

Dissecting Deep-Q-Networks: Understanding their Ability to Play Vision Games

Rohit Gandikota

Northeastern University

gandikota.ro@northeastern.edu

Abstract

Deep neural networks are one of the widely used function approximators in reinforcement learning. However, they remain a black box tool to approximate the Q values for state-action pairs. In this work, we attempt to dissect the deep Q networks to interpret their ability to play Atari games at a human level. Understanding DQNs will open up possibilities to debug and design improved solutions in the field of reinforcement learning. Interpretability can also help in better decision-making in RL as the designers can understand the weak-point in the design.

In this work, we dissect the deep neural network that is trained using Q-learning to play Ping-pong-v4 and visualize the importance of each neuron in decision-making. It is observed that certain neurons tend to actively look at the ping pong ball while some neurons kept a check on the opponent's scores. We also observed that the network does a cost-risk analysis based on the visual scores present in the frame of the Atari games before making a decision. To understand the importance of the input pixels, we also looked at the back-propagated gradients at the input and found that the ping pong ball was the most important part of the input image followed by the racquets and then the scores to make a decision. Our code, models, and results can be found at <https://github.com/rohitgandikota/interpret-dqn>

1 Introduction

Reinforcement learning has proven to be an effective technique utilized in a wide range of applications. For instance, it has enabled the achievement of superhuman performance in playing Atari 2600 video games and winning the Go competition against a professional player Mnih et al. (2015) Silver et al. (2016). Furthermore, reinforcement learning has been applied in the field of traditional computer vision tasks, such as image classification Ba et al. (2014); Mnih et al. (2014), captioning Ba et al. (2015), activity recognition Yeung et al. (2016), and object detection Caicedo and Lazebnik (2015); Bueno et al. (2017). In this work, we focus on RL agents playing Atari's PingPong-v4 using deep neural networks as function approximators (DQN).

Understanding deep neural networks has been active research in the recent decade. There are works that proposed visualizing the filters through deconvolution Zeiler and Fergus (2014). The initial findings of interpretability have shown progressive learning

of filters indicating a hierarchy of filtering Forsyth et al. (1999). However, these methods represent a holistic view of neural networks as feature extractors. This work mainly focuses on the specific roles of each neuron in the decision-making of the DQN. Some works have explored the behavior of a Convolutional Neural Network (CNN), by masking out image patches and visualizing the patches that have a maximum change in the activations Zeiler and Fergus (2014), Zhou et al. (2014). Alternatively, backpropagation variants can be employed to identify or generate significant image features Mahendran and Vedaldi (2015), Simonyan et al. (2014). Understanding the discriminative power of the CNN’s hidden layers can also be accomplished by isolating, transferring, or limiting portions of the network and testing their capabilities on specialized problems Yosinski et al. (2014), Agrawal et al. (2014), Sharif Razavian et al. (2014). However, visualizations reduce the mechanisms of the network to images that require interpretation, leading us to aim to match representations of CNNs with labeled interpretations directly and automatically.

The examination of individual units within neural networks is particularly relevant to our current work. For example, in a study by Zhou et al. (2014), human evaluation determined that individual units functioned as object detectors in a network trained to classify scenes. Nguyen et al. (2016) generated prototypical images for individual units using a feature inversion mapping, in contrast to our approach of automatically assigning concept labels. Alain and Bengio (2016) proposed a method for testing intermediate layers by training simple linear probes that analyze the information dynamics among layers and their effect on the final prediction.

2 Deep Q Learning

The Markov Decision Process (MDP) is formally defined in this section, along with a discussion of the Q learning setup and the model architecture. The training procedure designed for this work is also covered.

2.1 MDP Formulation

To understand the crux of the work, we first define the MDP formally including the states, rewards, actions and transitions:

State The state of an MDP is to be formulated such that it is self-sufficient for the agent to take an action. In vision games like ping-pong, a still frame is not MDP satisfied as the direction of the ball is unclear based on a single frame as shown in Figure 1(a). Therefore, to make the state MDP-satisfied, we stack the frames of the previous state such that the agent has all the information at the current state as shown in Figure 1(b). The state consists of the racquets of both the agent and the opponent, the ball, and the scores of the agents.

Actions There are 6 pre-defined actions in the game space. The first 2 actions are *IDLE* actions making the state unchanged. Actions 3 and 5 correspond to *LEFT* where the racquet moves to the left and actions 4 and 6 correspond to *RIGHT* where the agent moves right.

Rewards The reward function of the game state is very sparse. The agent incurs a -1 reward if the agent misses the ping pong ball at its side and gets a +1 reward if the ball is missed by the opponent.

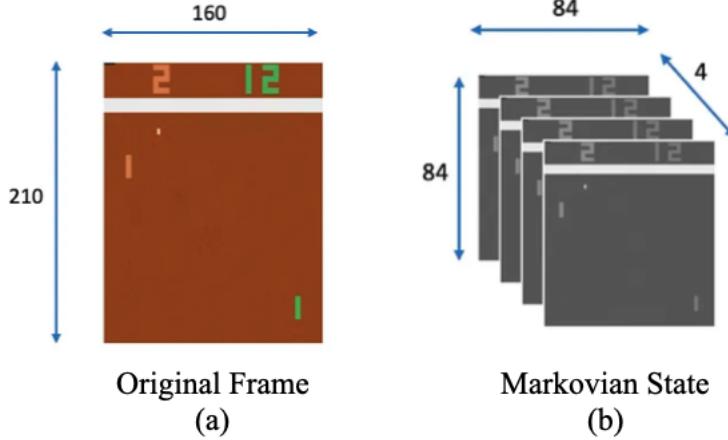


Figure 1: The original frame (a) from the game is an RGB image with no information on the direction of the ball making it non-Markovian. We, therefore, stack the frames (b) from previous time steps to impart the information to make the new state Markovian. We also preprocess the frames to make them gray-scaled and reduce the size to make them efficient.

