Mechanistic Interpretability of Reinforcement Learning Agents

Tristan Trim
University of Victoria
Victoria BC, Canada
tristantrim@gmail.com

Triston Grayston
University of Victoria
Victoria BC, Canada
graystontriston@gmail.com

***Abstract*—This document is a model and instructions for LaTeX. This and the IEEEtran.cls file define the components of your paper [title, text, heads, etc.]. *CRITICAL: Do Not Use Symbols, Special Characters, Footnotes, or Math in Paper Title or Abstract.**

***Index Terms*—component, formatting, style, styling, insert**

I. Introduction

Mechanistic interpretability is an emerging field within artificial intelligence research, dedicated to the reverse engineering of neural networks to understand their internal mechanisms after they have been trained to perform specific tasks. This area of study aims to unravel the complexities of how neural networks process information and make decisions, which is often considered a "black box" due to the intricate and opaque nature of these models. By dissecting the inner workings of neural networks, researchers can gain insights into the underlying principles that guide their functioning, potentially leading to more transparent, robust, and trustworthy AI systems.

This project continues the exploration of the models introduced in "Goal Misgeneralization in Deep Reinforcement Learning" [3]. The paper explored the effects of training a reinforcement learning agent on an environment modified to exhibit certain behaviours that would not be present in testing to explore what effects that might have on the agent itself. This includes having the goal exclusively exist in a specific area during testing, and having redundant actions. That paper explored the effects this would have on the model via primarily statistical methods. Our goal is to delve into pre-trained models and uncover the strategies and patterns employed to navigate and solve mazes using common methods found in the field of mechanistic interpretability.

Through this investigation, we hope to contribute to the broader field of AI interpretability by providing a detailed case study of a reinforcement learning model with well studied misgeneralization. Our findings could have significant implications for the development of more interpretable and explainable AI systems, which are essential for applications where transparency and accountability are paramount.

II. Related Works

The study "Leveraging Procedural Generation to Benchmark Reinforcement Learning" [1] introduced the Procgen Benchmark, a suite of 16 procedurally generated game environments specifically designed to evaluate reinforcement learning (RL) algorithms. These environments are characterized by their diversity and complexity, which are crucial for testing the generalization and robustness of RL agents. Each game environment in the Procgen Benchmark is created using procedural generation techniques, ensuring that every instance presents a unique scenario. This uniqueness is vital for benchmarking because it prevents overfitting to specific game layouts and promotes the development of RL models that can generalize across varied tasks. The benchmark includes a wide range of game genres and difficulty levels, providing a comprehensive platform for assessing the performance of different RL algorithms under diverse conditions.

Building on the Procgen Benchmark, the research presented in "Goal Misgeneralization in Deep Reinforcement Learning" explored the concept of goal misgeneralization [3]. Goal misgeneralization occurs when an RL agent, although capable of performing tasks outside its training distribution, ends up pursuing incorrect objectives. This misgeneralization poses significant challenges, as it indicates that the agent has not correctly internalized the intended goals of its tasks. The study extended the Procgen environments to investigate this phenomenon, using them as a testbed to analyze how and why RL agents might misgeneralize their goals. Through systematic experimentation, the researchers demonstrated instances where RL agents, despite showing competence in task execution, failed to align their behavior with the desired goals, highlighting the need for better training methodologies and evaluation metrics to ensure that RL agents not only learn effectively but also generalize appropriately to new situations.

In the subsequent work "Understanding and Controlling a Maze-Solving Policy Network," researchers selected a partially misgeneralizing policy network trained on the Procgen maze environment for in-depth mechanistic exploration [2]. The maze environment utilized in this study employs Kruskal's algorithm to generate complex maze structures. These mazes are presented as images, with a mouse icon representing the player's position and a cheese icon representing the goal. The study aimed to dissect the internal workings of the policy network, which had shown partial goal misgeneralization, to understand the specific pathways and decision-making processes it used to solve the mazes. By employing

