Imperial College
London

IMPERIAL COLLEGE LONDON

DEPARTMENT OF COMPUTING

Q-Search: Designing Neural Network Architectures Using Reinforcement Learning

Author:
Nattapat Juthaprachakul
Supervisor:
Prof. Wayne Luk

Submitted in partial fulfillment of the requirements for the MSc degree in M.Sc.Computing Science of Imperial College London

September 2018

Abstract

The project introduces Q-Search which is an algorithm based on Reinforcement Learning algorithm namely Q-Learning together with an epilon-greedy exploration strategy and experience replay technique to automatically design the convolutional neural network(CNN) architecture without human supervision. The Q-learning agent is trained to sequentially choose the CNN layers that later are formed into a model with the goal of maximizing the validation accuracy. The maximum number of layer and the type of layers that the agent can choose from are restricted to the convolutional layer, the maximum pooling layer and the softmax layer with pre-defined sets of parameters such as number of filter, size of kernel and number of stride. The project trains Q-Search models on two real datasets: CIFAR-10 and MNIST and compares results with random search and layerwise search. The project runs several experiments on Q-Search with same basic parameter settings and various different setups such as number of episode, number of model sampled from experience replay to update Q-table and how the experience replay is being used. On CIFAR-10 dataset, Q-Search is able to find well-performing topologies that beat the existing shallow network on image classification benchmarks, as well as beating architectures discovered by random and layerwise search.

Acknowledgments

I am grateful to my supervisor, Wayne Luk and Dr. Ce Guo for introducing me to this research and helping me throughout my work.

Contents

Chapter 1

Introduction

Over the past few years, deep convolutional networks(CNNs) have gained prominence and success on a variety of fields and major applications in computer vision domains such as face recognition (Beumer et al. (2006)), image recognition and classification (Chellappa et al. (1998); Krizhevsky et al. (2012)), document analysis (Oliveira and Viana (2017)) and etc. and in natural language processing domains such as speech recognition (Yu (2012)), text classification (Krendzelak and Jakab (2015)) and many more.

The success of deep learning not only contributes to feature designing but also architecture designing as it can be seen that the newly invented architectures are used to overcome challenging tasks in many areas which are hard to achieve in the past such as AlexNet (Krizhevsky et al. (2012)), VGG16/VGG19 (Simonyan and Zisserman (2014)), ResNet (He et al. (2015)), Inception V3 (Szegedy et al. (2015)), GoogleNet (Szegedy et al. (2014)) and Xception (Chollet (2016)).

The main components of typical CNN architecture are a different number of convolutional, pooling and fully-connected layers; and a softmax layer. Each of these layers serves different roles but works together in the overall network. Each layer has its own specific hyperparameters which play different roles that could affect how each layer works. For example, the convolutional layer has kernel size, number of filter, stride size and type of padding. The pooling layer has kernel size and stride size. Additionally, the CNN network can be constructed in any different number of layers and different ordering in overall network. The underlying architecture and hyperparameters are usually selected and adjusted under human supervision to perform well on different specific tasks.

However, overtime the architecture of convolutional networks becomes more complex with more depth and variation in parameter and underlying architecture. Thus, the number of possible choices that one can select makes the design space of CNN architectures combinatorially large and also each architecture may only work for specific problems; hence, it has become daunting tasks for hand-picking technique

or simple search technique like an exhaustive manual search.

One problem is that there is not much intuition about how to design a good CNN architecture. Although, thank to several existing supporting libraries such as PyTorch (Paszke et al. (2017)), Caffe (Jia et al. (2014)) and Tensorflow (Abadi et al. (2015)), it has become easier to design CNN architecture, designing well-performed architectures still requires a lot of expert knowledge and insights and sometimes needs a long period of trails and errors. In addition, finding well-performed architecture requires enough ample time.

Another problem is that though the several well-performed architectures can be directly imported from the above mentioned libraries, the idea of using one model to a new given task based on the past experiences might not work as expected since the CNN architecture that works well in one problem may not suit to others due to various limitations described above. This kind of problems lead to the idea of implementing automation process in designing neural architectures using specific algorithm without human oversight. However, several researches in the area of automating architecture design so far (Saxe et al. (2011); Bergstra and Bengio (2012); Domhan et al. (2015)) have still primarily relied on theoretical insights or intuition gained from experimentation which is still based on human knowledge.

In this paper, the process of automatically designing the convolutional architecture is called Q-Search which is based on Q-learning, one of reinforcement learning algorithm. The goal of Q-learning agent is to discover the CNN architectures that perform well on a given machine learning task with no human intervention. The task of Q-learning agent is to sequentially select the different types of limited numbers of CNN layers based on some pre-defined criteria. This paper focuses on the use of Q-learning on limited number of layer which is 4 layer(state space) and limited type of layer(action space) which is 16 types. The scope of layer type in this project is the convolutional layer, the maximum pooling layer and the softmax layer. Even though the agent is left with finite space to search from and finite actions to choose from, the overall search space is still combinatorially large. The episode ends when the agent reaches maximum number of given layer or softmax layer(termination state). The overall training process ends when the agent reaches the end of all episodes. At the end of each episode, the agent is given a reward for selecting an architecture in form of validation accuracy to update Q-table. The Q-table is used as a lookup table for Q-learning agent to decide which actions should it take. The episodes are divided into many different periods according to given value of ε. At the beginning of episode, the agent learns to explore through random exploration and as episode goes on the agent will begin to exploit its finding to select higher performing models using the ε greedy strategy (Mnih et al. (2015)). The experience replay sampling technique is used to speed up the learning process by helping the overall training to converge faster. The experiments in this paper are conducted on two standard image classification datasets: CIFAR-10 and MNIST by using different setups such as differ-

ent number of episodes and different strategies to update Q-table from experience replay techniques (ji Lin (1992)). Finally, the other hyperparameters besides layer type are in the same setups.(Figure 1.1)

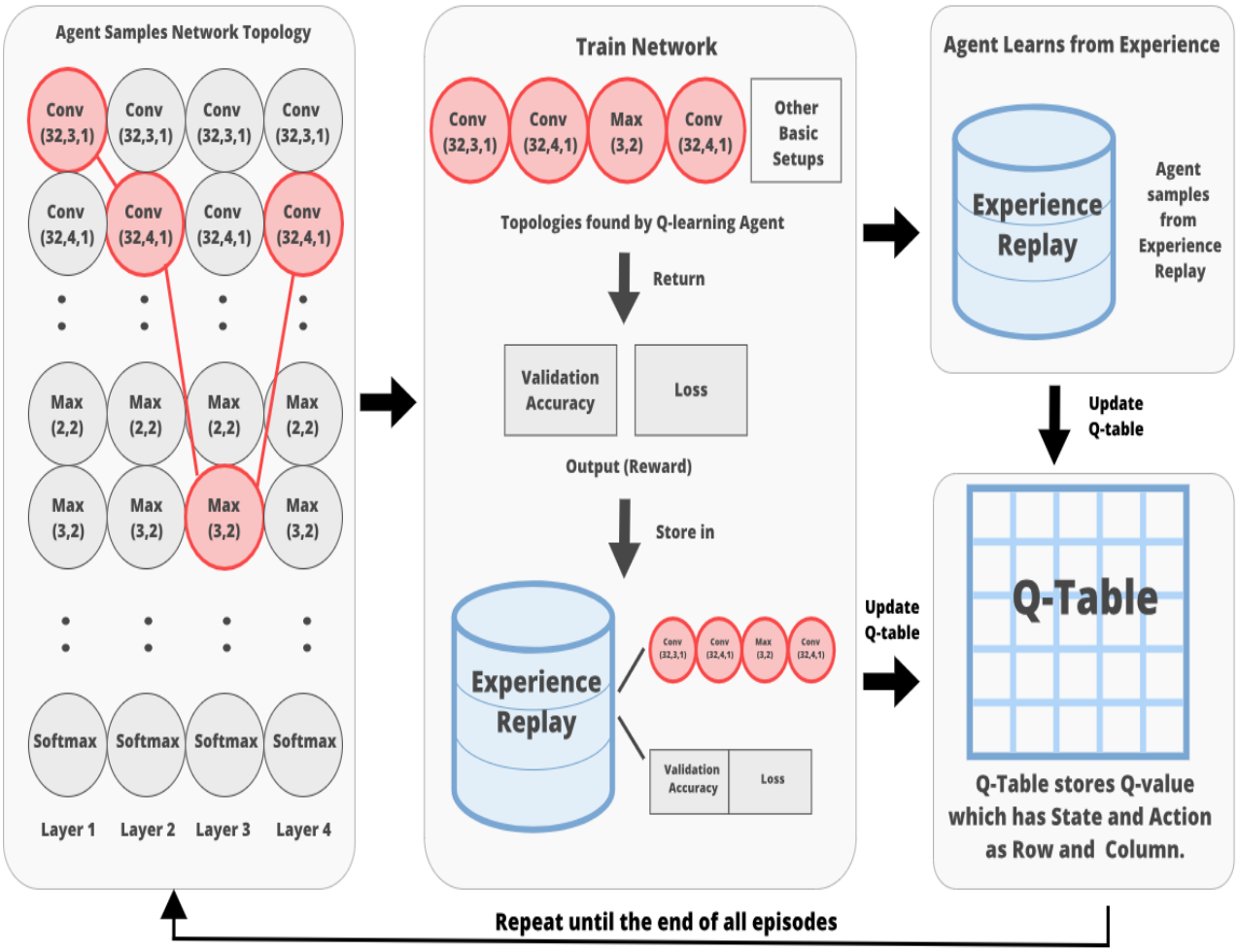

Figure 1.1: Designing CNN Architectures with Q-Search. Left box: the agent starts by selecting the CNN architectures based on pre-defined criteria. Center box: the network is trained and the agent is given the reward as validation accuracy. The topologies and reward are store in experience replay. Right box: the Q-table can be updated directly after network is trained or can be updated by sampling specified number of models from experience replay.

The result of experiments shows that even if the Q-learning agent can find well-performing architectures (97.67% validation accuracy) in the MNIST dataset, random search can outperform with validation accuracy of 98.13%. Nevertheless, there

are no benchmarks for shallow networks to compare the topologies the agent has discovered with. For CIFAR-10 dataset, in every experiment the agent can surely find the well-performing architectures (79.12%, 80%, 79.41%, 80.09%, 80.17%) that can outperformed validation accuracy benchmark of 77% for the shallow network (McDonnell and Vladusich (2015)) found by handpicking architecture. Although random search can discover well-performed model with 80.18% validation accuracy, there is no guarantee that it will work in other cases as shown in the distribution in figure1.2 since in this paper the random search is performed only on 2000 models. In addition, the models discovered by random search method that could compete with the benchmark are fewer than 5 models with mean and median of all models clustering around 72% validation accuracy. The reason that Q-Search does not work much better than random search in MNIST is that, from the observation, the MNIST is a very simple dataset, meaning that any kind of simple architecture can achieve validation accuracy more than 90% as seen in layerwise search while for more complex dataset like CIFAR-10 Q-learning agent performs much better than random search and layerwise search in term of certainty of discovering well-performing architectures. (Figure 1.2)

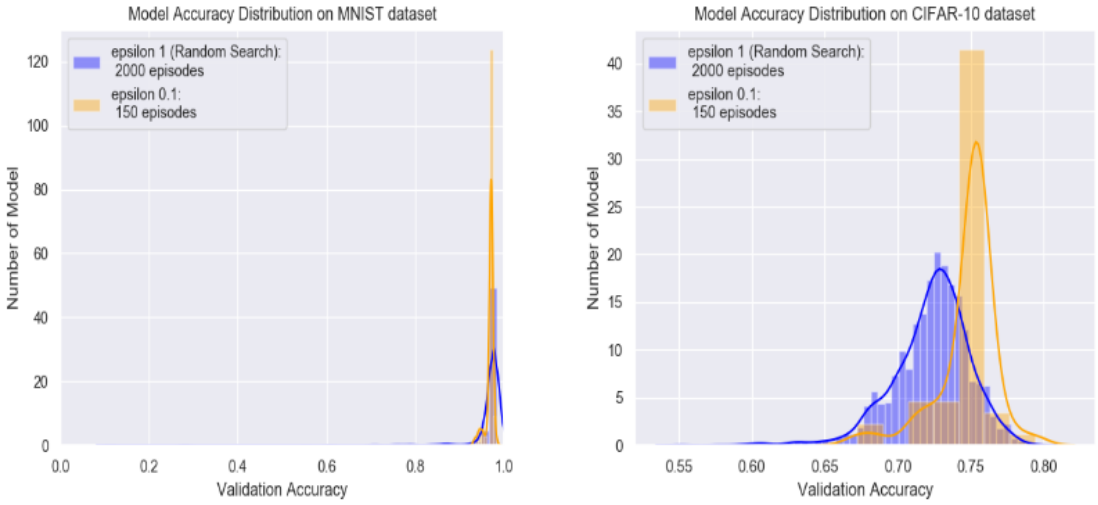

Figure 1.2: Distribution of Validation Accuracy of models found by Random Search and Q-Search on MNIST and CIFAR-10 dataset

Chapter 2

Background

This chapter aims at providing knowledge for main chapters of this paper. This chapter first reviews the essential background of Reinforcement Learning (Section 2.1). Then, this chapter presents related background in using Reinforcement learning to search for Artificial Neural Network Architecture(ANNs) (Section 2.2).

2.1 Reinforcement Learning

Reinforcement Learning(RL) is learning what to do –how to map situations to actions– by interacting with an environment in order to maximize a numerical reward signal which encodes the success of outcome of that action. The learner is not explicitly told which actions to take, but instead must discover which actions yield the most reward by trying them on basis of its past experiences (*exploitation*) and also by new choices (*exploration*), which is essentially trial-and-error learning. (Sutton and Barto (1998))

In short, RL solves the difficult problem of correlating immediate actions with the delayed returns they produce on basis of trial and error. An agent operates in an interactive delayed-return environment, where it can be difficult to understand which action leads to which outcome over many time steps over time.

2.1.1 Difference between Reinforcement Learning and Super/Unsupervised learning

2.1.1.1 Reinforcement learning and Supervised Learning

Supervised learning is the machine learning task of learning a function that maps an input to an output based on a training set of labeled input-output pairs which are provided by an external supervisor. The objective of this learning is to create an inferred function/system that is able to generalize its responses so that it acts correctly in situations not present in the training set. However, under interactive environment, there are so many subtasks that the agent needs to achieve in order to obtain examples of desired behavior that correctly represent all of the situations where the agent has to act. Therefore, in reinforcement learning the agent needs to

learn from its own experience based on reward function which arises from agent's action as a feedback from environment that the agent is operating in. This is the main difference between supervised learning and reinforcement learning.

2.1.1.2 Reinforcement learning and Unsupervised Learning

Unsupervised learning is learning by trying to find underlying structure hidden in collections of unlabeled data rather than the mapping function as in supervised learning. It does not use any kind of feedback from the environment regarding performances of the agent while the agent in reinforcement learning tries to maximize a reward signal without trying to find hidden patterns.

2.1.1.3 Reinforcement Learning

In supervised learning, one has a target label for each training example while, in unsupervised learning, one has to find hidden patterns without explicit labels at all. However, in reinforcement learning, one has time-delayed labels which are called the rewards. By relying upon those rewards, the agent needs to learn to interact with the environment.

In summary, according to Sutton and Barto (1998)," *RL is a computational approach to understanding and automating goal-directed learning and decision making. It is distinguished from other computational approaches by its emphasis on learning by an agent from direct interaction with its environment, without requiring exemplary supervision or complete models of the environment*".

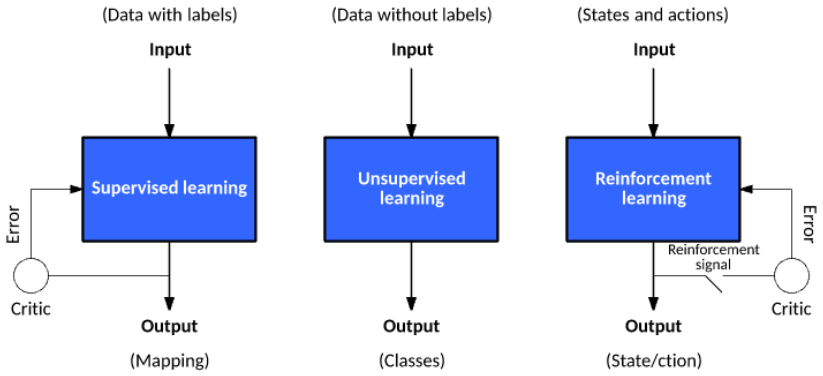

Figure 2.1: Three learning models for algorithms

2.1.2 Important terms and concepts in RL

- Agent: an agent takes actions.
- Action: an action is the set of all possible moves the agent can make.
- Environment: the environment is a place where the agent interacts with. The environment takes the current state and action of the agent as an input, and returns corresponded reward and next state as an output to the agent.

- State (S): a state is the situation where the agent finds itself in, which is a specific place and moment that puts the agent in relation to other things in the environment such opponents, obstacles or prizes.

- Reward (R): a reward is the feedback given by the environment as a result of agent's actions. The agents objective is to maximize the total reward it receives over the long run. The reward signal thus defines what are the good and bad events for the agent and can be immediate or delayed. Therefore, it is the primary basis for altering the policy; if an action selected by the policy is followed by low reward, then the policy may be changed to pick some other actions in that situation in the future. In short, reward effectively evaluate the agent's action.

- Policy (π): a policy is the strategy that the agent employs to determine the next action based on the current state, which is a mapping from perceived states of the environment to actions.

- Value function (V): a value function is the expected long-term return with discount, as opposed to the short-term reward R. $V_\pi(s)$ is defined as the expected return in the long term of the current state under policy π.

  Sutton and Barto (1998) states that "while the reward signal indicates what is good in an immediate sense, a value function specifies what is good in the long run. The value of a state is the accumulated amount of reward an agent expects over the future, starting from that state. Whereas rewards determine the immediate, intrinsic desirability of environmental states, values indicate the long-term desirability of states after taking into account the states that are likely to follow, and the rewards available in those states".

- Q-value or action-value (Q): Q-value is similar to Value but it takes an extra parameter, the current action a. $Q_\pi(s,a)$ refers to the long-term return of the current state *s*, taking action *a* and then following policy π. In short, Q maps state-action pairs to rewards.

- Discount factor(γ): the discount factor is multiplied with future rewards as discovered by the agent as it represent how much future rewards lose their value according to how far away in time they are. In short, it is a parameter indicating that how much future rewards worth less than immediate rewards.

2.1.3 Markov Decision Process(MDP)

According to Sutton and Barto (1998), Markov Decision Process(MDP) is the a straightforward framing of the problem of learning from interaction to achieve a goal. **Agent** is the learner and decision maker. **Environment** is the thing the agent interact with. **Action** is selected by agent. The environment responds to that action and then presents new situation to the agent. **Reward**, special numerical values that

the agent seek to maximize over time through its choice of action, is produced by the environment.

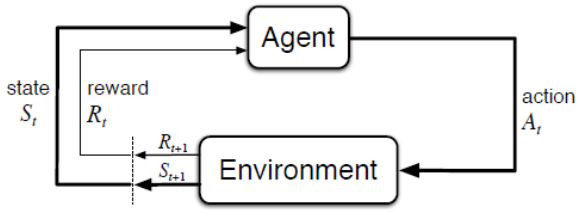

Figure 2.2: The agent-environment interaction in a Markov decision process.

