Convolutional Neural Network Architecture Seach with Q-Learning

Mike Phulsuksombati [1]

Abstract

We seeks to automate the process of designing the architecture of the convolutional neural network using reinforcemnet learning. The Q-learning agent is trained to sequentially select the CNN layers to achieve maximum accuracy on the validation set. We limited the layers that the agent can select the convolutional layers, the maximum pooling layers, and the softmax layer with pre-defined hyperparameters such as number of filters and kernel size. We experiment the Q-learning agent on three datasets. First, to test that the agent can converge and select the optimal architecture. We run the Q-learning agent on the randomply generated data from the preselected CNN architecture. The agent is tested on the two standard classification datasets, namely ProstateX and CIFAR-10. The agent is able to find the CNN architecture that achieves better test accuracy than the state-of-the-art model on the ProstateX dataset.

1. Introduction

Convolutional neural networks (CNNs) have achieved many success in computer vision related task. The architecture of the convolutional network becomes more complex with more dept and variation as we can observe from the large scale visual recognition challenge (LSVRC) (Russakovsky et al., 2015) over the past years. Although implementing the CNNs become much easier due to many existing library such as Caffe (Jia et al., 2014) or Tensorflow (Abadi et al., 2015), designing a good architecture for a specific problem still require a lot of insights and trials and errors in tuning.

The convolutional architecture mainly consists of the convolutional layers, the pooling layers, the fully-connected layers, and the softmax layer. Each of this layers has specific roles and contribution to the overall network. Moreover, each types of layers contains its own specific hyperparameters. For example, the convolutional layer has number of output filters, kernel size, stride, and type of padding. Therefore, there are many design choice just when to pick a particular layer. The decision for the neural network designer is not only to string these layers together, but also to adjust these hyperparameters in the way that such architecture performs well on the given task. Therefore, there are combinatorially large CNN architectures that one can select for a specific problem.

There is not many intuition of how to design a good CNN architecture. For a new task, human tends to design the CNN architecture based on their experiences of seeing an architecture that works well on other problems. However, such designs are not guarantee to work well for the task. This leads to an idea of using an algorithm to automate the process of designing the CNN architecture without human intervention. We construct a Q-learning agent that learn how to select the types of layers and layer parameters to achieve good result on the validation set. We scope the types of layer in this projec to the convolutional layer, the maximum pooling layer, and the softmax layers where the layer parameters are commonly used in practice.

2. Background/Related Work

Automatic selection of network architecture without human intervention is an important research area in machine learning. The traditional approach is based on using genetic algorithms to artificially evolve the neural network on both architecture and weights such as NeuroEvolution of Augmenting Topologies (NEAT) (Stanley & Miikkulainen, 2002), which achieve a good performance on classical reinforcement learning tasks. The method is also adpated to solve a more complex environment such as the Super Mario World (Baldominos et al., 2015) without looking at the state of the game. However, although the genetic based algorithms are good in discover new model with flexible parameter choices, the limitations are the large number of population and the number of generation required to produce a competitive result. Threfore, it cannot be used for a large scale problem.

Recently, reinforcement learning has been used to speed up and simplify the neural network architecture search. One approach the we followed in this work is based on Q-Learning called MetaQNN (Baker et al., 2016). The algorithm is iteratively learn to select a layer from a finite set of layers and connect them together to produce a competitive neural network architecture in the comparable level to several hand-picked architecture in the standard dataset such as CIFAR-100 and MNIST. However, the method is still limited in term of flexibility since it can only generate the network from a pre-selected pool of layers and the with fixed length. Therefore, it cannot generate a novel architecture in similar fashion to those generated by the genetic algorithm.

To combine the best of both approachs together, a Q-value table is replaced by a recurrent neural network to allow more flexibility in generating the architecture (Zoph & Le, 2016). The recurrent network generates the architecture similar to generating a sequence of tokens. The reccurent network is trained with a policy gradient method to achieve the best validation accuracy. The model generates by this method is able to achieve a better accuracy and less numberf of model parameters than the state-of-the-art model on the CIFAR-10 and Penn Treebank dataset. However, the method requires many more number of model to learn from than the standard Q-learning method and require huge computation resources on large dataset.

3. Approach

In the following setion, we will describe how to use the Q-learning agent to generate the convolutionl architectures. The main objective is to find the convolutional architectures that maximize the validation accuracy. Our approach follows the work of (Baker et al., 2016) with some modification on the transition of the Markov Decision process and the set of layers that the agent can select.

3.1. Markov Decision Process

We model the process of generating the convolutional architectures as the finite horizon Markov Decision Process (MDP). The time horizon represents the maximum number of layers that the agent allows to select. We limit the state space and action space to be finite by limiting the types and hyperparameter configuration of the layer. The layers that the agent can select is shown in Table 1. There is total of 16 layers that the agent can select. All configuration of the layers are adjust such that they can connect to all other layers via padding.

The overview of the architecture selection process is represented in Figure 1. The agent starts with the initial input layer and, at each step, takes the action to select the next layer of the network. The actions are deterministic and always succeed. The process is stopped when the agent select the termination state or reach the maximum number of layers. Note that although the numbers of permitted layers are finite, the search space for the architecture is still large, $\sum_{i=0}^{n} 16^i$ where n is the maximum number of layers.

After the agent successfully select the network architecture,

Type	Parameter	Total
Convolution	number of output filters $\in \{32, 36, 48, 64\}$ kernel size $\in \{1, 3, 5\}$ stride is 1 apply padding follow with ReLU	12
Max Pooling	(kernel size, stride) $\in \{(2,2),(3,2),(5,3)\}$ apply padding	3
Softmax (Termination)	Output class	1

Table 1. Configuration of CNNs layers that agent can select

the network is trained and tested on validation set to get the accuracy. The reward is defined as the validation accuracy. There is no immediate reward for each action, but rather the delayed reward after complete the episode (when the network is selected and trained.) Note that the reward is stochastic since the network is trained with the stochastic gradient descent; however, if the network is resampled, we assume that it has the same validation accuracy for simplicity. Therefore, the reward can be considered to be deterministic in this setting.

