Visual Diagnostics for Deep Reinforcement Learning Policy Development

Jieliang Luo*, Sam Green*, Peter Feghali, George Legrady, and Çetin Kaya Koç
University of California, Santa Barbara
jieliang@ucsb.edu

***Abstract*—Modern vision-based reinforcement learning techniques often use convolutional neural networks (CNN) as universal function approximators to choose which action to take for a given visual input. Until recently, CNNs have been treated like black-box functions, but this mindset is especially dangerous when used for control in safety-critical settings. In this paper, we present our extensions of CNN visualization algorithms to the domain of vision-based reinforcement learning. We use a simulated drone environment as an example scenario. These visualization algorithms are an important tool for behavior introspection and provide insight into the qualities and flaws of trained policies when interacting with the physical world. A video may be seen at https://sites.google.com/view/drlvisual.**

***Index Terms*—reinforcement learning, cyber-physical systems, convolutional neural networks, engineering visualization**

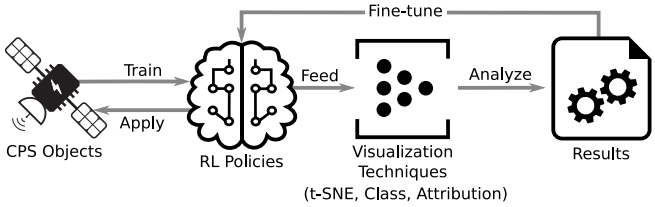


Fig. 1: Visualization development cycle when using convolutional neural networks for vision-based reinforcement learning. By iteratively visualizing *what* and *how* the CNN is perceiving, the engineer gains insight regarding *why* the RL policy makes its decisions.

I. Introduction

Reinforcement learning (RL) is a family of methods aimed at training an **agent** to collect rewards from an environment through trial-and-error approaches. Since the deep Q-network (DQN) algorithm was introduced in 2013, there has been a surge of interest in using convolutional neural networks (CNN) in vision-based RL algorithms [1]. In the context of cyber-physical systems, vision-based RL has exciting potential to provide high levels of autonomy in applications like robotics, self-driving cars, and infrastructure inspection. However, CNNs are known to be opaque to debugging and RL's emphasis on trial-and-error demands rigorous behavioral verification before they may be allowed control over safety-critical physical systems. This work adapts CNN visualization techniques to the domain of RL.

Existing CNN visualization techniques attempt to visualize classes, provide decision attribution, or cluster inputs according to their resulting label. Techniques considered here include:

- t-SNE maps [2] – Clusters similar inputs by the output classifications they trigger.
- Class visualization [3] – Generates inputs which trigger specified classifications.
- Attribution visualization [4] – Identifies image regions most responsible for a classification decision.

These techniques are useful for understanding *what* would cause a CNN to make a particular decision (through class visualization) and *why* a CNN made a particular decision (through t-SNE and attribution visualization).

* Equal contribution.

CNN visualizations have proven to be valuable for identifying strengths and weaknesses in a trained network. For example, the feature visualization for GoogLeNet's saxophone class indeed extracts a saxophone shaped object from the network. That is, the method generates an image, which, when input into the trained GoogLeNet CNN, will maximize the output probability of the saxophone class. However, when looking at the generated image, one can clearly see that the outline of a man has also been extracted from the network! The cause of this is the fact that CNNs, unlike humans, look at every pixel in the training set and will therefore extract all biases with which it was presented during training.

Existing visualization methods are powerful tools for gaining understanding and trust during CNN image classification development, but they are not adequate by themselves for use with RL. For example CNN-based RL often uses a **stochastic policy** which means that an agent's action is chosen randomly under a distribution defined by the CNN's output (i.e. "class") probabilities. Visual diagnostics for stochastic policies should capture such uncertainty. Finally, existing CNN visualizations are designed for static images, not for the types of time-series data collected when an agent interacts with an environment.