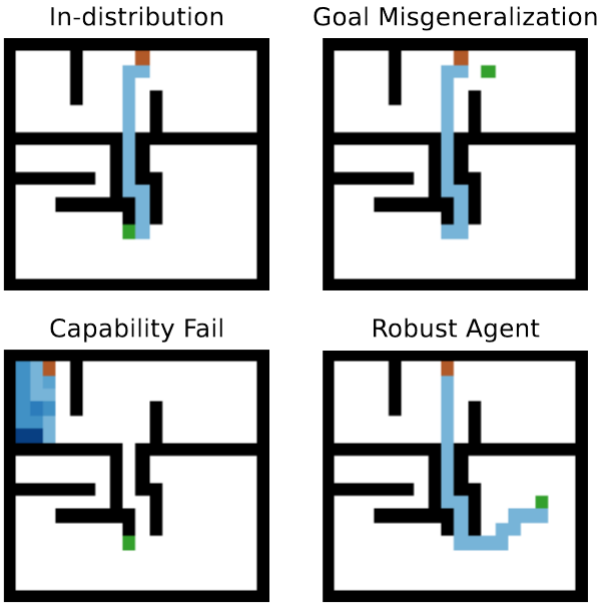


Fig. 1: Goal Misgeneralization Pic from [3]. Red is agent, green is the goal

techniques such as activation patching, the researchers were able to manipulate specific neuron activations within the network, effectively controlling its behavior. This approach not only provided insights into the network's decision-making process but also demonstrated the potential of mechanistic interpretability techniques to influence and correct the behavior of RL models.

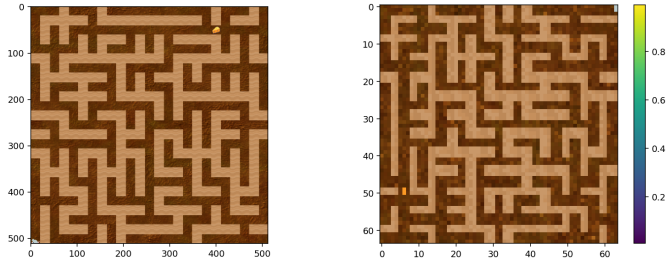
Our project aims to extend the foundational work laid out by these studies by employing a vast array of additional interpretability techniques beyond those already covered. We intend to apply a comprehensive suite of methods, including but not limited to saliency mapping and feature mapping to uncover deeper insights into the inner workings of the Impala network. By leveraging these advanced techniques, we hope to discover new aspects of the network's functionality and decision-making processes that were not previously understood. This extensive application of interpretability methods is intended to provide a more thorough and nuanced understanding of how RL models, like the Impala network, navigate and solve complex tasks such as maze environments.

III. Experimentation

This project has us conduct various forms of experimentation on an Impala model pre-trained on a fairly simple environment of procedural generated mazes.

A. Feature Mapping

The introduction of convolutional networks marked a significant milestone in machine learning, especially for image processing. This innovation enabled models to effectively understand spatial relationships between pixels within localised



(a) Upscaled Maze (b) Pre-Processed Maze

Fig. 2: Mazes

regions of an image. Additionally, convolutional networks offer greater interpretability compared to traditional dense layers, as their outputs are transformed versions of the input images. These visualisations depict kernel activation's, allowing us to understand the specific features each kernel captures from the input image upon a visual inspection.

The model that we're working with has 15 convolutional layers comprising the majority of the model's architecture. We used this as a very preliminary exploration into the model, trying to find redundancies and notable patterns that may arise from the model itself.

This section works with a 512x512x3 image which is then pre-processed into a 64x64x3 grid, and transposed as it goes into the model. These are shown in Fig 2, and we use them as a ground truth for the rest of this section.

Our initial exploratory work focuses on the first layer of the neural network, a crucial step in understanding the foundational patterns recognized by the model. In the context of our task, which involves navigating a maze, we hypothesize that this initial layer would capture the most elementary features present in the environment. These features likely include the walls delineating the boundaries of the maze, the pathways allowing navigation, the location of the cheese (as a reward signal), and the positions of the mice (agents).

To confirm our hypothesis, we analyze the activations of the neurons in the first layer. Activations refer to the responses of neurons to specific inputs. By visualizing these activations, we can infer what features the neurons are responsive to. Our results indicate that certain neurons are indeed selectively responsive to the walls and general structure of the maze. This is evident from the activation maps, which highlight the presence of linear and angular patterns corresponding to the walls and pathways. These are shown in Fig 3.

As a general rule, regions with higher activation levels are represented by the color green in our visualization. We have chosen to employ a diverging colormap rather than a sequential one, as it more effectively highlights both the high and low activation areas. This choice facilitates the differentiation between various levels of neural activity, thereby making it easier to identify significant patterns within the data. A diverging colormap is particularly advantageous for detecting subtle variations in activation that might otherwise be overlooked with a sequential colormap.