3.2. Q-learning

Q-learning is popular model-free reinforcement learning algorithm that has a great success on modern reinforcement learning benchmark such as Atari (Mnih et al., 2013). The main idea is to learn the action-value function Q by iteratively update the estimate of Bellman equation as

$$Q(S_t, A_t) = (1-\alpha)Q(S_t, A_t) + \alpha(R_{t+1} + \gamma \max_a Q(S_{t+1}, a))$$

where α is the Q-learning rate and γ is the discounted factor for reward. The above iterative method is shown to converge to the optimal Q^* with probability 1 (Sutton & Barto, 1998).

The Q-learning approach is suitable for the CNN architecture search problem because, although the search space is limited by finite state and action assumption, our search space is still combinatorially large. Hence, breaking the problem down to smaller subproblem with Q-learning will helps find the high-performing architecture faster. Moreover, our MDP is difficult to learn due to its size and limited amount of episode the agent is allow to observe (due to training time), thus a model free approach is more appropriate than the model-based approach for such problem.

3.3. ϵ-greedy exploration

The exploration is necessay for the Q function to converge. We use the ϵ-greedy exploration in this project. The

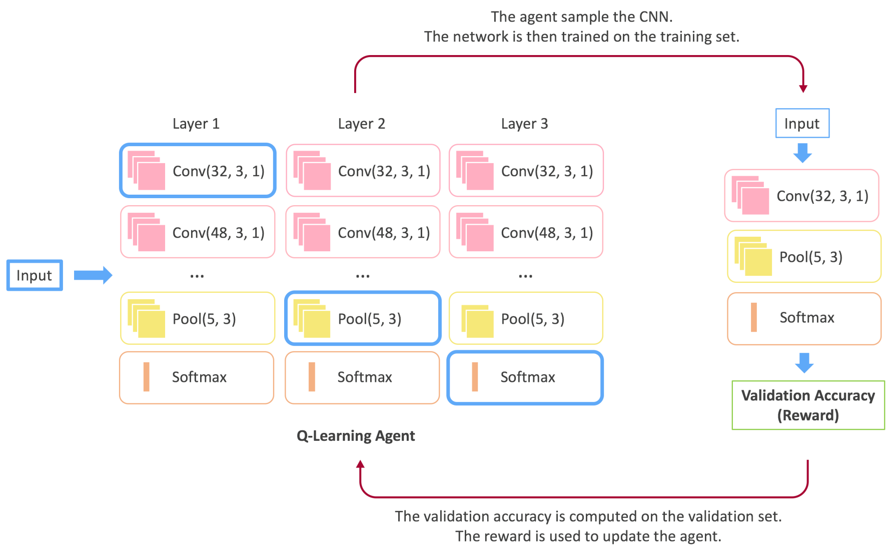

The agent sample the CNN.
The network is then trained on the training set.
Layer 1
Layer 2
Layer 3
Conv(32, 3, 1)
Conv(48, 3, 1)
Pool(5, 3)
Softmax
Input
Q-Learning Agent
Validation Accuracy (Reward)
The validation accuracy is computed on the validation set.
The reward is used to update the agent.

Figure 1. a nice plot

strategy is to take a random action with probability ϵ and the greedy action according to the $\operatorname{argmax}_a Q(S_t, a)$ with probability $1 - \epsilon$. We set the ϵ to decay overtime so that the the agent learn to explore the environment early and gradually exploit the knowledge to achieve the maximum reward. We have tried with both linear and exponentially decay to schedule the ϵ for the project; however, we find that the agent needs very large exploration phase before we can exploit for the reward. That is the agent needs to learn many number of architectures before it can decide which to use. This is due to the delayed reward, which is specific to our problem. We use the ϵ schedule described in (Baker et al., 2016) in Table 2, which we find our agent to converge in all of our experiment.

Table 2. ϵ reduction schedule

ϵ	**1.0**	**0.9 - 0.7**	**0.6 - 0.1**
# Models Trained	1500	100	150

3.4. Experience Replay

Experience replay is an important technique that speed up the converge rate for the agent to learn the environment. The main idea is to store the experience in the replay buffer and the experiences are sampled from this buffer to update the Q-table. The experience replay helps break the similarity of the sequential data that we get from observation, which makes training the agent similar to supervised learning. Once the ϵ is small, the agent will observe the similar data since the action is taken greedily. Therefore, agent will learn less about the environment. The experience replay helps the agent to learn from the data in the previous state, which stabilizes the training process.

4. Experiment Result

We test our approach on three datasets. The first data is randomly generated dataset from a fixed CNN. This is to show that our approach yields an optimal model if it is able to converge to the fixed CNN. The second and third datasets are the ProstateX challenge and CIFAR-10. The aim for these experiments are to test the application of our approach on the real dataset. The randomized dataset and the ProstateX dataset take around 1-2 days to complete and the CIFAR-10 took around 5 day to complete. All models is run on Google Cloud.

4.1. Experiment Setup

The neural networks are constructed using Tensorflow (Abadi et al., 2015) and run with GPU setup. Each model is trained for 10 epochs using the Adam optimizer (Kingma & Ba, 2014) with $\beta_1 = 0.9$, $\beta_2 = 0.999$, and $\epsilon = 10^{-8}$. We set the batch size to 128 and fixed learning rate at 10^{-4}. All weigth is initialized with Xavier initalization (Glorot & Bengio). We limit the maximum number of layers that the agent can pick to 4 layers. Note that the number of possible architecture in this setup is $\sum_{i=0}^{4} 16^i = 69,905$ which is fairly large to perform brute force search. If the model is resampled, the model will not be trained and is assigned the previous validation accuracy as the reward. This is to speed up the training process.