We have adapted CNN visualization techniques to the domain of CNN-based RL. We show that these tools are valuable for providing explanations regarding an agent's decision-making process and can help an engineer understand policy deficiencies. In the following section, we provide: 1) a formal introduction to the goals of RL, our methodology, and related work; 2) visualization results for simulated drone experiments; 3) conclusions and opportunities for further research.

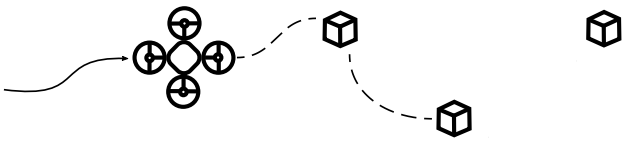


Fig. 2: Illustration of cube-collection environment. The drone policy is rewarded for "collecting" as many boxes as possible in each episode.

II. Problem formulation and related work

We have extended AirSim, a drone simulator created by Microsoft [5]. AirSim is an Unreal Engine plugin which provides physically realistic simulations for autonomous vehicles. Our extensions add support for a variety of reinforcement learning tasks.

As illustrated in Fig. 2, our AirSim environment features a drone which learns how to maneuver in order to "collect" cubes which are in front of it. At the beginning of each episode, the drone is reset to a starting point, and cubes are randomly distributed in front of it. The drone is controlled by a policy which is rewarded for collecting as many cubes as possible in each episode.

Formally, at each time step t, the drone's camera receives a partial observation of its state s_t and then makes an action a_t. The agent's policy $\pi_\theta : s_t \rightarrow a_t$ is the logic with takes state observations and returns action selections. The possible actions in our environment are: "left", "forward", and "right". After each action the environment will return a new state observation s_{t+1} and reward r_{t+1}. The policy is represented as a convolutional neural network parameterized by θ. The goal in RL is to find parameters which maximize the agent's ability to collect rewards. In a finite time-horizon, the goal is accomplished by finding CNN parameters $\theta^\star$:

$$\theta^\star = \arg\max_\theta \sum_{t=0}^{T-1} r(s_t, a_t), \tag{1}$$

where $T-1$ is the number of time steps experienced in the episode. In summary, the objective of our cube-collecting task is to find the CNN parameters to maximize the drone's ability to move toward cubes.

After each episode is finished, we use the **REINFORCE** algorithm to update the CNN parameters θ [6]. REINFORCE is an iterative algorithm which uses gradient decent to adjust θ in a direction which increases action probabilities that led to cube-collection in the prior episode. The amount of each adjustment is scaled by a **learning-rate**. This simple algorithm is ideal for the initial experiments presented here, because it allowed us to quickly understand the fundamental challenges in RL visualization.

We experiment with three algorithms to visualize the CNN policy's behavior: t-SNE, class visualization, attribution visualization.

A. t-SNE

T-Distributed Stochastic Neighbor Embedding (t-SNE) is a dimensionality reduction algorithm developed by [2]. It is well suited for visualizing high-dimensional datasets. The method positions each high-dimensional datapoint (e.g. images) in a two or three-dimensional map in a way that similar datapoints are nearby and dissimilar ones are distant. The most recent use of t-SNE is to use a trained convolutional neural network (CNN) to extract features from each image, feed the features to t-SNE to get the position of each image, and arrange the images on a 2D or 3D space based on the given positions.

B. Class Visualization

Class visualization methods generate visual inputs which activate a particular output in a *trained* neural network. This approach allows for a high-level of human comprehension about the behavior of a network, rather than treating the network like a black-box model. For our specific feature visualization approach, we use **Class Model Visualization** (CMV) [3]. CMV generates inputs which will trigger any specific output class in a trained convolutional neural network.

Formally, we let a represent the action for which we want to generate an input image to trigger, s is the input image which will be optimized such that the action probability is maximized. We let $\pi_\theta(a|s)$ represent the probability of taking action a given the image s. The goal then is to solve the following optimization problem:

$$s^* = \arg\max_s \pi_\theta(a|s). \tag{2}$$

In practice, the optimal image s^* is found using gradient ascent by an automatic differentiation tool like TensorFlow.

