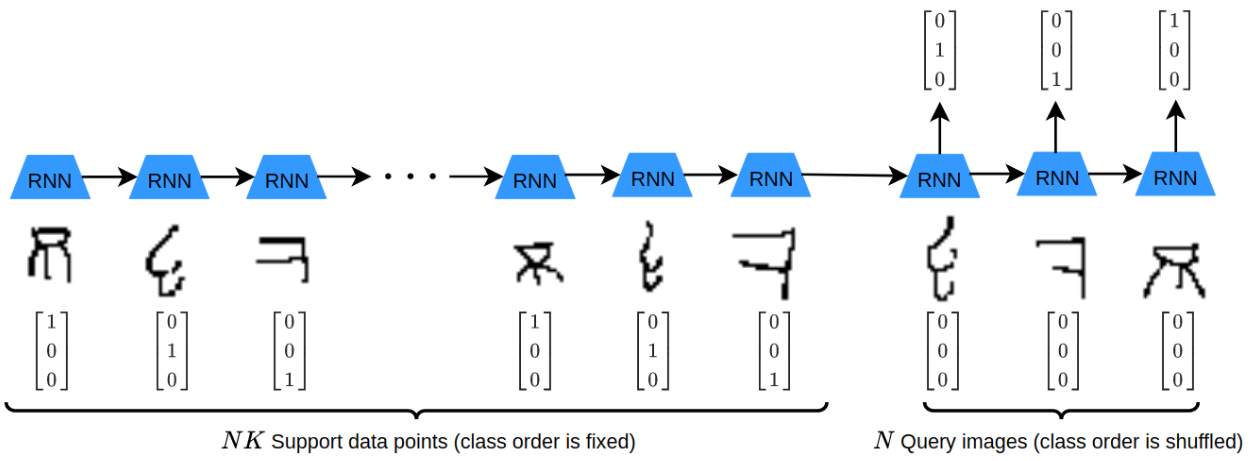

RNN
NK Support data points (class order is fixed)
N Query images (class order is shuffled)

Figure 2: Feed K labeled examples of each of N classes through the memory-augmented network. Then feed final set of N examples and optimize to minimize loss.

In the `mann.py` file:

1. Fill in the `forward` function of the `MANN` class to take in image tensor of shape $[B, K+1, N, 784]$ and a label tensor of shape $[B, K+1, N, N]$ and output label predictions of shape $[B, K+1, N, N]$. The layers to use have already been defined for you in the `__init__` function, and you can just call them in your `forward` code. *Hint: Remember to pass zeros, not the ground truth labels for the final N examples.*

2. Fill in the function called `loss_function` in the `MANN` class which takes as input the $[B, K+1, N, N]$ labels and $[B, K+1, N, N]$ predicted labels and computes the cross entropy loss only on the N test images.

Note: Both of the above functions will need to be backpropogated through, so they need to be written in PyTorch in a differentiable way.

[12 points (Coding)][4 points (Plot)][6 points (Written)] Problem 3: Analysis

Once you have completed problems 1 and 2, you can train your few shot classification model. You should observe both the support and query losses go down, and the query accuracy go up. Now we will examine how the performance varies for different size problems. Train models for the following values of K and N:

- $K = 1, N = 2$
- $K = 2, N = 2$
- $K = 1, N = 3$

- $K = 1, N = 4$

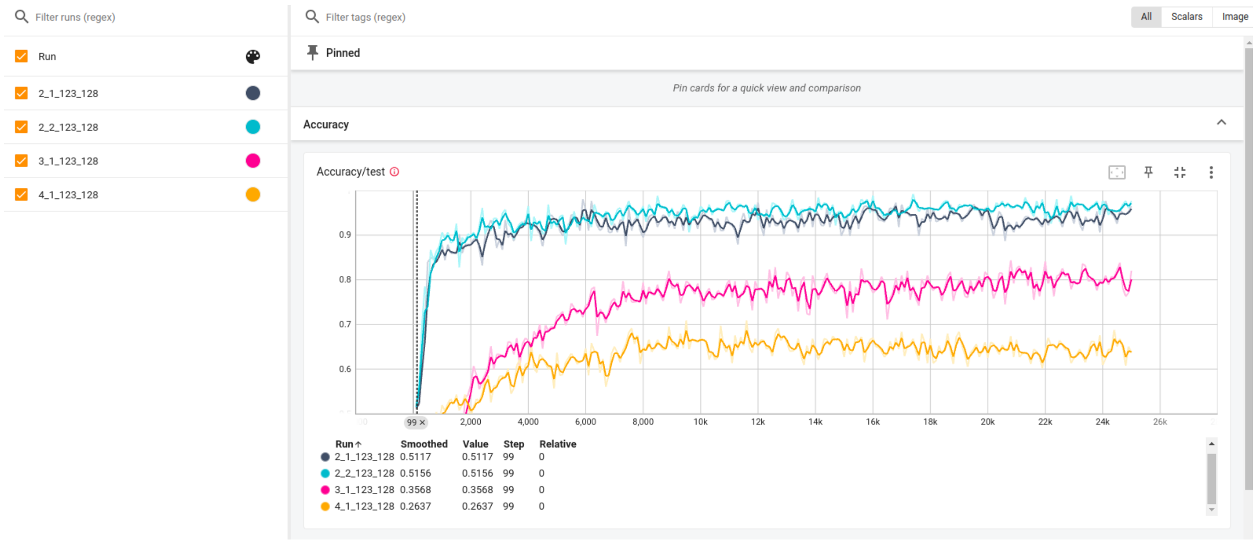
Example code:
```
python main.py --num_shot K --num_classes N
```
For checking training results and/or taking a screenshot for the writeup, use:
```
tensorboard --logdir runs/
```
You should start with the case $K = 1, N = 2$ as it can aid you in the implementation and debugging process. Your model should be able to achieve a query set accuracy of above 90% in this first two scenarios scenario on held-out test tasks, around 80% in the second scenario, and more than 60% in the final scenario.

