Tufts University

Department of Computer Computer Science

COMP150-04: Reinforcement Learning Final Project

Investigation Of The Effects Of Character-centric Cropping On Pacman DQN Performance

Authors:
Holt Spalding
Oliver Newland

Supervisor:
Dr. Jivko Sinapov

December 20, 2018

Abstract

In this paper we present our findings for an exploratory experiment over the performance of two deep Q-learning network reinforcement learning agents playing ATARI Ms. Pacman. The agents are distinct in that one has access to the entire game state (as represented by the pixels on screen), while the other only a cropped view of the screen, centered around Ms. Pacman. We hoped to explore whether the smaller feature space of the cropped-view agent would be able to train faster without a significant loss in in-game performance. Despite the paper's initial ambitions, we were ironically severely limited by the computational resources available to us. However, our results still provide a good starting point for fully understanding the relationship between agent performance and feature space reduction.

1 Introduction

Over the past few years there has been remarkable progress in the study of reinforcement learning agents in high-dimensional sensory environments. With the advent of so-called "deep" reinforcement learning, RL agents can now use non-linear function approximators such as neural networks in order to deconstruct complex patterns within abstract environments. Deep RL algorithms have been especially impactful in the field of computer vision, one such algorithm being deep Q-learning networks (or DQNs). DQNs were first pioneered by engineers at Deepmind, a Google subsidiary, in 2013, and since then they've come to be used extensively in development of game-playing RL agents. A benefit of DQNs is that DQN agents need only a visual representation of their game environment in order to learn optimal play strategies, no information about the internal state of the game is necessary. This is all thanks to convolutional neural networks (CNN) built into the DQN algorithm which are capable of breaking down images into their most relevant features. Mastering the use of purely visual stimuli in the training of deep RL agents has countless real world applications and it could hold the key to development of fully-autonomous robotic agents.

While there's been quite a lot of research into deep RL agents for computer vision over the last few years, few seem to have studied the effects of incomplete visual information on agent performance. Many researchers will often perform a basic crop and greyscaling of their visual stimuli in order to

expedite the training process, but few have seemed to formally study the effects of these and other feature compression techniques on agent performance. What if we could get away with much more extreme image compression without any visible effect on agent performance? Furthermore, what if we were cast into a situation in which not all of the visual information we wanted was available at once? It stands to reason that working with incomplete information in this way is highly applicable to many tasks, and it is a worthy cause to try understand how certain information loss can affect agent performance.

As effective as deep Q-learning networks may be in computer vision RL tasks, they are still limited by the massive amounts of computational power and time required to train them. Since the training time of a DQN is largely proportional to input image size, this paper seeks to investigate the effect of extreme cropping around a variable point of interest on agent performance. Our game of choice is openAI gym's MsPacman-v0 [**?**]. Using the same DQN implementation found in Mnih's (et. al.) iconic paper "Playing Atari with Deep Reinforcement Learning" [3], we implement a spotlight feature reduction in which a significant part of the image is cropped out with only a 40 by 40 pixel grid around Ms. Pacman being visible to the DQN. Like shining a spotlight on a dark stage, the DQN only receives information local to the game-playing agent. We compare performance of agents trained on this modified feature space with those trained on a 160 by 160 pixel grid which captures the entire game environment in order to determine if a reduction in the feature space results in huge speed ups of training time without a significant loss in agent performance.

This paper sought to demonstrate that a cropping the view field of the game would cause the agent to perform comparably well while drastically reducing the computation time. Our initial findings support this claim, however they are limited by the length of computation we were able to afford our agents. As it turns out, we were unprepared for how long a DQN agent takes to run on a high dimensional feature space, which caused major roadblocks in the development and debugging of our experiment. However, our results do show that the cropped/spotlight DQN is equally effective over the period it was allowed to train (200 episodes), and runs much faster (4 hours vs. 40 minutes). Our work leads us in the right direction, and while it does not disprove our hypothesis, it is hard to say with certainty that it supports it either. This project was mostly reinforces our initial claim that DQN's are limited by the amount of time and computation required to train them.