For the Q-learning setup, we set Q-learning rate $\alpha = 0.1$ and the discount factor $\gamma = 1$. This is to not prioritize short term reward from each selection of layer, but rather a contribution of the whole architecture. At the end of every episode, we sample 100 experiences from the experience buffer to perform an experience replay and update Q-table. For the exploration strategy, we decay the ϵ according to the Table 2.

To simplified our notation for architecture, we represent the convolutional layers as Conv(number of filters, kernel size, stride) and the max pooling layer as Pool(kernel size, stride), and the sofmax layer as Softmax(number of classes).

4.2. Randomly Generate Dataset

The goal of this experiment is to show that the proposed Q-learning agent will eventually select the optimal network architecture. We first construct the optimal network N^* exclusively from the layers in Table 1 to generate the dataset. We draw a random input image X_i where each entries is sample from the standard gaussian distribution. Then, we feed-forward X_i to get the output and add a Gaussian noise with variance 0.1 to the output and obtain the final y_i. The Q-leaning agent is trained to select the best architecture N_A based on the randomly generated $\{X_i, y_i\}$ dataset.

We created 5,100 random image of size $32 \times 32 \times 3$ and feeded forward through a fixed neural network N^* with architecture Conv(64, 3, 1), Conv(32, 3, 1), Pool(3, 2) and Softmax(4) to get the labels of 4 classes. Then, the data is splitted in to 5,000 training set and 100 validation set. We trained in total of $2,700$ models with the ϵ schedule in Table XXX. The performance for the randomly generated data is shown in Figure 2.

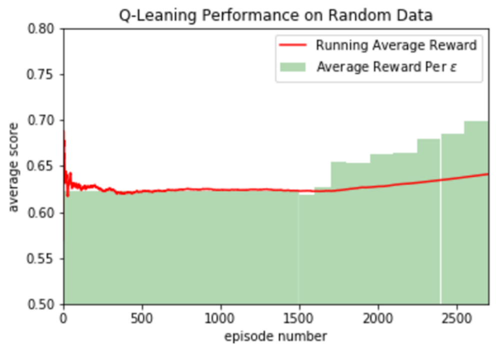


Figure 2. Q-Learning performance on randomly generated data. The red line shows a running average reward and the green bar show the average reward during an ϵ period.

We can see that as ϵ decrease, the running average reward increases because the agent learns to select the CNN architectures with better validation accuracy. The increasing is small because the full exploration period, $\epsilon = 1.0$, is long $(1,500$ models) in comparison to the exploitation periods $(1,200$ models). However, from the green bars, we can see that the average reward per ϵ phase is constantly increasing, which suggests that the training is stable.

Network Architecture	Validation Accuracy
Original model: Conv(64, 3, 1), Conv(32, 3, 1), Pool(3, 2), Softmax(4)	**72%**
Pool(2, 2), Conv(48, 3, 1), Conv(36, 3, 1), Softmax(4)	73%
Pool(2, 2), Conv(48, 3, 1), Conv(32, 3, 1), Softmax(4)	73%
Conv(36, 4, 1), Pool(2, 2), Conv(36, 3, 1), Softmax(4)	73%

Table 3. The top 3 architectures found by Q-learning agent for the randomly generated data.

Now, we observe the top 3 models sampled from the Q-learning agent in Table 4.2. Note that the top model is not necessary the model that the agent found in the last ϵ phase, but overall phases. The high-performing models are more likely to be found toward the end of ϵ schedule.

From Table 4.2, we observe that the agent is able to find the original architecture N^*; however, the top 3 models are not such architecture. We observe that the validation accuracy for the top 3 models are higher than that of the original model. The agent is able to find an original model; however, it cannot converge to the model because of the validation accuracy of the original model is lower than that of the top models. We suspect that this is because the reward is stochastic due to the stochasticity in training the model, specifically from stochastic gradient descent.

Moreover, we suspect that another cause that the agent does not converge may be from that we added too much Gaussian noise when we generate the data. Hence, we generate the new data set by reducing the variance of the noise to 0.01. We observe that now majority of the models achieve the validation accuracy of 1. This means that the model is able to fit the data perfectly. Hence, Q-learning is definitely not converge to a single model. However, we belive that with right amount of noise the Q-learning agent should converge to the original architecture.

4.3. ProstateX

Now, we apply the approach on the real data set to see whether it can discover the CNN architectures that are on the same par with the hand-picked architecture. We obtained the data set from the SPIE-AAPM-NCI Prostate MR Classification Challenge (ProstateX), which is preprocessed by the Radiology Department, Stanford University School of Medicine. The data consists of multiparametric MRI exams of 197 patients: 134 patients with non-aggressive prostate cancer and 59 patients with agressive prostate cancer. The task is to classify the 3 channels MR images of prostate cancer into two classes: agressive and non-aggressive prostate cancer. The sample data is shown in Figure 3. We cropped the images to size 50 x 50 pixels, with the center being the point annotated by the radiologist.

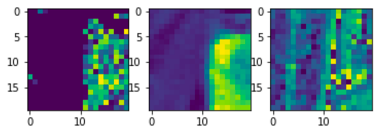


Figure 3. Sample MR images from the ProstateX

We separate the data into 157 samples for training set and 40 samples into validation set. To increase the size of training data, the MR images are augmented by rotation, reflection, scaling and adding Gaussian noise, which yield the total of around 2500 samples.