Note: The computation of this homework should be doable on your laptop's CPU. For reference, running one experiment takes around $3mins$ on our TA's Apple MacPro M1.

Answer the following questions:

1. **[4 points (Plot)]** For each configuration, submit a plot of the meta-test query set classification accuracy over training iterations (A TensorBoard screenshot with all plots in Figure is fine).

   Solution:

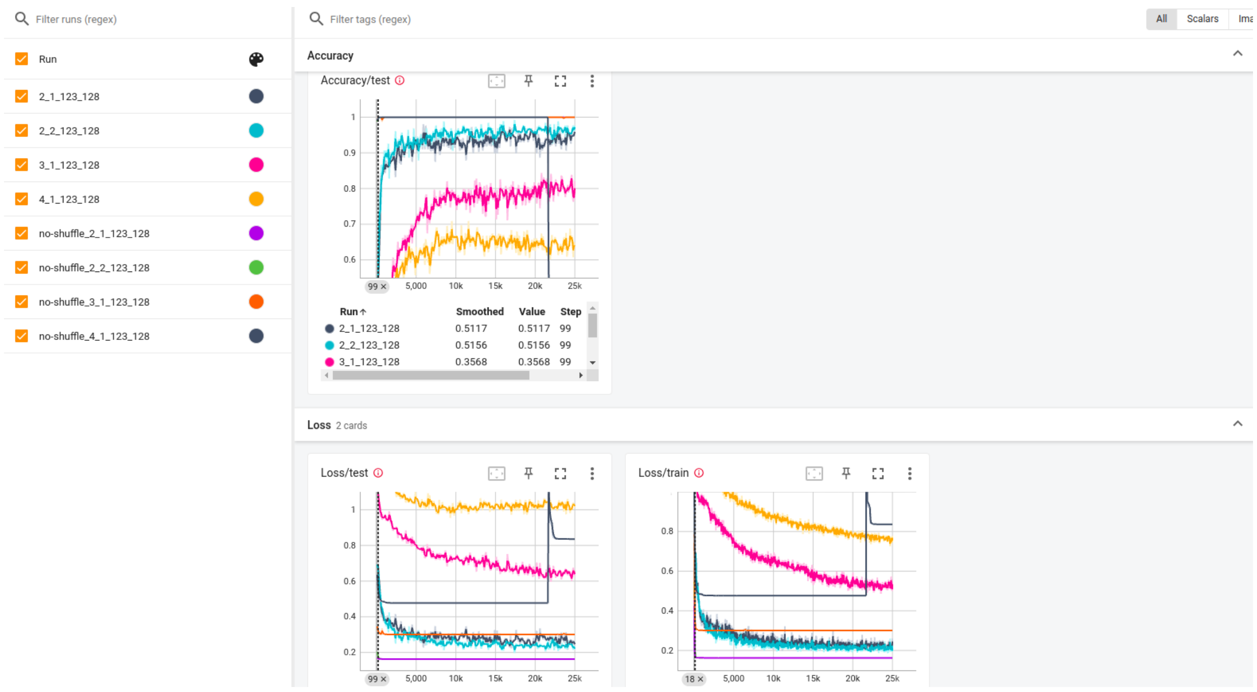


	Run ↑	Smoothed	Value	Step	Relative
●	2_1_123_128	0.5117	0.5117	99	0
●	2_2_123_128	0.5156	0.5156	99	0
●	3_1_123_128	0.3568	0.3568	99	0
●	4_1_123_128	0.2637	0.2637	99	0

Figure 3: Meta-test query set classification accuracy.

2. **[2 points (Written)]** In our problem, we shuffled the order of examples in the query set. Do we really need to perform this step? What happens if we do not shuffle?

   Solution: As shown in Figure 4, The model can use the order provided by the support set to predict the labels in the query set. It causes model to learn poorly or even not learn at all. So the shuffle step is necessary and essential for model training.



- ☑ Run
- ☑ 2_1_123_128 ●
- ☑ 2_2_123_128 ●
- ☑ 3_1_123_128 ●
- ☑ 4_1_123_128 ●
- ☑ no-shuffle_2_1_123_128 ●
- ☑ no-shuffle_2_2_123_128 ●
- ☑ no-shuffle_3_1_123_128 ●
- ☑ no-shuffle_4_1_123_128 ●

Run ↑	Smoothed	Value	Step
● 2_1_123_128	0.5117	0.5117	99
● 2_2_123_128	0.5156	0.5156	99
● 3_1_123_128	0.3568	0.3568	99

Figure 4: The performance of training and testing under shuffling query set and without shuffling.

3. **[2 points (Written)]** How does increasing the number of classes affect learning and performance?

