Transfer Deep Reinforcement Learning in 3D Environments: An Empirical Study

Devendra Singh Chaplot
School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213
chaplot@cs.cmu.edu

Guillaume Lample
School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213
glample@cs.cmu.edu

Kanthashree Mysore Sathyendra
School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213
ksathyen@cs.cmu.edu

Ruslan Salakhutdinov
School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213
rsalakhu@cs.cmu.edu

Abstract

The ability to transfer knowledge from previous experiences is critical for an agent to rapidly adapt to different environments and effectively learn new tasks. In this paper we conduct an empirical study of Deep Q-Networks (DQNs) where the agent is evaluated on previously unseen environments. We show that we can train a robust network for navigation in 3D environments and demonstrate its effectiveness in generalizing to unknown maps with unknown background textures. We further investigate the effectiveness of pretraining and finetuning for transferring knowledge between various scenarios in 3D environments. In particular, we show that the features learnt by the navigation network can be effectively utilized to transfer knowledge between a diverse set of tasks, such as object collection, deathmatch, and self-localization.

1 Introduction

Deep reinforcement learning (RL) has recently caught the attention of researchers for its effectiveness in achieving human-level performance in a wide variety of tasks, including playing Atari 2600 games [10], Go [13], high-dimensional robot control [7], and solving physics-based control problems [4].

Although it is possible to leverage the knowledge acquired from previous environments, many of the existing models are commonly trained and tested on different tasks independently [10, 15, 16], making it hard for the network to learn from previous experiences. As a consequence, some tasks turn out to be extremely challenging to learn, even though they may have good policies that would be easy to find if some previous knowledge could be transferred from other simpler environments. Furthermore, transfer learning could significantly reduce the long training times of RL models.

While there have been some recent approaches for transfer learning in Atari games [11, 3, 12], in this paper, we focus on 3D environments as they are comparatively more challenging to learn from scratch and ideal to study transfer learning as almost all scenarios require the knowledge of basic navigation. Furthermore, unlike most Atari games, states are partially observable, and the agent receives a first-person perspective, which makes the task more suitable for real-world robotics applications.

A number of previous transfer learning applications of deep RL in 3D environments often assume similar source and target environments. For example, [12] use object collection on a particular map

30th Conference on Neural Information Processing Systems (NIPS 2016), Barcelona, Spain.

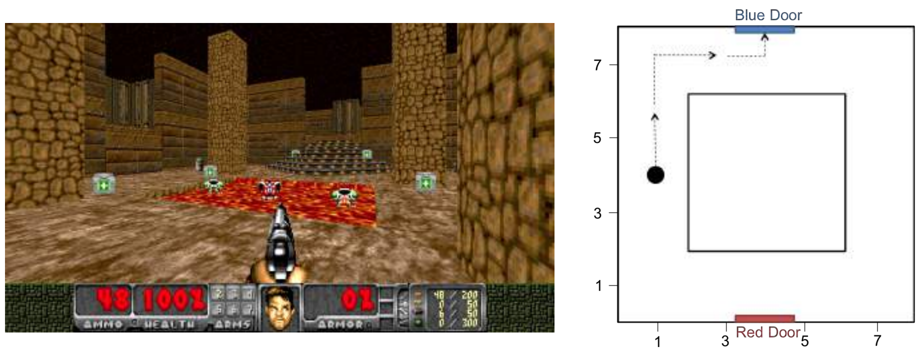

Figure 1: (a) A screenshot of Doom. [6] (b) Self-localization scenario

(e.g. an M-shaped maze) as the source task and the same objective on another map (e.g. a Y-shaped maze), having the same background texture, as the target task. Similarly, [18] show transfer learning of target-driven navigation from known to unknown scenes. In contrast, we train a navigation network that can handle unknown maps with unknown textures. We next investigate transfer learning between source and target tasks having different objectives (for instance, from navigation to object collection). Furthermore, the maps used in our experiments are much larger and more complex than simple M and Y-shaped mazes used by [12].