More precisely, Markov decision process is a discrete time stochastic control process which is a classical formalisation of sequential of decision making (BELLMAN (1957)). At each time step, the process is in some state *s*, and an agent may choose any action *a* that is available in state *s*. The environment responds at the next time step by randomly moving into a new state *s'*, and giving back the agent a corresponding reward R_a (*s*,*s'*).

Characterized as environment's dynamics in MDP, the probability that the current state *s* moves into its new state *s'* is influenced by the chosen action described in this the state transition function P_a (*s*,*s'*). Therefore, the next state *s'* depends on the current state *s* and the agent's action *a*. More importantly, the state is said to have the *Markov property* if it includes information about all aspects of the past agent-environment interaction that make a difference for the future (Sutton and Barto (1998)). In short, *Markov Property* will be satisfied if given current state *s* and action *a* is conditionally independent of all previous states and actions.

Definition: A Markov decision process is a 5-tuple (S , A ,P_a , R_a , γ) where

- S is a finite set of states.
- A is a finite set of actions. On the other hand, A_s is the finite set of actions available from state *s*.
- P_a (s , s') = Pr ($s_{t+1} = s' \mid s_t = s, a_t = a$) is the probability of an action a in state s at time t leading to state s' at time $t+1$.
- R_a (s , s') is the an immediate reward obtained after transitioning from state s to state s' because of action a.
- $\gamma \in [0,1]$ is the discount factor, which represents the difference in importance between future rewards and present rewards.

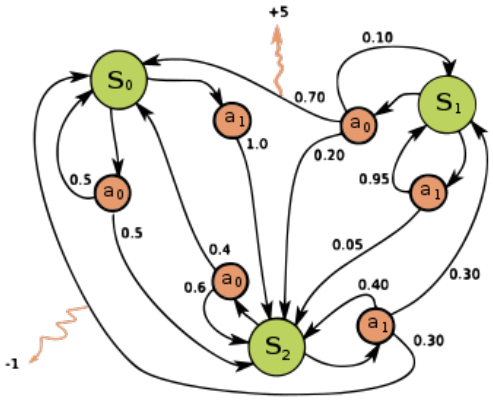

Figure 2.3: Example of a simple MDP with three states (green circles) and two actions (orange circles), with two rewards (orange arrows).

2.1.4 Policies and Value Functions

Sutton and Barto (1998) states that almost all reinforcement learning algorithms involve the estimating of *value functions* — functions of states (or of state-action pairs) that calculate *how good* it is to perform a given action in a given state or how good it is for the agent to be in a given state.

2.1.4.1 Value function

A ***value function*** is defined in terms of expected future rewards or, more precisely, in term of expected return, which depends on what actions the agent will take. Accordingly, value functions are associated with agent's particular actions which can be called *policy*. The *value function* of a state *s* under a policy π, denoted $v_\pi(s)$, is the expected return when starting in *s* and following π thereafter. For MDPs, we can define v_π formally by

$$v_\pi(s) \doteq \mathbb{E}_\pi[G_t \mid S_t = s] = \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \,\middle|\, S_t = s\right], \text{ for all } s \in \mathcal{S},$$

Figure 2.4: the function v_π is called the *state-value function for policy.* π

where $\mathbb{E}_\pi[\cdot]$ denotes the expected value of a random variable given that the agent follows policy π, and t is any time step.

2.1.4.2 Policy

A ***policy*** is a mapping from states to probabilities of selecting each possible action. If the agent is following policy π at time t, then $\pi(a \mid s)$ is the probability that $A_t = a$ if $S_t = s$. Sutton and Barto (1998) define that reinforcement learning methods specify how the agent's policy is changed as a result of its experience. $q_\pi(s,a)$ can be defined

$$q_\pi(s,a) \doteq \mathbb{E}_\pi[G_t \mid S_t=s, A_t=a] = \mathbb{E}_\pi\left[\sum_{k=0}^{\infty}\gamma^k R_{t+k+1} \,\middle|\, S_t=s, A_t=a\right].$$

Figure 2.5: q_π is called the action-value function for policy π.

as value or an expected return that an agent gets after taking action a in state s and thereafter following under a policy π.

2.1.4.3 Fundamental property of Value functions

A fundamental property of value functions used throughout reinforcement learning and dynamic programming is that they satisfy recursive relationships.

$$\begin{aligned} v_\pi(s) &\doteq \mathbb{E}_\pi[G_t \mid S_t=s] \\ &= \mathbb{E}_\pi[R_{t+1}+\gamma G_{t+1} \mid S_t=s] \\ &= \sum_a \pi(a|s)\sum_{s'}\sum_r p(s',r|s,a)\Big[r+\gamma\mathbb{E}_\pi[G_{t+1}|S_{t+1}=s']\Big] \\ &= \sum_a \pi(a|s)\sum_{s',r} p(s',r|s,a)\Big[r+\gamma v_\pi(s')\Big], \quad \text{for all } s\in\mathcal{S}, \end{aligned}$$

The last equation is the *Bellman equation for*v_π which expresses a relationship between the value of a state and the values of its successor states.

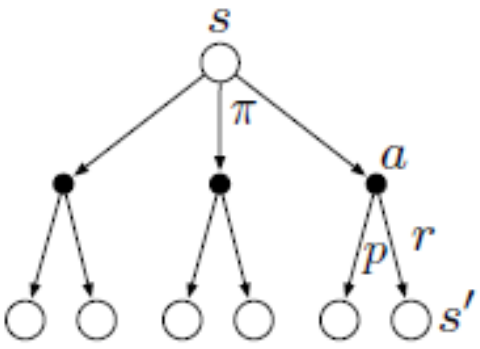

Figure 2.6: Backup diagram for v_π

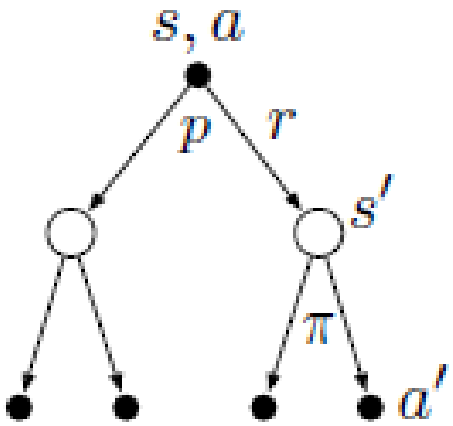

Figure 2.7: Backup diagram for q_π

2.1.5 Optimal Policies and Optimal Value Functions

Solving a reinforcement learning task means finding a policy that achieves a lot of reward over the long run.

Sutton and Barto (1998) proves that for *finite MDPs we can precisely define an optimal policy in the following way. Value functions define a partial ordering over policies. A policy π is defined to be equal to or better than a policy π' if its expected return is equal to or greater than that of π' for all states.*

- Optimal Value Function: *optimal state-value function*
  $v_*(s) = \max_\pi v_\pi(s)$, for all $s \in \mathbb{S}$.

- Optimal Policy: *optimal action-value function*
  $q_*(s,a) = \max_\pi q_\pi(s,a)$, for all $s \in \mathbb{S}$ and $a \in \mathbb{A}(s)$.

For the state-action pair *(s,a)*, this function expresses the expected return of action *a* taking in state *s* and thereafter following an optimal policy. Thus, q_* can be written in terms of v_* as follows:

$$q_*(s,a) = \mathbb{E}[R_{t+1} + \gamma v_*(S_{t+1}) \mid S_t = s, A_t = a].$$

Because v_* is the value function for a policy, it must satisfy the self-consistency condition given by the Bellman equation for state values.

$$\begin{aligned} v_*(s) &= \max_{a \in \mathcal{A}(s)} q_{\pi_*}(s,a) \\ &= \max_a \mathbb{E}_{\pi_*}[G_t \mid S_t = s, A_t = a] \\ &= \max_a \mathbb{E}_{\pi_*}[R_{t+1} + \gamma G_{t+1} \mid S_t = s, A_t = a] \\ &= \max_a \mathbb{E}[R_{t+1} + \gamma v_*(S_{t+1}) \mid S_t = s, A_t = a] \\ &= \max_a \sum_{s',r} p(s', r \mid s, a)\big[r + \gamma v_*(s')\big]. \end{aligned}$$

The last two equations are two forms of the **Bellman optimality equation for v_***.

Hence, **Bellman optimality equation for q_*** is

$$\begin{aligned} q_*(s,a) &= \mathbb{E}\Big[R_{t+1} + \gamma \max_{a'} q_*(S_{t+1}, a') \;\Big|\; S_t = s, A_t = a\Big] \\ &= \sum_{s',r} p(s', r \mid s, a)\Big[r + \gamma \max_{a'} q_*(s', a')\Big]. \end{aligned}$$

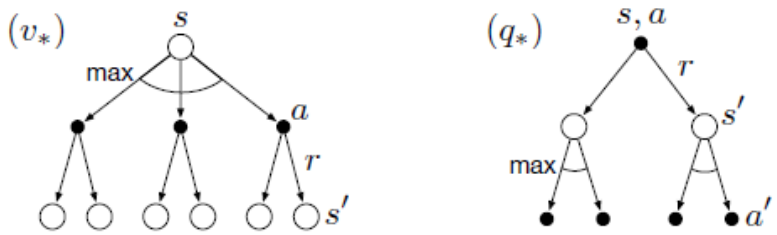

Figure 2.8: Backup diagram for v_* and q_*

2.1.6 Prediction and Control Problems

In general, RL methods are implemented to solve two related problems: the Prediction Problem and the Control Problem.

2.1.6.1 Prediction Problems

RL is used to learn the value function for the policy an agent pursues. At the end of learning, this value function describes for every visited state how much future reward we can expect when performing actions starting at this current state.

Algorithm: Monte Carlo methods(MC) and Temporal-Difference(TD) learning

1. Monte Carlo methods(MC)
MC methods must wait until the end of the episode to get the return and use that return as a target for update $V(S_t)$ (only then is G_t known). A simple every-visit MC method, which is suitable for non-stationary environments, is given by this following update formula:

$$V(S_t) \leftarrow V(S_t) + \alpha\Big[G_t - V(S_t)\Big],$$

where G_t is the actual return following time *t*, and α is a step-size parameter.

2. One-step TD or TD(0)
TD methods need to wait only until the next time step. At time t + 1, TD methods immediately set up a target and make a useful update using the observed reward R_{t+1} and the estimated $V(S_{t+1})$. The simplest TD method makes the update immediately on transition to S_{t+1} and receives R_{t+1}, which is called TD(0) and has an update rule as the following formula:

$$V(S_t) \leftarrow V(S_t) + \alpha\Big[R_{t+1} + \gamma V(S_{t+1}) - V(S_t)\Big]$$

In effect, the target for the MC update is G_t, whereas the target for the TD update is $R_{t+1} + \gamma V(S_{t+1})$.

Advantages of TD learning over MC method:

- TD method does not require a model of the environment, reward and next-state probability distributions.(Sutton and Barto (1998))

- TD method is naturally implemented in an online, fully incremental fashion as an agent makes an update immediately after one time-step while under MC method the agent needs to wait until the end of episode since only then return is known. This is major difference and must be considered carefully. For example, some applications have very long episodes, so that delaying all learning until the end of the episode is too slow. Other applications are continuing tasks and have no episodes at all.(Sutton and Barto (1998))

- Sutton (1988) shows that MC methods must ignore or discount episodes on which experimental actions are taken, which can greatly slow learning. TD methods are much less vulnerable to these kind of problems because they learn from each transition regardless of what subsequent actions are taken.

- TD methods are usually found to be faster converge approaches than MC methods on stochastic tasks. (Dayan (1994))

2.1.6.2 Control Problems

As an agent travels through state spaces and interacts with environment, the main objective is to find a policy that has the most rewards. Going through and optimizing several policies to obtain the optimal policy that maximizes the long-term accumulated reward is major goal for action planning and optimal control which is required for solving control problems. Since this is a predictive type of control, to solve the control problem would also require a solution to the prediction problem.

Algorithms for Control Problems: Sarsa and Q-learning

1.Sarsa: On-policy TD Control
To solve control problems, the first step is to learn an action-value function rather than state-value function. In particular, for an on-policy method we must estimate q_π(s,a) for the current behavior policy π and for all states s and actions a. We consider transitions from state-value pair to state-action pair and learn the values of state-action pairs. Since these cases are both Markov chains with a reward process, they are identical in formal. The theorems assuring the convergence of state values under TD(0) also apply to the corresponding algorithm for action values: (Rummery and Niranjan (1994))

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha\Big[R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t)\Big].$$

Sarsa (on-policy TD control) for estimating $Q \approx q_*$

Algorithm parameters: step size $\alpha \in (0, 1]$, small $\varepsilon > 0$
Initialize $Q(s, a)$, for all $s \in \mathcal{S}^+, a \in \mathcal{A}(s)$, arbitrarily except that $Q(terminal, \cdot) = 0$

Loop for each episode:
  Initialize S
  Choose A from S using policy derived from Q (e.g., ε-greedy)
  Loop for each step of episode:
    Take action A, observe R, S'
    Choose A' from S' using policy derived from Q (e.g., ε-greedy)
    $Q(S, A) \leftarrow Q(S, A) + \alpha\big[R + \gamma Q(S', A') - Q(S, A)\big]$
    $S \leftarrow S'$; $A \leftarrow A'$;
  until S is terminal

2.Q-learning: Off-policy TD Control
Introduced by Watkins (1989), the learned action-value function(Q) directly approximates the optimal action-value function (q_*), which is independent of the policy being followed. However, the policy still has an effect that determines which state-action pairs are visited and updated. This helps simplify the analysis and enable early convergence of the algorithm.

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha\Big[R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t)\Big].$$

Q-learning (off-policy TD control) for estimating $\pi \approx \pi_*$

Algorithm parameters: step size $\alpha \in (0, 1]$, small $\varepsilon > 0$
Initialize $Q(s, a)$, for all $s \in \mathcal{S}^+, a \in \mathcal{A}(s)$, arbitrarily except that $Q(terminal, \cdot) = 0$

Loop for each episode:
  Initialize S
  Loop for each step of episode:
    Choose A from S using policy derived from Q (e.g., ε-greedy)
    Take action A, observe R, S'
    $Q(S, A) \leftarrow Q(S, A) + \alpha\big[R + \gamma \max_a Q(S', a) - Q(S, A)\big]$
    $S \leftarrow S'$
  until S is terminal

2.1.7 Challenges in Reinforcement Learning

The following is common scenarios that use RL algorithm:

- RL agent does not know anything about the environment. It learns what to do

by exploring the environment.

- RL agent uses action and receives states and rewards correspondingly.
- RL agent can only change the environment through its own actions.

The above scenarios raise major difficulties in RL as following:

- Some actions take time to obtain a corresponding reward and learning this dynamics can be challenging.
- The reward received by the environment may not be related to the last action but some action in the past of the agent.

In summary
The agents take actions in an environment and receive states and rewards. However, it does not know which actions produce rewards and when an action will produce rewards since sometimes an agent does an action that takes time to produce rewards. The goal is to find a policy that maximizes agent's value function and it is needed to be learned from interaction with the environment. This is a major issue in RL which is known as *Exploration-Exploitation Dilemma*.

Exploration-Exploitation Dilemma
RL agents need to explore their environment in order to assess its reward structure. After some periods of exploration, the agent might discover a set of explicitly rewarding actions. However, it still does not know that the found actions are actually the best action. Therefore, it is a trade-off between making the best decision given current information (exploitation) and gathering more information (exploration).

2.2 Related Work

2.2.1 Designing Neural Network Architectures

2.2.1.1 Hyperparameter Optimization

Several techniques for automating selection of network architectures have been proposed but fail to achieve the similar level of performance of handcrafted networks. For example, several Bayesian optimization approaches have been proposed recently to search for network topologies (Shahriari et al. (2016); Bergstra et al. (2013); Domhan et al. (2015)) and hyperparameters. (Snoek et al. (2012) ; Swersky et al. (2013))

2.2.1.2 Genetic Algorithm and Neuro-evolution Algorithm

Various schemes for combining genetic algorithms and neural networks been proposed and tested for automating neural network design can be traced back to the 1980s (Schaffer et al. (1992)). However, several algorithms based on genetic algorithm proposed in the past fail to match the performance of handcrafted networks and less flexible in designing network topologies.(Verbancsics and Harguess

(2013)) For example, Stanley and Miikkulainen (2002) presents a method called NeuroEvolution of Augmenting Topologies (NEAT). Pinto and Cox (2012) proposes biologically-inspired high-throughput network selection algorithm which is based on screening method in genetics.

Wierstra et al. (2005) proposes the general framework for sequence learning, EVOlution of recurrent systems with LINear Outputs (Evolino), to discover good RNN hidden node weights through evolution, while using linear regression to calculate an optimal linear mapping from hidden state to output. Floreano et al. (2008) gives an overview of the most prominent methods for evolving artificial neural networks with a special focus on recent advances in the synthesis of learning architectures. Stanley et al. (2009) proposes a method to evolve artificial neural networks(ANNs) through evolutionary algorithms which is inspired by the evolution of biological brains. All of these approaches are modern neuro-evolution algorithms which are very flexible in creating new models. However, they are less practical at a large scale due to limitation of search-based methods which require external guided intuition.

2.2.1.3 Meta-Learning

Thrun and Pratt (1998) proposes the methods to use some particular values of base-model parameters that share some common properties across similar algorithms. This method can be applied to use in new task more quickly. Similar more modern approach (Bergstra et al. (2012)) which relies on a meta-modeling approach based on Tree of Parzen Estimators (TPE) has been proposed by Bergstra et al. (2011) to choose both the hyperparameters and type of layers of feed-forward networks. However, both methods fail to match the performance of hand-tuned networks.

2.2.2 MetaQNN Algorithm: Designing Neural Network Architectures Using Reinforcement Learning (Baker et al. (2016))

This report will focus on searching for best neural network architecture using Reinforcement Learning which relies heavily on existing algorithm,which in this case refers to MetaQNN (Baker et al. (2016)). MetaQNN is a meta-modeling algorithm based on reinforcement learning to automatically generate high-performing CNN architectures for a given learning task. The learning agent is trained to sequentially choose CNN layers using Q-learning with an ε-greedy exploration strategy (Mnih et al. (2015)) and experience replay. (ji Lin (1992)) The agent explores a large but finite space of possible architectures and iteratively discovers models with improved performance on the learning task.

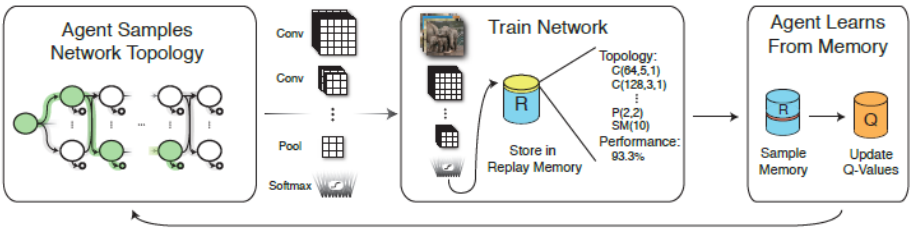

Figure 2.9: Designing CNN Architectures with Q-learning: The agent begins by sampling a Convolutional Neural Network (CNN) topology conditioned on a predefined behavior distribution and the agents prior experience (left block). That CNN topology is then trained on a specific task; the topology description and performance, e.g. validation accuracy, are then stored in the agents memory (middle block). Finally, the agent uses its memories to learn about the space of CNN topologies through Q-learning (right block).