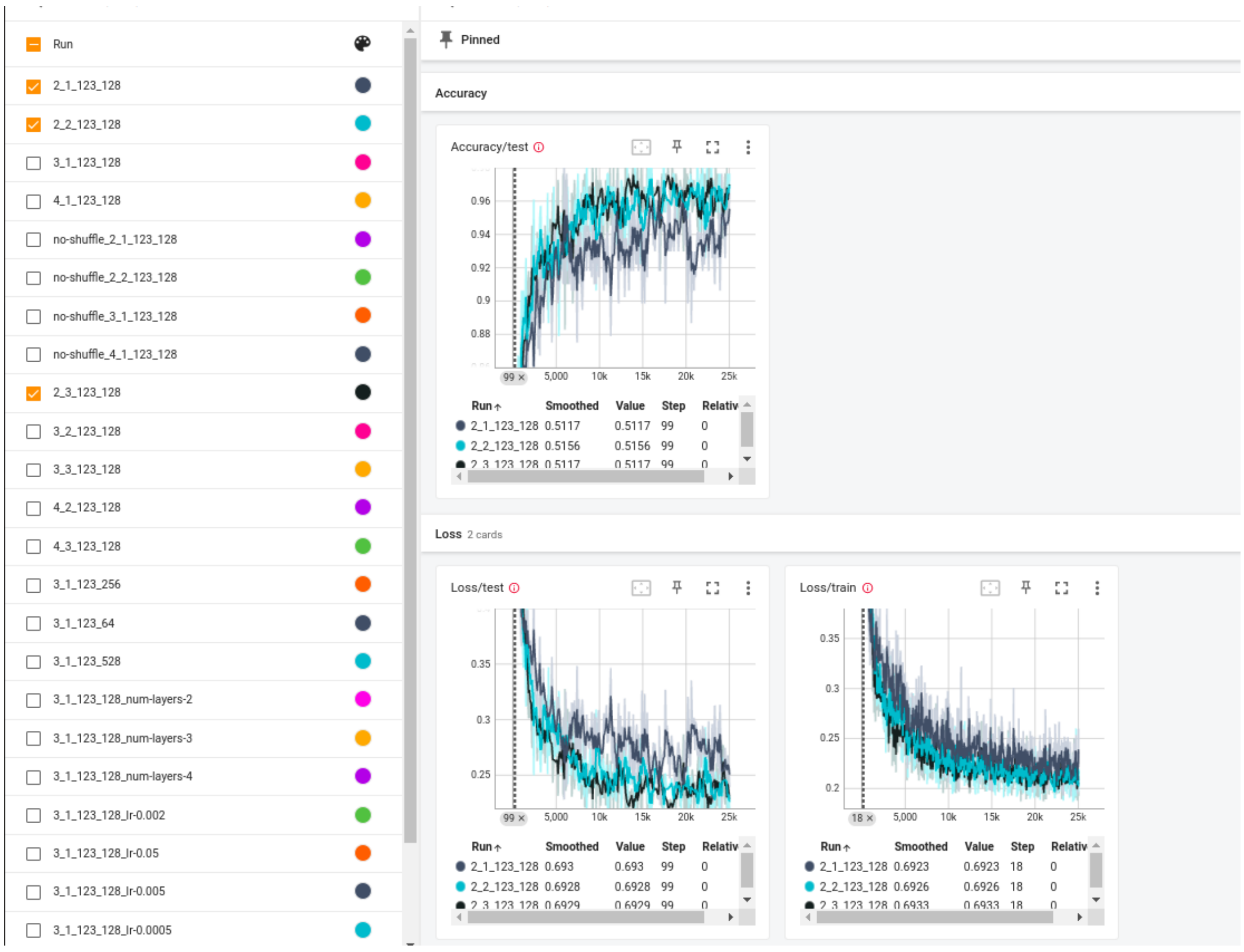
   Solution: From Figure 4, we can see that the performance of model is worse with more number of classes. With 2 classes, model attains the best performance on both meta-training and meta-testing data.

4. **[2 points (Written)]** How does increasing the number of examples in the support set affect performance?

   Solution: As shown in Figure 5, 6, 7. Typically with more examples in one class, the performance should be better. However, there's also not much performance increase if with more examples. The effect is much more clear under 4 task classes.

[5 points (Plot, Written)][5 points (Extra Credits)] Problem 4: Experimentation

a **[5 points (Plot, Written)]** Experiment with one hyperparameter that affects the performance of the model, such as the type of recurrent layer, size of hidden state, learning rate, or number of layers. Submit a plot that shows how the meta-test query set classification accuracy of the model changes on 1-shot, 3-way classification as you



▬	Run	
✓	2_1_123_128	●
✓	2_2_123_128	●
☐	3_1_123_128	●
☐	4_1_123_128	●
☐	no-shuffle_2_1_123_128	●
☐	no-shuffle_2_2_123_128	●
☐	no-shuffle_3_1_123_128	●
☐	no-shuffle_4_1_123_128	●
✓	2_3_123_128	●
☐	3_2_123_128	●
☐	3_3_123_128	●
☐	4_2_123_128	●
☐	4_3_123_128	●
☐	3_1_123_256	●
☐	3_1_123_64	●
☐	3_1_123_528	●
☐	3_1_123_128_num-layers-2	●
☐	3_1_123_128_num-layers-3	●
☐	3_1_123_128_num-layers-4	●
☐	3_1_123_128_lr-0.002	●
☐	3_1_123_128_lr-0.05	●
☐	3_1_123_128_lr-0.005	●
☐	3_1_123_128_lr-0.0005	●

Accuracy/test

Run ↑	Smoothed	Value	Step	Relativ
● 2_1_123_128	0.5117	0.5117	99	0
● 2_2_123_128	0.5156	0.5156	99	0
● 2_3_123_128	0.5117	0.5117	99	0