Transition The game state is deterministic as it is a competitive beginner’s game. The agent moves right with 100% probability if *RIGHT* action is taken and left if *LEFT* action is taken.

2.2 DQN

Although it may seem like the agent selects actions based on their immediate rewards, in reality, we teach the agent to calculate state-action values $Q(s, a)$ that consider both immediate and future rewards. These values can be computed using either a tabular approach, which requires tracking all state-action pairs, or function approximation. Given the size of our state-action space ($84 \times 84 \times 4 \times 255 \times 6$) where 84 is the patch size; 4 is the frame history; 255 is the DN range and 6 is the action space. We opted for function approximation through deep neural networks. Equation 1 presents the update rule for the sample-based, 1-step, tabular Q values, where γ indicates the discount factor that determines the priority between immediate and future rewards.

$$Q(s, a) = Q(s, a) + \alpha(r + \gamma \max_{a'} Q(s', a') - Q(s, a)) \quad (1)$$

Given our use of a function approximator in the form of a deep convolutional network to predict Q values, we update the parameter λ using the equation presented in Equation 2.

$$\lambda \leftarrow \lambda + \alpha(r + \gamma \max_{a'} Q_\lambda(s', a') - Q_\lambda(s, a)) \nabla_\lambda Q_\lambda(s, a) \quad (2)$$

Deep Q networks and Q learning algorithms are susceptible to overestimating Q values, which occurs as our Q value estimates approach their true values. This is because we use the same network to generate both values, and in our agent’s training, this issue manifested as a consistent request for higher-resolution images. To address this problem, we introduce a target network, which involves maintaining a copy of the

original network (with network parameters λ') and updating it less frequently than the primary network. By doing this, we can avoid any bias from the primary network because the target network can pull the bias back. This mechanism is illustrated in Equation 3.

$$\begin{aligned} a' &= \operatorname{argmax}_{a'} Q_{\lambda}(s', a') \\ \lambda &\leftarrow \lambda + \alpha(r + \gamma Q_{\lambda'}(s', a') - Q_{\lambda}(s, a)) \nabla_{\lambda} Q_{\lambda}(s, a) \\ \lambda' &\leftarrow \lambda \text{ (every 4000 steps)} \end{aligned} \quad (3)$$

2.3 Training Details

We discuss the main training details of the DQN we train for playing Ping Pong V4. We also give some information on the deep network architecture.

Architecture: The network takes in an input image with dimensions 4 and outputs a vector of Q-values corresponding to each possible action. The network has three convolutional layers with 32, 64, and 64 filters respectively, followed by two fully connected layers with 512 and 6 units respectively. ReLU activations are used.

Hyperparameters: We compared the effects of appending the history frames in state representation and chose to append the last 4 frames as this yielded satisfactory initial results. We use a discount factor of 0.99. We updated the target vector every 10000 iterations and trained the agent for 100000 iterations, with a batch size of 32. We used the RMS optimizer with a 1e-3 learning rate and random uniform initialization.

Replay Buffer: Since we used a function approximator, training the network with consecutive samples would lead to training inefficiency due to the high correlation between samples. Thus, we used a replay buffer to store the (s, a, r, s') tuples. We simulated the agent by starting from an image state s , taking action a based on Q values ($\operatorname{argmax}_a Q(s, a) + \text{noise}$), observing the reward r and next state s' , and storing the tuple in the buffer. We used a buffer size of 50000, and once the buffer was full, we trained the network by randomly sampling pairs from the buffer. We replaced the tuple with the least error with new samples, essentially making it a pseudo-prioritized buffer.

Exploration: At the start of training, the network had no information about the task or the environment. Therefore, it was important to explore the actions initially. To achieve this, we added an exploration constant ϵ , which decreased over time as the agent learned more about the task. ϵ represents the probability at which the agent chooses a random action compared to the optimal action. We start with 100% exploration and then reduce it at 1e-5 decay till 10%.

We show the training curves in Figure 2 where the average reward is plotted against the number of training interactions and decaying exploration constant.

3 Method

In this section, we discuss the methods used to interpret the DQN agent decision-making process. We use two different methods to interpret the process: first, a dissection method to understand the learned parameters of the neural network. Second, a visual method to understand the important pixels in the input image responsible for a decision made by the agent.