2 Background and Related Work

As mentioned, our work is primarily an extension of the landmark 2013 paper "Playing Atari with Deep Reinforcement Learning" by Volodymyr Mnih and company that first introduced the deep Q-learning network (DQN) learning agent [3]. They found that developing a policy by adjusting the weights of a CNN was more effective than any other learning agent before it when it came to playing ATARI games. In some cases, the agent became more competent than human experts. This paper launched an explosion of CNN use in reinforcement learning as it opens the door to running effective learning agents on high dimensional visual data without having to carefully select features for each task. The DQN we use in our experiments is inspired by this original DQN, however there are some noteworthy differences.

Firstly, the original paper never played Ms. Pacman, and instead stuck to a collection of 7, equally complex games (except pong, which is certainly less complex). We choose to run this experiment on Ms. Pacman because it lends itself well to the idea of spotlighting, in that Ms. Pacman herself moves around the board and only interacts with objects in her immediate vicinity. Because the original paper does not provide a benchmark for Ms. Pacman, we implemented the same greyscale preprocessing the Mnih and co. did, to give ourselves a baseline with which to compare our experiments.

Second, their DQN was smaller than ours. They only used two layers of convolution, while ours has three. Ours also defaults to the original 160x160 playing field, while theirs excludes the 1 pixel border around the edge of the screen, and downsamples the resolution by a factor of 2. These differences are largely in part due to the improved neural network architecture available to us through TensorFlow [1]. Our exact implementation is detailed in the next section.

Other work in feature reduction for DQNs has mostly focused on frame skipping and image resolution reduction. "Dynamic Frame skip Deep Q Network" By Aravind S. Lakshminarayanan and company took this to an extreme by creating an agent that dynamically learns how many frames to observe before selecting an action, to great success [2]. They also extended the original DQN Atari paper by basing their DQN off of the original, a model which we followed as well. However, there is no significant research we could find in reducing the field of view to be central around the playing character's location. Therefore, we believe this to be an important avenue in research to investigate, which we have done in this paper.

3 Problem Formulation and Technical Approach

3.1 OpenAI's MsPacman Python Environment

The OpenAI MsPacman environment offers a simple interface for RL researchers to emulate the classic MsPacman ATARI game in order to train RL agents. The MsPacman environment represents the game state by a 210 x 160 three-channel RGB image which it exposes to the developer. It also provides information on the number of lives MsPacman has remaining at any given frame, and the amount of in-game reward MsPacman has acrewed over the course of a game. At given frame, MsPacman is given the opportunity to perform one of 9 possible actions which include moving left, moving right, moving up, moving down, moving on a diagonal, or remaining where she was in the previous frame. Each action is repeatedly performed for a duration of k frames, where k is uniformly sampled from 2,3,4{2, 3, 4}2,3,4.

3.2 Q-Learning

Before explaining how a DQN can use images to teach a MsPacman agent the optimal play strategy, its important that we first describe the learning task in more formal terms. In keeping with classic RL paradigms, we can define our problem in terms of a Markov Decision Process. A Markov Decision Process (MDP) is just one of many models of how an agent interacts with its environment over discrete time steps, and it can defined in terms of three basic elements: a set of observations, a set of actions, and a set of rewards. Is is the goal of our learning agent to develop an understanding of environmental dynamics in order to determine the optimal action that ought be taken at given time in order to maximize some cumulative reward. In the context of this experiment, our learning agent is obviously embodied by MsPacman, the frames produced by the emulator represent the agent's observations of the state of the environment, the action space is the set of 9 actions MsPacman can take at any given frame as described above, and the rewards simply correspond to the in-game rewards MsPacman collects through her interaction with the environment. To reiterate, as our learning agent is optimized using a DQN, it has no access to the internal state of the game, only the raw pixels produced by the emulator. Over the course of the learning process the agent slowly develops a play strategy (also known as a *policy*) by approximating a so-called Q function. A Q function essentially places a value on each possible

action a at any given state of the environment by estimating the expected cumulative reward to be achieved once the action is performed (an action leading to the consumption of a power pill for example would have a high expected reward while an action leading to the death of MsPacman would have zero or negative reward). We define the cumulative reward associated with time t to be $R_t = \sum_{t'=t}^{T} \gamma^{t'-t} r_t$ representing the sum of discounted (discounted by a constant factor γ) rewards beginning at time t until the time of the game's termination T. An agent which has faithfully approximated the optimal action-value function, denoted Q^*, knows exactly what action to take at any given state of the environment in order to receive the largest reward. In mathematical terms, we say $Q^* = max_\pi \mathbb{E}[R_t | s_t = s, a_t = a, \pi]$ where π represents a *policy* that maps states or sequences of state-action pairs to a distribution over actions.