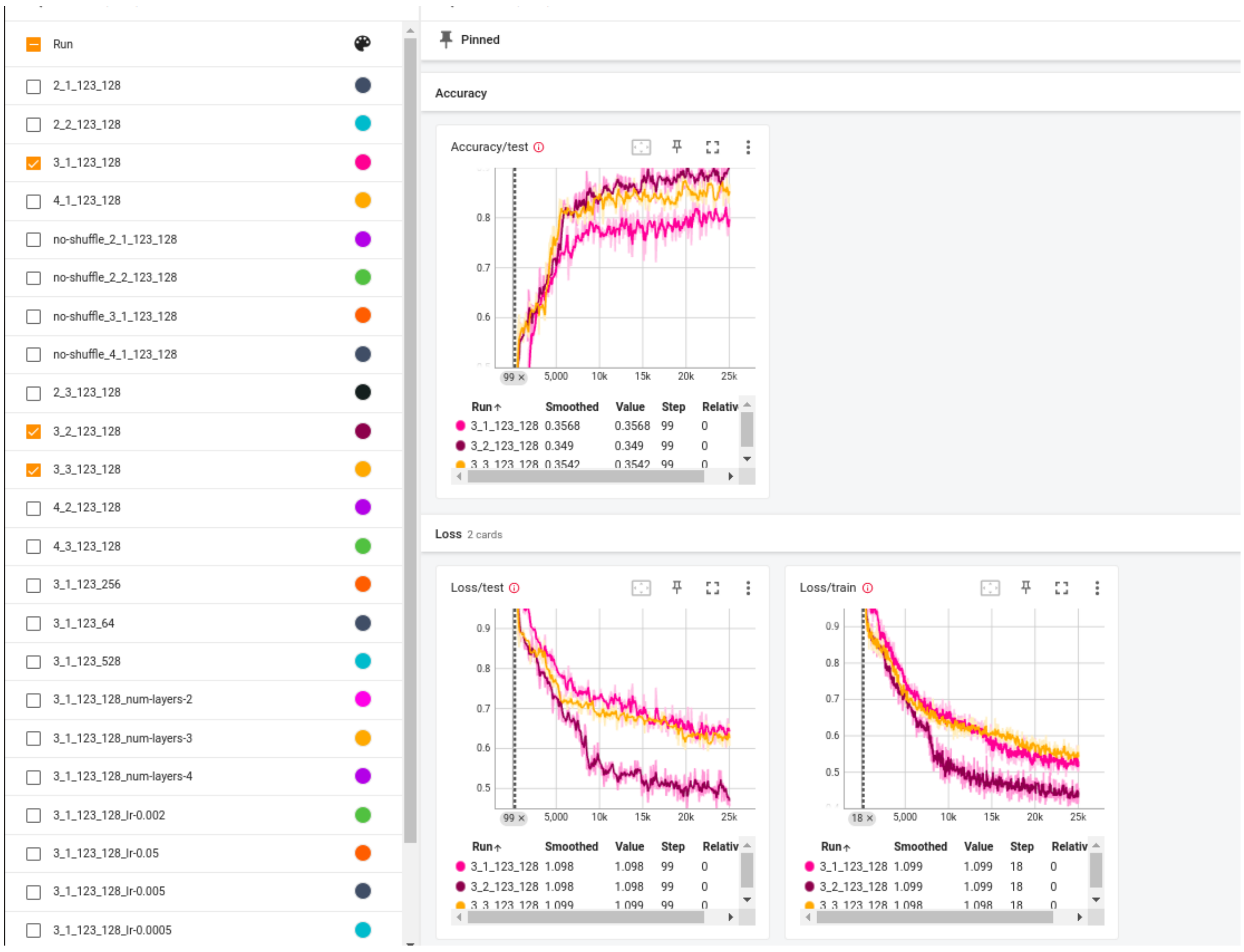
Loss/test

Run ↑	Smoothed	Value	Step	Relativ
● 2_1_123_128	0.693	0.693	99	0
● 2_2_123_128	0.6928	0.6928	99	0
● 2_3_123_128	0.6929	0.6929	99	0

Loss/train

Run ↑	Smoothed	Value	Step	Relativ
● 2_1_123_128	0.6923	0.6923	18	0
● 2_2_123_128	0.6926	0.6926	18	0
● 2_3_123_128	0.6933	0.6933	18	0

Figure 5: Performance with different number of examples in the support set, the results are attained under 2 task classes.



	Run	
☐	2_1_123_128	●
☐	2_2_123_128	●
☑	3_1_123_128	●
☐	4_1_123_128	●
☐	no-shuffle_2_1_123_128	●
☐	no-shuffle_2_2_123_128	●
☐	no-shuffle_3_1_123_128	●
☐	no-shuffle_4_1_123_128	●
☐	2_3_123_128	●
☑	3_2_123_128	●
☑	3_3_123_128	●
☐	4_2_123_128	●
☐	4_3_123_128	●
☐	3_1_123_256	●
☐	3_1_123_64	●
☐	3_1_123_528	●
☐	3_1_123_128_num-layers-2	●
☐	3_1_123_128_num-layers-3	●
☐	3_1_123_128_num-layers-4	●
☐	3_1_123_128_lr-0.002	●
☐	3_1_123_128_lr-0.05	●
☐	3_1_123_128_lr-0.005	●
☐	3_1_123_128_lr-0.0005	●

Accuracy/test

Run	Smoothed	Value	Step	Relativ
3_1_123_128	0.3568	0.3568	99	0
3_2_123_128	0.349	0.349	99	0
3_3_123_128	0.3542	0.3542	99	0