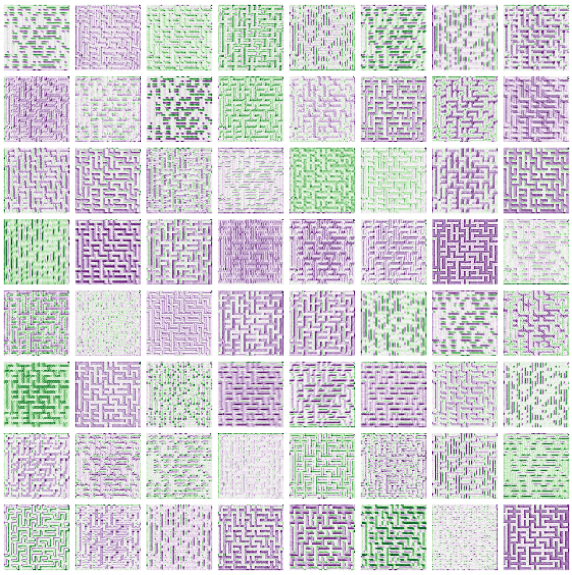


Fig. 3: First Convolutional Layer Activations. Green = Higher Activations.

Interpolating the data at the first convolutional layer turns out to be a simple task. Activation's are found pretty exclusively at either walls, paths, or corners. What may be interesting to point out, is that at this level there are no obvious kernels that light up specifically around the mouse (agent) or the cheese (goal) despite those being unique pixels highlighting some sort of functionality.

That being said, the true power of convolutional neural networks lies in the intermediate layers, where patterns detected by lower-level kernels are combined and refined. This process results in more sophisticated and abstract representations, significantly enhancing the model's ability to comprehend and represent the underlying data.

The next layer we look at is the convolutional layer hidden in the next impala block, referenced in [**?**]. The impala blocks themselves contain pooling and res-net layers, and as a result of this processing the resulting activation maps become significantly harder to interpret visually. We can still, discern several noteworthy observations.

It appears that, in contrast to the earlier layer, the neural network begins to distinctly identify the mouse (actor) within the environment, shown in Figure 4.

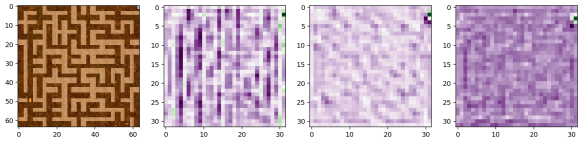


Fig. 4: Feature Maps Identifying Region Around Cheese

Several feature maps exhibit notable activations in the vicinity of the top right corner, which corresponds to the mouse's location. Interestingly, however, the peak activations are not situated precisely at the top right pixel, where the mouse is positioned in the environment. Instead, these peak activations seem to be slightly displaced downward, occurring just below the actual position of the mouse. Whether this shift is attributable to the decreased resolution of the feature maps or another underlying factor remains unclear.

It is particularly noteworthy that no feature maps exhibit a similar focused activation pattern for the cheese, which serves as the goal in the environment. This lack of recognition for the cheese is intriguing, as we observe no singular feature map that is activated exclusively at the cheese's location. This poses an interesting question: why would arguably the most unique point on the map be effectively ignored by the model at this level? We conducted several reasons for why this might be the case. The most immediate hypothesis is this being a direct result of goal misgeneralization. That is to say, the cheese in training may be ignored entirely, as it is not a feature necessary for prediction due to it's similar spawning locations. Another hypothesis is that the resolution at this level might obscure the features necessary to recognize the cheese. The network might require a combination of higher resolution and more context-specific information, which is typically processed in deeper layers, to identify the goal accurately. This seems to support previous works exploration on channel 55 [2].

As we delve deeper into the neural network, the subsequent layers become increasingly complex and challenging to interpret visually. The intricacy of the feature maps in these deeper layers necessitates the adoption of more sophisticated analytical techniques to understand the network's behavior and decision-making processes.

To streamline the exploration of these deeper layers, it is essential to first comprehend the underlying mechanisms by which mazes are generated and structured.