3.3 DQN

A Q-network is a neural network whose weights are represented by θ which serves as a nonlinear function approximator to approximate Q^*. A Q-network is trained by adjusting θ in order minimize a series of loss functions $L_i(\theta_i)$ which change at each iteration of back-propagation i. Formally,

$$L_i(\theta_i) = \mathbb{E}_{s,a\sim\rho(\cdot)}[(y_i - Q(s, a; \theta_i))^2]$$

where $y_i = \mathbb{E}_{s'\sim\varepsilon}[r + \gamma max_{a'} Q(s', a'; \theta_{i-1}) | s, a]$ is the target for iteration i and $\rho(s, a)$ is a probability distribution over sequences s and an action a which is referred to as a *behavior distribution*. We say that DQN is an *off-policy* learning algorithm since it updates the Q-value of the state-action pair it has chosen under its behavior policy by comparing it with the Q-value of the next state s' and action a' it would have chosen under a greedy policy. In other words, it estimates the return (total discounted future reward) for state-action pairs assuming a greedy policy were followed despite the fact that it's not following a greedy policy. This serves to provide adequate approximations of the Q-value of state-action pairs without hindering exploration under a behavior policy.

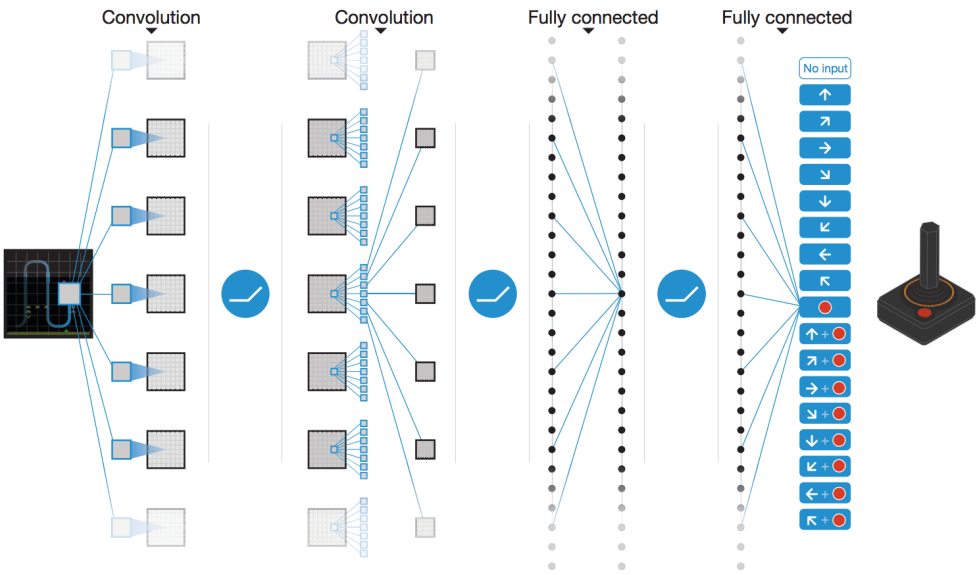

Figure 1: Schematic illustration of DQN architecture taken from Google images. Source unknown.

The above image illustrates the basic architecture which underlies a DQN. As you can see, first a vector of raw pixel representing a frame from the game is fed into a feed-forward convolutional neural network. This CNN serves to extract abstract but nonetheless important features from the image, providing a more nuanced picture of the state of the game. After feeding through a series of convolution and pooling layers, the resulting tensor is then fed into a series of fully connected layers which connect to our final output layer, which in this case contains 9 units, the values of each representing the Q-values of each of the 9 actions given the input state and the current network weights. The Q-value of our current state-action pair under our behavior policy as well as the Q-value of our state-action pair under a greedy policy can be calculated with a simple feed-forward pass through the network. Network weights can then by adjusted to reduce the discrepency's between our target and behavior policies by means of backpropagation which uses gradient descent for optimization. The subject of adjusting network weights lies outside the scope of this paper and need not be explained any further.