Our contribution in this paper is threefold: First, we study the application of Deep Q-Networks (DQNs) [10] where the agent is evaluated on previously unseen environments and is shown to be as effective as in the training environment. Second, we train a robust network for navigation in 3D environments with a simple trick of using random textures during training, and show its effectiveness in generalizing to unknown maps with unknown background textures. Finally, we investigate the effectiveness of pretraining and finetuning for transferring knowledge between various scenarios in 3D environments. Specifically, we show that the features learned by the navigation network can be utilized to effectively transfer the knowledge of navigation to a diverse and complex set of scenarios, including object collection, deathmatch, and self-localization.

2 Related Work

Transfer learning for RL algorithms has been widely studied in the past two decades. [14] provide a survey of transfer learning methods for RL prior to the introduction of deep learning for RL. In the context of deep learning, transfer learning approaches of pretraining and finetuning have been shown to be effective in multiple domains [2, 9, 17]. These methods can be adapted for transfer learning in deep RL applications. One of the closest approaches to our work is the work of [11], who pretrain a multi-task Actor-Mimic Network on several Atari games, and use the weights of this network to train a DQN on unseen Atari games after removing the final softmax layer. We employ a similar idea in the context of 3D environments in the game of Doom, but train the source network on a single task.

Transfer learning for deep RL in 3D environments has also gained some attention recently. [12] introduced an architecture called progressive neural networks, that use lateral connections for sequentially transferring learned features from previous tasks without forgetting. This approach is shown to be effective in transferring knowledge between different mazes in 3D Labyrinth environment with the same objective of object collection. Similarly, [18] introduce deep siamese actor-critic network for transfer learning from known to unknown scenes with the same goal of target-driven navigation. In this work, we focus on a setting where the source and target tasks have very different objectives: for example, transferring knowledge from navigation to the object collection, or self-localization task.

3 Background: Deep Q-Learning

Reinforcement learning deals with learning a policy for an agent interacting in an unknown environment. At each step, an agent observes the current state s_t of the environment, decides on an action a_t according to a policy π, and observes a reward signal r_t. The goal of the agent is to find a policy

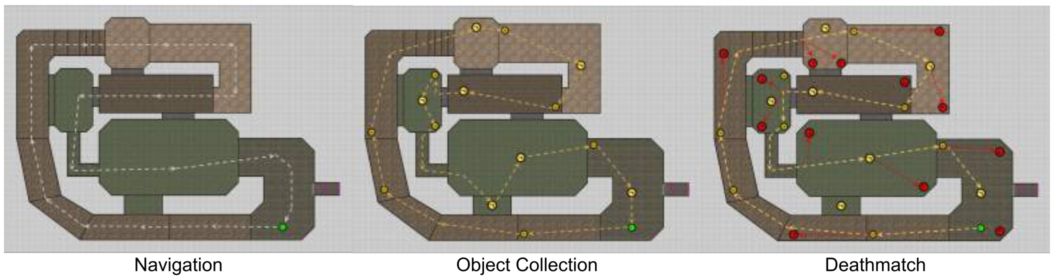

Figure 2: Figure showing the optimal sequence of actions in various scenarios. The agent is shown in green, objects in yellow and enemies in red. The dashed line shows the optimal path.

that maximizes the expected sum of discounted rewards R_t. The Q-function of a given policy π is defined as the expected return from executing an action a in a state s. It is common to use a function approximator to estimate the action-value function Q. In particular, Deep Q-Learning uses a neural network to obtain an estimate of the Q-function of the current policy which is close to the optimal Q-function Q^* defined as the highest return we can expect to achieve by following any strategy:

$$Q^*(s,a) = \max_{\pi} \mathbb{E}\left[R_t | s_t = s, a_t = a\right] = \max_{\pi} Q^{\pi}(s,a).$$

In other words, the goal is to find θ such that $Q_\theta(s,a) \approx Q^*(s,a)$. The optimal Q-function verifies the Bellman optimality equation, $Q^*(s,a) = \mathbb{E}\big[r + \gamma \max_{a'} Q^*(s',a') | s,a\big]$. If $Q_\theta \approx Q^*$, it is natural to think that Q_θ should be close from also verifying the Bellman equation. This leads to the loss function:

$$L_t(\theta_t) = \mathbb{E}_{s,a,r,s'}\left[\big(y_t - Q_{\theta_t}(s,a)\big)^2\right],$$

where t is the current time step, and $y_t = r + \gamma \max_{a'} Q_{\theta_t}(s',a')$.