B. Systemic Maze Investigation.

"Leveraging Procedural Generation to Benchmark Reinforcement Learning" [1] introduces the Procgen Benchmark, a suite of 16 procedurally generated environments designed to evaluate the sample efficiency and generalization capabilities of reinforcement learning (RL) agents. The core motivation behind Procgen Benchmark is to address the limitations of traditional benchmarks, such as the Arcade Learning Environment (ALE), which can lead to overfitting due to their static and repetitive nature. By employing procedural content generation, Procgen Benchmark ensures a near-infinite supply of highly varied and randomized environments, challenging RL agents to develop robust policies that generalize well across different scenarios.

In this project, we utilize procedurally generated mazes within a Python virtual environment (venv) to create a dynamic and challenging setting for training neural networks. The mazes are designed to have one clear path from the agent (mouse) to the goal (cheese), ensuring a consistent objective across different trials. The maze generation process employs various algorithms that ensure a single, solvable path, enhancing the training efficiency and consistency of the reinforcement

learning models. These mazes are first created at a high resolution to capture intricate details of the environment.

Once the mazes are generated in some grid format, their high-resolution images are processed by upscaling to enhance visibility and detail. The mazes are then downscaled to a 64x64x3 grid, standardizing the input size for the neural network. This downscaling process reduces the image complexity while preserving essential features, such as walls and paths.

To introduce variability and prevent overfitting, different seeds are used to generate mazes of varying sizes and configurations. Each seed produces a unique maze, ensuring that the neural network is exposed to a diverse set of environments. This diversity is crucial for developing a model that can generalize well to new, unseen mazes. Additionally, the project includes an interactive tool that allows users to custom-make mazes of a specified size, as shown in 5

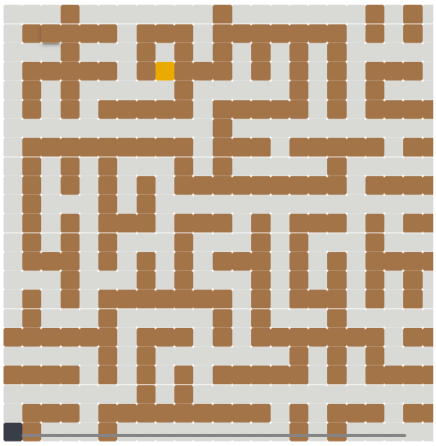


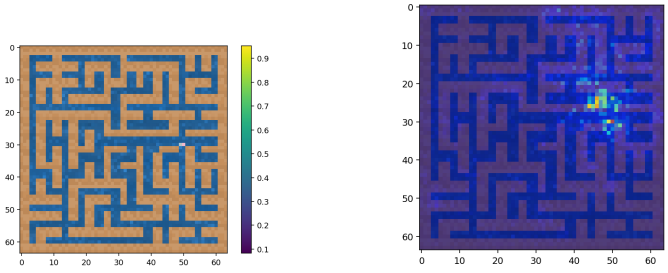
Fig. 5: Interactive Maze Plotter. (Yellow is cheese, black is mouse)

This tool provides a hands-on approach to designing maze environments, offering flexibility and control over the training scenarios. Users can create mazes tailored to specific research needs or difficulty levels, further enriching the training dataset.

C. Saliency Mapping

Saliency mapping is a technique used in neural networks to highlight the regions of an input image that contribute most to the model's prediction. By generating a heatmap over the input, saliency maps indicate which pixels or features the model focuses on, providing insights into the model's decision-making process. This visualization is particularly useful for interpreting and debugging models, as it helps identify potential biases, understand feature importance, and ensure that the model is learning relevant patterns rather than spurious correlations.

Our goal with saliency mapping is to identify specific navigational behaviors of the neural network. We hypothesize that due to the methods of training, the network inherently prefers navigating to the top right corner of the maze, even in the absence of the cheese. This would indicate if the model has developed a bias or default strategy favoring that corner as an



(a) Maze With No Cheese (b) Saliency Map of Maze

Fig. 6: Saliency Mapping of a Maze with No Goal (Cheese)

optimal path. Removing the cheese from the map and feeding it back into the model, we can see that given complete freedom of movement, the mouse will still prioritize movement to the top-right corner. Given that the model was trained with the PPO algorithm [9] we can extract exact probabilities of how likely the model is to make certain moves.