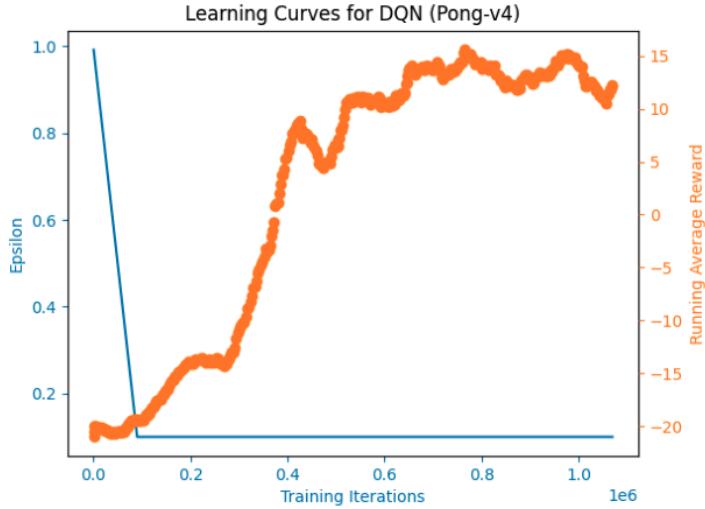


Figure 2: The learning curves of the DQN agent show a healthy learning curve and increasing rewards with reduced exploration. This shows that the agent learns to take better greedy action over time as the exploration is clearly reduced. We achieve an average reward of +15 for a game.

3.1 Neuron Dissection

To understand the importance of the learned parameters in the DQN, we dissect the network and visualize the most important features being extracted by the neurons. To elaborate, we probe the DQN at a convolution layer L and collect the activation maps \mathcal{A}_L^i . We then track the most important maps that are responsible for the prediction p at the final layer N . To do this, we use a method inspired by Viterbi backtracking. We collect the maximum activations responsible for the prediction from the previous layer $\max(\mathcal{A}_{N-1}^1, \mathcal{A}_{N-1}^2, \dots, \mathcal{A}_{N-1}^i)$ where i is the total number of neurons/filters in the layer $N - 1$. We repeat this till the layer L and visualize the top-important activation maps. Suppose the activation \mathcal{A}_{L+1}^j is the highest activation in layer $L + 1$, the top activating neuron activation in layer L with a total of n neurons would be:

$$\mathcal{A}_L^i = \max(\mathcal{A}_L^{1,j}, \mathcal{A}_L^{2,j}, \mathcal{A}_L^{3,j}, \dots, \mathcal{A}_L^{n,j}) \quad (4)$$

Where $\mathcal{A}_L^{i,j}$ represents all the activation map that goes from neuron i in layer L to neuron j on layer $L + 1$

In Figure 3, we visualize the top activations from the first convolution layer for the current frame in Figure 3(a). The network decides to move *RIGHT* and we backtrack the neurons responsible for that action. We find that the top neurons at the first layer are very specific and look at the ping pong ball and the racquets. We find that all the top 6 neurons look precisely at the ball and racquets while a few also keep a track of the boundaries to keep a track of the bounce (like neuron30 in Figure 3(b))

In Figure 4, we visualize the top activations from the second convolution layer for the current frame in Figure 4(a). We observed that the top 2 neurons (15 and 59) mainly focus on the ball. Interestingly the DQN agent assigns neuron 47 to keep a check on the opponent's score in Figure 4 and neuron 31 to keep track of its own score.

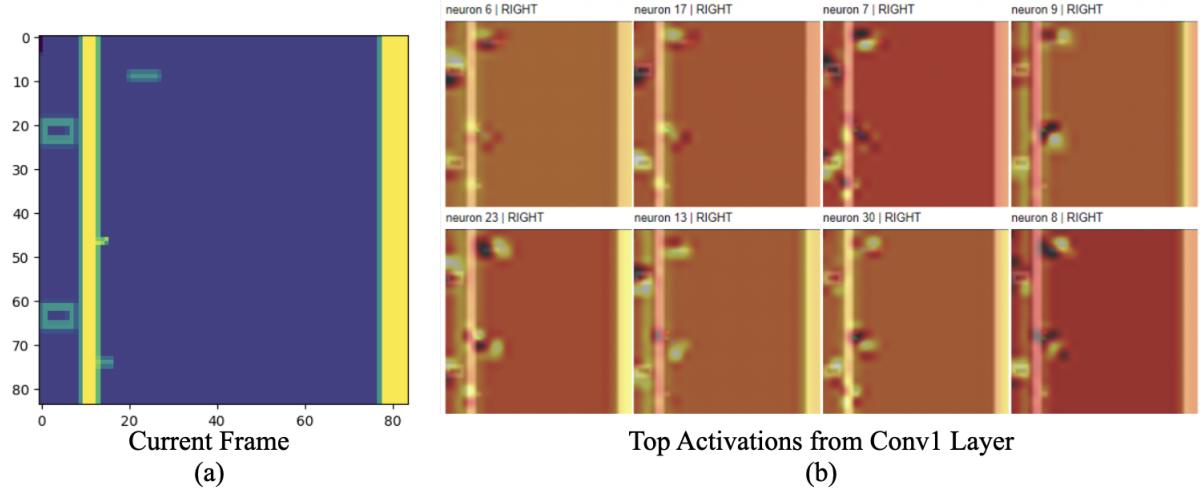


Figure 3: We look at the top activating activation maps at convolution layer 1 for the current frame in (a). We find that neuron8 looks particularly at the ball and racquets, while neuron30 also looks at the boundary to the left to understand the bounce of the ball.

