

Lab 2 Report

Learning to Control SPOT using Camera Inputs

1. **Explain briefly how you train your model to achieve good performance, e.g., through data augmentation, regularization, or any other techniques. (maximum 250 words).**

Note: The model was trained using Macbook Air M2 GPU using the 2.1.0.dev20230311 version of Pytorch (nightly). GPU_NAME specified to torch was “mps” instead of “cuda”.

Specifications of the model:

Model

Pretrained Resnet18 with require_grad=True for fine-tuning

+

FC layer with intention and output of Resnet as input

MyModel(

```
(model_ft): ResNet(
  (conv1): Conv2d(3, 64, kernel_size=(7, 7), stride=(2, 2), padding=(3, 3), bias=False)
  (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (relu): ReLU(inplace=True)
  (maxpool): MaxPool2d(kernel_size=3, stride=2, padding=1, dilation=1, ceil_mode=False)
  (layer1): Sequential(
    (0): BasicBlock(
      (conv1): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    )
    (1): BasicBlock(
      (conv1): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    )
  )
  (layer2): Sequential(
    (0): BasicBlock(
      (conv1): Conv2d(64, 128, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1), bias=False)
      (bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (downsample): Sequential(
        (0): Conv2d(64, 128, kernel_size=(1, 1), stride=(2, 2), bias=False)
        (1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      )
    )
    (1): BasicBlock(
      (conv1): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    )
  )
  (layer3): Sequential(
    (0): BasicBlock(
      (conv1): Conv2d(128, 256, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1), bias=False)
      (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (downsample): Sequential(
        (0): Conv2d(128, 256, kernel_size=(1, 1), stride=(2, 2), bias=False)
        (1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      )
    )
    (1): BasicBlock(
      (conv1): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    )
  )
)
```

```

    )
)
(layer4): Sequential(
  (0): BasicBlock(
    (conv1): Conv2d(256, 512, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1), bias=False)
    (bn1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (relu): ReLU(inplace=True)
    (conv2): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
    (bn2): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (downsample): Sequential(
      (0): Conv2d(256, 512, kernel_size=(1, 1), stride=(2, 2), bias=False)
      (1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    )
  )
  (1): BasicBlock(
    (conv1): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
    (bn1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (relu): ReLU(inplace=True)
    (conv2): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
    (bn2): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  )
)
(avgpool): AdaptiveAvgPool2d(output_size=(1, 1))
(fc): Linear(in_features=512, out_features=5, bias=True)
)
(fc): Sequential(
  (0): Linear(in_features=8, out_features=5, bias=True)
)
)

```

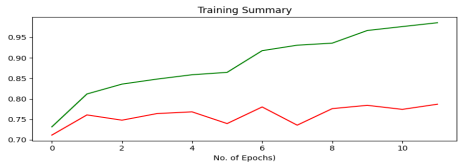
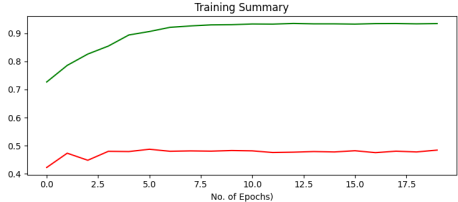
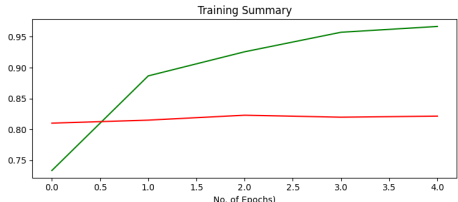
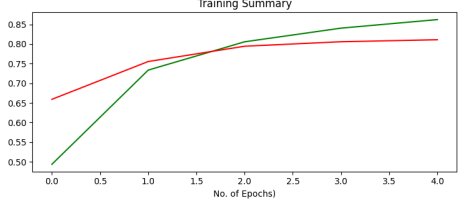
Hyperparameters:

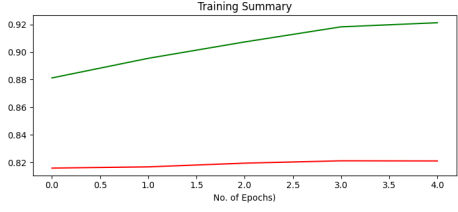
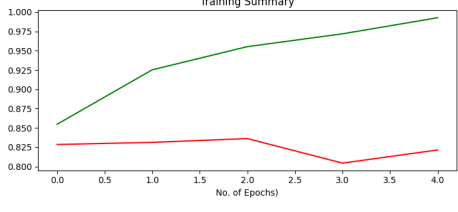
- batch_size = 64
- num_epochs = 3
- num_workers = 2
- seed = 42
- optimizer = AdamW with lr=0.0005, weight_decay=0.01
- Loss criterion = Cross Entropy Loss
- scheduler = ReduceLROnPlateau(optimizer, 'min', patience=1, factor=0.3)

Dataset preprocessing

- Image size = (224, 224)
- Normalize: mean=[0.485, 0.456, 0.406], std= [0.229, 0.224, 0.225]

Multiple iterations of model design and hyperparameter tuning led to the final model choice (**Number 6**). The following table summarizes the most important results. The hyperparameter set are all same unless specified.