Algorithm 1 Deep Q-learning with Experience Replay

Initialize replay memory $\mathcal{D}$ to capacity N
Initialize action-value function Q with random weights
for episode $= 1, M$ **do**
  Initialise sequence $s_1 = \{x_1\}$ and preprocessed sequenced $\phi_1 = \phi(s_1)$
  for $t = 1, T$ **do**
    With probability ϵ select a random action a_t
    otherwise select $a_t = \max_a Q^*(\phi(s_t), a; \theta)$
    Execute action a_t in emulator and observe reward r_t and image x_{t+1}
    Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$
    Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in $\mathcal{D}$
    Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from $\mathcal{D}$
    Set $y_j = \begin{cases} r_j & \text{for terminal } \phi_{j+1} \\ r_j + \gamma \max_{a'} Q(\phi_{j+1}, a'; \theta) & \text{for non-terminal } \phi_{j+1} \end{cases}$
    Perform a gradient descent step on $(y_j - Q(\phi_j, a_j; \theta))^2$ according to equation 3
  end for
end for

Figure 2: Mnih et. al, 2013

To get a slightly better sense of the DQN algorithm, look to the pseudocode above from the 2013 Deepmind paper. This algorithm describes two aspects of the algorithm yet to be explained. The first is the process of action selection, which is done by means of an epsilon-greedy policy. Epsilon is slowly annealed over the course of the training process, in order to allow exploration early on in the learning process and more focussed action selection later in the learning process. The other concept yet to be described is "experience replay." The DQN algorithm employed by the engineers at Deepmind does not learn online, but it actually learns by means of randomly sampling small sequences of state-action pairs recorded from the past. This is done in order to prevent the preference of any particular action.

4 Experiment

For this experiment we compared two different agents, one trained on a 160 x 160 pixel, greyscale image of the Ms. Pacman learning environment, and another trained on a 40 x 40 pixel greyscale image of the environment centered around Ms. Pacman. Both agents were run for a 10,000 time steps before training began in order to gather experience which could be used during experience replay. Both agents also had their frame skip set to 4, a discount factor 0.99, a replay memory buffer with a maximum size of 60,000

frames, and a replay memory mini batch size of 32 frames, as is done in the DeepMind paper. Due to computational limitations, both agents could only be run for 200 episodes each, each episode providing MsPacman with 3 lives. The average score of all 3 rounds of every episode were then recorded and averaged in order to assess performance.

5 Results

Experiment 1 (Figure 3) trained on 200 episodes in about 4 hours. This is significantly less time than Mnih and company allowed for their agent in their original paper (on a more powerful computer no less), but it is about the extent of what our computers could manage. We see a steady increase in the peaks of performance, but our troughs are fairly consistently low, and our average scores are fairly low in general (however they are averaged over 3 lifes played on the same board). At the very end of training, for the last 25 episodes or so, we see performance improve, however not to a degree that we can be certain beyond doubt it is statistically significant. Still, the improvement exists.