Instead of performing the Q-learning updates in an online fashion, it is common to use experience replay [8] to break the correlation between successive samples. At each time step, agent experiences (s_t, a_t, r_t, s_{t+1}) are stored in a replay memory, and the Q-learning updates are carried out on mini-batches of experiences randomly sampled from the memory.

At each training step, the next action is generated using an ϵ-greedy strategy: with a probability ϵ, the next action is selected randomly, and with probability $1-\epsilon$, the next action is chosen to be the best action according to the current network. In practice, it is common to start with $\epsilon = 1$ and to progressively decay ϵ.

4 Experimental setup

In this section, we describe the various scenarios used to investigate transfer learning in DQNs. We developed these scenarios in the Doom game environment using the ViZdoom API [5] and open source Doom editor, Slade 3. Figure 1(a) shows a screenshot of Doom environment. The ViZDoom API gives a direct access to the Doom game engine and allows to synchronously send commands to the game agent and receive inputs of the current state of the game. We interacted with the Doom game engine using ACS scripts inside the Doom editor to calculate rewards for different scenarios. We plan to release the code and game scenario files for all the experiments.

4.1 Navigation in Unknown Maps

In the navigation scenario, the objective is to cover as much distance as possible in the map. The motivation behind this task is to learn to not be stuck against walls or in alternating actions (left and right). The action space contains 3 actions: “Move Forward”, “Turn Left”, and “Turn Right”. The reward at each time step is the distance travelled since the last time step. Figure 2(a) shows the optimal path in the navigation scenario.

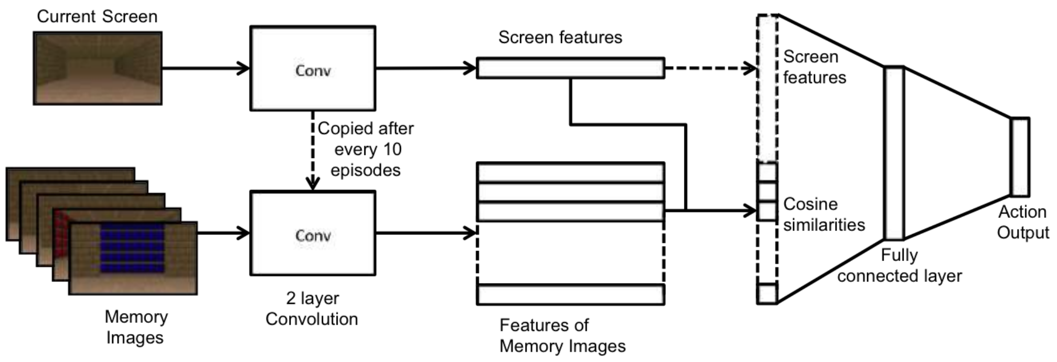

Figure 3: An illustration of the architecture of Deep Q-Network used for the Self-localization task.

In our setting, the distance information was obtained from the game engine and might not be available in other applications. However, we argue that most robotics applications will have this information from their motor sensors. In the future, we plan to estimate the distance travelled using the change in visuals seen by the agent to make the learning process robust to applications where this information is not available.

We next introduce a simple trick of using random textures on multiple maps to generalize to unknown maps with unknown textures. For each episode during training, we randomly select a map from a set of 10 training maps and then use random textures for each wall, floor and ceiling from a set of 300 textures. For evaluation, we use a set of 3 different test maps, and a separate set of 100 test textures. This means that our agent is tested to navigate in unknown maps with unknown textures. Each episode lasts 60 seconds.