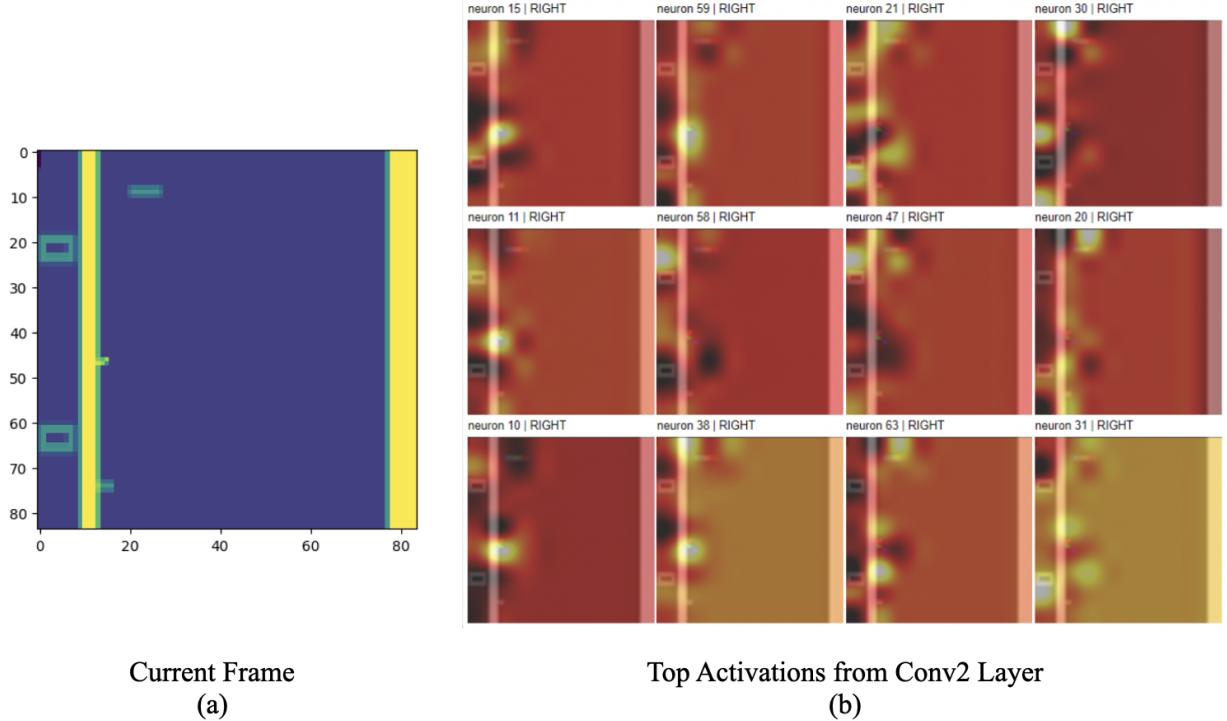


Figure 4: We look at the top activating activation maps at convolution layer 2 for the current frame in (a). We find that the top 2 neurons (15 and 59) look at the ball and racquets. Interestingly, we observed that neuron 47 looks at the opponent's score, and neuron 31 looks at its own score. We observed that the agent takes more risks if the agent score is much lower compared to its own to finish the game earlier and plays defense when the scores are not too drastic. Interesting cost-risk analysis that the agent does before making a move.

Neuron31 @ Conv2

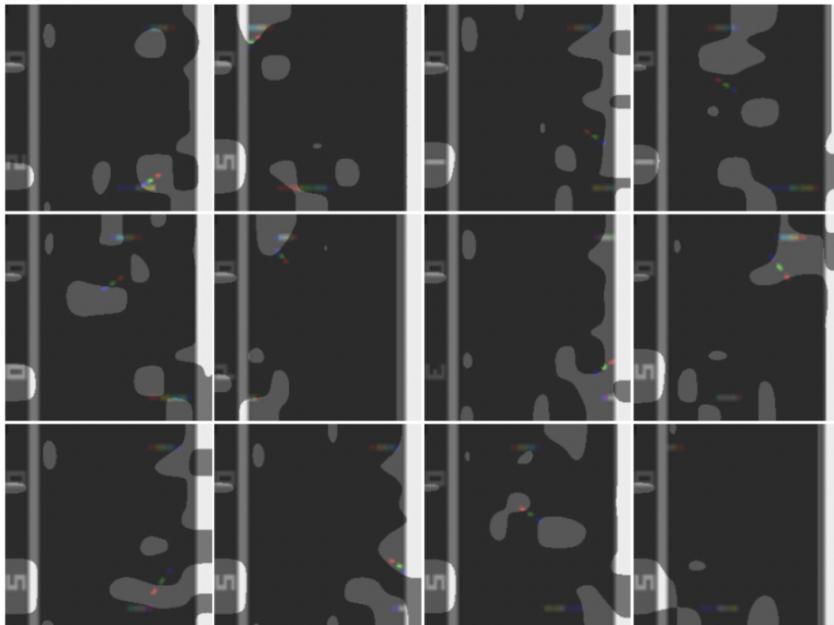


Figure 5: The top activating regions of the neuron31 in the Conv2 layer has always been the agent’s score. This shows that each neuron has a fixed role in looking at certain regions of the input frame.

We observed that based on these scores, the agent does a cost-risk analysis to take an action. If the opponent’s score is high, and the agent is at risk of losing, it plays defensively. When the score is low and the agent is winning, it takes an aggressive role.

3.2 Network Dissection

Now instead of sample-based analysis, we also do a network-based analysis where we analyze the network parameters rather than the activation of a single sample. We run the network over a large sample of images over 50000 frames and we show the 12 regions where each neuron activated strongest in this data set. In Figure 5, we show the Neuron31 from the convolution 2 layer to understand if it is predominantly designed to focus on the agent’s score. It is observed that most of the time these neurons have a fixed designation of what to observe in a frame. We show more results in Figures 7, 8, 9.