S.No	Name	Training Validation Accuracy	Hyperparameters	Graph (Y-axis: Accuracy, Train , Validation)	Comments
1.	Baseline (2 layer CNN)	64.07 63.14	num_epochs=10; optimizer=SGD with lr=0.001, weight_decay=0.01 , momentum=0.5	Didn't plot	1. Needs some design changes and tuning.
2.	5 layer CNN with BatchNorm2D (CNN-5 + BN)	99.7 77.8	num_epochs=10; optimizer=SGD with lr=0.001, weight_decay=0.01 , momentum=0.9		<ol style="list-style-type: none"> 1. A bigger architecture is able to learn more features of the data. 2. Batch normalization after each layer helps to stabilize the network for faster training. 3. SGD with momentum generalizes decently on the validation set because of its ability to surpass local minima.
3.	(CNN-5 + BN) + Data Augmentation	98.96 47.1	num_epochs=20; optimizer=Adam with lr=0.001, weight_decay=0.01		<ol style="list-style-type: none"> 1. Adam converges on the training set very fast but generalizes poorly on the validation set compared to SGD with momentum. 2. Augmenting the training data by <ol style="list-style-type: none"> a. Resize → (32, 32) b. Rotate and Crop by $\pm 10^\circ$ doesn't result in <i>additional</i> data but rather changes the existing data with a certain probability, thereby decreasing the dataset's variance. However, the augmentations were extreme (50% probability) resulting in worse validation performance for even existing images.
4.	ResNet18 (high learning rate)	97.3 81.2	num_epochs=5; optimizer=SGD with lr=0.001, weight_decay=0.01 , momentum=0.9		<ol style="list-style-type: none"> 1. Pretrained ResNet18 incorporates bigger architecture with BatchNorm which is fine-tuned to fit the small dataset. 2. Training and validation start to diverge after 3 epochs due to a high learning rate.
5.	ResNet18 (low learning rate)	92.1 82.3	num_epochs=10; optimizer=SGD with lr=0.0001, weight_decay=0.01 , momentum=0.9		<ol style="list-style-type: none"> 1. Training and validation have more linear stability before starting to diverge. Convergence is slow, however, generalization is better.

S.No	Name	Training Validation Accuracy	Hyperparameters	Graph (Y-axis: Accuracy, Train , Validation)	Comments
					
6.	ResNet18 (changed optimizer)	98.6 82.4	<pre>num_epochs=5; optimizer=AdamW with lr=0.0005, weight_decay=0.01</pre>		<ol style="list-style-type: none"> AdamW generalizes comparably to SGD with momentum due to the stable regularization, while also resulting in faster training convergence. Model seems to overfit after epoch 2.

2. Consider the following experience, which is often encountered in practice. Alice's model demonstrates remarkable accuracy (>80%) during testing, and yet its navigation performance on a real robot drops significantly. For example, it may completely fail to complete a loop along the corridors in the COM1 building. One noticeable issue is the robot's gradual deviation from its

intended path. Discuss potential reasons for this performance decrease and suggest possible solutions. (maximum 500 words)

Hint. You may google for distributional shift or covariate shift in behavior cloning imitation learning.

A potential reason for Alice's poor model performance on a real robot is mainly because of covariate shift while using an Imitation Learning algorithm (like Behavior Cloning) which refers to learning a model that can make arbitrary mistakes in parts of the state space not covered by the expert dataset. Since the robot greedily imitates demonstrated actions, it can drift away from demonstrated/seen states due to error accumulation. This can lead to the robot never traversing its intended path. These have various causes:

1. **Noisy sensors:** Sensors and cameras used to collect datasets were noisy causing incorrect/erroneous mapping of real-world objects to the simulated objects.
 - a. **Solution:** Calibrate sensors with precision. Add more sensors (in number, and variety) to average out the noise component in their readings. Use high-quality sensors that give better resolution thereby possibly providing a greater number of features..
2. **Out of Distribution (OOD) Data:** The robot encountered some parts of the path which were either not present in the training dataset or were not well covered causing the robot to not know what actions to take in those states with high confidence.
 - a. **Solution:** Incorporate more diverse and thorough data of the environment in the training set. Alternatively, perform reward engineering such that once an unseen state is encountered, the robot is encouraged/rewarded to take an action that helps it to recover back to seen states. We can also reward the robot negatively to reach unseen states. For e.g. SQIL [1].
3. **New Obstacles:** The real-world robot may have encountered new unseen obstacles in the same environment and the model wasn't able to take appropriate actions to avoid them and choose another path.
Solution: Incorporate methods to identify uncertainty (newly added obstacles in this case) and take appropriate actions. For e.g., the agent can do online re-planning, i.e., compute a new action at each timestep by taking the current state and observation as input [2]. Alternatively, we can opt for a model-based RL algorithm that isn't affected by OOD easily [3].

References:

- [1] Reddy, S., Dragan, A. D., & Levine, S. (2019). SQIL: Imitation Learning via Reinforcement Learning with Sparse Rewards. *ArXiv*. /abs/1905.11108
- [2] Wang, T., & Ba, J. (2019). Exploring Model-based Planning with Policy Networks. *ArXiv*. /abs/1906.08649
- [3] Lu, C., Ball, P. J., Osborne, M. A., & Roberts, S. J. (2021). Revisiting Design Choices in Offline Model-Based Reinforcement Learning. *ArXiv*. /abs/2110.04135