4.2 Object Collection

In this scenario, the agent receives a reward for picking up objects. Different types of objects carry different rewards: +6 for a weapon, +2 for ammo and +10% of health increased (for different sizes of health packs). The agent's health is set to 50% and inventory is cleared at every time step to ensure that the agent always receives a reward when it walks over an object. A negative reward, or decreasing health, is also incorporated for walking on lava. A lava texture was only replaced by other lava or acid textures during both training and testing so that the agent can distinguish between lava and floor. Figure 2(b) shows the optimal path in the object collection scenario.

Similar to the navigation task, we use 60 seconds episodes on the set of training and test maps with random textures. The appearance and location of different objects in each map is also randomized.

4.3 Deathmatch

In the deathmatch scenario, the agent plays against built-in Doom bots on the same map with the objective to frag as many enemies as possible within a fixed time limit. When an agent dies, it immediately respawns on the same map at a different location. We used a +50 reward for a frag, -50 for a death and -100 for a suicide. We define the final score as the kill to death ratio of the agent. Figure 2(c) shows the optimal set of actions in the deathmatch scenario.

4.4 Self-localization

In the self-localization scenario, the objective is to navigate to a unique location in an ambiguous environment in order to localize. We created a simple square map for this task, containing a blue and a red door, as shown in Figure 1(b). All the other walls have the same texture, which makes the location ambiguous given just the current frame. The agent needs to navigate to the blue or the red door in order to self-localize. As opposed to navigation or object collection, this task requires some degree of high-level planning. The set of possible actions contains "Move Forward", "Turn Right", "Turn Left" and "No Action".

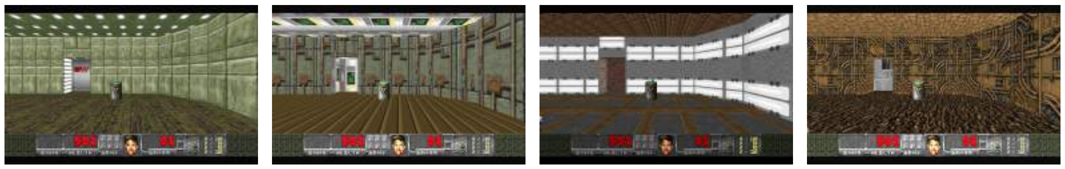
Figure 4: Examples of random textures

Since the objective is to localize, some information about the map is required by the agent. We provide 96 images of the visuals (100 $\times$ 75 images) seen by the agent at 24 locations evenly placed around the map in 4 orientations: North, South, East and West. This includes few images of both the blue and red doors. We augment the vanilla Deep Q-Network [10] with cosine similarity of the current screen features with each of the 96 "memory" image features, as shown in Figure 3. The features for the current screen are obtained by passing them through the 2 convolutional layers in the Q-network. The convolutional features for the images in the memory are fixed and updated based on the Q-network after every 10 episodes.

The episodes have a fixed length of 30 seconds. At the end of the episode, the agent needs to self-localize, or navigate to the blue or the red door (see Figure 1b). In our setting, the location prediction is the location of the image in the memory which has the highest similarity with the screen visible to the agent at the end of the episode. The agent receives a positive reward for correct predictions (correct location on the map) and a negative reward for incorrect ones. This means that the agent only receives a single reward in the whole episode, leading to a delayed reward and a sparse replay table with respect to non-zero rewards. This makes the task very challenging as it is difficult to learn what actions are responsible for which reward.

The self-localization scenario is similar to the target-driven navigation scenario considered by [18], who introduced a deep siamese actor-critic network for this task. This network learns a general embedding given the current screen and the target image and then learns scene or map specific layers to capture the layout and object arrangements in a given scene. Unlike the target-driven navigation where the target is given as input to the network, in the self-localization task the agent needs to learn to find a unique target. However, no other information about the map is given as input in the deep siamese network, while self-localization network requires a few images from the map. Our architecture for self-localization can potentially be generalized to unknown maps, while deep siamese networks need to learn a scene-specific layer for new scenes.