We find that Neuron 17 in Conv 1 is assigned to look at the boundary for the occurrence of bounce in the game as shown in Figure 7. On the other hand, Neuron 17 in Conv 2 has the job to guard the regions near the boundaries. We feel like, it activates the network to alert the decision before the ball is about to hit the boundary which would change the trajectory drastically as shown in Figure 8. Finally, there are some neurons that sincerely keep track of the ball and the racquets only like Neuron 13 in the Conv 2 layer as shown in Figure 9.

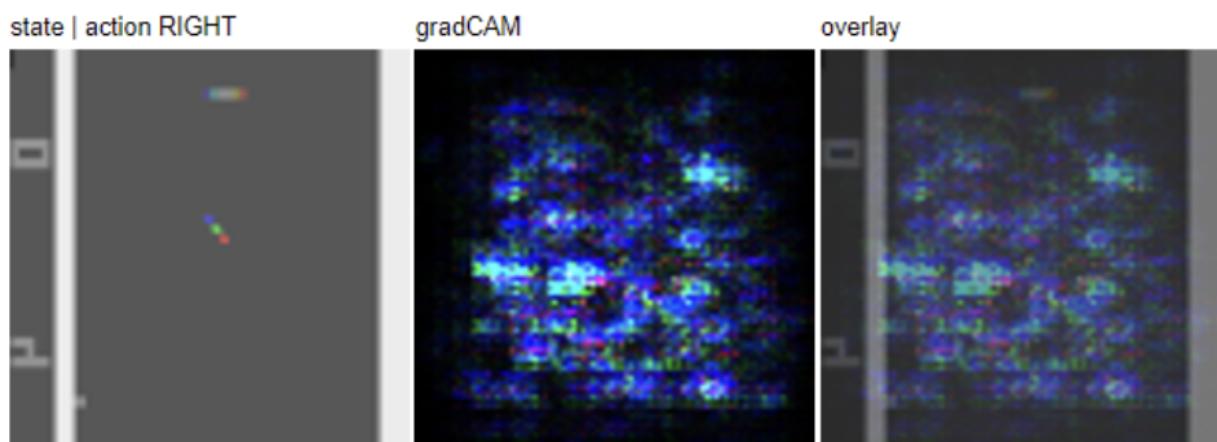
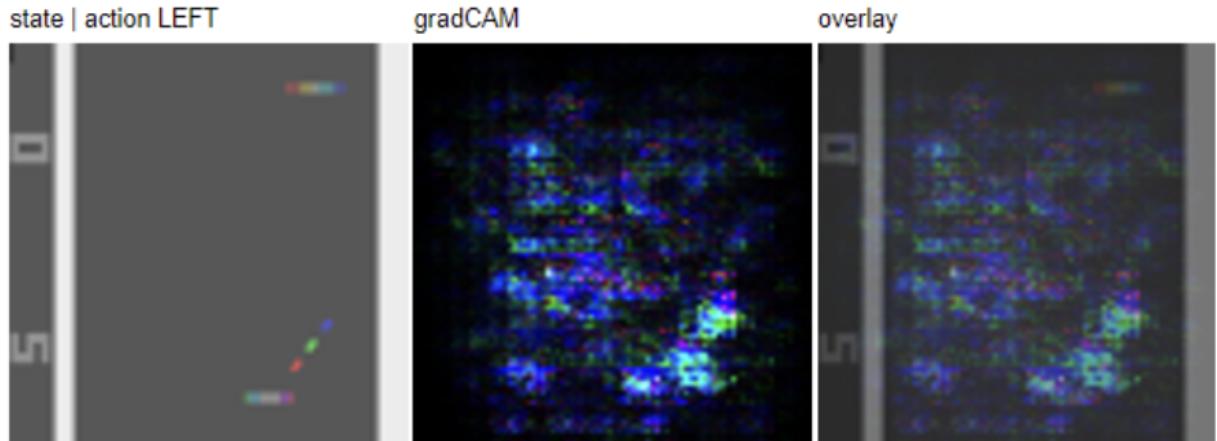


Figure 6: We look at the gradients at the perturbed inputs. We find that the gradients are high near the regions where the ball is present and also at the regions of the racquet. This is a very good interpretation of the DQN that suggests that the regions of the images with balls and racquets are the most important in decision-making.

3.3 Smooth-Grad CAM

Since neurons are differentiable it is natural to look at their gradients and visualize the rate of change. Smilkov et al. (2017) proposed a method to look at the gradients of the perturbated inputs to understand the importance of input pixels in classification. We use a similar strategy, but instead, we use it to understand the important pixels in the input region to make a decision for Q-values by the network. In Figure 6, we show the gradients at the input for the input images on the left. We find that the most important pixels in the image are the region of the ball and the racquets.

To elaborate, we add some noise to the input frames, pass it through the network and run a backprop on the network using the desired action as a label. Now we do not use the gradients to update the network, rather we visualize these gradients at the input level. A higher level interpretation of these gradients is that, the higher the gradient levels, the more important it is in the decision-making as they are updated more than the unnecessary ones.

However, we notice a great amount of noise in the gradients. This is obvious as the gradients correspond to not only the learned distribution but also the higher frequencies in the image domain and the perturbations due to ReLU activations. Similar observations have been made by Adebayo et al. (2018), where they notice that gradients are a neat idea, but it can be hard to get them to work well for visualization.