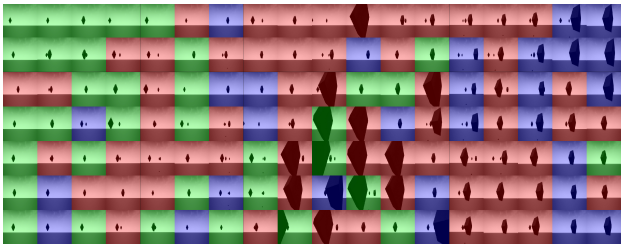
C. Attribution Visualization

Attribution visualization techniques highlight regions in an input which are most responsible for a particular action in a CNN-based policy. We will use an attribution visualization technique called **Gradient-weighted Class Activation Mapping** (Grad-CAM) [4], which highlights regions in the input image most responsible for an action probability.

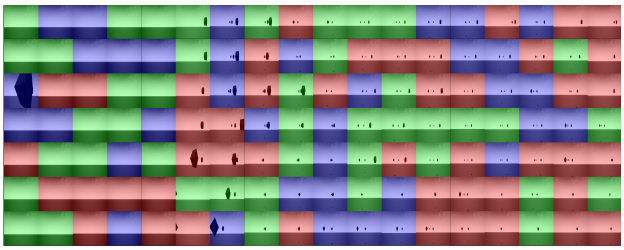
CNNs are particularly well-suited for attribution visualization, because they maintain spatial structure of the input as it flows through the network, this is why we can extract meaning from the last layer. A **feature map** is the output of a convolutional layer after it has passed through a nonlinearity (e.g. ReLU) function. Feature maps typically have many channels, and the goal of Grad-CAM is to find which channels contribute the most to an action taken. Grad-CAM achieves that goal by calculating the average derivative of the policy network, given a specific action a and input image s, with respect to the feature map of interest:

$$\alpha_k = \frac{1}{Z} \sum_i \sum_j \frac{\partial \pi_\theta(a|s)}{\partial A^k_{ij}}, \tag{3}$$

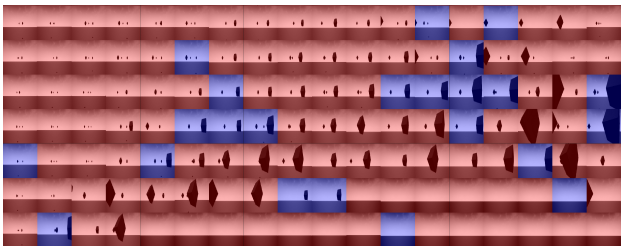
where A is the feature map of our target convolutional layer, A^k is channel k of A, A^k_{ij} is the neuron at position i, j, and $Z = i \times j$. α_k is known as the **importance weight** for feature map channel k.



(a) High-performance policy



(b) Poor-performance policy



(c) Forward-and-right-only policy

Fig. 3: T-SNE visualizations for the (a) high-performance policy, (b) poor-performance policy and (c) a policy which only moves right and forward. The tinting of each visual input is based on the action taken by the policy, given the input: red indicates "forward", green indicates "left", and blue indicates "right".

Once importance weights $\alpha_1, \alpha_2, \cdots, \alpha_K$ are known, they may be used to linearly combine feature map channels 1 through K, giving a "class activation map":

$$\text{Grad-CAM} = \text{ReLU}\Big(\sum_k \alpha^k A^k\Big), \tag{4}$$

where ReLU is being used to filter for derivatives with a positive effect.

D. Related work

There has only been a marginal amount of previous literature regarding the visualization of reinforcement learning policies. The creators of the DQN algorithm used t-SNE to cluster actions according to Atari game screens [7]. More in-depth exploration of t-SNE and Atari games was performed by [8]. Attribution visualization was used by [9] to analyze Atari video games. To the best of the authors' knowledge, this is the first work to apply t-SNE, class visualization, and attribution visualization to cyber-physical systems.