The state-of-the-art result for machine learning algorithm on the ProstateX data set is 73% accuracy using Elastic Net (Banerjee et al., 2016). The official test set is available online at `http://spiechallenges.cloudapp.net/`. Our goal to use Q-learning agent to search for a CNN architecture that has higher accuracy on the test data than the state-of-the-art benchmark. The performance of the Q-learning algorithm of the ProstateX dataset is shown in Figure 4.

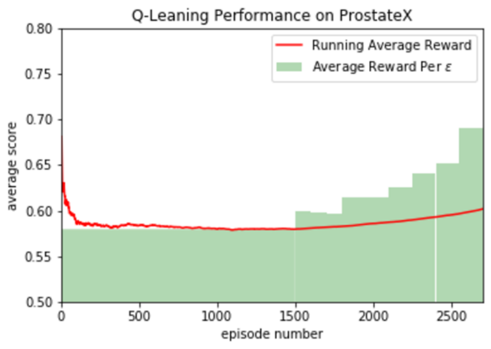


Figure 4. Q-Learning performance on ProstateX.

We can see that as ϵ decrease, the running average reward slightly increases, which suggests that our training is stable and agent learn to select the high-performing architectures. Since there is no published benchmark on the problem using CNNs, we have tried to hand-picked several architectures for the problem and the best we found achieve the accuracy of around $70 - 71\%$ on the test set. From Figure 4, we can see that the average performance when $\epsilon = 0.1$ is around 70% which suggests that the agent is able to pickthe architecture at the same level as human.

The top 3 models found by our Q-learning agent are presented in the Table 4.3. We can see that the top model found, which has the best validation accuracy, is able to achive better result than the current state-of-the-art model on the test set.

Network Architecture	Accuracy (val, test)
Conv(64, 5, 1), Conv(32, 3, 1), Pool(3, 2), Softmax(2)	(83%, 76%)
Conv(36, 4, 1), Conv(32, 5, 1), Conv(32, 3, 1), Softmax(2)	(80%, 73%)
Conv(32, 3, 1), Conv(36, 3, 1), Pool(3, 2) Softmax(4)	(80%, 71%)

Table 4. The top 3 architectures found by Q-learning agent for the ProstateX.

This top models found here are rather interesting that they incorporate the Pool(3,2), which is not as commonly use as Pool(2, 2). Moreover in the thrid network there is an upsampling from 32 filters to 36 filters, which is also not as common as the downsampling. This shows the effectiveness of the Q-learning method as there is no human bias in selecting the architecture. The algorithm simply learns from the experience and selecting the layers based only its experience with the samples with no prior bias.

4.4. CIFAR-10

We experiment on the standard CIFAR-10 image classification problem. The task is to classify color images of size 32×32 into 10 classes. The state-of-the-art model using fractional max-pooling (Graham, 2014) achieved an accuracy of 96.53% on the official test data set. However, there is no official benchmark for the network with limited depth to 5 layers. Therefore, we only show the accuracy that our top three model from our Q-learning agent with no benchmark result.

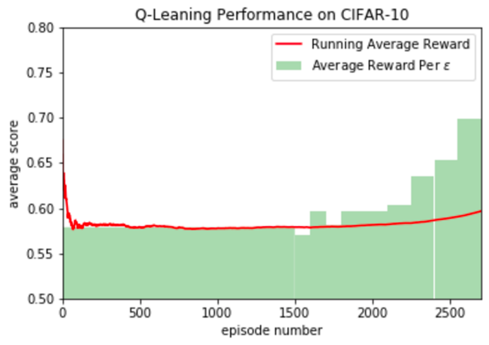


Figure 5. Q-Learning performance on CIFAR-10.

We sample the CIFAR-10 dataset into $10,000$ training data, $1,000$ validation data, and $1,000$ training data. We can see from 5 that Q-learning training is stable since the running average reward is constantly increase. However, we observe that for the $\epsilon = (0.9, 0.2)$ the average reward per ϵ is slightly increase from when the agent always explore. This shows that our problem requires the agent to sample a lot of model before it can learn and exploit for the reward.

Network Architecture	Accuracy (val, test)
Conv(48, 5, 1), Conv(32, 3, 1), Conv(36, 3, 1), Softmax(10)	(78%, 73%)
Conv(32, 3, 1), Conv(36, 3, 1), Conv(48, 5, 1), Pool(2, 2), Softmax(10)	(75%, 77%)
Conv(32, 3, 1), Conv(36, 3, 1), Conv(32, 5, 1), Conv(48, 5, 1), Softmax(10)	(75%, 70%)

Table 5. The top 3 architectures found by Q-learning agent for the CIFAR-10.

The top three models trained so far is in Table 4.4. As expected, we did not see very high accuracy from the Q-learning agent. However, the best model achive the test accuracy of 77%, which is on par with the best desgin shallow network from (McDonnell & Vladusich, 2015). We believe that this is due to the limited representation of the shallow network.

We also observe that from the top three models the agent always pick the Conv(32, 3, 1) following with Conv(36, 3, 1), which is a upsampling. This is similar to what we observe in the ProstateX dataset. There is no intuition on why this help the CNN to achieve good result; however, the same phenomenon is also observed in the RNN agent trained with policy gradient (Zoph & Le, 2016). This also second the advantages of using the reinforcement learning to find the best architectures since there is no biased in layer selection.

5. Conclusion

In this project, we show the effectiveness of using Q-learning in discovering the novel CNN architectures that on par with the human picked approach. Although the agent is not able to find the optimal architecture in the randomly generated dataset, the agent is able to find the CNN architecture that rivals human performance in the ProstateX and CIFAR-10 dataset.