2.2.2.1 Architectures behind MetaQNN

- Q-learning

  $$Q_{t+1}(s_i, u) = (1-\alpha)Q_t(s_i, u) + \alpha\left[r_t + \gamma \max_{u' \in \mathcal{U}(s_j)} Q_t(s_j, u')\right].$$

  The Bellman update equation has two parameters: (i) α is a Q-learning rate which determines the rate allocated to new information over old information, and (ii) γ is the discount factor which determines the weight given to short-term rewards over future rewards.

- ε-greedy exploration strategy (Mnih et al. (2015))
  With this strategy, a random action is taken with probability and the greedy action(Mnih et al. (2015)), $\max_{u \in U(s_i)} Q_t(s_i, u)$, is chosen with probability 1 - ε. ε is gradually annealed from 1 to 0 for the purpose of balancing exploration-exploitation tradeoff, meaning that the agent begins in an exploration phase and slowly starts moving towards the exploitation phase.

- experience replay (ji Lin (1992))
  When the exploration space is large, it is practical and useful to use the experience replay technique for faster convergence (ji Lin (1992)). In experience replay, the agent is presented with a memory of its past explored paths and rewards. At a given interval, the agent samples from the memory and updates its Q-values from Q-learning equation.

2.2.2.2 MetaQNN Algorithms: Training Procedure

1. The agent models the layer selection process as a MDP with the hierarchical nature of the feature representation assumption which learned by neural networks with many hidden layers (Lecun et al. (2015)). Therefore, a well-performing layer in one network should also perform well in another network.

2. The agent sequentially selects layers via the ε-greedy strategy until it reaches a termination state.

3. The CNN architecture defined by the agent's path is trained on the chosen learning problem and the agent is given a reward in term of validation accuracy.

4. The validation accuracy and architecture description are stored in a replay memory, and experiences are sampled periodically from the replay memory to update Q-values.

5. The agent follows an ε-schedule which determines the shifting phase from exploration to exploitation.

2.2.2.3 MetaQNN Space Requirements

State Space: reducing CNN layer definitions to simple state tuples.
There are 5 different types of layer.

1. Convolution layer(C)

2. Pooling layer(P)

3. Fully Connected layer(FC)

4. Global Average Pooling layer(GAP)

5. Softmax layer(SM)

Layer Type	Layer Parameters
Convolution Layer (C)	layer depth(i), receptive field size(f), stride(l), number of receptive field(d), representation size(n)
Pooling Layer (P)	layer depth(i), receptive field size(f), stride(l), representation size(n)
Fully Connected Layer (FC)	layer depth(i), number of consecutive FC layers(n), number of neurons(d)
Global Average Pooling/Softmax Layer	previous state(s), type(t)

Table 2.1: Experimental State Space. For each layer type, the relevant parameters and the values of each parameter that the agent is allowed to take.

2.2.2.4 Q-learning training procedure

Main idea: to balance the size of the state-action space and the model capacity with the enough exploration paths needed by the agent to converge.

1. Q-learning rate (α): the Q-learning rate (α) is set to 0.01.

2. Discount factor (γ): the discount factor (γ) is set to 1 in order to avoid over-prioritize short-term rewards.

3. ε-greedy: ε is gradually decreased from 1.0 to 0.1 in steps. The step-size is defined by the number of unique models trained. At $\varepsilon = 1.0$, in the beginning the agent samples CNN topology randomly along a uniformly weighted Markov chain. MetaQNN procedure is used to train every topology sampled by the agent and the prediction performance of this trained network topology on the validation set is recorded (Baker et al. (2016))

   A larger number of models are trained at $\varepsilon = 1.0$ as compared to other values of ε which helps ensure that exploration phase of the agent is adequate before beginning exploitation phase.

   The training procedure is stopped at $\varepsilon = 0.1$ since if stopping at $\varepsilon = 0$, the final policy may end up as non-stochastic final policy and in non-global minimum. Ideally, the project wants to identify several well-performing model topologies, which can then be ensembled to improve prediction performance.

Chapter 3

Q-Search: Approach

The project focuses on Q-Learning Algorithm to generate the best topology of convolutional neural network architectures that maximize validation accuracy. The states that the agent are in are hierarchical layers of neural network and the actions that the agent can perform are different type of layers that the agent can choose. The project follows the main concepts of MetaQNN (Baker et al. (2016)) i.e. Q-Learning, ε-Greedy and Experience Replay with some modification on number of layers (State) that the agent is in and type of layers (Action) that the agent can select. In addition, the project also runs several experiments on various different configuration settings in term of number of episodes, number of models sampled from experience replay to update Q-table and how and when to sample models from experience replay.

3.1 Layer type, number of layer and layer selection procedure

The Markov Decision Process(MDP) is the main concept that the project relies upon by allowing the agent to find optimal paths only in a finite-time-horizon environment. The time horizon represents the space that the agent is in and the agent can perform certain actions in that space. Therefore, the agent is allowed to select only limited types of the layer (Action space) with some hyperparameter configurations and there are limited numbers of layers (State space) that the agent can transverse.

3.1.1 Layer Type and Number of Layer

There are total number of 16 layers that the agent can select and the maximum number of layers that the agent is in is 4. The agent will stop whenever it reaches the last layer or gets softmax layer (termination layer) and there are total number of 16 different types of layers that the agent can select. Padding is applied as to allow different configurations of layers to be able to connect to each other. 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. (note that for 4 layers, the number of possible architecture in this setup is 69905 which is too large to perform

brute force search.)

Layer Type	Layer Parameters	Total Numbers of Layer
Convolution Layer(C)	Number of output filters: 32,36,48,64 Kernel size: 3,4,5 Stride size: 1 Apply padding and Follow by ReLU	12
Pooling Layer(P)	(Kernel size, Stride): (2,2), (3,2), (5,3) Apply padding	3
Softmax Layer(S)	Output class	1

Table 3.1: Experimental State and Action Space. Configuration of CNNs layers that agent can select.

3.1.2 Layer Selection Procedure

The layer selection procedure starts with the initial input layer. At each step, the agent performs action by selecting the next layer of the network and the action the agent took always succeeds. The procedure stops when the agent reaches the maximum number of layers or selects the softmax layer(the termination state). When the agent has a complete topology of the model, the model is trained and tested on validation set to get the accuracy. The action of the agent is considered to be deterministic since if the agent selects a model that has already been trained, it will be given previously-found validation accuracy without needing to retrain the found model for simplicity.

Layer Name	Layer Type	Filter Number	Kernel Size	Stride Number
c1	Convolutional	32	3x3	1
c2	Convolutional	32	4x4	1
c3	Convolutional	32	5x5	1
c4	Convolutional	36	3x3	1
c5	Convolutional	36	4x4	1
c6	Convolutional	36	5x5	1
c7	Convolutional	48	3x3	1
c8	Convolutional	48	4x4	1
c9	Convolutional	48	5x5	1
c10	Convolutional	64	3x3	1
c11	Convolutional	64	4x4	1
c12	Convolutional	64	5x5	1
m1	Max Pooling	-	2x2	2
m2	Max Pooling	-	3x3	2
m3	Max Pooling	-	3x3	3
s	Softmax	-	-	-

Table 3.2: Actions The agent has 16 different types of layers as its action space.

3.2 Q-Search

Q-Search employs Q-Learning, a model-free reinforcement learning algorithm, with Q-table to record value of actions in every state. The main idea is to learn the action-value function Q by iteratively updating the estimate of Bellman equation.

The Q-learning rate(α) is set to 0.01 and discount factor(γ) is set to 1 as not to overprioritize short-term rewards over long-term rewards. This implies that the way that the agent updates Q-Table will not be biased toward models with fewer number of layers over models with higher number of layers but rather based on the whole contribution of overall architecture.

Though the state and action space for agent are limited, the search space is still combinatorially large. In addition, due to limited number of episodes restricted by training time, the environment is partially observable. This is why Q-Learning algorithm is suitable for this kind of problem as it is a model-free learning. The Q-learning algorithm breaks up large problem into many smaller subproblems and iteratively updates Q-table overtime.

3.3 Epsilon-Greedy(ε)

The exploration is necessary for the Q-function to converge. The strategy is to take random action with probability ε and to take greedy action according to argmax Q(s,a) with probability 1 - ε. ε is gradually reduced overtime so that the agent starts with exploration phase early and gradually exploits that knowledge to achieve the maximum reward. The agent needs very large exploration phase before entering exploitation phase for the reward since the agent needs to learn several models before it can decide which one to exploit.

The way the value of epsilon being assigned to each episode also plays an important role in determining the behavior of the agent. If few episodes are given with high value of ε, the agent will not have enough spaces to explore. The agent could end up into getting a suboptimal result.

Epsilon(ε)	**1.0**	**0.9-0.7**	**0.6-0.1**
Number of model trained	1500-2500	500-1000	500-800

Table 3.3: ε Schedule. The agent trains the specified number of models at each ε.

3.4 Experience Replay

The experience replay is used to store name, topology, accuracy and loss of models and used by the agent as a memory of its past explored paths and rewards to randomly sample to update Q-table at a given interval. This technique helps Q-Learning algorithm converge faster, stabilize training process and break the similarity of some data as the action of agent is performed more greedily as training goes on.

3.5 Comparison between Q-Search and MetaQNN approach

Approach	**MetaQNN**	**Q-Search**
State Space	Unlimited number of layer	Maximum number of 4 layers
Action Space	Unlimited type of layer	16 different types of layers
Dataset	MNIST, CIFAR-10, CIFAR-100, SVHN	MNIST, CIFAR-10
Total number of episodes	2700 episodes	2500-3800 episodes
Reinforcement Algorithm	Q-learning without Q-table	Q-learning using Q-table
Exploration Strategy	ε-greedy strategy	ε-greedy strategy
When to sample models from Experience Replay to update Q-table	Every time agent trains new model	3 Updating modes: 1. Every 100 episodes 2. Every time agent trains new model 3. No update
Number of models sampled from Experience Replay to update Q-table	100 models	0, 5, 10 or 50 models
Deep Learning Framework	Caffe framework	Tensorflow framework
Number of GPUs required	at least 10 GPUs	1-2 GPU(s)

Table 3.4: Comparison Table between this project and MetaQNN approach.

3.6 Overview of overall procedures

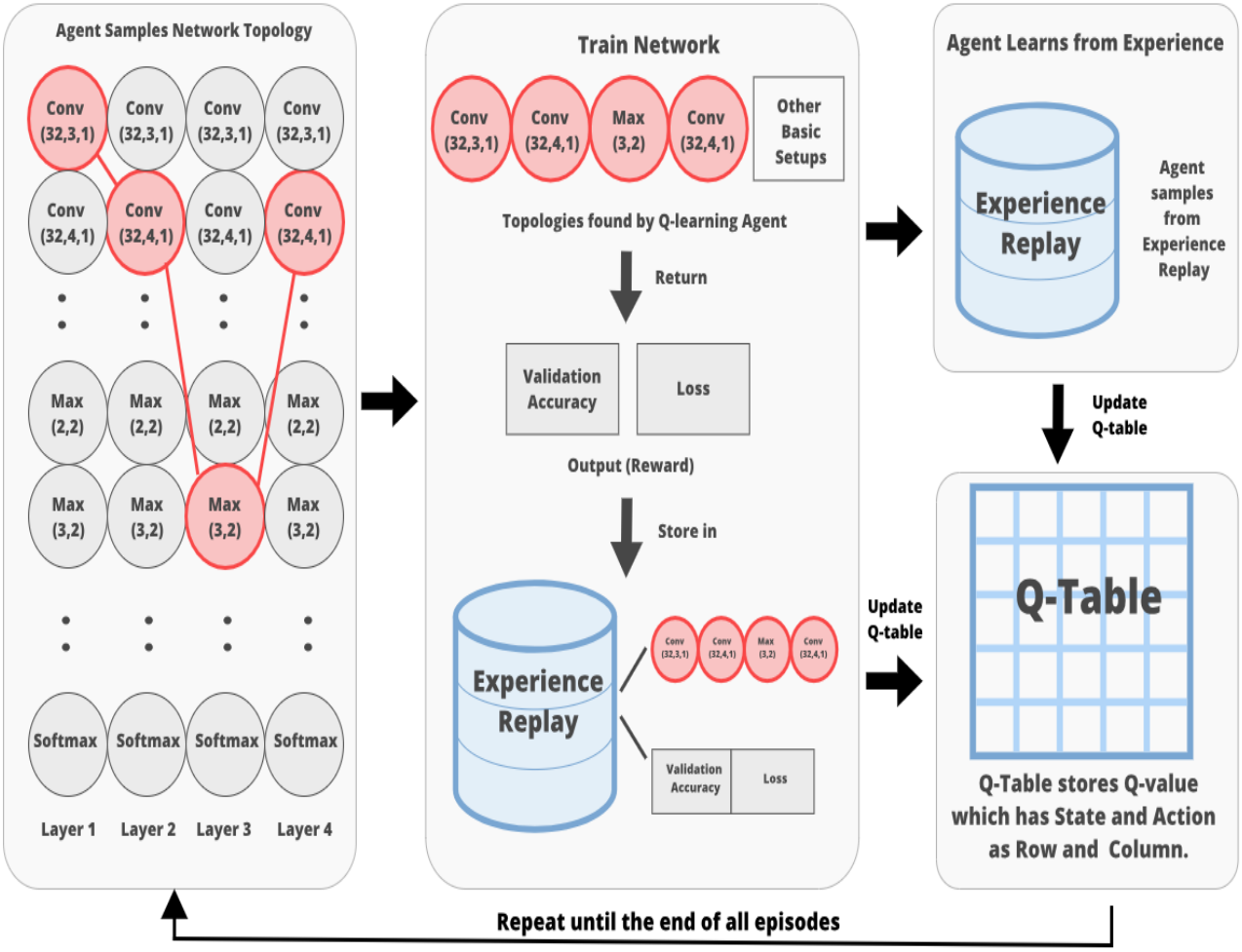

Figure 3.1: Designing CNN Architectures with Q-Search Left box: the agent starts by sampling the CNN architectures based on some pre-defined criteria. Center box: the network is trained and the agent is given the reward as validation accuracy. The topologies and reward are stored in experience replay. Right box: the Q-table can be updated directly after network is trained or can be updated by sampling from experience replay.

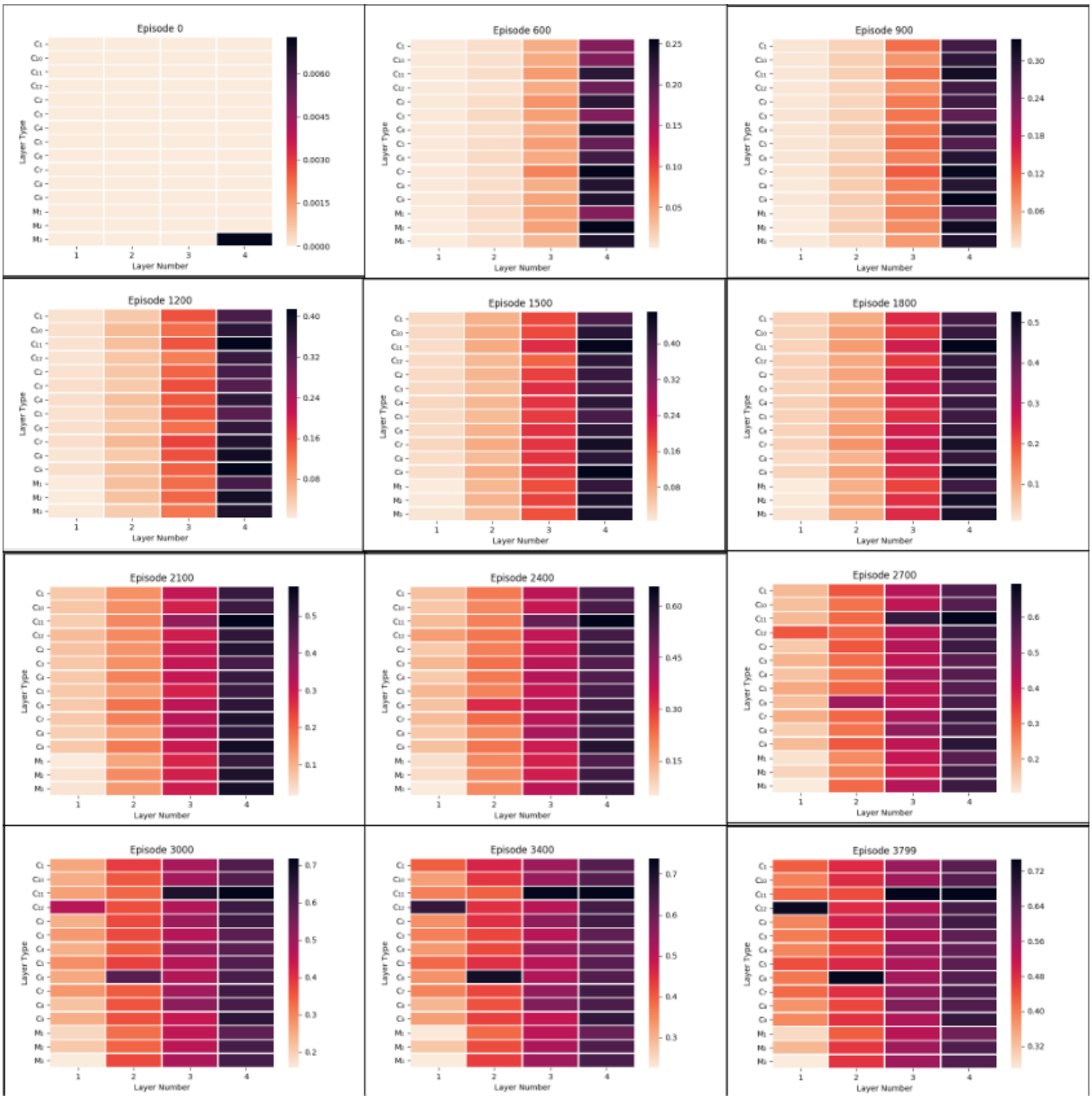

Figure 3.2: Snapshot of Q-Table by Q-Search in different episodes: Row consists of Action space while Column consists of State space. In the beginning of the episode, the agent explores spaces by using random strategy. As the training goes on, the agent starts employing exploitation strategy by greedily selecting topologies that give highest validation accuracy. The snapshot shows the convergence of overall training as in the last episode the action in each layer settles in just one highest value.

Chapter 4

Experimental Setups and Results

The goal of all of the experiments is to compare several searching algorithms such as Random Search and Layerwise Search with Q-Search on validation accuracy of topologies found by the above algorithms. In addition, the goal is to find the best setting of overall training of Q-Search as to for the agent be able to discover the topologies that maximizes validation accuracy and as to for the overall training minimize the overall training time. The main idea of Q-Search is to use Reinforcement Learning algorithm namely Q-learning with different exploration techniques and different setups to automate the design process of CNN architecture.

The Q-learning agent role is to select topologies that give the highest validation accuracy. With different settings, the project tries to find the best setups for the agent that can discover the best topologies while using the least training time. For example, the number of episode is set to be varied in order to observe the effect. The ways that the agent uses experience replay in both frequency of model sampling and number of model to be sampled from are also set differently. In sum, the project tests the described approach in several experiments with different setups on two real datasets: MNIST and CIFAR-10. The MNIST dataset took around 3-6 days to complete and the CIFAR-10 dataset took 6-11 days. All models is run on just one GPU which is either Nvidia GeForce GTX TitanX Black or Nvidia GeForce GTX 1080.