It is important to note, this model was trained with redundancies on purpose to increase the complexity of generalization. One of the ways in which this was enforced was in a redundant moveset. The agent has 5 effective actions, being all non-diagonal directions and an action that does nothing. The model however, has distinctly 15 different outputs. The largest redundancy is having 7 distinct outputs that do nothing. This is an important note from here on out, as the PPO architecture creates a probability distribution over all of these actions despite redundancies in those actions. Saliency mappings have been manufactured for both singular model outcomes (1 of 15) as well as overall action probabilities (A weighted sum of saliency mapping for the action of moving in a certain direction) for particularly notable examples.

In the scenario depicted in figure 6, the model exhibits a tendency to move 'UP' with a probability exceeding 80%, even in the absence of cheese. This behavior suggests that the model has developed a preference for moving upwards, which would be consistent in the agent striving towards the top right corner. This dawns the question, if these training mazes often featured successful outcomes associated with moving 'UP', the model could have encoded this direction as a favorable action, regardless of if that is moving towards the top right corner. We then can test if the actions of moving up, or moving to the top right corner are preferable. We test this with the map shown in Figure 7, where the model can either move up, a move it might find favourable, or left, a move that achieves a step closer to the goal. Indeed, it decides to go left with 96% probability.

Secondly, we seek to assess whether the network can be directed to move towards the top right corner even when the cheese, the primary goal, is located in the opposite direction. It will indicate the network's ability to prioritize alternative objectives over the default reward-based goal.

Initial tests involve placing the cheese in a region of the map distant from the top right corner to evaluate the model's

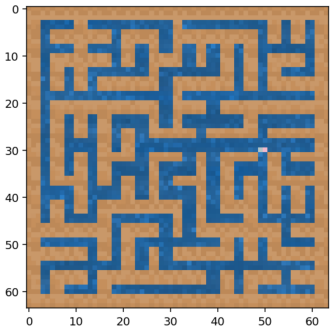
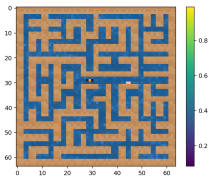
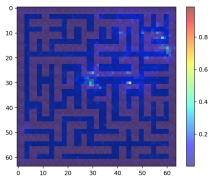


Fig. 7: Map Testing Action Probabilities

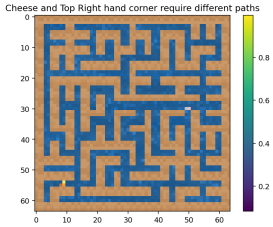


(a) Cheese Equidistant from Top Right

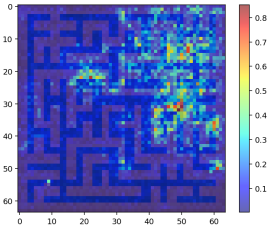


(b) Saliency of Equidistant Map

Fig. 9: Saliency Map of Mouse Equidistant from Cheese and Top Right Boundary

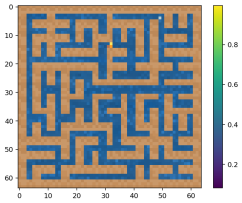


(a) Maze With Cheese in Bottom Left

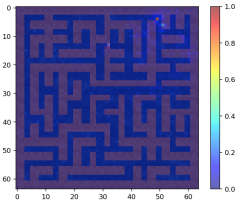


(b) Saliency Map of Maze

Fig. 8: Saliency Mapping with Cheese in Bottom Left



(a) Cheese Bottom Left



(b) Saliency Map

Fig. 10: Saliency Lack of Activation to Cheese after being in Top Right Corner

perception of its objective. If the model prioritizes the cheese in this scenario, it would suggest minimal misgeneralization, indicating that the model correctly identifies and pursues the primary goal. Conversely, if the model prioritizes the top right corner instead, it would indicate a significant misgeneralization, revealing a potential bias in its learned navigation strategy.

Figure 8 shows that the model heavily highlights the top right region of the map, noting both the mouse's current location and a broad area in the top right corner. The agent predicts it will move 'RIGHT' with a 69% probability. Interestingly, the agent does recognize the cheese when making this decision, but to a minimal extent. This observation led us to question whether the model perceives both regions as 'goals' and if it uses an intrinsic metric to compare the time steps needed to reach these goals.

One might start to question the model's process of reasoning. The prioritization of the top right hand corner might be due to the large distance away it is from the cheese. Further, there exists a possibility that the top right corner serves as a subgoal for the cheese is intriguing because it suggests a hierarchical approach in the model's navigation strategy. This would imply that the model might be using intermediate targets to simplify the pathfinding task, breaking down the journey into more manageable segments.