Although the approach is not as flexible in terms of the architectures as the RNN agent, the Q-learning is simple and able to find a good architecture using less computational resources. However, it still need ample amount of time to search for the architectures because the approach requires a lot of sample architecture until the Q-learning agent is able to learn and pick the network with high validation accuracy.

The benefit of using reinforcement learning to select the CNN architecture is that there is no bias involve in the selection process. The process of selection the CNN layers to achive good performance is not intuitive for human understanding and we tend to learn from example of the successful networks. That is, we already have prior of what combination of layers we should connect together to get good accuracy. However, this is not the case for every problem. Good architecture is really specific to the problem. For example VGG-16 which performs very well on the CIFAR-10 (Simonyan & Zisserman, 2014) can not achieve very low accuracy on the ProstateX.Overall, automating the CNN architecture search should help save time in designing and tuning the CNNs to achieve good result for the particular dataset.

One question for future work is to figure out how to get the Q-learning to converge to the fixed network in the randomized dataset. We believe that, by adding the right amount of noise to the randomly generated dataset, the Q-learning agent should converge. Moreover, due to the slow training time of Q-learning agent, we believe that the process can be implemented in the parallel to speed up the training time. We believe that the RNN architecture should provide the flexibility to the architecture search process. Our future work is to combine the advantages of both methods to get the speed of the Q-learning agent and better accuracy from the flexibility of the RNN agent.

References

Abadi, Martín, Agarwal, Ashish, Barham, Paul, Brevdo, Eugene, Chen, Zhifeng, Citro, Craig, Corrado, Greg S., Davis, Andy, Dean, Jeffrey, Devin, Matthieu, Ghemawat, Sanjay, Goodfellow, Ian, Harp, Andrew, Irving, Geoffrey, Isard, Michael, Jia, Yangqing, Jozefowicz, Rafal, Kaiser, Lukasz, Kudlur, Manjunath, Levenberg, Josh, Mané, Dan, Monga, Rajat, Moore, Sherry, Murray, Derek, Olah, Chris, Schuster, Mike, Shlens, Jonathon, Steiner, Benoit, Sutskever, Ilya, Talwar, Kunal, Tucker, Paul, Vanhoucke, Vincent, Vasudevan, Vijay, Viégas, Fernanda, Vinyals, Oriol, Warden, Pete, Wattenberg, Martin, Wicke, Martin, Yu, Yuan, and Zheng, Xiaoqiang. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. URL `http://tensorflow.org/`. Software available from tensorflow.org.

Baker, Bowen, Gupta, Otkrist, Naik, Nikhil, and Raskar, Ramesh. Designing neural network architectures using reinforcement learning. *arXiv preprint arXiv:1611.02167*, 2016.

Baldominos, Alejandro, Saez, Yago, Recio, Gustavo, and Calle, Javier. *Learning Levels of Mario AI Using Genetic Algorithms*, pp. 267–277. Springer International Publishing, Cham, 2015. ISBN 978-3-319-24598-0. doi:

10.1007/978-3-319-24598-0_24. URL http://dx.doi.org/10.1007/978-3-319-24598-0_24.

Banerjee, Imon, Hahn, Lewis, Sonn, Geoffrey, Fan, Richard, and Rubin, Daniel L. Computerized multiparametric mr image analysis for prostate cancer aggressiveness-assessment. *arXiv preprint arXiv:1612.00408*, 2016.

Glorot, Xavier and Bengio, Yoshua. Understanding the difficulty of training deep feedforward neural networks.

Graham, Benjamin. Fractional max-pooling. *arXiv preprint arXiv:1412.6071*, 2014.

Jia, Yangqing, Shelhamer, Evan, Donahue, Jeff, Karayev, Sergey, Long, Jonathan, Girshick, Ross, Guadarrama, Sergio, and Darrell, Trevor. Caffe: Convolutional architecture for fast feature embedding. In *Proceedings of the 22nd ACM international conference on Multimedia*, pp. 675–678. ACM, 2014.

Kingma, Diederik and Ba, Jimmy. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

McDonnell, Mark D and Vladusich, Tony. Enhanced image classification with a fast-learning shallow convolutional neural network. In *Neural Networks (IJCNN), 2015 International Joint Conference on*, pp. 1–7. IEEE, 2015.

Mnih, Volodymyr, Kavukcuoglu, Koray, Silver, David, Graves, Alex, Antonoglou, Ioannis, Wierstra, Daan, and Riedmiller, Martin. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

Russakovsky, Olga, Deng, Jia, Su, Hao, Krause, Jonathan, Satheesh, Sanjeev, Ma, Sean, Huang, Zhiheng, Karpathy, Andrej, Khosla, Aditya, Bernstein, Michael, et al. Imagenet large scale visual recognition challenge. *International Journal of Computer Vision*, 115(3):211–252, 2015.

Simonyan, Karen and Zisserman, Andrew. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

Stanley, Kenneth O and Miikkulainen, Risto. Evolving neural networks through augmenting topologies. *Evolutionary computation*, 10(2):99–127, 2002.

Sutton, Richard S and Barto, Andrew G. *Reinforcement learning: An introduction*, volume 1. MIT press Cambridge, 1998.

Zoph, Barret and Le, Quoc V. Neural architecture search with reinforcement learning. *arXiv preprint arXiv:1611.01578*, 2016.