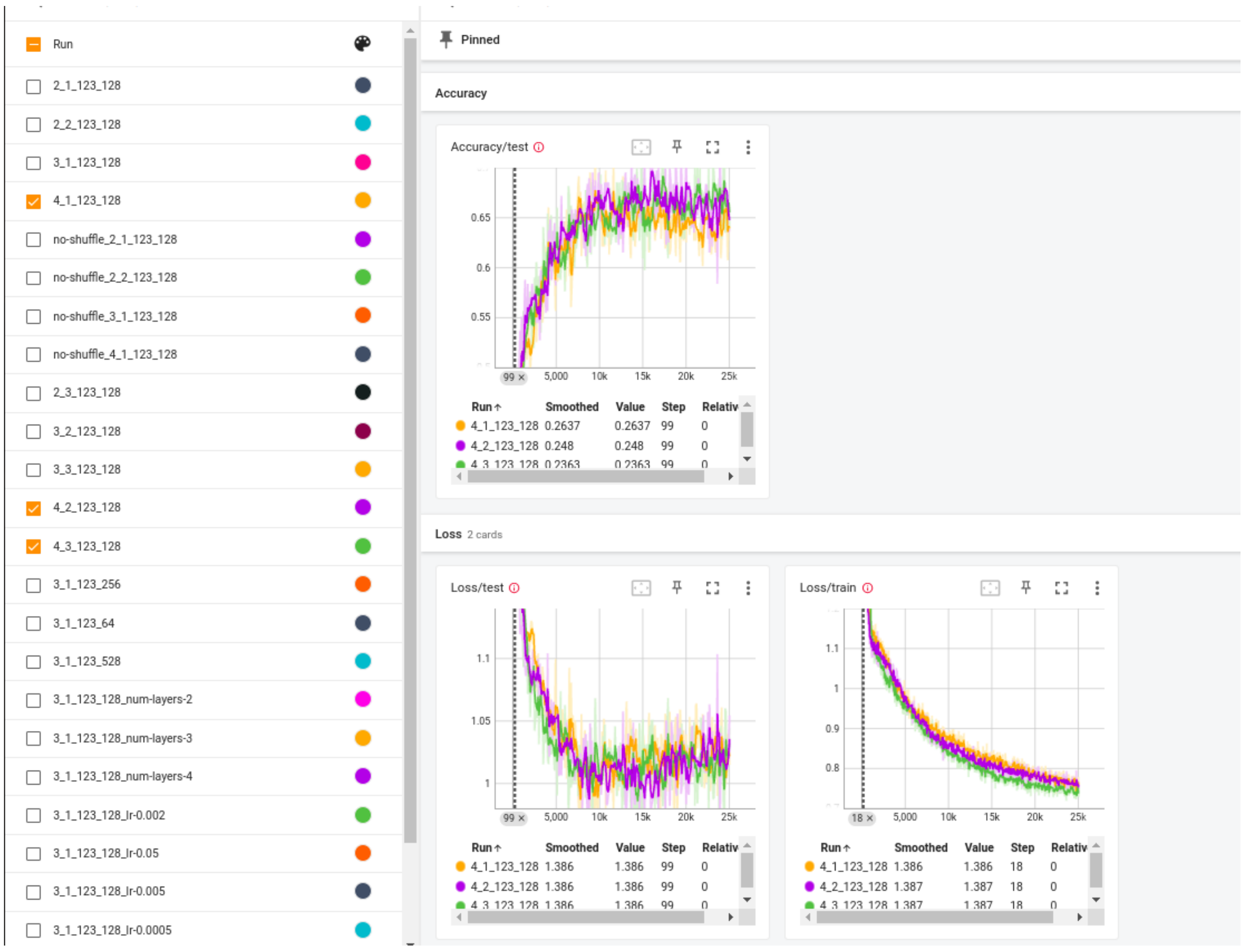
Loss/test

Run	Smoothed	Value	Step	Relativ
3_1_123_128	1.098	1.098	99	0
3_2_123_128	1.098	1.098	99	0
3_3_123_128	1.099	1.099	99	0

Loss/train

Run	Smoothed	Value	Step	Relativ
3_1_123_128	1.099	1.099	18	0
3_2_123_128	1.099	1.099	18	0
3_3_123_128	1.098	1.098	18	0

Figure 6: Performance with different number of examples in the support set, the results are attained under 3 task classes.



	Run	
☐	2_1_123_128	●
☐	2_2_123_128	●
☐	3_1_123_128	●
✓	4_1_123_128	●
☐	no-shuffle_2_1_123_128	●
☐	no-shuffle_2_2_123_128	●
☐	no-shuffle_3_1_123_128	●
☐	no-shuffle_4_1_123_128	●
☐	2_3_123_128	●
☐	3_2_123_128	●
☐	3_3_123_128	●
✓	4_2_123_128	●
✓	4_3_123_128	●
☐	3_1_123_256	●
☐	3_1_123_64	●
☐	3_1_123_528	●
☐	3_1_123_128_num-layers-2	●
☐	3_1_123_128_num-layers-3	●
☐	3_1_123_128_num-layers-4	●
☐	3_1_123_128_lr-0.002	●
☐	3_1_123_128_lr-0.05	●
☐	3_1_123_128_lr-0.005	●
☐	3_1_123_128_lr-0.0005	●

Accuracy/test

Run ↑	Smoothed	Value	Step	Relativ
4_1_123_128	0.2637	0.2637	99	0
4_2_123_128	0.248	0.248	99	0
4_3_123_128	0.2363	0.2363	99	0

Loss/test

Run ↑	Smoothed	Value	Step	Relativ
4_1_123_128	1.386	1.386	99	0
4_2_123_128	1.386	1.386	99	0
4_3_123_128	1.386	1.386	99	0

Loss/train

Run ↑	Smoothed	Value	Step	Relativ
4_1_123_128	1.386	1.386	18	0
4_2_123_128	1.387	1.387	18	0
4_3_123_128	1.387	1.387	18	0

Figure 7: Performance with different number of examples in the support set, the results are attained under 4 task classes.

change the parameter. Provide a brief rationale for why you chose the parameter and what you observed in the caption for the plot. **If you aim to do the following Extra Credit question, you need to test something different than the LSTM hidden dimension here.**

Solution: The results can be seen by Figure 8, 9, 10, 11.