4.1 Experiment Setups

The neural networks are constructed using Tensorflow and run with GPU setup. All weights are initialized with Xavier initialization. The maximum number of layers is limited such that the agent ends up in 1-4 layers and there are only 16 different types of layers that the agent can select.

For Q-Search setup, Q-learning rate(α) is set to 0.1 and discount factor(γ) is set to 1. At the end of every episode, Q-table is updated by the reward obtained by the agent in form of validation accuracy of the model. The experience replay updating technique varies across the different experiments. For example, one technique is to update the Q-table every time the agent finishes training new model by sampling

different numbers of models from the experience replay. Another technique is to sample 50 models from experience replay every 100 episodes to update Q-table. For trade-off between exploration and exploitation period in Q-Search, the ε is annealed overtime as training goes on according to ε table on different experiments.

4.1.1 MNIST

MNIST is a simple computer vision dataset. It consists of images of handwritten digits of 10 classes and also includes labels for each image telling which digit it is. In this project, MNIST dataset is split into 3 parts: 55000 data points of training data, 10000 points of test data, and 5000 points of validation data. Each image is 28 pixels by 28 pixels. It is trained with Gradient descent algorithm with 0.001 learning rate and with Batch size of 100.

Standard procedure:
The input image is flatten and fed into a single matrix. The Q-learning agent selects CNN topologies according to pre-defined specifications. The architecture that is chosen by the agent is then appended with fully-connected layer(FC Net) of 1024 units with ReLU activation function (Agarap (2018)) and dropout with probability of 50 percent (Srivastava et al. (2014)). Lastly, the network is finished with last layer of fully-connected layer of 10 unit output classes.

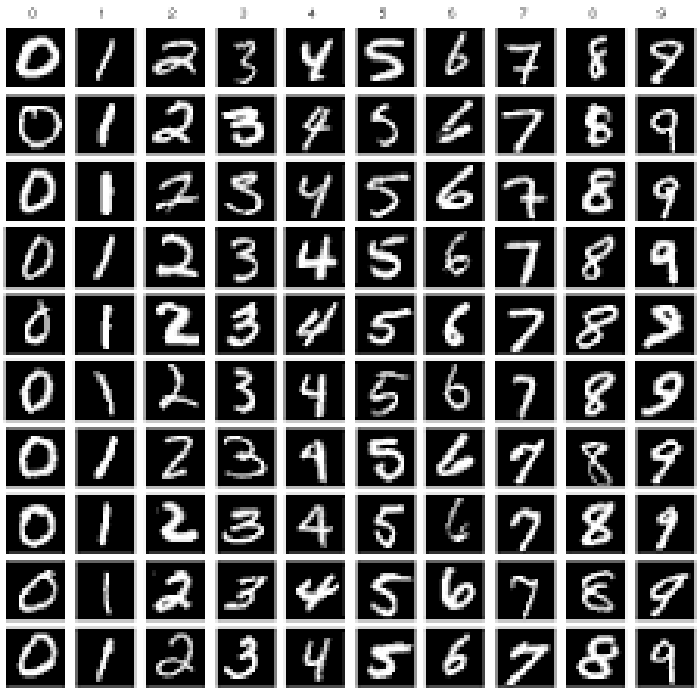

Figure 4.1: Example images from the MNIST dataset

4.1.2 CIFAR-10

The project experiment on the CIFAR-10 dataset which consists of colour 32x32 images with 10 classes. CIFAR-10 dataset consists of 60000 32x32 colour images in 10 classes with 6000 images per class. There are 50000 training images and 10000 test images.(Krizhevsky (2009)) Batch size is set to 32 and Epoch is set to 100. RMSProp Optimizer is used with learning rate equal to 0.0001 and decay rate is set to 0.000001. Data augmentation is applied by flipping image horizontally, randomly shifting image vertically and horizontally. Early stopping is used when the change in monitored quantity like loss and validation accuracy is less than 0.001 (which is considered to be no improvement) and 3 consecutively epochs without improvement, the training will stop.

The state-of-the-art model using fractional max-pooling (Graham (2014)) achieves an accuracy of 96.53% on the official test data set. However, the official benchmark for the shallow network with limited depth of 4 layers is 77% (McDonnell and Vladusich (2015)). Therefore, the project will show the accuracy of top 5 model that is found by Q-Search.

Standard procedure: the standard procedure is as same as MNIST dataset.

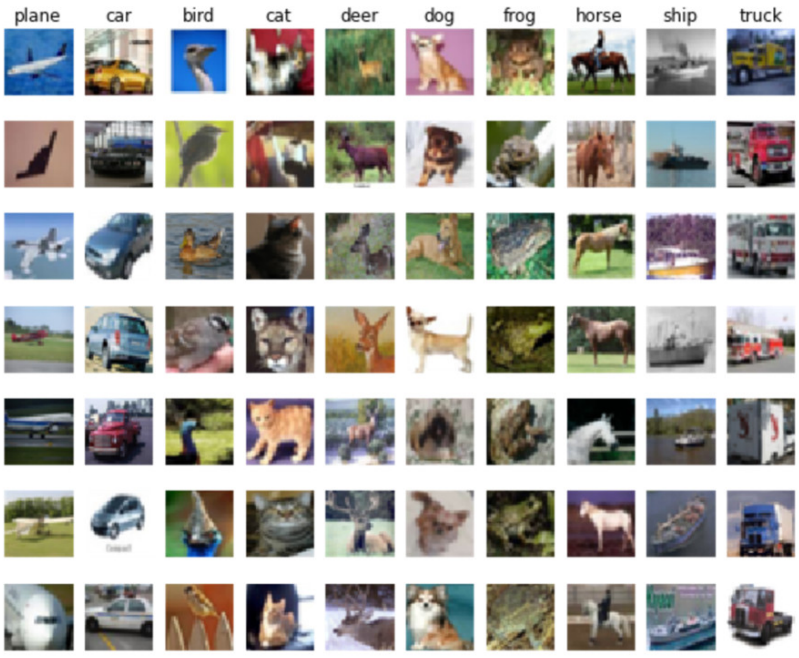

Figure 4.2: Example images from the CIFAR-10 dataset

4.2 Standard Architecture for MNIST and CIFAR-10

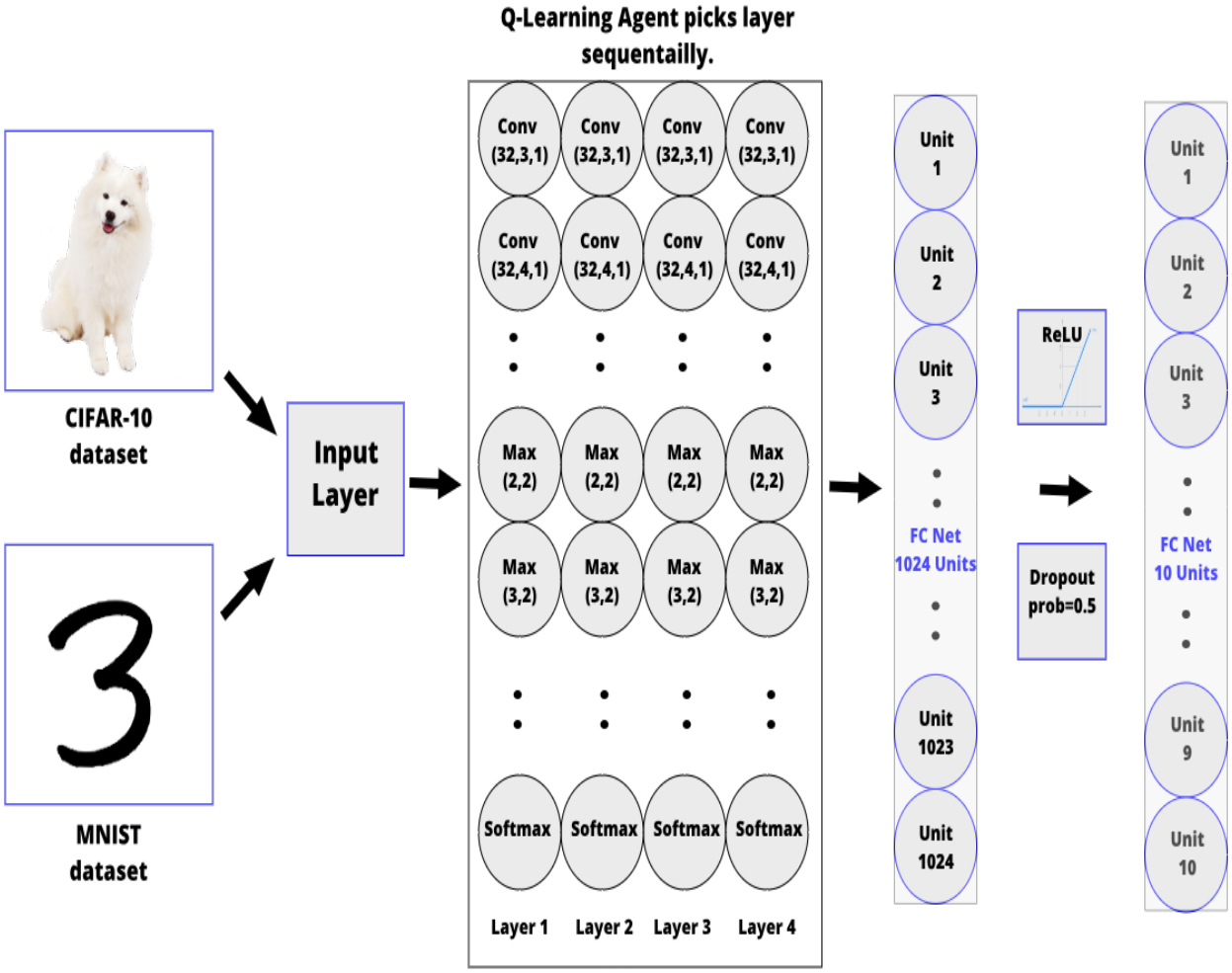

Figure 4.3: The input image is fed into input layer. The Q-learning agent selects CNN topologies according to pre-defined criteria and follows by The fully-connected layer(FC Net) of 1024 units with ReLU activation and dropout with probability of 50 percent. The last layer of overall network is fully-connected layer of 10 unit output classes.

4.3 Experiments and Results

4.3.1 Random Search

4.3.1.1 MNIST

- Total Episode: 2000
- Total number of models trained: 2000
- Training Time per model: 5-10 minutes
- Overall Training Time: 5-6 days

No.	Model	Network Topologies	Loss	Accuracy
1	model-520	Conv(32,3,1),Conv(48,4,1), Conv(64,5,1),Conv(48,5,1)	0.06094	98.13%
2	model-535	Conv(64,3,1),Conv(48,3,1), Conv(32,3,1),Conv(36,5,1)	0.05800	98.13%
3	model-858	Conv(32,3,1),Conv(48,4,1), Conv(64,3,1),Conv(48,5,1)	0.06094	98.13%
4	model-873	Conv(64,3,1),Conv(48,3,1), Conv(32,3,1),Conv(36,5,1)	0.05800	98.13%
5	model-530	Conv(64,5,1),Conv(36,3,1), Conv(48,4,1),Max(2,2)	0.06217	98.09%

Table 4.1: Summary of Top 5 Models found by Random Search on MNIST

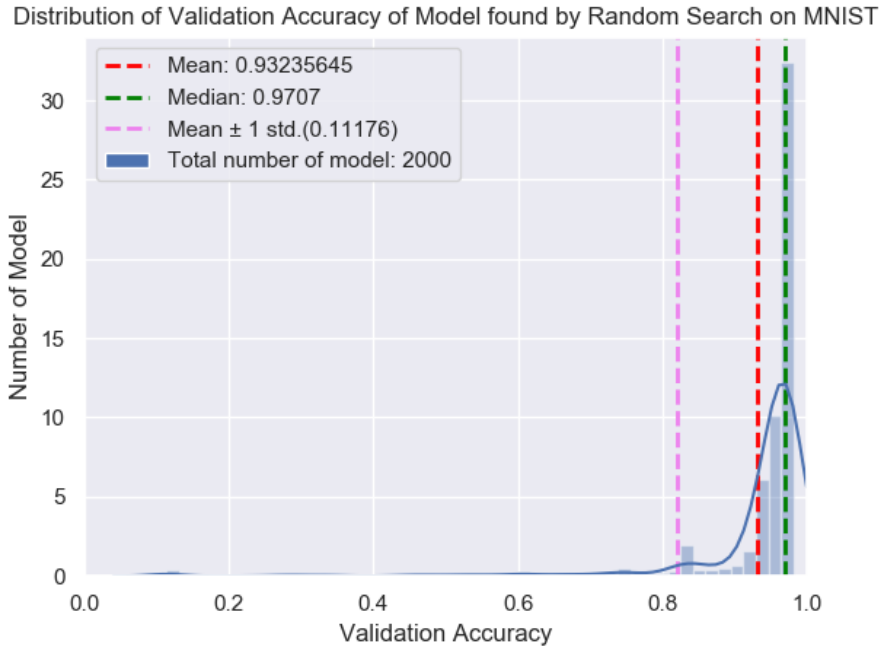

Figure 4.4: Distribution of Validation Accuracy of 2000 models found by Random Search on MNIST dataset

4.3.1.2 CIFAR-10

- Total Episode: 2000
- Total number of models trained: 2000
- Training Time per model: 15-20 minutes
- Overall Training Time: 10-11 days

No.	Model	Network Topologies	Loss	Accuracy
1	model-774	Conv(64,4,1),Conv(64,5,1), Max(5,3),Conv(64,5,1)	0.60828	80.18%
2	model-444	Conv(48,5,1),Max(3,2), Conv(64,5,1),Conv(48,4,1)	0.62098	79.97%
3	model-927	Conv(32,4,1),Max(5,3), Conv(64,4,1),Conv(64,5,1)	0.66346	79.31%
4	model-650	Conv(64,5,1),Max(5,3), Conv(64,4,1),Conv(64,3,1)	0.63902	79.16%
5	model-1806	Conv(64,4,1),Max(1,1), Conv(48,5,1),Conv(32,5,1)	0.65147	79.08%

Table 4.2: Summary of Top 5 Models found by Random Search on CIFAR-10

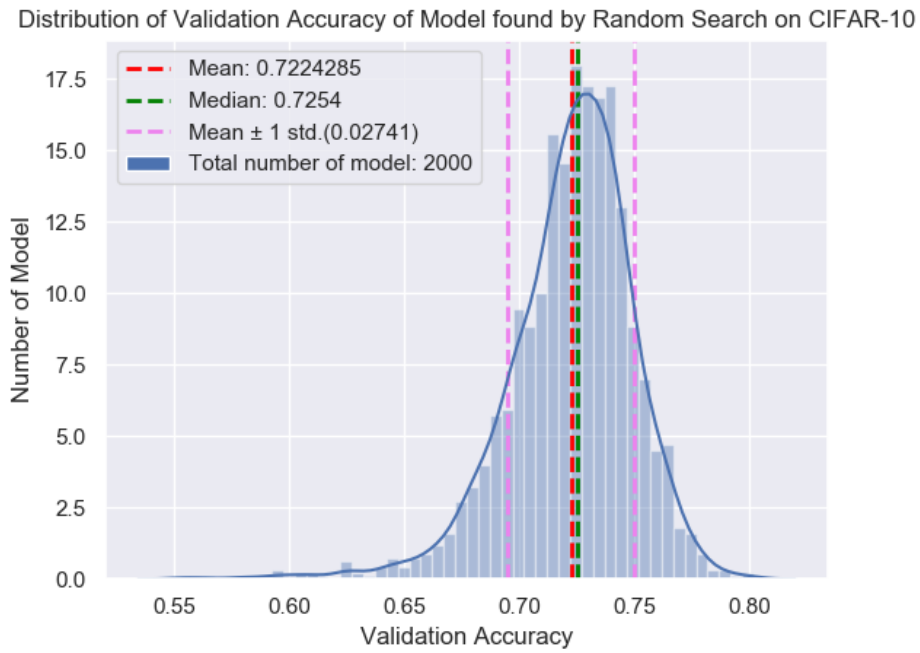

Figure 4.5: Distribution of Validation Accuracy of 2000 models found by Random Search on CIFAR-10 dataset

4.3.2 Layerwise Search

4.3.2.1 MNIST

- Total Episode: 64
- Total number of models trained: 64
- Training Time per model: 5-10 minutes
- Overall Training Time: 8 hours

No.	Model	Network Topologies	Loss	Accuracy
1	model-44	Conv(36,5,1),Conv(48,4,1), Conv(64,5,1),Softmax	0.07997	97.46%
2	model-36	Conv(36,5,1),Conv(48,4,1), Conv(36,3,1),Softmax	0.08287	97.41%
3	model-41	Conv(36,5,1),Conv(48,4,1), Conv(48,5,1),Softmax	0.08569	97.41%
4	model-63	Conv(36,5,1),Conv(48,4,1), Conv(64,5,1),Max(5,3)	0.08547	97.41%
5	model-56	Conv(36,5,1),Conv(48,4,1), Conv(64,5,1),Conv(48,4,1)	0.08361	97.39%

Table 4.3: Summary of Top 5 Models found by Layerwise Search on MNIST

4.3.2.2 CIFAR-10

- Total Episode: 64
- Total number of model trained: 64
- Training Time per model: 15-20 minutes
- Overall Training Time: 1 day

No.	Model	Network Topologies	Loss	Accuracy
1	model-63	Conv(48,5,1),Conv(36,5,1), Conv(32,4,1),Max(5,3)	0.72411	75.72%
2	model-57	Conv(48,5,1),Conv(36,5,1), Conv(32,4,1),Conv(48,5,1)	0.72874	75.67%
3	model-51	Conv(48,5,1),Conv(36,5,1), Conv(32,4,1),Conv(32,5,1)	0.73973	75.08%
4	model-34	Conv(48,5,1),Conv(36,5,1), Conv(32,4,1),Softmax	0.75530	74.91%
5	model-56	Conv(48,5,1),Conv(36,5,1), Conv(32,4,1),Conv(48,4,1)	0.80207	74.56%

Table 4.4: Summary of Top 5 Models found by Layerwise Search on CIFAR-10

4.3.3 Q-Search - MNIST Dataset

4.3.3.1 Experiment 1 on MNIST

- Total Episode: 3500
- Epsilon Table:

Epsilon(ε)	**1.0**	**0.9-0.7**	**0.6-0.1**
Number of model trained	2500	500	500

Table 4.5: ε Schedule of Experiment 1 on MNIST. The learning agent trains the specified number of unique models at each ε.

- Discount rate(γ): 1.0
- Q-Learning rate(α): 0.01
- Number of model randomly selected to update the Q-table from experience replay: 10
- Experience Replay update mode: update after training new model
- Training Time: 5 days

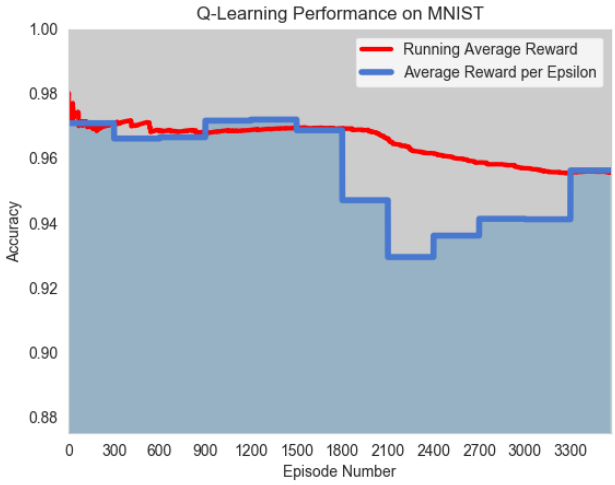

Figure 4.6: Q-Learning Performance on MNIST on Experiment 1

At the beginning of 2500 episodes during $\varepsilon = 1$, the agent explores a search space by using random strategy to select network topologies. However, from episode 1600 up to 2500, the agent randomly picked architectures that do not perform well compared with the early episodes. As the training goes on, when ε starts to decrease, the agent begins to exploit the knowledge from several learned models by focusing on selecting topologies that give good validation accuracy. There is some plateau that the agent is struck in but rather can recover as blue line rises back up again. Nevertheless, the result is still not decisive as blue line still runs up before the training ends. This indicates the episode that is set for an exploitation phase may be too low as the blue line still rises up and does not hit same plateau yet.