(a) High-performance policy

(b) Poor-performance policy

(c) Forward-and-right-only policy

Fig. 4: Class visualizations for the (a) high-performance policy, (b) poor-performance policy and (c) a policy which only moves right and forward. Generated images maximize action probabilities for their respective policies. The left-most image triggers the "left" drone action, middle triggers "forward", and right triggers "right".

III. Results

We trained three separate policies for the cube-collection task. A **high-performance** policy was trained such that it almost always collected all cubes. A **poor-performance** policy was trained only briefly so it never learned. And a "broken" **forward-and-right-only** policy was trained to collect only boxes in front of it or to the right, and avoiding any box to the left. We applied t-SNE, class visualization, and attribution visualization to each of the policies to demonstrate how these tools aid in policy development and understanding.

A. t-SNE

Fig. 3 provides t-SNE visualizations of the three policies. The tinting of each patch is based on the action taken by the policy, given the drone's observation: red indicates "forward", green indicates "left", and blue indicates "right".

In Fig. 3(a) we observe from the high-performance policy that the visual inputs with the same tinted color are generally clustered together by the content of the image.[1] In comparison, the poor-performance policy Fig. 3(b) scatters colors all over the map. This is because each action is equally likely, and uncorrelated with the drone's observation. In Fig. 3(c) the forward-and-right-only policy is insightful, as there are no green ("left") patches. It can also be observed that Fig. 3(c) shows mostly "forward" (red) actions and fewer "right" (blue) actions.

B. Class Visualization

The high-performance policy's class visualization in Fig. 4(a) clearly explains what the policy is looking for, where the bias towards the "left", "forward", or "right" depends on the position of the cube. Specifically, if a box blocks the view of the camera on the left, then the policy will most

[1]Clustering is not guaranteed because the policy is stochastic.

likely choose the "left" action. Similarly for the "forward" and "right" actions.

Class visualization of the poor-performance policy is also insightful. Fig. 4(b) illustrates that the actions of the poorly trained model are equally triggered by noise.

Fig. 4(c) visualizes the forward-and-right-only policy, where only the "right" action visualization makes sense. That is, when the camera is occluded on the right, the policy will choose to move to the right. The "left" action shows that there is a small response to camera occlusion on the left, but the "forward" action dominates the left action.

Class visualizations for the forward-and-right-only policy highlight one of the challenges in reinforcement learning. If the learning-rate in REINFORCE is too high, the policy might finalize its decision making process based on early experience. In this case, the drone experienced an early success by moving "right" and "forward", which resulted in the elimination of "left" action probabilities. The remedy for this was to lower the learning-rate of the policy updates.

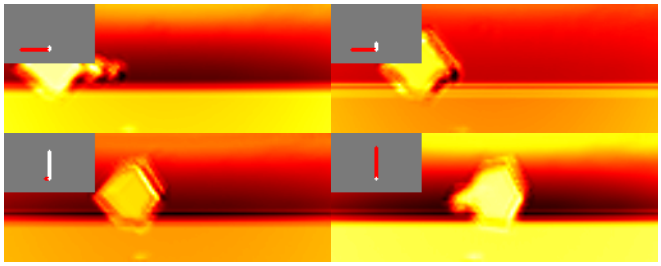
C. Attribution Visualization

Fig. 5 provides attribution visualizations. The T-shape in the corner of each image visualizes action probabilities, and the red color indicates the (stochastic) action taken. The bright areas of each image indicate image features which most contributed to the probability of the action taken.

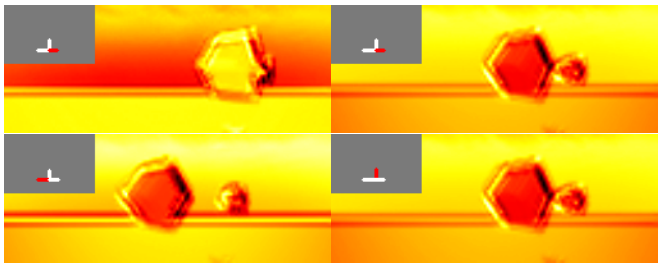
Observing the upper-left image in Fig. 5(a), we see that the "right" action was the only action with a high probability. Furthermore, the closest cube is the brightest (indicating its importance for the decision), followed by the second cube. Also note that the lower-left image visualizes what contributed to the low-probability "left" action which was made.