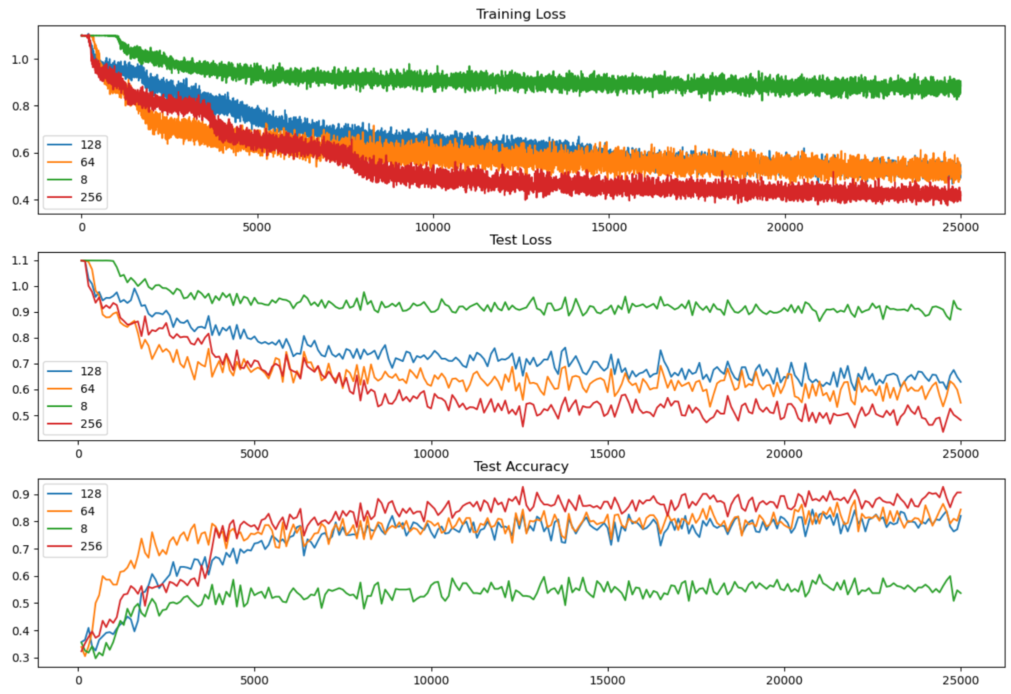


Figure 8: The effects of different sizes of hidden state on 1-shot, 3-way classification. The larger hidden dimension leads to greater performance.

b **[5 points (Extra Credit)]** In this question we'll explore the effect of memory representation on model performance. We will focus on the $K = 1$, $N = 3$ case.

In the previous experiments we used an LSTM model with 128 units. Consider additional memory sizes of 256, and 8. How does increasing and decreasing the memory capacity influence performance?

Solution: The results can be seen by Figure 12, 13, 14.

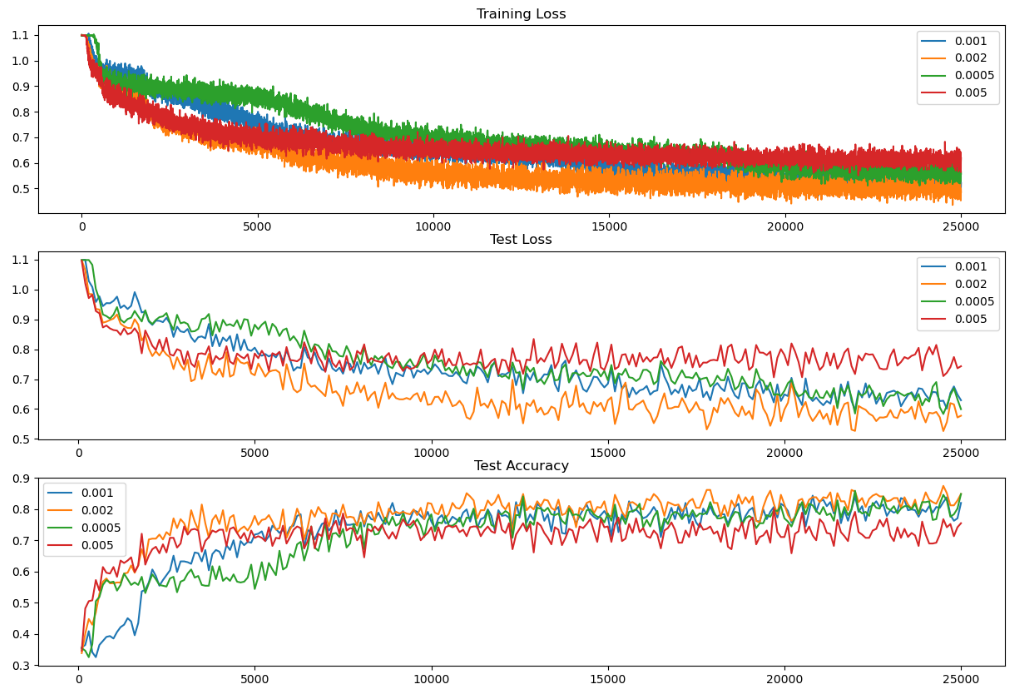


Figure 9: The effects of different learning rates on 1-shot, 3-way classification. The learning rates of 1e-3 and 2e-3 lead to the best performance.

References

[1] Adam Santoro, Sergey Bartunov, Matthew Botvinick, Daan Wierstra, and Timothy Lillicrap. Meta-learning with memory-augmented neural networks. In Maria Florina Balcan and Kilian Q. Weinberger, editors, *Proceedings of The 33rd International Conference on Machine Learning*, volume 48 of *Proceedings of Machine Learning Research*, pages 1842–1850, New York, New York, USA, 20–22 Jun 2016. PMLR.

[2] Brenden M. Lake, Ruslan Salakhutdinov, and Joshua B. Tenenbaum. Human-level concept learning through probabilistic program induction. *Science*, 350(6266):1332–1338, 2015.

[3] Nikhil Mishra, Mostafa Rohaninejad, Xi Chen, and Pieter Abbeel. Meta-learning with temporal convolutions. *CoRR*, abs/1707.03141, 2017.