Table 4.6: Result of Experiment 1 on MNIST

Total Episode	No. of Model trained at ε = 1	No. of Pre-trained model	Total Model Trained	No. of Model sampling to update Q-table/ Mode	Final Q-Table	Training Time	Average Accuracy of Model during ε = 1
3500	2500	1500	4481	10/ After new model trained	c2 c6 c3 c1	5 days	96.49%

Table 4.7: Summary of Top 5 Models found by Q-learning agent

No.	Model	Network Topologies	Loss	Accuracy	Top 5 models found by Random Search
1	model-3186	c1: Conv. Layer:(32,3,1) c5 : Conv. Layer:(36,4,1) c10 : Conv. Layer:(64,3,1) c1 : Conv. Layer:(32,3,1)	0.07844	97.65%	Model No.: 1.model-520 Topology: c1 c8 c12 c9 Loss: 0.06094 Accuracy: **98.13%**
2	model-2796	c4 : Conv. Layer:(36,3,1) c10 : Conv. Layer:(64,3,1) c8 : Conv. Layer:(48,4,1) m3 : MaxPool Layer:(5,3)	0.08101	97.62%	Model No.: 2.model-535 Topology: c10 c7 c1 c6 Loss: 0.05800 Accuracy: **98.13%**
3	model-3357	c2 : Conv. Layer:(32,4,1) c1 : Conv. Layer:(32,3,1) c9 : Conv. Layer:(48,5,1) m2 : MaxPool Layer:(3,2)	0.07830	97.61%	Model No.: 3.model-858 Topology: c1 c8 c12 c9 Loss: 0.06094 Accuracy: **98.13%**
4	model-4120	c2 : Conv. Layer:(32,4,1) c1 : Conv. Layer:(32,3,1) c12 : Conv. Layer:(64,5,1) c1 : Conv. Layer:(32,3,1)	0.07826	97.60%	Model No.: 4.model-873 Topology: c10 c7 c1 c6 Loss: 0.05800 Accuracy: **98.13%**
5	model-3045	c5 : Conv. Layer:(36,4,1) c11 : Conv. Layer:(64,4,1) c7 : Conv. Layer:(48,3,1) c4 : Conv. Layer:(36,3,1)	0.08240	97.59%	Model No.: 5.model-530 Topology: c12 c4 c8 m1 Loss: 0.06217 Accuracy: **98.09%**

4.3.3.2 Experiment 2 on MNIST

- Total Episode: 3800
- Epsilon Table:

Epsilon(ε)	**1.0**	**0.9-0.7**	**0.6-0.1**
Number of model trained	2000	1000	800

Table 4.8: ε Schedule of Experiment 2 on MNIST. The learning agent trains the specified number of unique models at each ε.

- Discount rate(γ): 1.0
- Q-Learning rate(α): 0.01
- Number of model randomly selected to update the Q-table from experience replay: 0
- Experience Replay update mode: -
- Training Time: 5 days

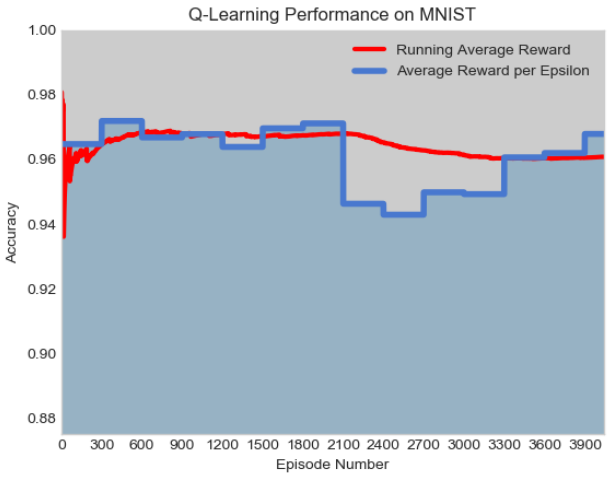

Figure 4.7: Q-Learning Performance on MNIST on Experiment 2

At the beginning of 2000 episodes during $\varepsilon = 1$, the agent uses random strategy to select network topologies. As the training goes on, when ε starts to go down toward 0.1, the agent starts to use exploitation strategy by focusing on selecting topologies that give good validation accuracy. Nevertheless, at about episode 2000, the blue curve collapses down, meaning that the agent selects the network architectures that do not give good results. From the observation, there are many periods that the blue curve is struck into some plateau for several episodes. This might indicate the fact that the agent who does not implement an experience replay technique employs only greedy strategy and is struck into local optima. The agent may find the well-performing architectures in the end of episodes but there is no guarantee that it will achieve that good results every time before struck into local optima forever.

Table 4.9: Result of Experiment 2 on MNIST

Total Episode	No. of Model trained at ε = 1	No. of Pre-trained model	Total Model Trained	No. of Model sampling to update Q-table/ Mode	Final Q-Table	Training Time	Average Accuracy of Model during ε = 1
3800	2000	500	3101	0/ -	c1 c6 c5 m2	5 days	96.89%

Table 4.10: Summary of Top 5 Models found by Q-learning agent

No.	Model	Network Topologies	Loss	Accuracy	Top 5 models found by Random Search
1	model-2996	c5 : Conv. Layer:(36,4,1) c2 : Conv. Layer:(32,4,1) c8 : Conv. Layer:(48,4,1) m2 : MaxPool Layer:(3,2	0.08380	97.66%	Model No.: 1.model-520 Topology: c1 c8 c12 c9 Loss: 0.06094 Accuracy: **98.13%**
2	model-2284	c1 : Conv. Layer:(32,3,1) c4 : Conv. Layer:(36,3,1) c9 : Conv.Layer:(48,5,1) m3 : MaxPool Layer:(5,3)	0.08101	97.62%	Model No.: 2.model-535 Topology: c10 c7 c1 c6 Loss: 0.05800 Accuracy: **98.13%**
3	model-2306	c11 : Conv. Layer:(64,4,1) c3 : Conv. Layer:(32,5,1) c10 : Conv. Layer:(64,3,1) c3 : Conv. Layer:(32,5,1)	0.08001	97.62%	Model No.: 3.model-858 Topology: c1 c8 c12 c9 Loss: 0.06094 Accuracy: **98.13%**
4	model-2589	c1 : Conv. Layer:(32,3,1) c2 : Conv. Layer:(32,4,1) c2 : Conv. Layer:(32,4,1) c8 : Conv. Layer:(48,4,1)	0.07943	97.62%	Model No.: 4.model-873 Topology: c10 c7 c1 c6 Loss: 0.05800 Accuracy: **98.13%**
5	model-2998	c1 : Conv. Layer:(32,3,1) c7 : Conv. Layer:(48,3,1) c10 : Conv. Layer:(64,3,1) c8 : Conv. Layer:(48,4,1)	0.08010	97.60%	Model No.: 5.model-530 Topology: c12 c4 c8 m1 Loss: 0.06217 Accuracy: **98.09%**

4.3.3.3 Experiment 3 on MNIST

- Total Episode: 3800
- Epsilon Table:

Epsilon(ε)	**1.0**	**0.9-0.7**	**0.6-0.1**
Number of model trained	2000	1000	800

Table 4.11: ε Schedule of Experiment 3 on MNIST. The learning agent trains the specified number of unique models at each ε.

- Discount rate(γ): 1.0
- Q-Learning rate(α): 0.01
- Number of model randomly selected to update the Q-table from experience replay: 50
- Experience Replay update mode: update after training new model
- Training Time: 6 days

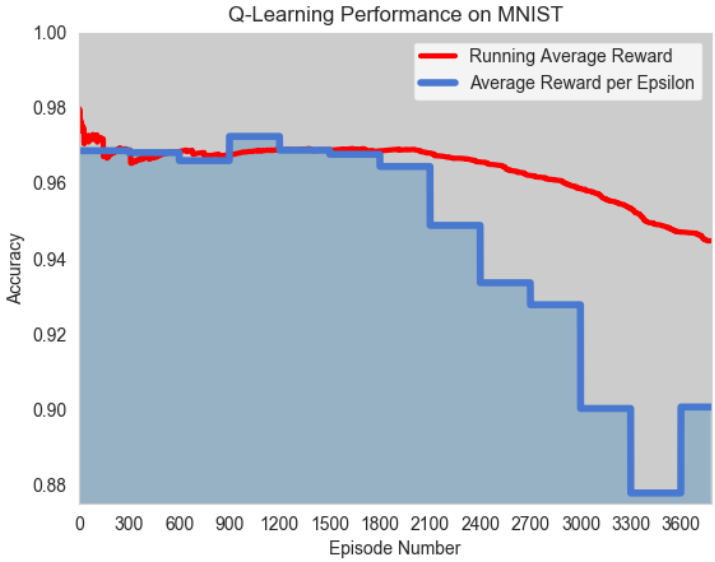

Figure 4.8: Q-Learning Performance on MNIST on Experiment 3

At the beginning of episodes, during exploration phase when ε is equal to 1, the agent uses random strategy to select model topologies. As ε goes down from 1 toward 0.1, every time that the agent finds new model it begins to sample 50 models from the experience replay to update Q-table which is far higher than experiment 1 (5 models). This causes Q-table to be updated too frequently, making the agent exploit topologies randomly until the end of episode as indicated by blue line. Although the agent can found well-performing architectures as shown in the next table, the actions that agent takes are random actions and might get struck in some bad topologies indicated by blue line goes down over time.

Table 4.12: Result of Experiment 3 on MNIST

Total Episode	No. of Model trained at ε = 1	No. of Pre-trained model	Total Model Trained	No. of Model sampling to update Q-table/ Mode	Final Q-Table	Training Time	Average Accuracy of Model during ε = 1
3800	2000	500	3517	50/ After new model trained	c8 c9 c8 c9	6 days	96.90%

Table 4.13: Summary of Top 5 Models found by Q-learning agent

No.	Model	Network Topologies	Loss	Accuracy	Top 5 models found by Random Search
1	model-2074	c1 : Conv. Layer:(32,3,1) c4 : Conv. Layer:(36,3,1) c5 : Conv. Layer:(36,4,1) m1 : MaxPool Layer:(2,2)	0.08085	97.59%	Model No.: 1.model-520 Topology: c1 c8 c12 c9 Loss: 0.06094 Accuracy: **98.13%**
2	model-3389	c4 : Conv.Layer:(36,3,1) c2 : Conv. Layer:(32,4,1) c12 : Conv. Layer:(64,5,1) m1 : MaxPool Layer:(2,2)	0.07729	97.59%	Model No.: 2.model-535 Topology: c10 c7 c1 c6 Loss: 0.05800 Accuracy: **98.13%**
3	model-2534	c7 : Conv. ayer:(48,3,1) c5 : Conv. Layer:(36,4,1) c4 : Conv. Layer:(36,3,1) m2 : MaxPool Layer:(3,2)	0.07848	97.58%	Model No.: 3.model-858 Topology: c1 c8 c12 c9 Loss: 0.06094 Accuracy: **98.13%**
4	model-2080	c4 : Conv. Layer:(36,3,1) c4 : Conv. Layer:(36,3,1) c4 : Conv. Layer:(36,3,1) c3 : Conv. Layer:(32,5,1)	0.08001	97.57%	Model No.: 4.model-873 Topology: c10 c7 c1 c6 Loss: 0.05800 Accuracy: **98.13%**
5	model-2352	c7 : Conv. Layer:(48,3,1) c10 : Conv. Layer:(64,3,1) c10 : Conv. Layer:(64,3,1) c8 : Conv. Layer:(48,4,1)	0.07876	97.57%	Model No.: 5.model-530 Topology: c12 c4 c8 m1 Loss: 0.06217 Accuracy: **98.09%**

4.3.3.4 Experiment 4 on MNIST

- Total Episode: 3800
- Epsilon Table:

Epsilon(ε)	**1.0**	**0.9-0.7**	**0.6-0.1**
Number of model trained	2000	1000	800

Table 4.14: ε Schedule of Experiment 4 on MNIST. The learning agent trains the specified number of unique models at each ε.

- Discount rate(γ): 1.0
- Q-Learning rate(α): 0.01
- Number of model randomly selected to update the Q-table from experience replay: 50
- Experience Replay update mode: update periodically every 100 episodes
- Training Time: 3 days

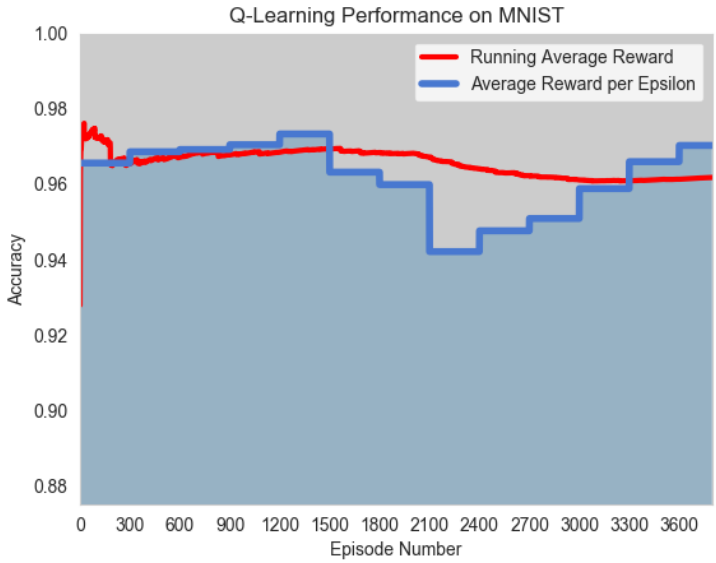

Figure 4.9: Q-Learning Performance on MNIST on Experiment 4

The Figure 4.9 shows the greatest improvement over all previous experiments. At the beginning of episodes up to the middle, there are some drops in performance of the agent as the agent explores topology spaces. However, thank to the periodical Q-table update from experience replay as ε begin to go down toward 0.1, the agent starts to exploit the well-performing topologies greedily leading up to steady improvement as indicated by the blue line gradually rises overtime without any decrease in performance. This is a result of the way that Q-table is updated as there is not much randomness in this experiment compared with experiment 1 and 3. The use of experience replay with periodic update also helps training converge faster compared with the experiment 2 that does not use experience replay technique.

Table 4.15: Result of Experiment 4 on MNIST

Total Episode	No. of Model trained at ε = 1	No. of Pre-trained model	Total Model Trained	No. of Model sampling to update Q-table/ Mode	Final Q-Table	Training Time	Average Accuracy of Model during ε = 1
3800	2000	500	3110	50/ Every 100 episodes	c8 c9 c8 c9	3 days	96.89%

Table 4.16: Summary of Top 5 Models found by Q-learning agent

No.	Model	Network Topologies	Loss	Accuracy	Top 5 models found by Random Search
1	model-2625	c8 : Conv. Layer:(48,4,1) c5 : Conv. Layer:(36,4,1) c3 : Conv. Layer:(32,5,1) c1 : Conv. Layer:(32,3,1)	0.08028	97.67%	Model No.: 1.model-520 Topology: c1 c8 c12 c9 Loss: 0.06094 Accuracy: **98.13%**
2	model-2816	c10 : Conv. Layer:(64,3,1) c7 : Conv. Layer:(48,3,1) c10 : Conv. Layer:(64,3,1) c1 : Conv. Layer:(32,3,1)	0.07784	97.60%	Model No.: 2.model-535 Topology: c10 c7 c1 c6 Loss: 0.05800 Accuracy: **98.13%**
3	model-2547	c2 : Conv. Layer:(32,4,1) c1 : Conv. Layer:(32,3,1) c5 : Conv. Layer:(36,4,1) c3 : Conv. Layer:(32,5,1)	0.08090	97.56%	Model No.: 3.model-858 Topology: c1 c8 c12 c9 Loss: 0.06094 Accuracy: **98.13%**
4	model-2612	c5 : Conv. Layer:(36,4,1) c8 : Conv. Layer:(48,4,1) c5 : Conv. Layer:(36,4,1) c8 : Conv. Layer:(48,4,1)	0.07732	97.55%	Model No.: 4.model-873 Topology: c10 c7 c1 c6 Loss: 0.05800 Accuracy: **98.13%**
5	model-2760	c5 : Conv. Layer:(36,4,1) c3 : Conv. Layer:(32,5,1) c5 : Conv. Layer:(36,4,1) c1 : Conv. Layer:(32,3,1)	0.08051	97.55%	Model No.: 5.model-530 Topology: c12 c4 c8 m1 Loss: 0.06217 Accuracy: **98.09%**

4.3.3.5 Result of Best Topologies in every experiment found by Q-Search, Random Search and Layerwise Search on MNIST

Table 4.17: Result of Best Topologies in every experiment found by Q-Learning agent on MNIST dataset

Experiment No./ No. of pre-trained Model	Total Episode	Total Model Trained	No. of Model sampled to update Q-table/ Mode	Best Topologies Q-Agent have found	Training Time	Best Loss	Best Accuracy
1/ 1500	3500	4481	10/ Update after getting new model	Conv(32,3,1), Conv(36,4,1), Conv(64,3,1), Conv(32,3,1)	5 days	0.0785	97.65%
2/ 500	3800	3101	0/ -	Conv(36,4,1), Conv(32,4,1), Conv(48,4,1), Max(5,3)	5 days	0.0837	97.66%
3/ 500	3800	3517	50/ Update after getting new model	Conv(32,3,1), Conv(36,3,1), Conv(36,4,1), Max(2,2)	6 days	0.0809	97.59%
4/ 500	3800	3110	50/ Update every 100 episodes	Conv(48,4,1), Conv(36,4,1), Conv(32,4,1), Conv(32,3,1)	**3 days**	0.0803	**97.67%**

Table 4.18: Result of Best Topologies found by Random Search and Layerwise Search methods on MNIST dataset

Method	Total Episode	Total Model Trained	Training Time per Model	Best Topologies Q-Agent have found	Training Time	Best Loss	Best Accuracy
Random Search	2000	2000	5-10 mins	Conv(32,3,1), Conv(48,4,1), Conv(64,5,1), Conv(48,5,1)	5-6 days	0.0609	**98.13%**
Layerwise Search	64	64	5-10 mins	Conv(36,5,1), Conv(48,4,1), Conv(64,5,1), Softmax	8 hours	0.0800	97.46%