4 Conclusion

This work explores various methods to interpret the ability of DQN networks to play atari games at a human level. We show this by dissecting the neurons in the network and visualizing their top activations. We found that certain neurons in the network had designated roles like keeping track of the scores or tracking the ball. We also found that these neurons always do the same task for all the inference/test samples. We also interpret the models based on the importance they give to the input pixels. By visualizing the smooth gradients at the input, we found that the model gives the highest preference to the regions with balls and racquets. Interpretable research, we believe, would open possibilities for better and safer reinforcement learning agents.

Neuron17 @ Conv1

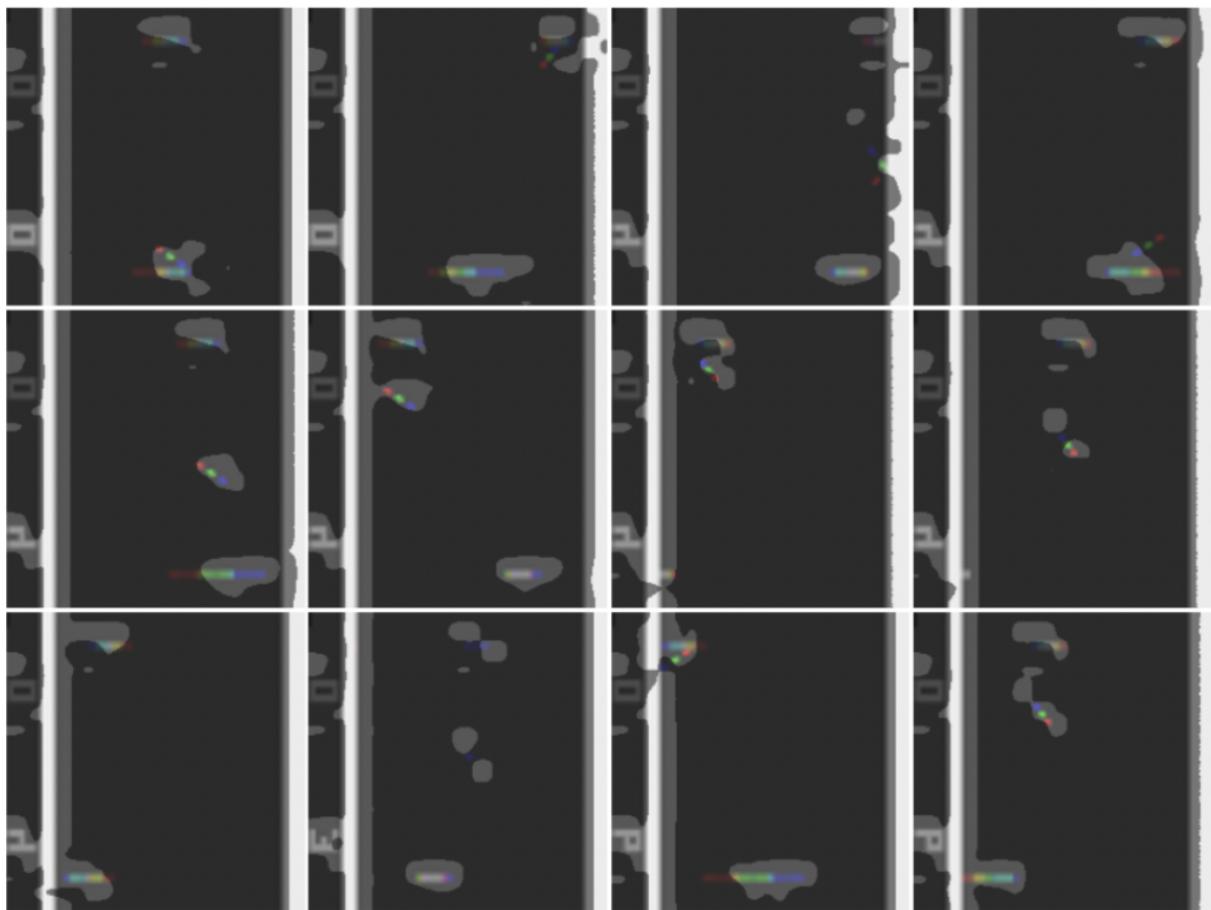


Figure 7: The top activating regions of the neuron17 in the Conv1 layer has always been the ball and the racquets. It is interesting to find that this neuron also looks at the boundaries of the game keeping track of the bounce. This shows that each neuron has a fixed role in looking at certain regions of the input frame.