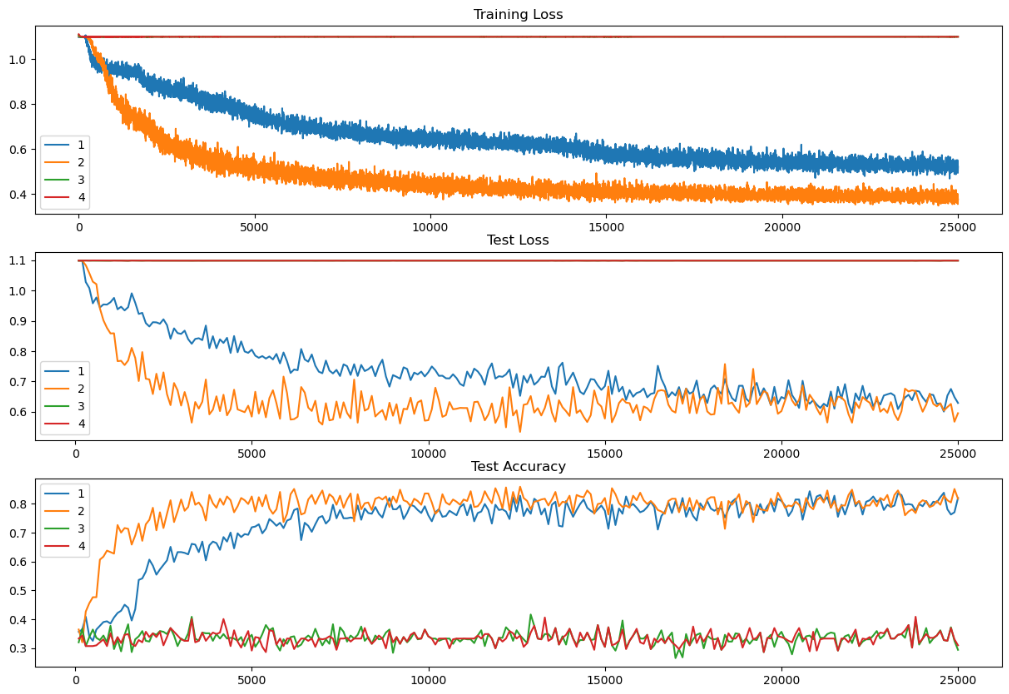


Figure 10: The effects of different number of model layers on 1-shot, 3-way classification. More layers doesn't lead to better performance, the model even doesn't learn at all.

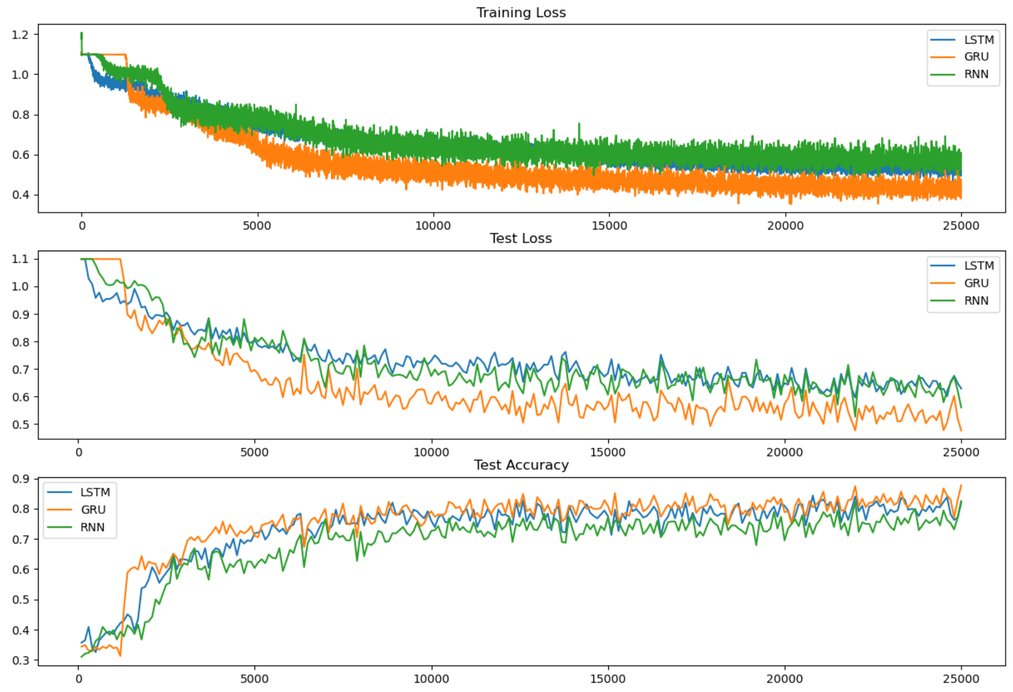


Figure 11: The effects of different types of recurrent layer on 1-shot, 3-way classification. GRU attains the best performance even though it's a simpler architecture compared to LSTM, naive RNN performs worst.

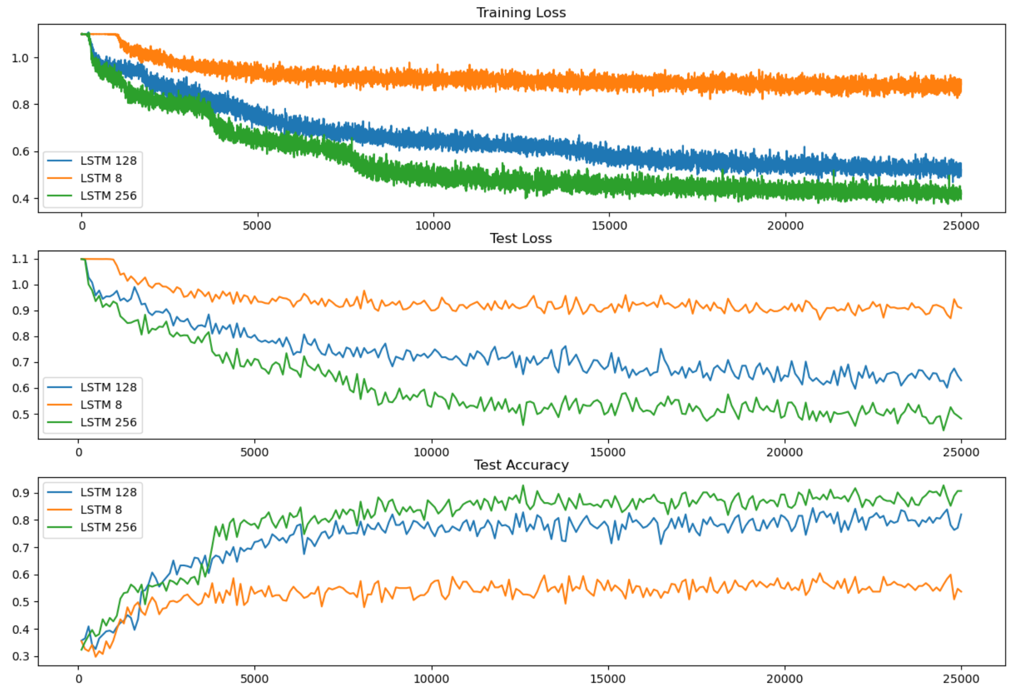


Figure 12: The effects of different sizes of hidden state using LSTM on 1-shot, 3-way classification. Adding more hidden dimensions leads to better performance. But the situation may not consist with more hidden states.