4.5 Hyper-parameters

We used the same architecture as the vanilla DQN [10] for all scenarios except self-localization. All networks were trained using the RMSProp algorithm and mini-batches of size 32. Network weights were updated every 4 steps, so experiences were sampled 8 times on average during the training [15]. The size of the replay memory was set to $1,000,000$. The discount factor was set to $\gamma = 0.99$. We used an ϵ-greedy policy during the training, where ϵ was linearly decreased from 1 to 0.1 over the first million steps, and then fixed to 0.1. We also used the frame-skip technique of [1]. In this approach, the agent only receives a screen input every $k + 1$ frames, where k is the number of frames skipped between each step. The action decided by the network is then repeated over all the skipped frames. A higher frame-skip rate accelerates the training, but can hurt the performance. We use a screen resolution of 320×240 in the game, resize it to 100×75 and stack the last 5 frames before passing them to the network.

5 Results

5.1 Navigation in Unknown Maps

We used a vanilla DQN[10] for training the agent in the Navigation scenario. Figure 5a shows the plot of the average reward as a function of the training time. Note that all the plots in Figure 5 are smoothed with a Gaussian kernel for better visibility. The network was evaluated every 5 minutes of training time and each evaluation consisted of 50 episodes on the test maps.

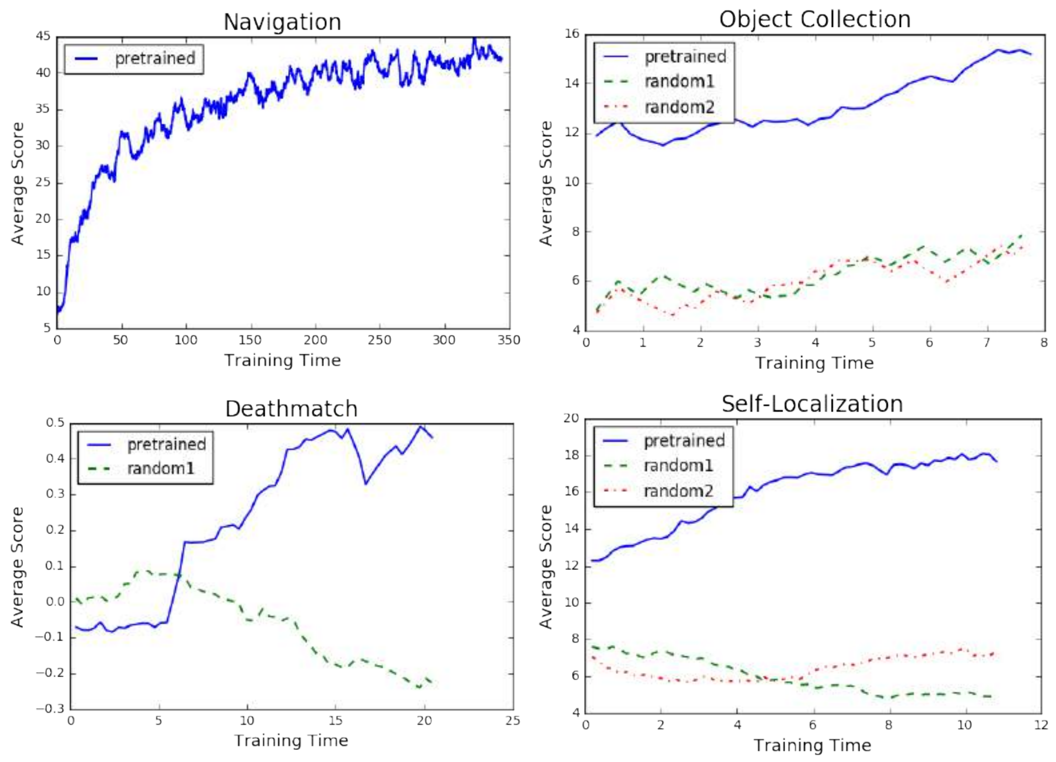

Figure 5: Plots of Average Score vs. Training time (hours) in: (a) Navigation task and Transfer Learning in 3 scenarios: (b) Object Collection, (c) Deathmatches, and (d) Self-localization