Neuron17 @ Conv2

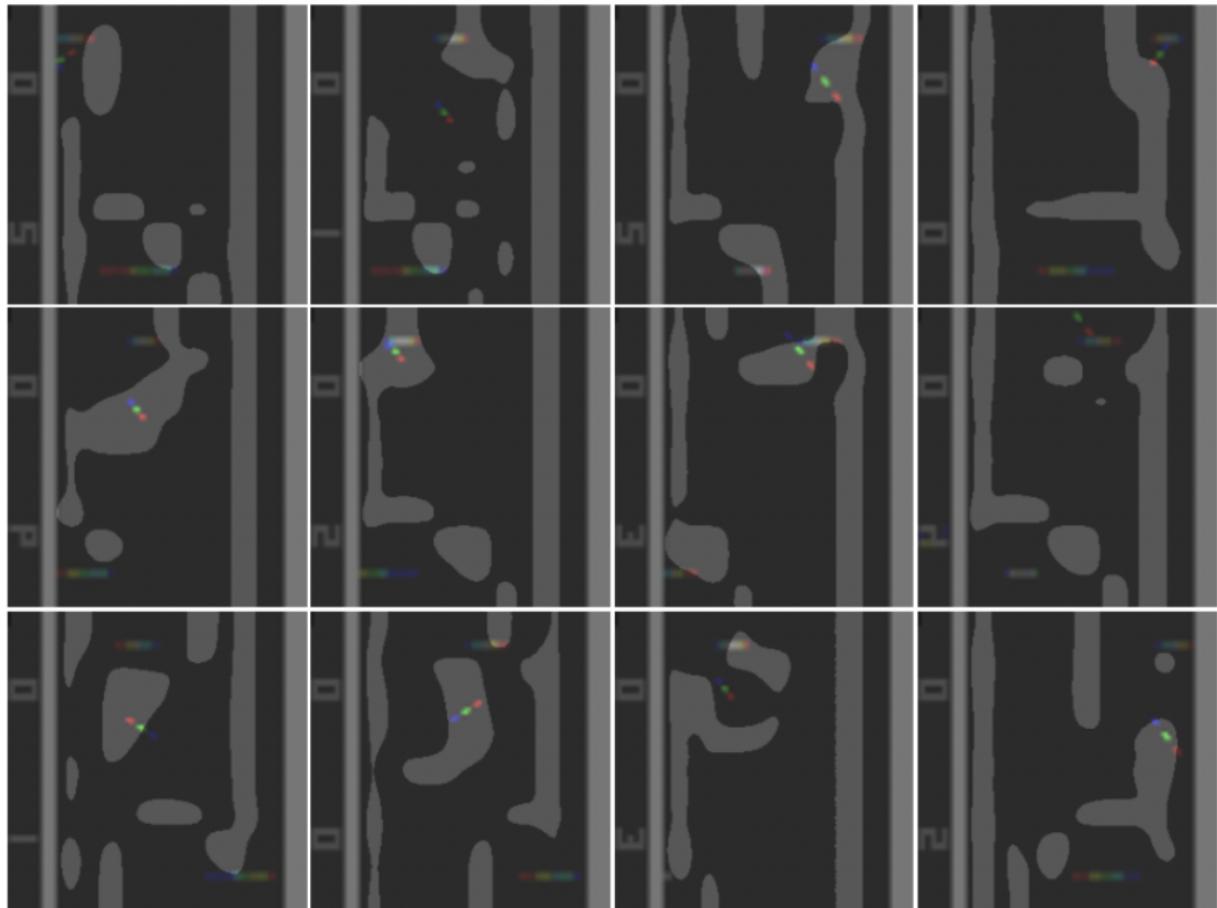


Figure 8: The top activating regions of the neuron17 in the Conv2 layer has always been the regions near This neuron and seems to be the one that is responsible to alert the model before the ball is about to hit the boundary. This shows that each neuron has a fixed role in looking at certain regions of the input frame.