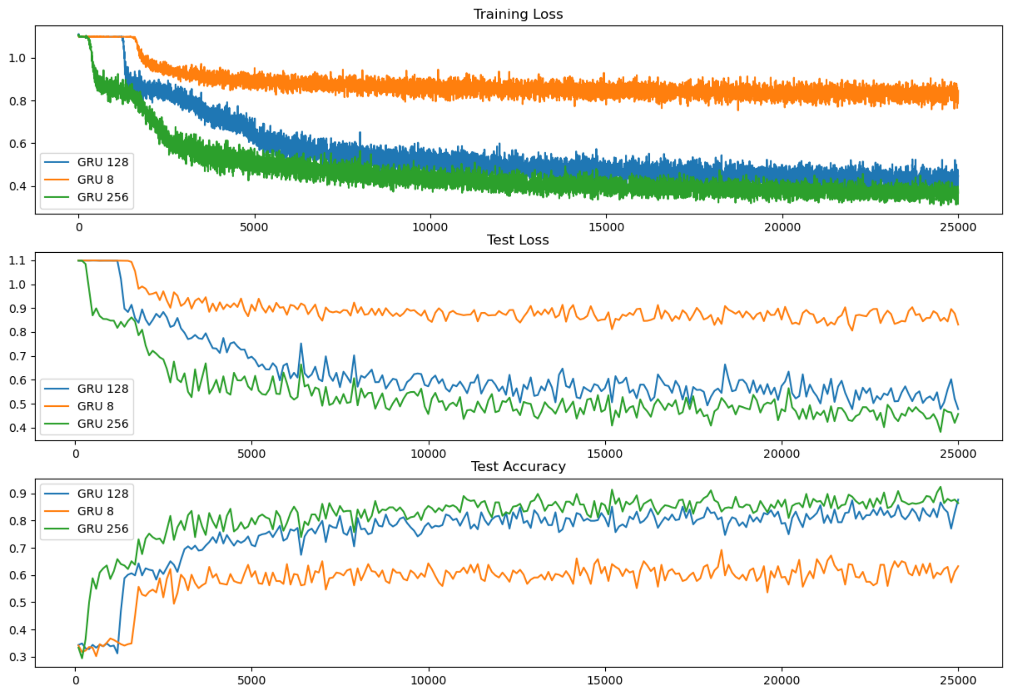


Figure 13: The effects of different sizes of hidden state using GRU on 1-shot, 3-way classification. It's similar to LSTM that adding more hidden dimensions leads to better performance.

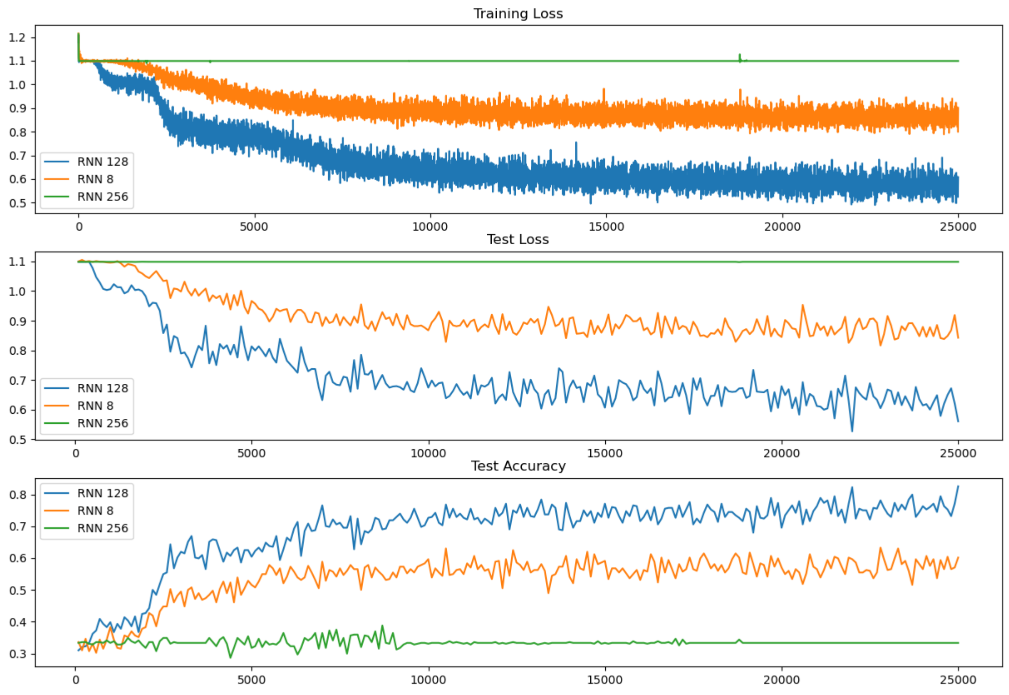


Figure 14: The effects of different sizes of hidden state using RNN on 1-shot, 3-way classification. Model with 256 hidden dimension does not learn. Compared to the lower 8 dimension, it’s more strong with 128 dimension.