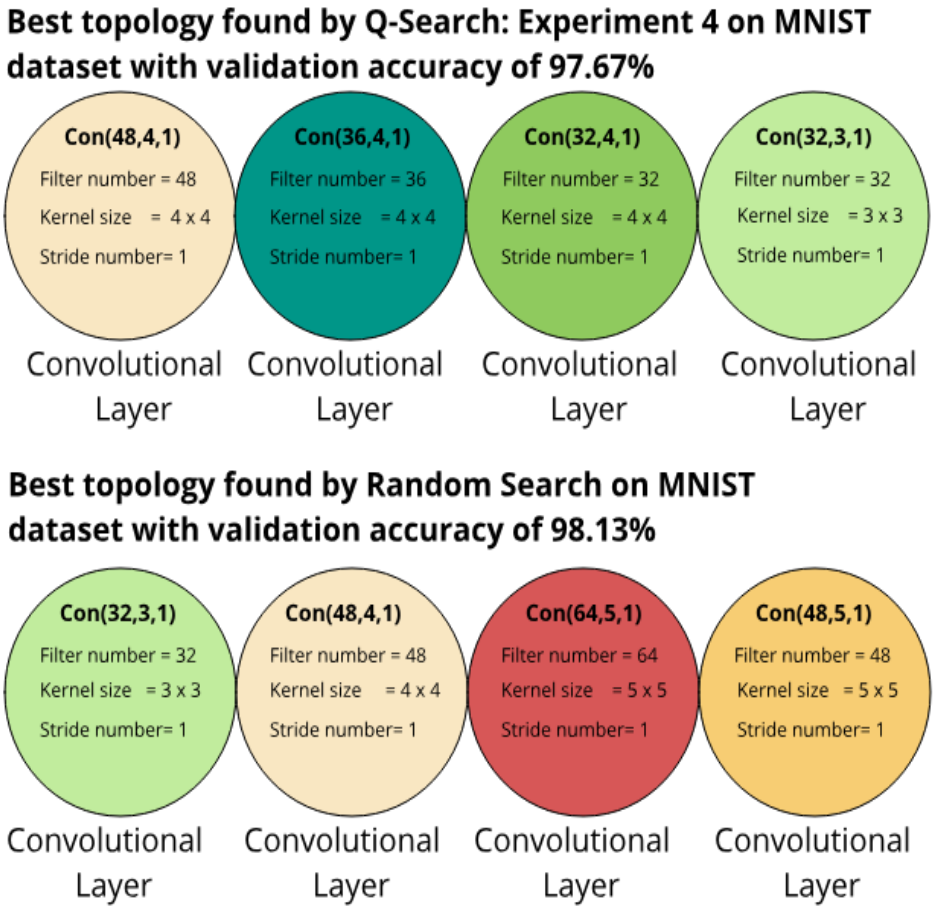

Figure 4.10: The best topology discovered by Q-Search of Experiment 4 and Random Search on MNIST dataset

4.3.3.6 Summary of Results from every Experiment between Q-Search, Random Search and Layerwise Search on MNIST dataset

The table 4.17 shows the best architecture in every experiments discovered by the agent according to the highest validation accuracy with different setups on CIFAR-10 dataset: total number of episode, number of model to be sampled from experience replay to update Q-table and the updating frequency (updating mode) while the table 4.18 shows the best architecture that has been found according to validation accuracy by random search method with the total of 2000 models with 5-10 minute for training time per each model and by layerwise search method with the total of 64 models with the training time per each model same as the random search. The pre-trained model is the model that is put into experience replay before starting running overall training in order to increase more model options for experience replay sampling purpose and this does not affect overall training in both training time and ability of the agent to find good topologies.

The experiment 1 has 3500 episodes in total and uses experience replay technique to update Q-table by sampling 10 models every time the agent has finished training new model while the experiment 2 has the total of 3800 episodes without experience replay technique. However, the training time for experiment 1 and 2 are equal while the validation accuracy is slightly different. This indicates that the experience replay technique used in experiment 1 not only does not help the agent to find better topologies but only makes training time longer since the experiment 1 has lower number of episode but the training time is as same as the experiment 2 with higher number of episodes. This conclusion can be confirmed by the comparison between experiment 2 and experiment 3 (with the experience replay technique similar to experiment 1) as both of them have same number of episodes but the training time in the experiment 3 takes longer without any advantage of validation accuracy over topologies discovered by the agent.

In the experiment 1 and 3, the Q-learning agent performs Q-table update every time the agent finishes training new model by randomly sampling 10 and 50 models from experience replay respectively while, in the experiment 4, Q-table update iterates over every 100 episode. In the experiment 3 and 4, even if the total numbers of episodes are equal (3800 episodes), the validation accuracy of topologies found the agent in experience 4 is higher and the training time of experiment 4 is considerably lower than the experiment 3. The reason that the experiment 4 takes less time to finish the training is that in the experiment 4 the total of models trained are 3110 compared to 3517 in the experiment 3.

In sum, the frequency of Q-table update by randomly sampling models from experience replay plays an important role in helping speed up the convergence of the overall training; for example, given all things are equal and even if the experiment 4 has the highest number of episodes, it can still find the well-performed topologies

that beat all other experiments in term of both training time and validation accuracy. The training time of experiment 4 is lowest which is even lower than experiment 1 with higher number of 3500 episodes.

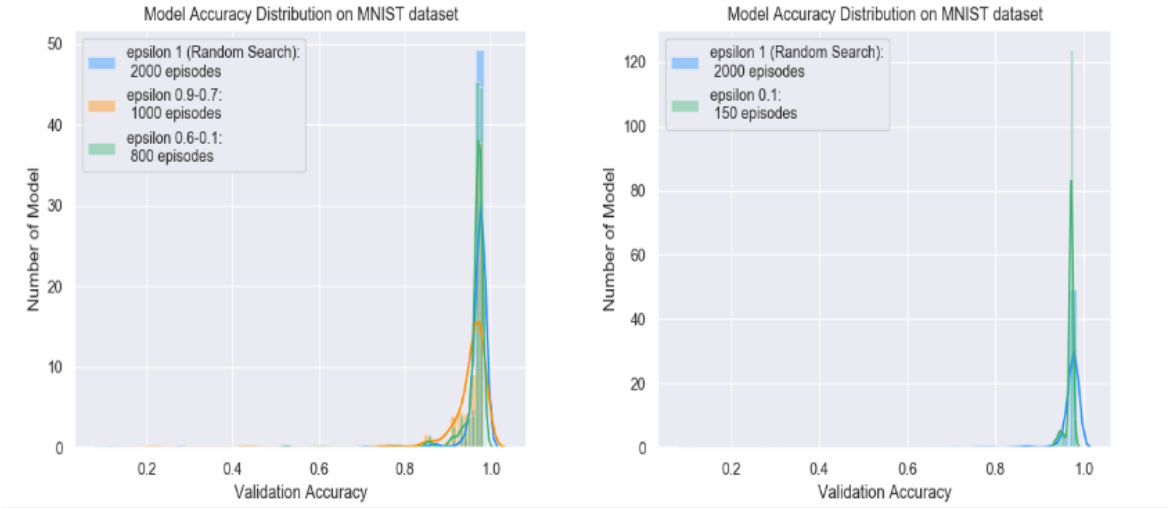

Figure 4.11: Distribution of Validation Accuracy over all episodes with different values of epsilon by Q-Search on MNIST dataset

Figure 4.11 shows the distribution of validation accuracy of topologies discovered by Q-learning agent over different of episodes which are assigned according to different epsilons. The higher the epsilon the more the agent chooses to explore more spaces while the lower the epsilon the more the agent chooses to exploit the good architectures (greedy strategy). For epsilon 1, the agent will perform random action which is similar to random search. For epsilon 0.1 with 150 episodes, the distribution of validation accuracy is directed toward the right side of the whole distribution which indicates that the agent chooses to exploit the well-performing topologies.

From the figure (4.18), the distribution in both random search and Q-learning at the very end of episodes does not show much difference as both have well-performing models with mean clustered around 96-97%. The conclusion is that for the small simple dataset like MNIST the simple topologies can achieve high accuracy; therefore, the simple technique like random search can comfortably compete with more complicated technique like Q-learning since the search space is very small.

4.3.4 Q-Search : CIFAR-10 Dataset

4.3.4.1 Experiment 1 on CIFAR-10

- Total Episode: 2500
- Epsilon Table:

Epsilon(ε)	**1.0**	**0.9-0.7**	**0.6-0.1**
Number of model trained	1500	500	500

Table 4.19: ε Schedule of Experiment 1 on CIFAR-10. The learning agent trains the specified number of unique models at each ε.

- Discount rate(γ): 1.0
- Q-Learning rate(α): 0.01
- Number of model randomly selected to update the Q-table from experience replay: 5
- Experience Replay update mode: update after finishing training new model
- Training Time: 8 days

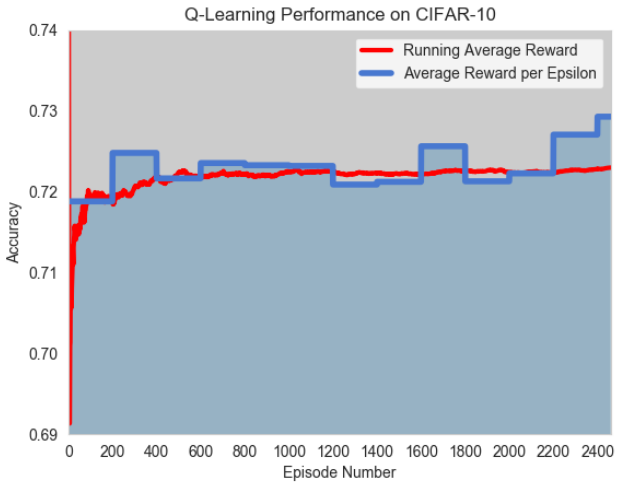

Figure 4.12: Q-Learning Performance on CIFAR-10 on Experiment 1

At the beginning of 1500 episodes during $\varepsilon = 1$, the agent uses random strategy to select network topologies. As the training goes on, when ε begins to go down to less than 0.5, the agent begins to use exploitation strategy by focusing on selecting topologies that give good validation accuracy. From episodes 2200 up to the end, the blue line gradually goes up. However, there is some improvement to be made as blue line still runs up. This indicates the episode that is set for an exploitation phase is too low as the blue line still does not hit some plateau yet.

Table 4.20: Result of Experiment 1 on CIFAR-10

Total Episode	No. of Model trained at ε = 1	No. of Pre-trained model	Total Model Trained	No. of Model sampling to update Q-table/ Mode	Final Q-Table	Training Time	Average Accuracy of Model during ε = 1
2500	1500	500	2303	5/ After new model trained	c2 c6 c11 c9	8 days	72.22%

Table 4.21: Summary of Top 5 Models found by Q-learning agent

No.	Model	Network Topologies	Loss	Accuracy	Top 5 models found by Random Search
1	model-2332	c12 : Conv. Layer:(64,5,1) m2 : MaxPool Layer:(3,2) c11 : Conv. Layer:(64,4,1) c11 : Conv. Layer:(64,4,1)	0.67386	79.12%	Model No.: 1.model-774 Topology: c11 c12 m3 c12 Loss: 0.60828 Accuracy: **80.18%**
2	model-2048	c11 : Conv. Layer:(64,4,1) m1 : MaxPool Layer:(2,2) c9 : Conv. Layer:(48,5,1) c3 : Conv. Layer:(32,5,1)	0.65148	79.08%	Model No.: 2.model-444 Topology: c9 m2 c12 c8 Loss: 0.62098 Accuracy: **79.97%**
3	model-2141	c11 : Conv. Layer:(64,4,1) m3 : MaxPool Layer:(5,3) c11 : Conv. Layer:(64,4,1) c12 : Conv. Layer:(64,5,1)	0.65148	78.84%	Model No.: 3.model-927 Topology: c2 m3 c11 c12 Loss: 0.66346 Accuracy: **79.31%**
4	model-2225	c12 : Conv. Layer:(64,5,1) c6 : Conv. Layer:(36,5,1) m3 : MaxPool Layer:(5,3) c9 : Conv. Layer:(48,5,1)	0.64787	78.57%	Model No.: 4.model-650 Topology: c12 m3 c11 c10 Loss: 0.63902 Accuracy: **79.16%**
5	model-1568	c11 : Conv. Layer:(64,4,1) c9 : Conv. Layer:(48,5,1) m3 : MaxPool Layer:(5,3) c9 : Conv. Layer:(48,5,1)	0.64622	77.82%	Model No.: 5.model-1806 Topology: c11 m1 c9 c3 Loss: 0.65147 Accuracy: **79.08%**

4.3.4.2 Experiment 2 on CIFAR-10

- Total Episode: 3300
- Epsilon Table:

Epsilon(ε)	**1.0**	**0.9-0.7**	**0.6-0.1**
Number of model trained	2000	800	500

Table 4.22: ε Schedule of Experiment 2 on CIFAR-10. The learning agent trains the specified number of unique models at each ε.

- Discount rate(γ): 1.0
- Q-Learning rate(α): 0.01
- Number of model randomly selected to update the Q-table from experience replay: 5
- Experience Replay update mode: update after training new model
- Training Time: 10 days

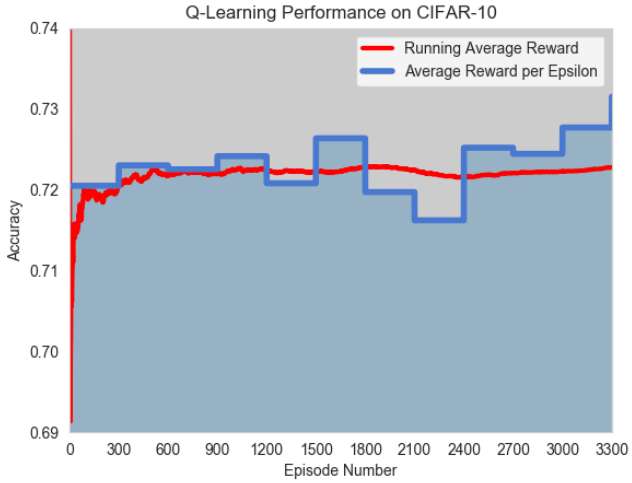

Figure 4.13: Q-Learning Performance on CIFAR-10 on Experiment 2

At the beginning of 2000 episodes during $\varepsilon = 1$, the agent uses random strategy to select network topologies. After some period of episode, the agent starts to exploit the knowledge of the trained models by focusing on selecting topologies that give good validation accuracy. From episodes 2500 up to the end, the blue line goes up; however, during some phases when ε starts decreasing from 1, there are some periods that blue line starts to go down and rise up back again. This may be the result of the way that the agent use experience replay as it finishes training new model it samples 5 models to update Q-table. This might induce too much stochasticity to overall training as the agent trains several new models. Another problem is that the blue line seems not to be stabilized yet. This can be solved by extending an exploitation phase by increasing number of episodes at the end when ε approaches 0.1.

Table 4.23: Result of Experiment 2 on CIFAR-10

Total Episode	No. of Model trained at ε = 1	No. of Pre-trained model	Total Model Trained	No. of Model sampling to update Q-table/ Mode	Final Q-Table	Training Time	Average Accuracy of Model during ε = 1
3300	2000	500	2979	5/ After training new model	c2 c6 c11 c6	10 days	72.26%

Table 4.24: Summary of Top 5 Models found by Q-learning agent

No.	Model	Network Topologies	Loss	Accuracy	Top 5 models found by Random Search
1	model-2556	c11 : Conv. Layer:(64,4,1) m3 : MaxPool Layer:(5,3) c11 : Conv. Layer:(64,4,1) c11 : Conv. Layer:(64,4,1)	0.65491	80.00%	Model No.: 1.model-774 Topology: c11 c12 m3 c12 Loss: 0.60828 Accuracy: **80.18%**
2	model-2547	c2 : Conv. Layer:(32,4,1) c12 : Conv. Layer:(64,5,1) m3 : MaxPool Layer:(5,3) c12 : Conv. Layer:(64,5,1)	0.64978	78.62%	Model No.: 2.model-444 Topology: c9 m2 c12 c8 Loss: 0.62098 Accuracy: **79.97%**
3	model-2678	c2 : Conv. Layer:(32,4,1) m3 : MaxPool Layer:(5,3) c11 : Conv. Layer:(64,4,1) c11 : Conv. Layer:(64,4,1)	0.66416	78.22%	Model No.: 3.model-927 Topology: c2 m3 c11 c12 Loss: 0.66346 Accuracy: **79.31%**
4	model-2415	c2 : Conv. Layer:(32,4,1) m1 : MaxPool Layer:(2,2) c12 : Conv. Layer:(64,5,1) c11 : Conv. Layer:(64,4,1)	0.66621	78.21%	Model No.: 4.model-650 Topology: c12 m3 c11 c10 Loss: 0.63902 Accuracy: **79.16%**
5	model-2289	c5 : Conv. Layer:(36,4,1) m1 : MaxPool Layer:(2,2) c8 : Conv. Layer:(48,4,1) c9 : Conv. Layer:(48,5,1)	0.65306	78.16%	Model No.: 5.model-1806 Topology: c11 m1 c9 c3 Loss: 0.65147 Accuracy: **79.08%**

4.3.4.3 Experiment 3 on CIFAR-10

- Total Episode: 3800
- Epsilon Table:

Epsilon(ε)	**1.0**	**0.9-0.7**	**0.6-0.1**
Number of model trained	2000	1000	800

Table 4.25: ε Schedule of Experiment 3 on CIFAR-10. The learning agent trains the specified number of unique models at each ε.

- Discount rate(γ): 1.0
- Q-Learning rate(α): 0.01
- Number of model randomly selected to update the Q-table from experience replay: 5
- Experience Replay update mode: update after training new model
- Training Time: 11 days

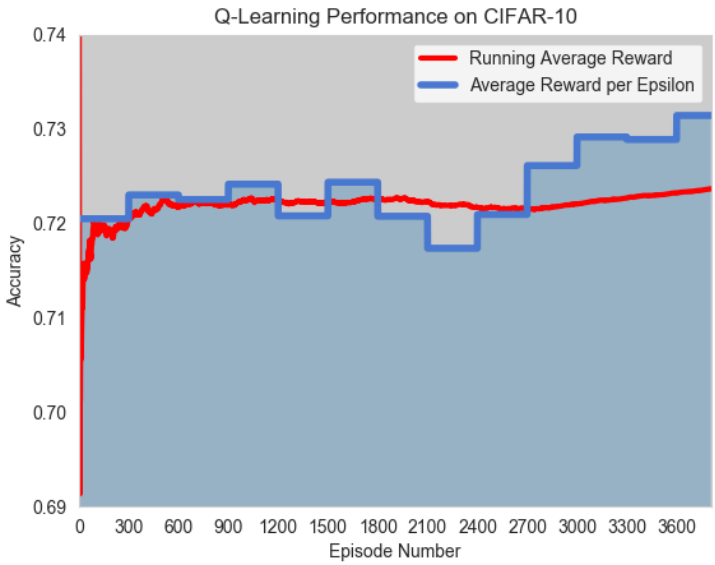

Figure 4.14: Q-Learning Performance on CIFAR-10 on Experiment 3

At the beginning of 2000 episodes during $\varepsilon = 1$, the agent uses random strategy to select network topologies. As the training goes on, the agent begins to use exploitation strategy by focusing on selecting topologies that give good validation accuracy. The Figure 4.14 shows that the same setup that is used in experiment 2 can achieve good results by increasing number of episodes for both exploration and exploitation phases as blue and red line begins to gradually rise up.