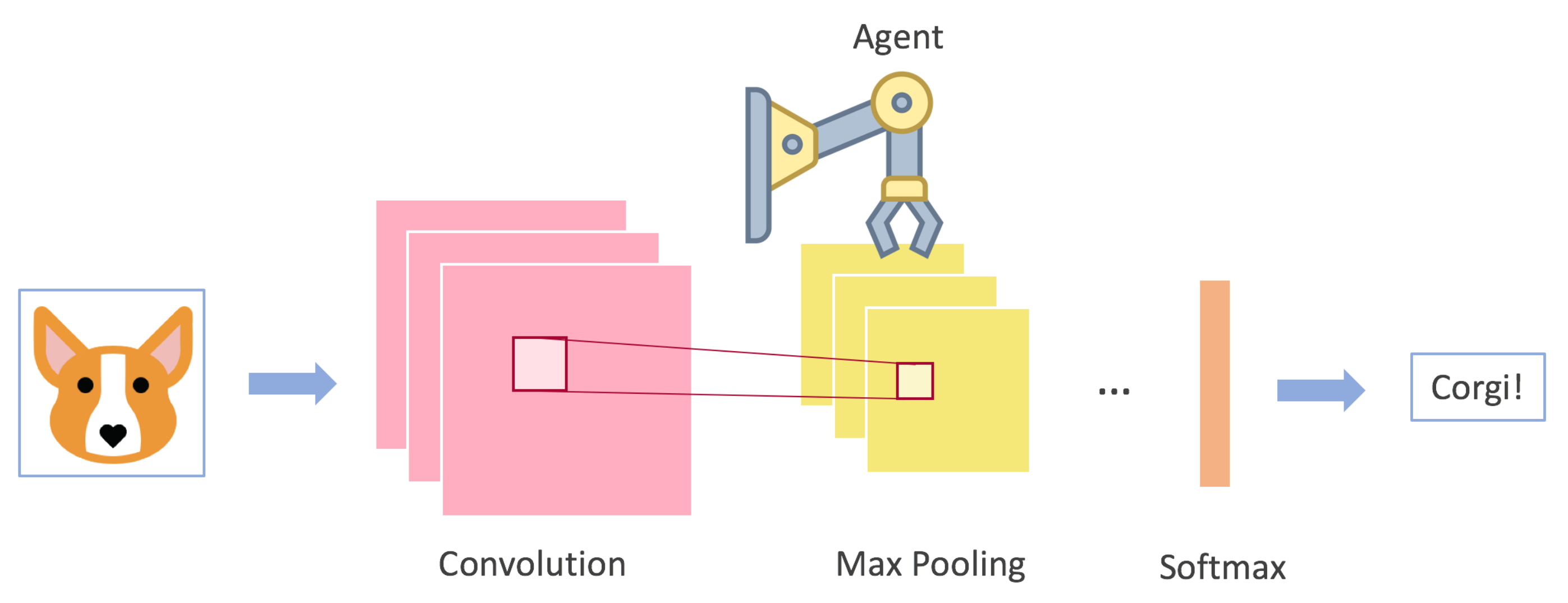
ICME Stanford | Institute for Computational & Mathematical Engineering

Convolutional Neural Network Architecture Design with Q-Learning

Mike Phulsuksombati

Background

- Convolutional neural networks (CNNs) have achieved many success in computer vision related task.
- Although implementing the CNNs becomes easier due to library like Tensorflow or PyTorch, designing architectures for CNNs still requires a lot of experience and many trials and errors to get a good architecture for a particular machine learning task.
- There are many design choices to consider such as layer types and layer parameters. Even if we limit the choices to be finite, the search space is still to large.
- We seek to **automate the process of designing the CNN architecture with Q-Learning**. We construct an agent that learn to select the types of layer and layer parameter to achieve high validation accuracy of a specific machine learning task.