Random Textures. The navigation network was able to generalize to unknown maps containing unseen textures without any finetuning. This was achieved by a simple trick of training the network using random textures. Figure 4 shows some examples of using random textures in the same frame. We observed that this simple trick improved the performance of the network on the test maps by over 300% as compared to network trained without random textures, while the training performance was comparable. This navigation network is also a part of the Action-Navigation Architecture introduced by [6] to play deathmatches in Doom.

In the next section, we analyze the filters learned by this network and show that the network is able to detect depth as well as textures in any given given frame.

5.2 Transfer Learning

We now study the transfer of the navigation network to other scenarios in 3D environments, by simply initializing the weights of the target task network with the navigation network weights. Table 1 shows a summary of results of transfer learning on various scenarios, which we discuss in the following subsections.

Navigation to Object Collection

Figure 5(b) compares the performance of a Deep Q-Network for object collection pretrained with navigation network vs. two randomly initialised Deep Q-Networks. All networks are evaluated for 50 episodes every 5 minutes of training. The plot shows that the pretrained network performs significantly better than randomly initialized networks, while maintaining a transfer ratio of around 2 until 8 hours of training. The final object collection network pretrained with the navigation network weights is effective at collecting objects in unknown maps as shown in the demo video at https://youtu.be/8IjAT-tEG-E.

	Object Collection		Deathmatch		Self-Localization	
Score	Random	Pretrained	Random	Pretrained	Random	Pretrained
Jumpstart	4.8	11.9	0	-0.08	7.2	12.2
Final Score	8.0	15.2	-0.21	0.44	7.4	17.8
Transfer Ratio		1.9		-		2.4

Table 1: Results of transfer learning in various scenarios.

Navigation to Deathmatch

In Figure 5(c), we compare the performance of a network initialized with the Navigation filters to a randomly initialized network on the deathmatch task. The network is evaluated for 15 mins of simulated game time after every 20 minutes of real time training. The superior performance of the pretrained network indicates some positive transfer between the tasks, however the final network after 20 hours of training is not as effective as Action-Navigation Model [6]. The network struggles to recognize, aim, and shoot enemies accurately, which indicates that the knowledge of navigation is not sufficient for effectively learning to play deathmatches.

Navigation to Self-localization

Since the architecture of the self-localization network is slightly different from the vanilla DQN, we simply initialize the convolutional layers of the self-localization network with the navigation network. Figure 5(d) compares the average score of the pretrained network with two randomly initialized networks as a function of training time. All networks are evaluated for 50 episodes every 5 minutes. The best pretrained network can predict the location at the end of the episode with $\tilde{9}4\%$ accuracy. Visualizing the performance of the agent shows that it mostly follows the shortest path to the blue door and stays there in each episode[1] (see demo at https://youtu.be/pzWSjE68hRU).

6 Analysis

We now analyze the convolutional filters learned by the navigation network and discuss the reasons behind effective generalization to unknown maps and transfer to new scenarios. The convolutional filters indicated that the last frame is most important for the scenarios considered in this paper. Figure 6 compares some of the convolutional features learnt by the navigation, object collection, and self-localization networks corresponding to the RGB layers of the last frame. Even though both the object collection and self-localization models were initialized with the navigation network, the similarity of the fine-tuned features indicates that the filters learnt by the navigation network are extremely useful for object collection and self-localization tasks.

We also analyze the similarity of features for different images in the self-localization scenario. We took 500 images in random orientations at random locations in the self-localization map. We then compared the similarity of different frames with these 500 images by plotting them as a heatmap according to their x-y coordinates as shown in Figure 7. The frame containing a long corridor is similar to the images from locations that have long corridors, while the frame where the agent is facing a wall is similar to the images that face the wall. The figure also shows that the frame containing the blue door is only similar to images containing blue doors.