Table 4.26: Result of Experiment 3 on CIFAR-10

Total Episode	No. of Model trained at ε = 1	No. of Pre-trained model	Total Model Trained	No. of Model sampling to update Q-table/ Mode	Final Q-Table	Training Time	Average Accuracy of Model during ε = 1
3800	2000	500	3407	5/ After training new model	c7 c6 c11 c9	11 days	72.24%

Table 4.27: Summary of Top 5 Models found by Q-learning agent

No.	Model	Network Topologies	Loss	Accuracy	Top 5 models found by Random Search
1	model-3228	c10 : Conv. Layer:(64,3,1) m3 : MaxPool Layer:(5,3) c8 : Conv. Layer:(48,4,1) c9 : Conv. Layer:(48,5,1)	0.66015	79.41%	Model No.: 1.model-774 Topology: c11 c12 m3 c12 Loss: 0.60828 Accuracy: **80.18%**
2	model-2988	c10 : Conv. Layer:(64,3,1) m1 : MaxPool Layer:(2,2) c12 : Conv. Layer:(64,5,1) c9 : Conv. Layer:(48,5,1)	0.66507	79.11%	Model No.: 2.model-444 Topology: c9 m2 c12 c8 Loss: 0.62098 Accuracy: **79.97%**
3	model-2273	c12 : Conv. Layer:(64,5,1) m1 : MaxPool Layer:(2,2) c12 : Conv. Layer:(64,5,1) c8 : Conv. Layer:(48,4,1)	0.65573	78.74%	Model No.: 3.model-927 Topology: c2 m3 c11 c12 Loss: 0.66346 Accuracy: **79.31%**
4	model-2855	c12 : Conv. Layer:(64,5,1) m1 : MaxPool Layer:(2,2) c9 : Conv. Layer:(48,5,1) c9 : Conv. Layer:(48,5,1)	0.66427	78.62%	Model No.: 4.model-650 Topology: c12 m3 c11 c10 Loss: 0.63902 Accuracy: **79.16%**
5	model-2318	c10 : Conv. Layer:(64,3,1) m3 : MaxPool Layer:(5,3) c5 : Conv. Layer:(36,4,1) c9 : Conv. Layer:(48,5,1)	0.64440	78.55%	Model No.: 5.model-1806 Topology: c11 m1 c9 c3 Loss: 0.65147 Accuracy: **79.08%**

4.3.4.4 Experiment 4 on CIFAR-10

- Total Episode: 3600
- Epsilon Table:

Epsilon(ε)	**1.0**	**0.9-0.7**	**0.6-0.1**
Number of model trained	1800	1000	800

Table 4.28: ε Schedule of Experiment 4 on CIFAR-10. The learning agent trains the specified number of unique models at each ε.

- Discount rate(γ): 1.0
- Q-Learning rate(α): 0.01
- Number of model randomly selected to update the Q-table from experience replay: 0
- Experience Replay update mode: -
- Training Time: 11 days

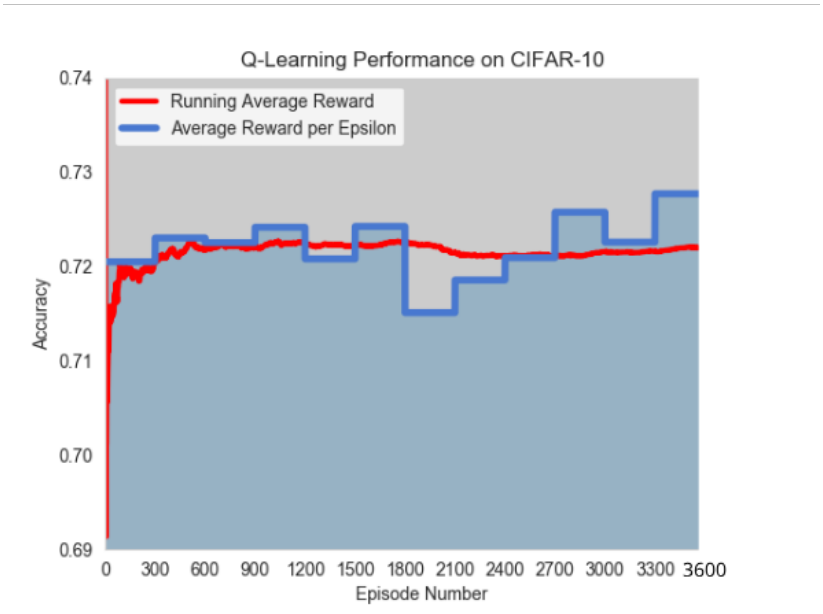

Figure 4.15: Q-Learning Performance on CIFAR-10 on Experiment 4

In this experiment, the agent does not use the experience replay technique. At the very beginning of episode, the agent explores topology space from episode 0 to 1800 at $\varepsilon = 1$ and starts to use exploitation strategy afterward when the epsilon slowly goes down toward 0.1. However, from blue curve there are some drops between episodes from episode 3000 even if the agent starts to exploit and find good topologies, this indicates the training unstability as agent follows greedy strategy and ends up struck into local optima.

Table 4.29: Result of Experiment 4 on CIFAR-10

Total Episode	No. of Model trained at ε = 1	No. of Pre-trained model	Total Model Trained	No. of Model sampling to update Q-table/ Mode	Final Q-Table	Training Time	Average Accuracy of Model during ε = 1
3600	1800	500	3304	0/ -	c4 m2 c11 c9	11 days	72.25%

Table 4.30: Summary of Top 5 Models found by Q-learning agent

No.	Model	Network Topologies	Loss	Accuracy	Top 5 models found by Random Search
1	model-3027	c11 : Conv. Layer:(64,4,1) m2 : MaxPool Layer:(3,2) c11 : Conv. Layer:(64,4,1) c9 : Conv. Layer:(48,5,1)	0.62262	80.09%	Model No.: 1.model-774 Topology: c11 c12 m3 c12 Loss: 0.60828 Accuracy: **80.18%**
2	model-2670	c12 : Conv. Layer:(64,5,1) m3 : MaxPool Layer:(5,3) c11 : Conv. Layer:(64,4,1) c12 : Conv. Layer:(64,5,1)	0.62147	79.60%	Model No.: 2.model-444 Topology: c9 m2 c12 c8 Loss: 0.62098 Accuracy: **79.97%**
3	model-2379	c1 : Conv. Layer:(32,3,1) c12 : Conv. Layer:(64,5,1) m3 : MaxPool Layer:(5,3) c9 : Conv. Layer:(48,5,1)	0.61925	**79.57%**	Model No.: 3.model-927 Topology: c2 m3 c11 c12 Loss: 0.66346 Accuracy: 79.31%
4	model-3055	c7 : Conv. Layer:(48,3,1) m2 : MaxPool Layer:(3,2) c11 : Conv. Layer:(64,4,1) c9 : Conv. Layer:(48,5,1)	0.65129	**79.31%**	Model No.: 4.model-650 Topology: c12 m3 c11 c10 Loss: 0.63902 Accuracy: 79.16%
5	model-2694	c9 : Conv. Layer:(48,5,1) m3 : MaxPool Layer:(5,3) c10 : Conv. Layer:(64,3,1) c8 : Conv. Layer:(48,4,1)	0.35289	**79.19%**	Model No.: 5.model-1806 Topology: c11 m1 c9 c3 Loss: 0.65147 Accuracy: 79.08%

4.3.4.5 Experiment 5 on CIFAR-10

- Total Episode: 3800
- Epsilon Table:

Epsilon(ε)	**1.0**	**0.9-0.7**	**0.6-0.1**
Number of model trained	2000	1000	800

Table 4.31: ε Schedule of Experiment 5 on CIFAR-10. The learning agent trains the specified number of unique models at each ε.

- Discount rate(γ): 1.0
- Q-Learning rate(α): 0.01
- Number of model randomly selected to update the Q-table from experience replay: 50
- Experience Replay update mode: update periodically every 100 episodes
- Training Time: 6 days

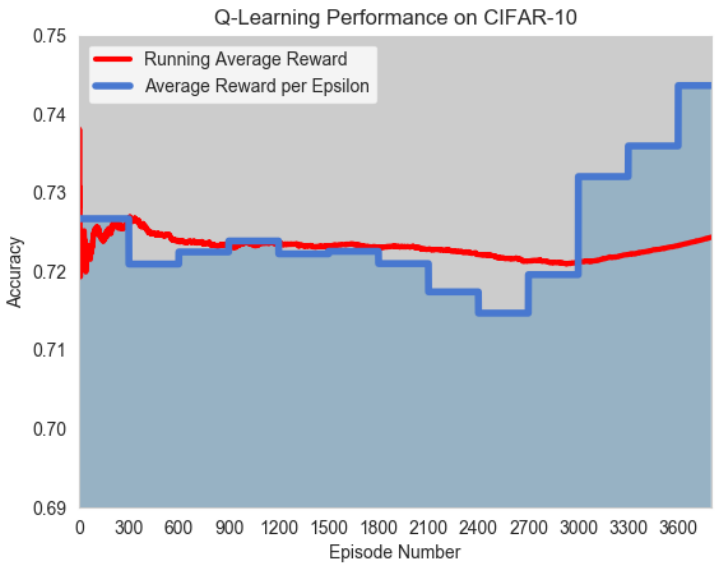

Figure 4.16: Q-Learning Performance on CIFAR-10 on Experiment 5

The Figure 4.16 shows the rising and converging blue line which is the greatest improvement over all previous experiments. At the beginning of episodes, the agent explores topology spaces from episode 0 to about 2800 from ε to 0.7 and after ε is annealed down toward 0.1, the agent starts to exploit the well-performing topologies greedily leading up to huge improvement as indicated by the blue line dramatically increases overtime without any drop in performance thank to the periodical Q-table update from experience replay which is different than all other experiments. This update mode does not induce too much stochasticity compared with experiment 1,2, and 3 and does help the overall training converge faster compared with the experiment 4 that does not use experience replay technique.

Table 4.32: Result of Experiment 5 on CIFAR-10

Total Episode	No. of Model trained at ε = 1	No. of Pre-trained model	Total Model Trained	No. Model sampled to update Q-table/ Mode	Final Q-Table	Training Time	Average Accuracy of Model during ε = 1
3800	2000	500	3096	50/ Every 100 episodes	c12 c6 c11 c11	6 days	72.26%

Table 4.33: Summary of Top 5 Models found by Q-learning agent

No.	Model	Network Topologies	Loss	Accuracy	Top 5 models found by Random Search
1	model-2467	c11 : Conv. Layer:(64,4,1) m3 : MaxPool Layer:(5,3) c9 : Conv. Layer:(48,5,1) c12 : Conv. Layer:(64,5,1)	0.65138	80.17%	Model No.: 1.model-774 Topology: c11 c12 m3 c12 Loss: 0.60828 Accuracy: **80.18%**
2	model-2563	c10 : Conv. Layer:(64,3,1) m3 : MaxPool Layer:(5,3) c12 : Conv. Layer:(64,5,1) c12 : Conv. Layer:(64,5,1)	0.623005	79.67%	Model No.: 2.model-444 Topology: c9 m2 c12 c8 Loss: 0.62098 Accuracy: **79.97%**
3	model-2965	c12 : Conv. Layer:(64,5,1) m1 : MaxPool Layer:(2,2) c11 : Conv. Layer:(64,4,1) c11 : Conv. Layer:(64,4,1)	0.65260	**79.66%**	Model No.: 3.model-927 Topology: c2 m3 c11 c12 Loss: 0.66346 Accuracy: 79.31%
4	model-2594	c12 : Conv. Layer:(64,5,1) m3 : MaxPool Layer:(5,3) c11 : Conv. Layer:(64,4,1) c11 : Conv. Layer:(64,4,1)	0.66937	**79.37%**	Model No.: 4.model-650 Topology: c12 m3 c11 c10 Loss: 0.63902 Accuracy: 79.16%
5	model-3014	c10 : Conv. Layer:(64,3,1) m1 : MaxPool Layer:(2,2) c12 : Conv. Layer:(64,5,1) c3 : Conv. Layer:(32,5,1)	0.65445	78.96%	Model No.: 5.model-1806 Topology: c11 m1 c9 c3 Loss: 0.65147 Accuracy: **79.08%**

4.3.4.6 Result of Best Topologies in every experiment found by Q-Search, Random Search and Layerwise Search method on CIFAR-10 dataset

Table 4.34: Result of Best Topologies in every experiment found by Q-Learning agent on CIFAR-10 dataset

No.	Total Episode	Total Model Trained	No. Model sampled/ Mode	Best Topologies Q-Agent has found	Training Time	Best Loss	Best Accuracy
1	2500	2303	5/ Update after getting new model	Conv(64,5,1),Max(5,3), Conv(64,4,1),Conv(64,4,1)	8 days	0.674	79.12%
2	3300	2979	5/ Update after getting new model	Conv(64,4,1), Max(5,3), Conv(64,4,1),Conv(64,4,1)	10 days	0.655	80.00%
3	3800	3407	5/ Update after getting new model	Conv(64,3,1), Max(5,3), Conv(48,4,1),Conv(48,5,1)	11 days	0.660	79.41%
4	3600	3204	0/ -	Conv(64,4,1),Max(3,2), Conv(64,4,1), Conv(48,5,1)	11 days	0.623	80.09%
5	3800	3096	50/ Update every 100 episodes	Conv(64,4,1),Max(5,3), Conv(48,5,1),Conv(64,5,1)	**6 days**	0.653	**80.17%**
Topology handpicked by research "Enhanced Image Classification With a Fast-Learning Shallow Convolutional Neural Network" by McDonnell and Vladusich (2015)						-	**77.00%**

Table 4.35: Result of Best Topologies found by Random Search and Layerwise Search methods on CIFAR-10 dataset

Method	Total Episode	Total Model Trained	Training Time per Model	Best Topologies Q-Agent found	Training Time	Best Loss	Best Accuracy
Random Search	2000	2000	15-20 mins	Conv(64,4,1), Conv(64,5,1), Max(5,3), Conv(64,5,1)	10-11 days	0.6083	**80.18%**
Layerwise Search	64	64	15-20 mins	Conv(48,5,1), Conv(36,5,1), Conv(32,4,1), Max(5,3)	1 day	0.7241	75.72%

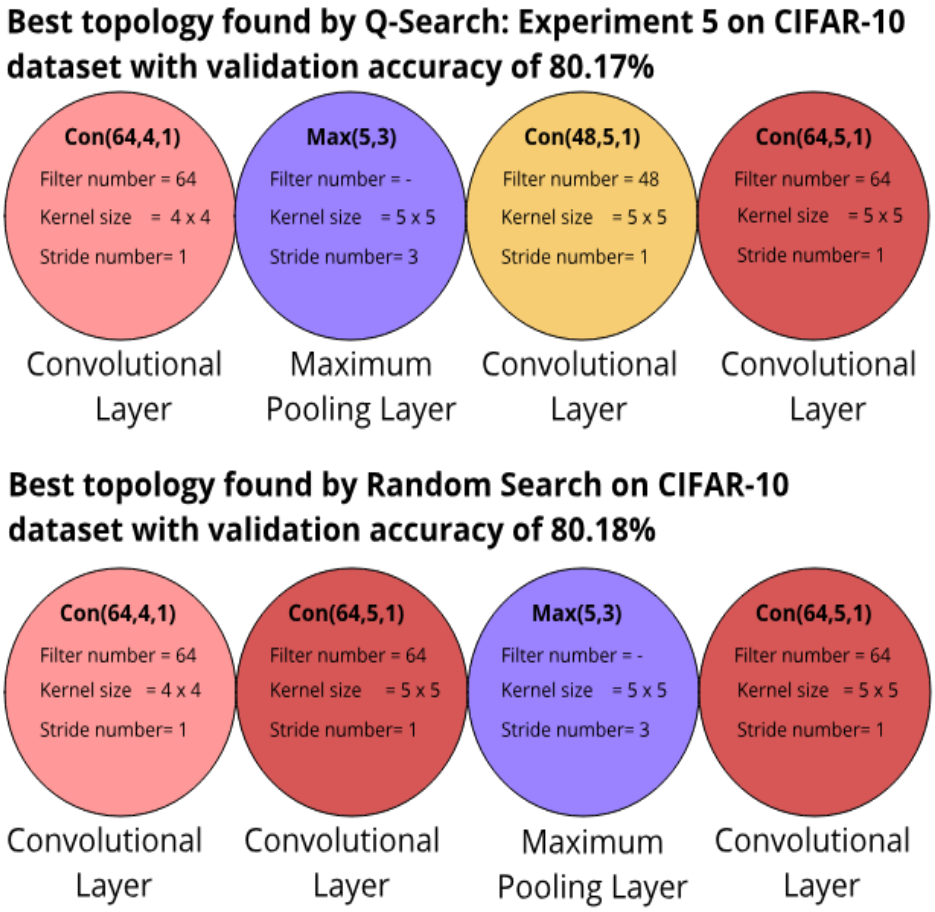

Figure 4.17: The best topology discovered by Q-Search of Experiment 5 and Random Search on CIFAR-10 dataset

4.3.4.7 Summary of Results from every Experiment between Q-Search, Random Search and Layerwise Search method on CIFAR-10 dataset

The table 4.34 shows the best architecture in every experiments discovered by the agent according to the highest validation accuracy with different setups on CIFAR-10 dataset: total number of episode, number of model to be sampled from experience replay to update Q-table and the updating frequency (updating mode). The table 4.35 shows the best architecture that has been found according to validation accuracy by random search method with the total of 2000 models with 15-20 minutes for training time per each model and by layerwise search method with the total of 64 models with same training time per each model as random search.

From table 4.34 in the experiment 1, 2 and 3, while the experience replay updating technique and number of model sampled to update Q-table are the same, the total number of episodes are set to be different in increasing manner from 2500 in experiment 1 to 3800 in experiment 3. The validation accuracy is barely different; however, the training time is rising corresponding to the number of episodes. While the experiment 4 has lower number episode than experiment 3, training time is the same. This indicates that even if validation accuracy of experiment 3 is slightly less than experiment 4, with experience replay technique the overall training will converge faster than those without.

In the experiment 3, the Q-learning agent performs a Q-table update every time the agent finishes training new model by randomly sampling 5 models from experience replay while, in the experiment 5, Q-table update iterates over every 100 episode without considering about new model found by the agent. In the experiment 3 and 5, even if the total numbers of episodes are equal (3800 episodes), the number of model trained in experiment 5 are 3096 models which is lower than 3407 models by 311 models. In the experiment 4, even if it has lower number of episode than experiment 5, the total of models trained are still higher than the experiment 5. Since training one model of CIFAR-10 takes about 15-20 minutes, this is why training time of experiment 5 is lower than experiment 3 and 4.

The conclusion for Q-learning is that the frequency of Q-table update by randomly sampling models from experience replay plays an important role in helping speed up the convergence of the overall training as shown in the experiment 5 that the actions in each layer gradually converge to the highest single value. For example, given all things are equal and even if the experiment 5 has the highest number of episodes, it can still find the well-performed topologies that beat all other experiments in term of both training time and validation accuracy. The training time of experiment 5 is the lowest which is even lower than experiment 1 with just only 2500 episodes.