In Fig. 5(b) we see visualizations for the poor-performance policy. In that figure, observe that all action probabilities are roughly the same, as indicated by the T-shape, regardless of the position of the drone relative to the cubes.

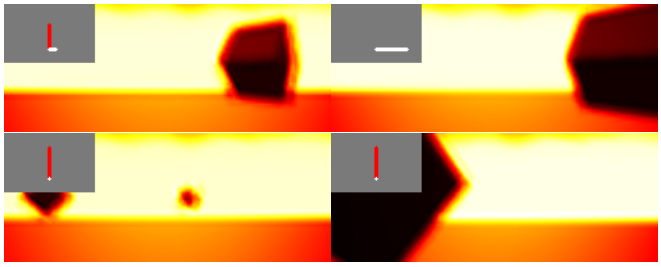
Fig. 5(c) provides attribution visualizations for a model that will only move "right" and "forward". Notice that the colors here are inverted compared to Fig. 5(a), which is odd. The policy pays more attention to the horizon than anything else. Observe that the lower-left image shows no "left" reaction to the cube on the left.



(a) High-performance policy



(b) Poor-performance policy



(c) Forward-and-right-only policy

Fig. 5: Attribution visualizations for state observations and actions taken with the (a) high-performance policy, (b) poor-performance policy, and (c) a policy which only moves right and forward. The T-shape in the corner image visualizes action probabilities, and the red color indicates action taken.

IV. Conclusions and Future Work

We see an increasing number of businesses using convolutional neural networks in vision-based cyber-physical systems. Simultaneously, the research community has been actively coupling CNNs with traditional reinforcement learning algorithms. We may soon begin to interact with RL-enabled CPS in our day-to-day lives. But this is currently dangerous because of the opacity of CNNs.

Our research is a step in the direction of creating understandable and trustworthy vision-based RL systems through policy visualization. We have adapted three existing CNN visualization techniques to our drone simulation environment: t-SNE, class visualization, and attribution visualization. But the adaptation of existing CNN visualization techniques addressed here are not adequate on their own.

Numerous opportunities exist for advancing this domain. For example reinforcement learning is inherently time-series based, it often makes stochastic decisions, and real-time visualization would be useful for some applications. Our future efforts will explore these avenues.

References

[1] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing atari with deep reinforcement learning," *arXiv preprint arXiv:1312.5602*, 2013.

[2] L. v. d. Maaten and G. Hinton, "Visualizing data using t-sne," *Journal of Machine Learning Research*, vol. 9, no. Nov, pp. 2579–2605, 2008.

[3] K. Simonyan, A. Vedaldi, and A. Zisserman, "Deep inside convolutional networks: Visualising image classification models and saliency maps," *arXiv preprint arXiv:1312.6034*, 2013.

[4] R. R. Selvaraju, M. Cogswell, A. Das, R. Vedantam, D. Parikh, and D. Batra, "Grad-cam: Visual explanations from deep networks via gradient-based localization," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017, pp. 618–626.

[5] S. Shah, D. Dey, C. Lovett, and A. Kapoor, "Airsim: High-fidelity visual and physical simulation for autonomous vehicles," in *Field and Service Robotics*. Springer, 2018, pp. 621–635.

[6] R. J. Williams, "Simple statistical gradient-following algorithms for connectionist reinforcement learning," in *Reinforcement Learning*. Springer, 1992, pp. 5–32.
[7] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, p. 529, 2015.
[8] T. Zahavy, N. Ben-Zrihem, and S. Mannor, "Graying the black box: Understanding dqns," in *International Conference on Machine Learning*, 2016, pp. 1899–1908.
[9] S. Greydanus, A. Koul, J. Dodge, and A. Fern, "Visualizing and understanding atari agents," *arXiv preprint arXiv:1711.00138*, 2017.