We further observed that the agent exploring with a randomly initialized network, spends most of its time facing the wall and is therefore not able to learn much. In contrast, the agent pretrained with a navigation network is much more efficient at exploring the environment and finding objects or blue/red doors, which substantially improves the learning speed.

[1]The agent does not tend to go to the red door a lot perhaps because it is easier to disambiguate the blue door from the brown walls as compared to the red door.

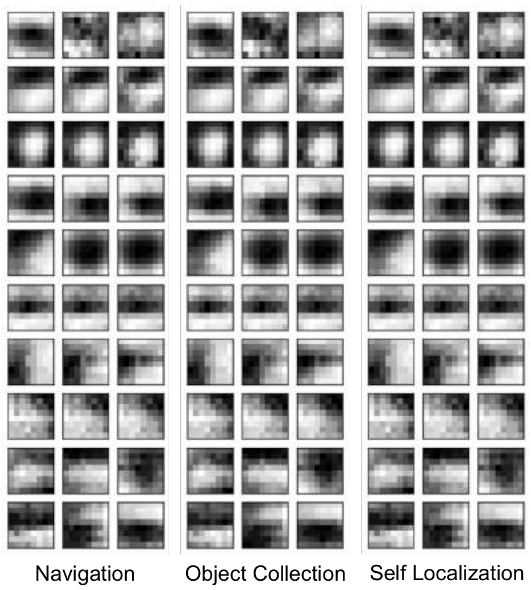

Figure 6: Visualization of the (subset of the) first convolutional layer filters for the RGB layers (right to left) of the last frame in (a) Navigation network (b) Object Collection network and (c) Self-localization network. Note that the filters are very similar in all the scenarios which may explain the effectiveness of transfer learning.

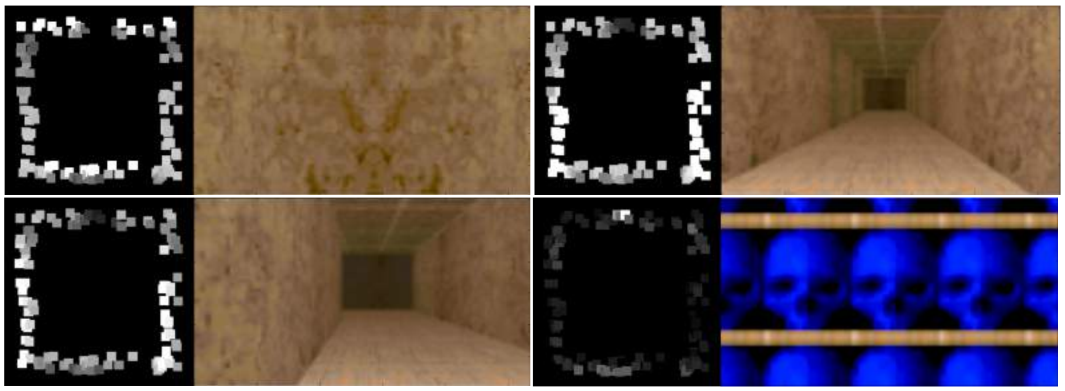

Figure 7: Visualization of the heatmap (on the **left**) of the cosine similarity of the given input frame (on the **right**) with 96 images in the memory in the north orientation, plotted according to their x-y coordinates on the map (Figure 1(b) shows the self-localisation map). The heatmap indicates that the navigation network is able to differentiate between images containing a long corridor vs. images that face a wall, and between images having different textures.