Neuron13 @ Conv2

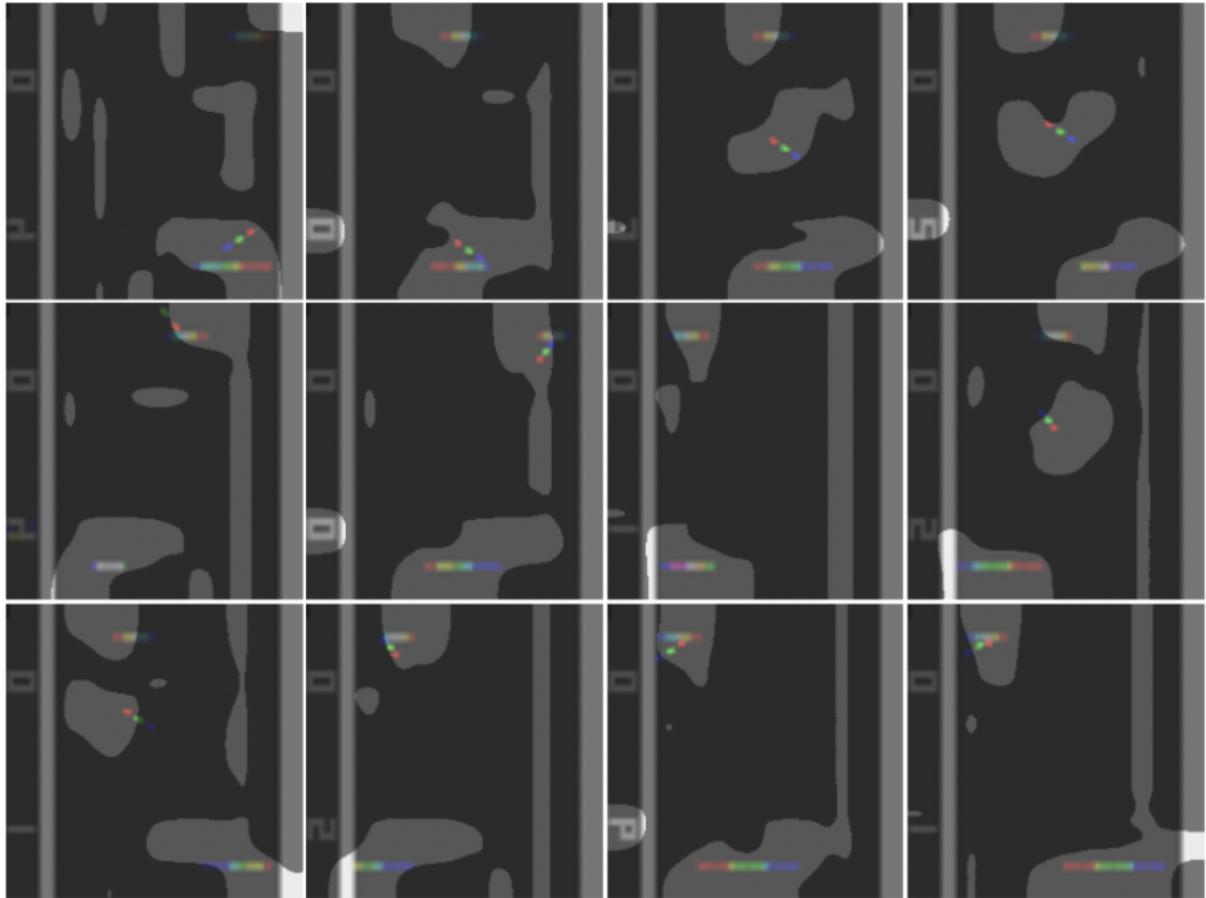


Figure 9: The top activating regions of the neuron13 in the Conv2 layer has always been only the ball and the racquets. This shows that each neuron has a fixed role in looking at certain regions of the input frame.

References

- Adebayo, J., Gilmer, J., Muelly, M., Goodfellow, I., Hardt, M., and Kim, B. (2018). Sanity checks for saliency maps. *Advances in neural information processing systems*, 31.
- Agrawal, P., Girshick, R., and Malik, J. (2014). Analyzing the performance of multi-layer neural networks for object recognition. In *Computer Vision–ECCV 2014: 13th European Conference, Zurich, Switzerland, September 6–12, 2014, Proceedings, Part VII 13*, pages 329–344. Springer.
- Alain, G. and Bengio, Y. (2016). Understanding intermediate layers using linear classifier probes. *arXiv preprint arXiv:1610.01644*.
- Ba, J., Kiros, R., Cho, K., Courville, A., Salakhutdinov, R., Zemel, R., and Bengio, Y. (2015). Tell: Neural image caption generation with visual attention kelvin xu. *arXiv (2015-02-10) https://arxiv.org/abs/1502.03044 v3*.
- Ba, J., Mnih, V., and Kavukcuoglu, K. (2014). Multiple object recognition with visual attention. *arXiv preprint arXiv:1412.7755*.
- Bueno, M. B., Giró-i Nieto, X., Marqués, F., and Torres, J. (2017). Hierarchical object detection with deep reinforcement learning. *Deep Learning for Image Processing Applications*, 31(164):3.
- Caicedo, J. C. and Lazebnik, S. (2015). Active object localization with deep reinforcement learning. In *Proceedings of the IEEE international conference on computer vision*, pages 2488–2496.
- Forsyth, D. A., Mundy, J. L., di Gesú, V., Cipolla, R., LeCun, Y., Haffner, P., Bottou, L., and Bengio, Y. (1999). Object recognition with gradient-based learning. *Shape, contour and grouping in computer vision*, pages 319–345.
- Mahendran, A. and Vedaldi, A. (2015). Understanding deep image representations by inverting them. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 5188–5196.
- Mnih, V., Heess, N., Graves, A., et al. (2014). Recurrent models of visual attention. *Advances in neural information processing systems*, 27.
- Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., et al. (2015). Human-level control through deep reinforcement learning. *nature*, 518(7540):529–533.
- Nguyen, A., Dosovitskiy, A., Yosinski, J., Brox, T., and Clune, J. (2016). Synthesizing the preferred inputs for neurons in neural networks via deep generator networks. *Advances in neural information processing systems*, 29.
- Sharif Razavian, A., Azizpour, H., Sullivan, J., and Carlsson, S. (2014). Cnn features off-the-shelf: an astounding baseline for recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition workshops*, pages 806–813.

- Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., Van Den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., et al. (2016). Mastering the game of go with deep neural networks and tree search. *nature*, 529(7587):484–489.
- Simonyan, K., Vedaldi, A., and Zisserman, A. (2014). Visualising image classification models and saliency maps. *Deep Inside Convolutional Networks*.
- Smilkov, D., Thorat, N., Kim, B., Viégas, F., and Wattenberg, M. (2017). Smoothgrad: removing noise by adding noise. *arXiv preprint arXiv:1706.03825*.
- Yeung, S., Russakovsky, O., Mori, G., and Fei-Fei, L. (2016). End-to-end learning of action detection from frame glimpses in videos. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2678–2687.
- Yosinski, J., Clune, J., Bengio, Y., and Lipson, H. (2014). How transferable are features in deep neural networks? *Advances in neural information processing systems*, 27.
- Zeiler, M. D. and Fergus, R. (2014). Visualizing and understanding convolutional networks. In *Computer Vision–ECCV 2014: 13th European Conference, Zurich, Switzerland, September 6–12, 2014, Proceedings, Part I 13*, pages 818–833. Springer.
- Zhou, B., Khosla, A., Lapedriza, A., Oliva, A., and Torralba, A. (2014). Object detectors emerge in deep scene cnns. *arXiv preprint arXiv:1412.6856*.