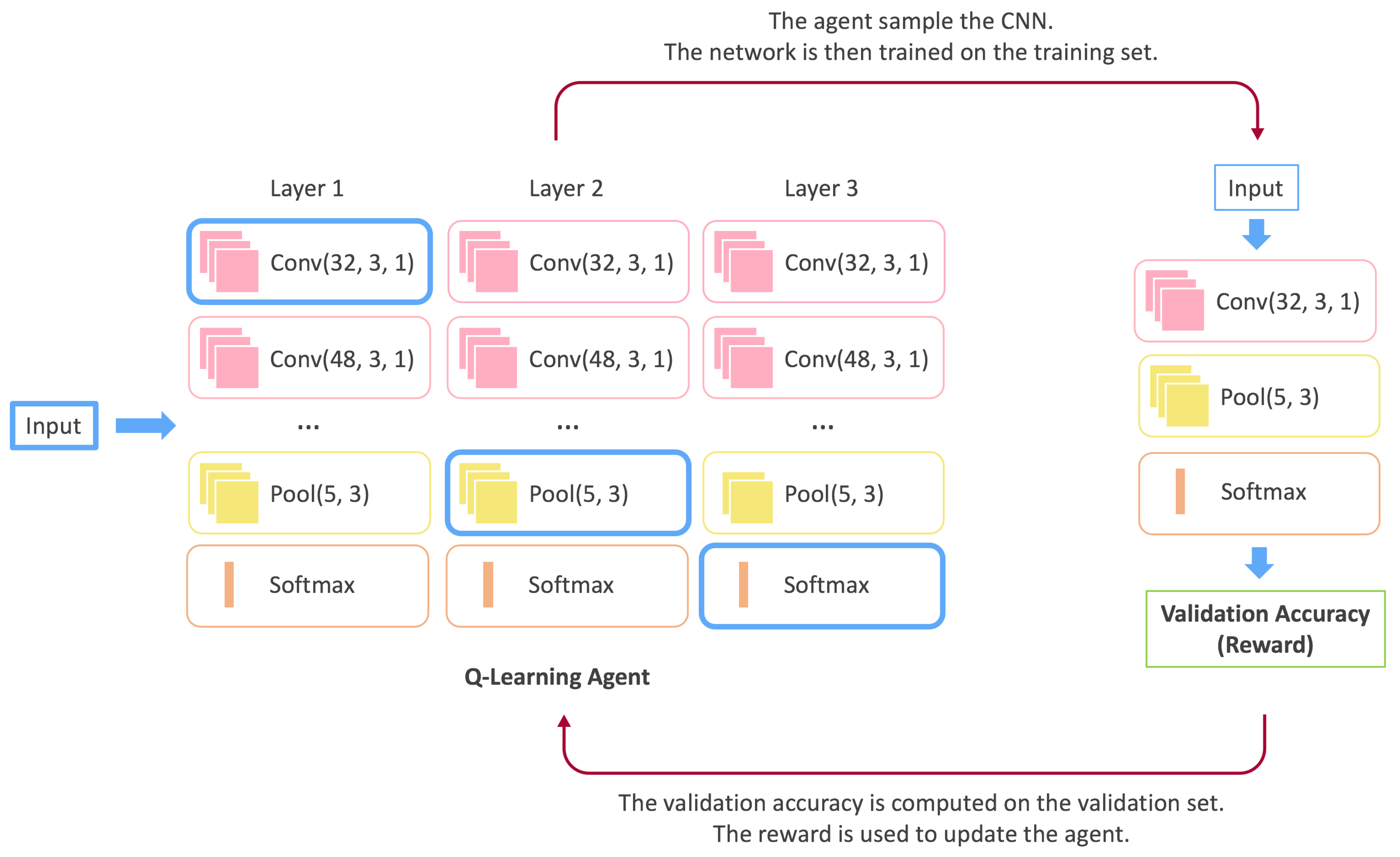
Problem Statement

- The process of designing the CNN architectures is modeled as the **finite horizon Markov Decision Process (MDP)**.
- By limiting the types and layer parameters, **the state and action space for such MDP are finite**.
- The layers that the agent allows to select are

Type	Parameter		Total
Convolution + ReLU	Number of output filter	32, 36, 48, 64	12
	Kernel size	3, 4, 5	
	Stride	1	
Max Pooling	(kernel size, stride)	(2, 2), (3, 2), (5, 3)	3
Softmax (Termination)	Output classes (depend on the task)		1
	Total		**16**

- The **actions are deterministic** and always succeed.
- The **reward is validation accuracy**, which is stochastic.
- The process terminates once the agent selects the softmax layer.
- The agent can select up to 4 layers until it is forced to terminate. A softmax layer is then added to the architecture.

Markov Decision Process


The agent sample the CNN.
The network is then trained on the training set.
Layer 1
Layer 2
Layer 3
Input
Conv(32, 3, 1)
Conv(48, 3, 1)
...
Pool(5, 3)
Softmax
Q-Learning Agent
Input
Conv(32, 3, 1)
Pool(5, 3)
Softmax
Validation Accuracy (Reward)
The validation accuracy is computed on the validation set.
The reward is used to update the agent.

Q-Learning

- It is **difficult to learn the model** because, although the transition of our MDP is deterministic, the reward is stochastic.
- Hence, we consider using a **model-free reinforcement learning algorithm**, namely Q-learning.
- The main idea is to learn the action-value function Q by iteratively update the Bellman equation using the experience learned from the environment as

$$Q(S_t, A_t) = (1-\alpha)Q(S_t, A_t) + \alpha(R_{t+1} + \gamma \max_a Q(S_{t+1}, a))$$

where α is the Q-learning rate and γ is the discounted factor for reward.

- The **experience replay** is used to speed up the convergence rate of Q-learning. Each episode (a CNN is trained), the agent sample 100 experiences from the replay buffer to update the Q function.

ϵ- Greedy Exploration

ϵ	1.0	0.9	0.8	0.7	0.6	0.5	0.4	0.3	0.2	0.1	Total
# of model trained	1,500	100	100	100	150	150	150	150	150	150	2,700

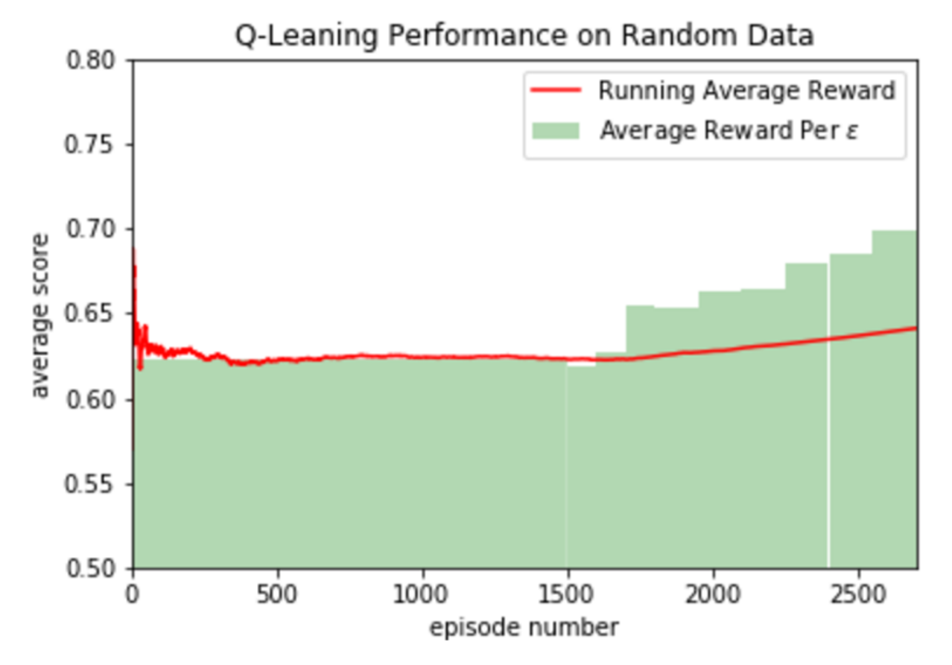
- The exploration is necessary for the Q function to converge.
- Take action according to $argmax_a Q(S_t, a)$ with probability $1-\epsilon$ and take a random action with probability ϵ.
- We use a step function decay in the table above.