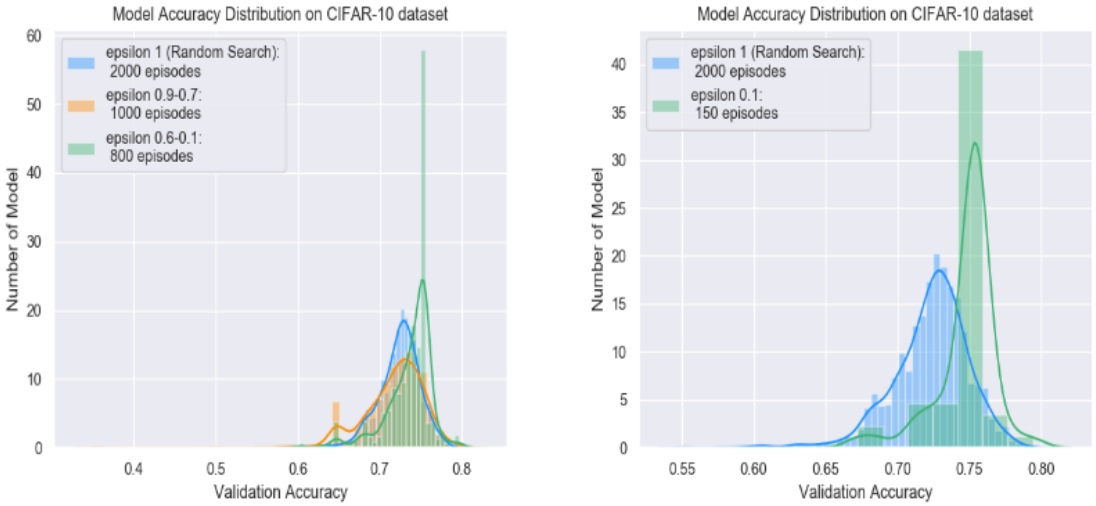

Figure 4.18: Distribution of Validation Accuracy over all episodes with different values of epsilon by Q-Search in Experiment 5 on CIFAR-10 dataset

Figure 4.18 shows the distribution of validation accuracy of topologies discovered by Q-learning agent over different of episodes which are assigned according to different epsilons. The higher the epsilon the more the agent chooses to explore more spaces while the lower the epsilon the more the agent chooses to exploit the good architectures (greedy strategy). For epsilon 1, the agent will perform random action which is similar to random search. For epsilon 0.1 with 150 episodes, the distribution of validation accuracy is moved toward the right side of the whole distribution which indicates that the agent chooses to exploit the well-performing topologies.

Although random search method can find the well-performing architectures that achieve validation accuracy of 80.18% in Table 4.35 which is about equal to best topologies discovered by Q-learning agent in experiment 5 (80.17%), Figure 4.18 shows that the distribution of random search method of 2000 models has mean clustered around just 72% percent, while Q-learning agent at last phase of episodes is able to get well-performing models with mean clustered around 75% with higher number of models with different topologies (y-axis). This indicates that random search is good compared to Q-learning method but there is some uncertainty of discovering the well-performing architectures compared to Q-learning method.

Chapter 5

Concluding Result

From the results of all experiments above, the common pattern can be observed that the experience replay technique can be useful if the number of models being sampled is set properly and the frequency that the agent samples models from experience replay also plays an important part for overall training. For example, in the last of experiment on both MNIST and CIFAR-10 datasets, the number of episodes are set to be highest and the agent samples 50 models from the experience replay to update Q-table in periodic manner namely every 100 episodes. The agent is able to find the well-performing topologies with the highest accuracy that beat all other experiments and the training time is also lowest. However, if the agent uses the technique to samples models from experience replay every time it finishes training new model to update Q-table, this technique can help avoid the local suboptimal problem but it does not help speed up training time compared to the experiment without experiment replay technique.

Dataset / Algorithms	Validation Accuracy for MNIST	Validation Accuracy for CIFAR-10
Random Search	98.13%	80.18%
Layerwise Search	97.46%	75.72%
Q-Search	97.67%	80.17%

Table 5.1: Result of Best Validation Accuracy of topologies found by Random Search, Layerwise Search and Q-Search methods on MNIST and CIFAR-10 dataset

From Table(5.1), although Random Search may beat Q-Search on both dataset, the distribution of best topologies found by Random Search in both dataset shows the uncertainty in finding good models, meaning that there is no guarantee that how long and how many models are needed for training in order to get a well-performing architecture as dataset becomes more complex. As seen in more complicated dataset like CIFAR-10 compared to MNIST, the distribution of validation accuracy of models is more varying. However, for Q-Search, it is certain to find well-performing topologies at the very end of episode (though the agent sometimes can find them early on). Although the layerwise search method is very time-efficient and simple, it has some limitations as shown in a decrease in performance in validation accuracy for more complicated dataset like CIFAR-10 over simple data like MNIST.

Chapter 6

Conclusion and Future work

This project shows that the reinforcement learning can be applied to the application of CNN architecture optimisation. The project runs several experiments on two datasets: MNIST and CIFAR-10 with same basic settings and different parameters such as a number of episodes, a number of models sampled to update Q-table and updating modes for experience replay technique. The project shows the effectiveness of using Q-Search to select well-performing network topology which can compete with the methods under human supervision. The project also shows the comparison between a variety of experiments with different settings against simple optimisation technique like random search and layerwise search.

In the MNIST dataset, the Q-Search does not show any advantage over random search in term of performance, time, and distribution rate of good topologies being discovered. However, for the CIFAR-10 dataset, Q-Search approach can beat random search and layerwise search in term of performance and rate of well-performing topologies being found because the experiments under Q-Search approach demonstrate that with proper setups the agent will discover good models with certainty at the end of episode (though the well-performing model can be found early on). Therefore, this indicates that as task becomes more challenging from handwritten numbers in MNIST dataset to real life objects in CIFAR-10 dataset, the complexity of model architecture will increase in term of both parameters and underlying network topologies, the random search method may work but there is no guarantee that it will find a well-performing model in the end. Even if the random search can find that good model,the training time can not be determined in advance, making the overall training unreliable and uncertain.

Another disadvantage of the approach used in this project is that the project limits the state and action space, making the overall network inflexible compared to MetaQNN (Baker et al. (2016)) and RNN agent method (Zoph and Le (2016)). Additionally, the great amount of time for overall training is required since the Q-learning agent needs enough sample spaces before capitalizing on them. Nevertheless, the project proves that the Q-Search method can find well-performing topologies with

limited computational resources and is more simple than others in term of code and infrastructure setups. The agent is capable of finding the CNN architecture for a shallow network that can beat the hand-picked model (McDonnell and Vladusich (2015)) in the CIFAR-10 even if there is no benchmark of the shallow network for MNIST.

In conclusion, the application of reinforcement learning helps compensate for the fact that there are some biases in selecting network as human tends to create models based on their own experience and on successful model topologies. In addition, there is not much intuition behind how to select network topologies in order to obtain well-performing models as one model can perform well in one dataset but may not work on other dataset. For example, the DropConnect network (Wan et al. (2013)) beats all existing models on MNIST but it does not work so well on CIFAR-10 compared to average benchmark of top networks. In sum, the process of automating network architecture selection could help save time with reliable results without concerning about any kind of datasets.

For the future work, the state and action spaces can be set to be more flexible and to be expanded by using other reinforcement algorithms besides simple Q-learning such as Q-learning approximation. In addition, the overall training of Q-learning can be combined with Recurrent Neural Network architecture to make underlying network topologies more flexible and suitable to any kind of machine learning problems. Additionally, the ε-greedy technique can be combined with other exploration techniques such as Boltzmann approach and Bayesian approach to help overall training converge faster.

Chapter 7

Ethics Checklist

	Yes	No
Section 1: HUMAN EMBRYOS/FOETUSES		
Does your project involve Human Embryonic Stem Cells?		✓
Does your project involve the use of human embryos?		✓
Does your project involve the use of human foetal tissues / cells?		✓
Section 2: HUMANS		
Does your project involve human participants?		✓
Section 3: HUMAN CELLS / TISSUES		
Does your project involve human cells or tissues? (Other than from Human Embryos/Foetuses i.e. Section 1)?		✓
Section 4: PROTECTION OF PERSONAL DATA		
Does your project involve personal data collection and/or processing?		✓
Does it involve the collection and/or processing of sensitive personal data (e.g. health, sexual lifestyle, ethnicity, political opinion, religious or philosophical conviction)?		✓
Does it involve processing of genetic information?		✓
Does it involve tracking or observation of participants? It should be noted that this issue is not limited to surveillance or localization data. It also applies to Wan data such as IP address, MACs, cookies etc.		✓
Does your project involve further processing of previously collected personal data (secondary use)? For example Does your project involve merging existing data sets?		✓
Section 5: ANIMALS		
Does your project involve animals?		✓
Section 6: DEVELOPING COUNTRIES		

Does your project involve developing countries?		✓
If your project involves low and/or lower-middle income countries, are any benefit-sharing actions planned?		✓
Could the situation in the country put the individuals taking part in the project at risk?		✓
Section 7: ENVIRONMENTAL PROTECTION AND SAFETY		
Does your project involve the use of elements that may cause harm to the environment, animals or plants?		✓
Does your project deal with endangered fauna and/or flora /protected areas?		✓
Does your project involve the use of elements that may cause harm to humans, including project staff?		✓
Does your project involve other harmful materials or equipment, e.g. high-powered laser systems?		✓
Section 8: DUAL USE		
Does your project have the potential for military applications?		✓
Does your project have an exclusive civilian application focus?		✓
Will your project use or produce goods or information that will require export licenses in accordance with legislation on dual use items?		✓
Does your project affect current standards in military ethics e.g., global ban on weapons of mass destruction, issues of proportionality, discrimination of combatants and accountability in drone and autonomous robotics developments, incendiary or laser weapons?		✓
Section 9: MISUSE		
Does your project have the potential for malevolent/criminal/terrorist abuse?		✓
Does your project involve information on/or the use of biological-, chemical-, nuclear/radiological-security sensitive materials and explosives, and means of their delivery?		✓
Does your project involve the development of technologies or the creation of information that could have severe negative impacts on human rights standards (e.g. privacy, stigmatization, discrimination), if misapplied?		✓
Does your project have the potential for terrorist or criminal abuse e.g. infrastructural vulnerability studies, cybersecurity related project?		✓
Section 10: LEGAL ISSUES		

Will your project use or produce software for which there are copyright licensing implications?		✓
Will your project use or produce goods or information for which there are data protection, or other legal implications?		✓
Section 11: OTHER ETHICS ISSUES		
Are there any other ethics issues that should be taken into consideration?		✓

Chapter 8

Bibliography

Bibliography

Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G. S., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I., Harp, A., Irving, G., Isard, M., Jia, Y., Jozefowicz, R., Kaiser, L., Kudlur, M., Levenberg, J., Mané, D., Monga, R., Moore, S., Murray, D., Olah, C., Schuster, M., Shlens, J., Steiner, B., Sutskever, I., Talwar, K., Tucker, P., Vanhoucke, V., Vasudevan, V., Viégas, F., Vinyals, O., Warden, P., Wattenberg, M., Wicke, M., Yu, Y., and Zheng, X. (2015). TensorFlow: Large-scale machine learning on heterogeneous systems. Software available from tensorflow.org. pages 2

Agarap, A. F. (2018). Deep learning using rectified linear units (relu). *CoRR*, abs/1803.08375. pages 27

Baker, B., Gupta, O., Naik, N., and Raskar, R. (2016). Designing Neural Network Architectures using Reinforcement Learning. pages v, 16, 19, 20, 60

BELLMAN, R. (1957). A markovian decision process. *Journal of Mathematics and Mechanics*, 6(5):679–684. pages 8

Bergstra, J., Bardenet, R., Bengio, Y., and Kégl, B. (2011). Algorithms for hyper-parameter optimization. In *Proceedings of the 24th International Conference on Neural Information Processing Systems*, NIPS'11, pages 2546–2554, USA. Curran Associates Inc. pages 16

Bergstra, J. and Bengio, Y. (2012). Random search for hyper-parameter optimization. *J. Mach. Learn. Res.*, 13:281–305. pages 2

Bergstra, J., Yamins, D., and Cox, D. D. (2012). Making a science of model search. *CoRR*, abs/1209.5111. pages 16

Bergstra, J., Yamins, D., and Cox, D. D. (2013). Making a science of model search: Hyperparameter optimization in hundreds of dimensions for vision architectures. In *Proceedings of the 30th International Conference on International Conference on Machine Learning - Volume 28*, ICML'13, pages I–115–I–123. JMLR.org. pages 15

Beumer, G. M., Tao, Q., Bazen, A. M., and Veldhuis, R. N. J. (2006). A landmark paper in face recognition. In *7th International Conference on Automatic Face and Gesture Recognition (FGR06)*, pages 6 pp.–78. pages 1

Chellappa, R., Fukushima, K., Katsaggelos, A., Kung, S.-Y., LeCun, Y., Nasrabadi, N. M., and Poggio, T. A. (1998). Applications of artificial neural networks to image processing (guest editorial). *IEEE Transactions on Image Processing*, 7(8):1093–1097. pages 1

Chollet, F. (2016). Xception: Deep learning with depthwise separable convolutions. *CoRR*, abs/1610.02357. pages 1

Dayan, P. (1994). The convergence of td(lambda) for general lambda. pages 13

Domhan, T., Springenberg, J. T., and Hutter, F. (2015). Speeding up automatic hyperparameter optimization of deep neural networks by extrapolation of learning curves. In *Proceedings of the 24th International Conference on Artificial Intelligence*, IJCAI'15, pages 3460–3468. AAAI Press. pages 2, 15

Floreano, D., Drr, P., and Mattiussi, C. (2008). Neuroevolution: from architectures to learning. *Evolutionary Intelligence*, 1(1):47–62. pages 16

Graham, B. (2014). Fractional max-pooling. *CoRR*, abs/1412.6071. pages 28

He, K., Zhang, X., Ren, S., and Sun, J. (2015). Deep residual learning for image recognition. *CoRR*, abs/1512.03385. pages 1

ji Lin, L. (1992). Self-improving reactive agents based on reinforcement learning, planning and teaching. In *Machine Learning*, pages 293–321. pages 3, 16, 17

Jia, Y., Shelhamer, E., Donahue, J., Karayev, S., Long, J., Girshick, R., Guadarrama, S., and Darrell, T. (2014). Caffe: Convolutional architecture for fast feature embedding. In *Proceedings of the 22Nd ACM International Conference on Multimedia*, MM '14, pages 675–678, New York, NY, USA. ACM. pages 2

Krendzelak, M. and Jakab, F. (2015). Text categorization with machine learning and hierarchical structures. In *2015 13th International Conference on Emerging eLearning Technologies and Applications (ICETA)*, pages 1–5. pages 1

Krizhevsky, A. (2009). *Learning Multiple Layers of Features from Tiny Images*. PhD thesis. pages 28

Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. In *Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 1*, NIPS'12, pages 1097–1105, USA. Curran Associates Inc. pages 1

Lecun, Y., Bengio, Y., and Hinton, G. (2015). Deep learning. *Nature*, 521(7553):436–444. pages 17

McDonnell, M. D. and Vladusich, T. (2015). Enhanced image classification with a fast-learning shallow convolutional neural network. *CoRR*, abs/1503.04596. pages 4, 28, 55, 61

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., Petersen, S., Beattie, C., Sadik, A., Antonoglou, I., King, H., Kumaran, D., Wierstra, D., Legg, S., and Hassabis, D. (2015). Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533. pages 2, 16, 17

Oliveira, D. A. B. and Viana, M. P. (2017). Fast cnn-based document layout analysis. In *2017 IEEE International Conference on Computer Vision Workshops (ICCVW)*, pages 1173–1180. pages 1

Paszke, A., Gross, S., Chintala, S., Chanan, G., Yang, E., DeVito, Z., Lin, Z., Desmaison, A., Antiga, L., and Lerer, A. (2017). Automatic differentiation in pytorch. pages 2

Pinto, N. and Cox, D. D. (2012). High-throughput-derived biologically-inspired features for unconstrained face recognition. *Image Vision Comput.*, 30(3):159–168. pages 16

Rummery, G. A. and Niranjan, M. (1994). On-line q-learning using connectionist systems. Technical report. pages 13

Saxe, A. M., Koh, P. W., Chen, Z., Bhand, M., Suresh, B., and Ng, A. Y. (2011). On random weights and unsupervised feature learning. In *Proceedings of the 28th International Conference on International Conference on Machine Learning*, ICML'11, pages 1089–1096, USA. Omnipress. pages 2

Schaffer, J. D., Whitley, D., and Eshelman, L. J. (1992). Combinations of genetic algorithms and neural networks: a survey of the state of the art. In *[Proceedings] COGANN-92: International Workshop on Combinations of Genetic Algorithms and Neural Networks*, pages 1–37. pages 15

Shahriari, B., Swersky, K., Wang, Z., Adams, R. P., and de Freitas, N. (2016). Taking the human out of the loop: A review of bayesian optimization. *Proceedings of the IEEE*, 104(1):148–175. pages 15

Simonyan, K. and Zisserman, A. (2014). Very deep convolutional networks for large-scale image recognition. *CoRR*, abs/1409.1556. pages 1

Snoek, J., Larochelle, H., and Adams, R. P. (2012). Practical Bayesian Optimization of Machine Learning Algorithms. *ArXiv e-prints*. pages 15

Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. (2014). Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15:1929–1958. pages 27

Stanley, K. O., D'Ambrosio, D. B., and Gauci, J. (2009). A hypercube-based encoding for evolving large-scale neural networks. *Artif. Life*, 15(2):185–212. pages 16

Stanley, K. O. and Miikkulainen, R. (2002). Evolving neural networks through augmenting topologies. *Evolutionary Computation*, 10(2):99–127. pages 16

Sutton, R. S. (1988). Learning to predict by the methods of temporal differences. In *MACHINE LEARNING*, pages 9–44. Kluwer Academic Publishers. pages 13

Sutton, R. S. and Barto, A. G. (1998). *Introduction to Reinforcement Learning*. MIT Press, Cambridge, MA, USA, 1st edition. pages 5, 6, 7, 8, 9, 11, 13

Swersky, K., Snoek, J., and Adams, R. P. (2013). Multi-task bayesian optimization. In *Proceedings of the 26th International Conference on Neural Information Processing Systems - Volume 2*, NIPS'13, pages 2004–2012, USA. Curran Associates Inc. pages 15

Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S. E., Anguelov, D., Erhan, D., Vanhoucke, V., and Rabinovich, A. (2014). Going deeper with convolutions. *CoRR*, abs/1409.4842. pages 1

Szegedy, C., Vanhoucke, V., Ioffe, S., Shlens, J., and Wojna, Z. (2015). Rethinking the inception architecture for computer vision. *CoRR*, abs/1512.00567. pages 1

Thrun, S. and Pratt, L., editors (1998). *Learning to Learn*. Kluwer Academic Publishers, Norwell, MA, USA. pages 16

Verbancsics, P. and Harguess, J. (2013). Generative neuroevolution for deep learning. *CoRR*, abs/1312.5355. pages 15

Wan, L., Zeiler, M., Zhang, S., Cun, Y. L., and Fergus, R. (2013). Regularization of neural networks using dropconnect. In Dasgupta, S. and McAllester, D., editors, *Proceedings of the 30th International Conference on Machine Learning*, volume 28 of *Proceedings of Machine Learning Research*, pages 1058–1066, Atlanta, Georgia, USA. PMLR. pages 61

Watkins, C. J. C. H. (1989). *Learning from Delayed Rewards*. PhD thesis, King's College, Cambridge, UK. pages 14

Wierstra, D., Gomez, F. J., and Schmidhuber, J. (2005). Modeling systems with internal state using evolino. In *Proceedings of the 7th Annual Conference on Genetic and Evolutionary Computation*, GECCO '05, pages 1795–1802, New York, NY, USA. ACM. pages 16

Yu, Y. (2012). Research on speech recognition technology and its application. In *2012 International Conference on Computer Science and Electronics Engineering*, volume 1, pages 306–309. pages 1

Zoph, B. and Le, Q. V. (2016). Neural architecture search with reinforcement learning. *CoRR*, abs/1611.01578. pages 60