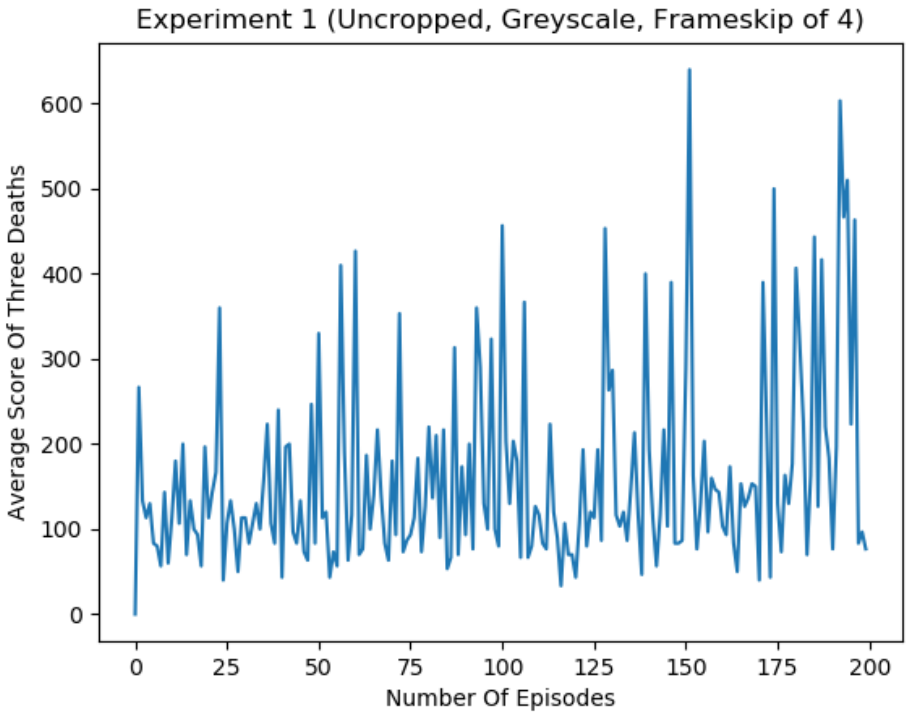

Figure 3: The score recorded for each episodes is the average of three lives played. Experiment 1 featured the full field of play.

Experiment 2 (Figure 4) trained on 200 episodes as well, taking about 40 minutes. The additional speed is due to the significantly smaller feature space. Experiment 2 only runs about 6 times as fast as Experiment 1, despite running on an image 16 times smaller. Ultimately, we see no definite improvement in performance from Experiment 1, however we see no real drop in performance either. The scores take a similar pattern with increasingly high peaks and consistently low troughs. However, where Experiment 1 picks up at the end, Experiment 2 does not. But overall, they are remarkably similar.

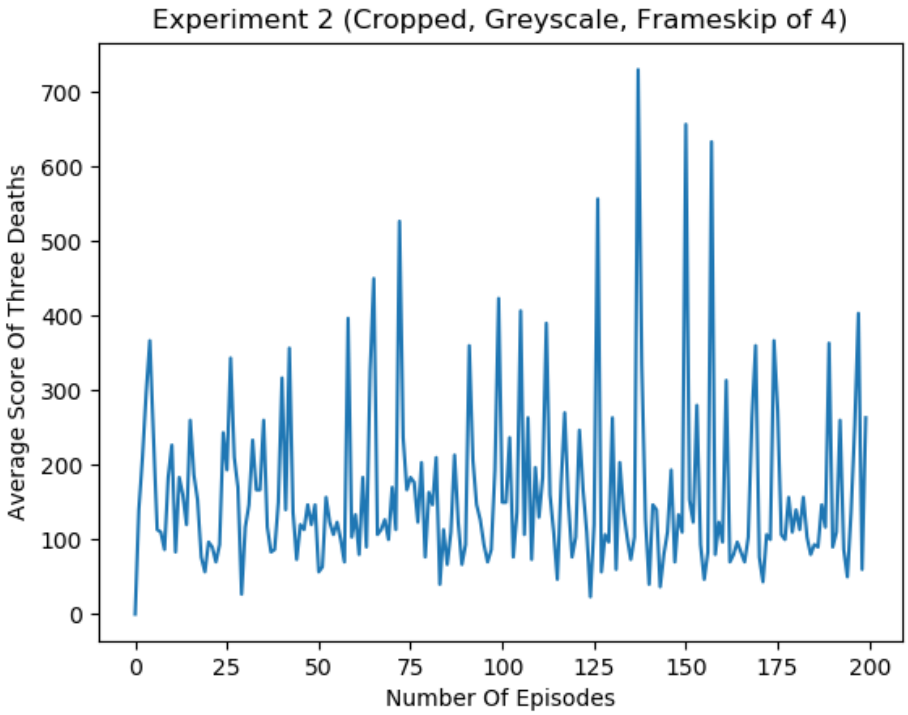

Figure 4: The score recorded for each episodes is the average of three lives played. Experiment 2 featured the cropped field of play.

When the running averages (over the last 10 scores) of the experiments are viewed together (Figure 5), that similarity becomes very clear. The averages stay in the band of values between 100 and 200, until the very end where Experiment 1 improves. This suggests at best, the two learning agents perform the same, and at worst, uncropped begins to improve more quickly per episode, but not by time.