7 Conclusion & Future Work

We showed that we can train a robust network for navigation in 3D environments by training on multiple maps with random textures. We demonstrated that this navigation network is able to generalize to unknown maps with unseen textures.The features learnt by this navigation network are shown to be effective in a diverse set of tasks, including object collection, deathmatch, and self-localization. In our future work, we plan to estimate the distance travelled using just the change in visuals seen by the agent rather than extracting this information from the game engine, in order to make the learning process robust to applications where this information is not available. We further plan to expand the self-localization scenario to handle multiple maps and generalize to new unknown maps, as current experiments used only a single map.

References

[1] Marc G Bellemare, Yavar Naddaf, Joel Veness, and Michael Bowling. The arcade learning environment: An evaluation platform for general agents. *Journal of Artificial Intelligence Research*, 2012.

[2] Yoshua Bengio et al. Deep learning of representations for unsupervised and transfer learning. *ICML Unsupervised and Transfer Learning*, 27:17–36, 2012.

[3] Yunshu Du, V Gabriel, James Irwin, and Matthew E Taylor. Initial progress in transfer for deep reinforcement learning algorithms.

[4] Nicolas Heess, Gregory Wayne, David Silver, Tim Lillicrap, Tom Erez, and Yuval Tassa. Learning continuous control policies by stochastic value gradients. In *Advances in Neural Information Processing Systems*, pages 2944–2952, 2015.

[5] Michał Kempka, Marek Wydmuch, Grzegorz Runc, Jakub Toczek, and Wojciech Jaśkowski. Vizdoom: A doom-based ai research platform for visual reinforcement learning. *arXiv preprint arXiv:1605.02097*, 2016.

[6] Guillaume Lample and Devendra Singh Chaplot. Playing FPS games with deep reinforcement learning. In *Thirty-First AAAI Conference on Artificial Intelligence*, 2017.

[7] Sergey Levine, Chelsea Finn, Trevor Darrell, and Pieter Abbeel. End-to-end training of deep visuomotor policies. *Journal of Machine Learning Research*, 17(39):1–40, 2016.

[8] Long-Ji Lin. Reinforcement learning for robots using neural networks. Technical report, DTIC Document, 1993.

[9] Grégoire Mesnil, Yann Dauphin, Xavier Glorot, Salah Rifai, Yoshua Bengio, Ian J Goodfellow, Erick Lavoie, Xavier Muller, Guillaume Desjardins, David Warde-Farley, et al. Unsupervised and transfer learning challenge: a deep learning approach. *ICML Unsupervised and Transfer Learning*, 27:97–110, 2012.

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

[11] Emilio Parisotto, Jimmy Lei Ba, and Ruslan Salakhutdinov. Actor-mimic: Deep multitask and transfer reinforcement learning. *arXiv preprint arXiv:1511.06342*, 2015.

[12] Andrei A Rusu, Neil C Rabinowitz, Guillaume Desjardins, Hubert Soyer, James Kirkpatrick, Koray Kavukcuoglu, Razvan Pascanu, and Raia Hadsell. Progressive neural networks. *arXiv preprint arXiv:1606.04671*, 2016.

[13] David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *Nature*, 529(7587):484–489, 2016.

[14] Matthew E Taylor and Peter Stone. Transfer learning for reinforcement learning domains: A survey. *Journal of Machine Learning Research*, 10(Jul):1633–1685, 2009.

[15] Hado Van Hasselt, Arthur Guez, and David Silver. Deep reinforcement learning with double q-learning. *CoRR, abs/1509.06461*, 2015.

[16] Ziyu Wang, Nando de Freitas, and Marc Lanctot. Dueling network architectures for deep reinforcement learning. *arXiv preprint arXiv:1511.06581*, 2015.

[17] Jason Yosinski, Jeff Clune, Yoshua Bengio, and Hod Lipson. How transferable are features in deep neural networks? In *Advances in neural information processing systems*, pages 3320–3328, 2014.

[18] Yuke Zhu, Roozbeh Mottaghi, Eric Kolve, Joseph J Lim, Abhinav Gupta, Li Fei-Fei, and Ali Farhadi. Target-driven visual navigation in indoor scenes using deep reinforcement learning. *arXiv preprint arXiv:1609.05143*, 2016.