Experiment Result

Randomized dataset

- We randomly generate the data set with a particular CNN and observe whether the Q-learning agent is able to recover the true CNN architecture.
- The data consists of 5,000 training and 100 validation of 32 x 32 x 3 images randomly sampled from Gaussian distribution with 4 classes.

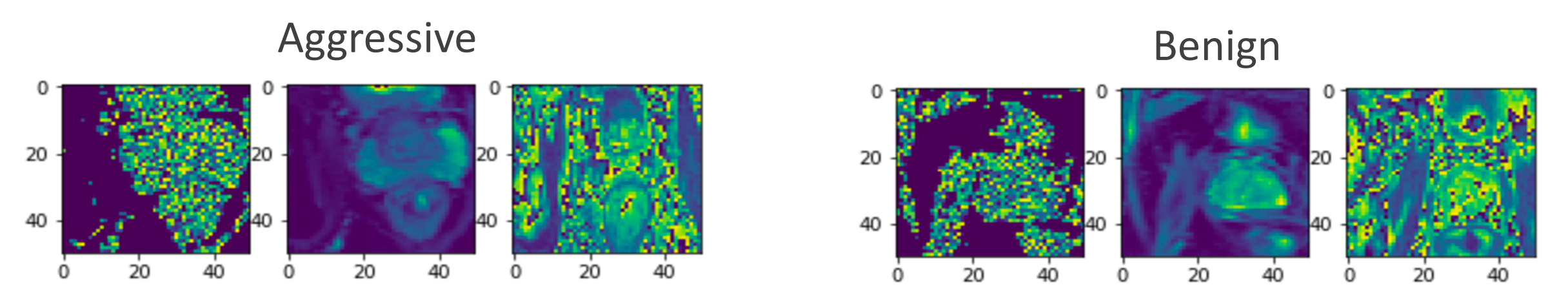
Model	Validation Accuracy
CONV(64, 3, 1), CONV(32, 3, 1), POOL(3, 2), Softmax [Original Network]	**72 %**
POOL(2, 2), CONV(48, 3, 1), COV(36, 3, 1), Softmax	73 %
POOL(2, 2), CONV(48, 3, 1), COV(32, 3, 1), Softmax	73 %
CONV(36, 4, 1), POOL(2, 2) COV(36, 3, 1), Softmax	73 %



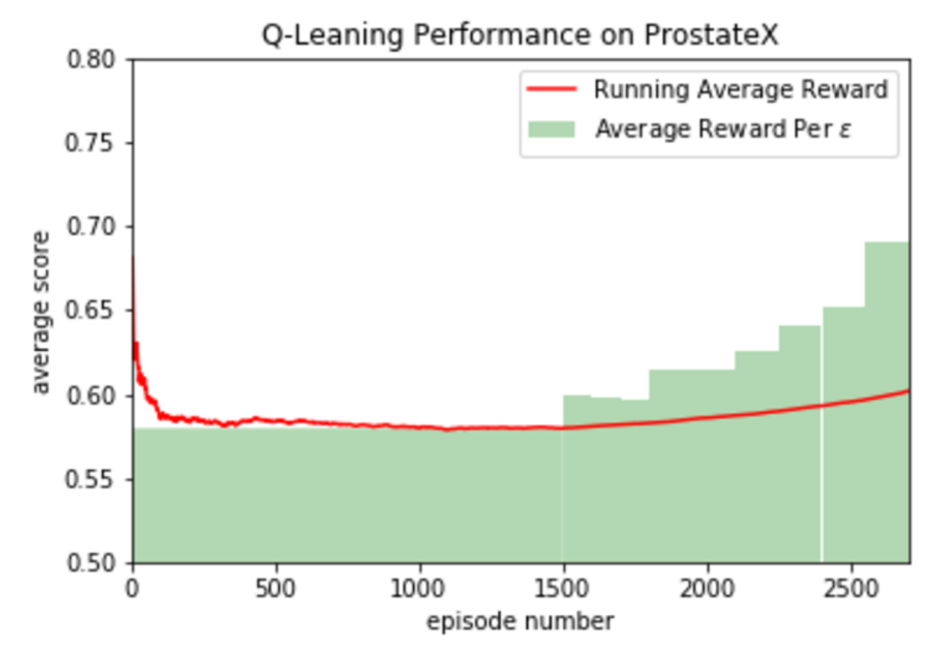
- The Q-Learning agent is not able to recover the original network. This is due to the stochasticity of the validation accuracy and the noise we added to the data. (The reward is lower than those from top CNNs.)

ProstateX

- The data set consists of 197 patients MR images obtained from the Radiology Department, Stanford Medical School. The data is augmented to increase the sample size.
- We separate the data into 157 training (~2,500 after augmented) and 40 validation of 50 x 50 x 3 MRI. The CNN needs to classify the MRI into two classes: aggressive and benign.



Model	Validation Accuracy
CONV(64, 5, 1), COV(32, 3, 1), POOL(3,2), Softmax	82.5 %
CONV(36, 4, 1), COV(32, 5, 1), COV(32, 3, 1), Softmax	80 %
CONV(32, 3, 1), CONV(36, 3, 1), POOL(3, 2), CONV(36, 5, 1), Softmax	80 %



- The architectures found exceed the state-of-the-art performance of 73% (ElasticNet) on the same dataset.

Next Step

- Show that the Q-learning agent is able to converge to the original network.
- Try the method on the larger standard image classification dataset like CIFAR-100 or MNIST.