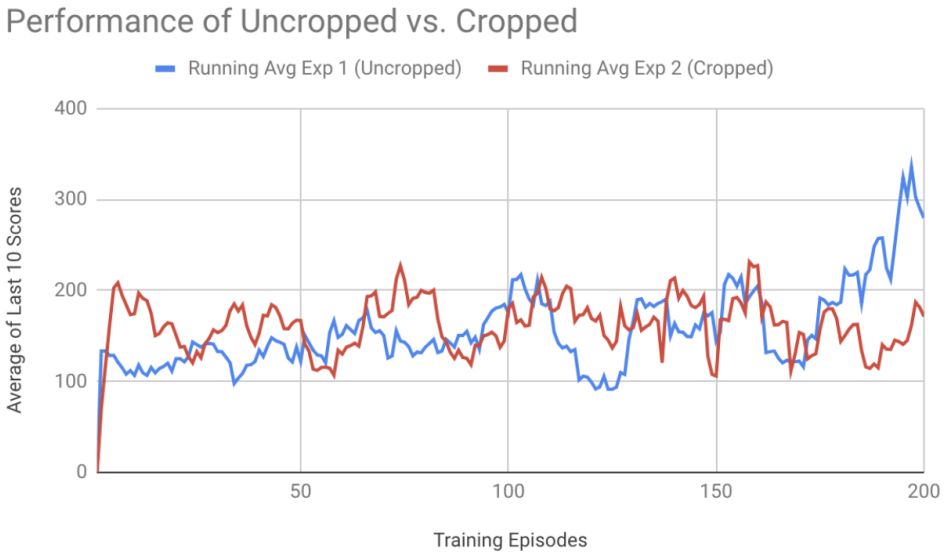

Figure 5: The score reported here is the average of the scores from the last 10 episodes.

6 Conclusion

By and large, this project showed how valuable it would be to reduce the training time of DQN agents. Even knowing that training time would be a huge obstacle, we were still limited in the number and length of experiments we could run, and that severely hindered our results. However, there is still a lot of valuable information we were able to gather. First, a reduction in feature space from 160 * 160 = 25600 to 40 * 40 = 1600, a reduction by a factor of 16, only resulted in a training speed increase by a factor of 6, so returns are not proportional here. Still, over the same number of episodes trained, both the standard and spotlight agents performed at about the same level, however the standard agent eventually began to pull away. In hindsight, there were several improvements that we could have made to achieve more meaningful results (discussed in the next section), but as of now we cannot rule out the possible efficacy of cropping the feature space for a better performance to training time tradeoff.

7 Future Work

Studying DQN's requires very specific experiment design and a wealth of computational power and time. We can improve in all of those areas. The goal of improving our future work would be to move beyond not rejecting a hypothesis to being able to describe the exact details of cropping across different crop sizes and domains, in the hopes that problems in the future can be identified as potentially benefiting from feature space cropping.

As far as experimental design goes, we initially set out to accomplish a much more robust study. Instead of two experiments, it would be ideal to experiment over every degree of cropping to find an optimal level of crop. Doing so would likely take weeks, and that is without having to debug half way through hour long test runs. We also wanted to run other baseline experiments to compare our cropping results to the performance of other feature space reductions such as binary color mapping, downsampling, and frame skipping.

In fact, there is likely a relationship between frame skipping and the effectiveness of a crop. If a string of the same move, provided all at once because of frame skipping, takes Ms. Pacman out of the current field of view, then surely it becomes much harder to train the agent. We did not explore this relationship in our project, but it is certainly an area of interest for the future.

Ultimately, even with warning, we underestimated how much time a DQN takes to run and debug. This project has given both of us a much greater appreciation for planning around long training sessions, as well as for GPU enabled Tensorflow, which neither of us were able to use (it is currently Linux only). We look to carry this wisdom forward into our work in the future.

References

[1] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Van-

houcke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[2] Aravind S. Lakshminarayanan, Sahil Sharma, and Balaraman Ravindran. Dynamic frame skip deep Q network. *CoRR*, abs/1605.05365, 2016.

[3] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin A. Riedmiller. Playing atari with deep reinforcement learning. *CoRR*, abs/1312.5602, 2013.