We can test both of these hypothesise with figures 9 and 10.

Figure 9 illustrates an intriguing behavior of the model, where despite high activation near the cheese, the agent initially moves in the opposite direction towards the top right corner. The observed pattern implies that the presence of the cheese in the environment influences the agent's movement towards the top right. This result, at first, is quite un-intuitive. A high activation of the goal resulting in an action moving in the exact opposite direction may hint at a model's inability to orient itself. Yet, that consideration does not stay consistent with the agent itself, as previous studies show the mouse can travel to the top right hand corners consistently with no orientation problems causing traversal into dead ends. This caused us to consider the possibility of hierarchical sub goals. This decision to move right instead of directly towards the cheese, which shows minimal activation via the saliency maps, suggests that the model might treat the top right corner as a sub goal.

If our agent has established the top right corner as a hierarchical subgoal, one would expect it to transition towards the cheese after reaching that subgoal. We can test this theory using Figure 10, which shows the agent positioned in the top right corner. Surprisingly, even in this position, the agent does not follow the straightforward path to the cheese, and instead decides to move down with a probability of roughly 51%. Furthermore, it has a combined probability of less than 10% of moving left, towards the cheese.

D. Interactive Distribution Coloring

Feature mapping has the advantage that it very directly shows you what the neural network is doing, but it has a shortcoming which can be explained by analogy. In block printing, multiple blocks are carved to be used as stamps, each adding their own color to the image that is printed. Fig 11 shows what these blocks might look like for the input image

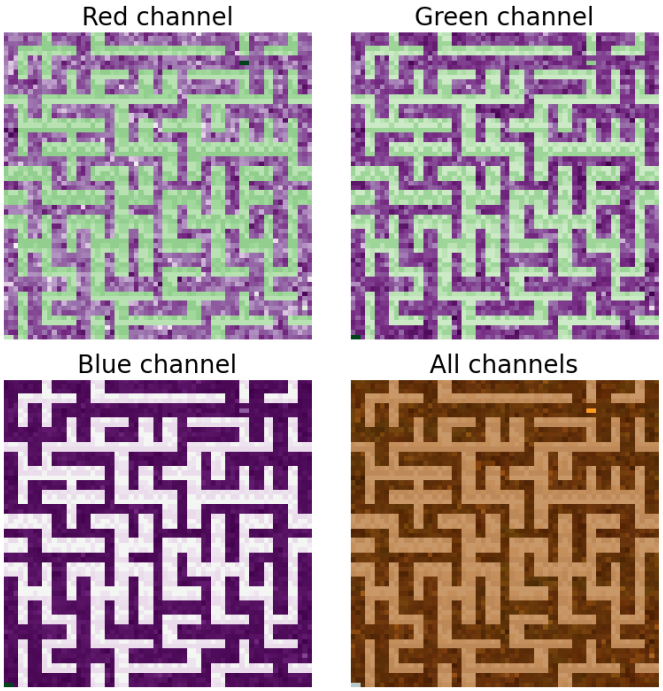


Fig. 11: The input image is shown separated into individual channels as is done in feature mapping, and is shown with all channels together. Notice that with all channels together the 4 classes of pixels (block, floor, mouse, and cheese) can easily be seen. To distinguish them in the other images requires a comparison of the pixel values between images. In general, you don't know how close two pixels are to the same color until you have compared their values across all channels.

to our policy network. Each convolutional layer transforms the image, taking it from 3 channels to 64 and then to 128, but it remains an image that could be printed on a single image if we could see 64 color channels rather than only 3. The key insight is to notice about the input that there are really only 4 colors, not $\mathbb{R}^3$. Or rather, The pixel values cluster around 4 locations in red-green-blue-space: dark brown, light brown, yellow, and gray. If the same thing is true of the images produced by the 64 "printing blocks" of feature mapping, then we could identify the clusters and color pixels based on which cluster they are in. What this looks like for the input image can be seen in Fig 12.

To explore this concept, we have developed a methodology involving 2 novel tools we have developed: PixCol, an interactive pixel class coloring tool, and NDSP, an n-dimensional scatter plot and distribution classification explorer. As an overview, the method involves: 1) using a clustering method in color space to classify the pixels, 2) using PixCol to color and label each pixel class, 3) using NDSP for viewing and editing the classification in the pixel distribution. Finally, that classification and coloring can be used as a reference for iterating the process on the next layer of the neural net.

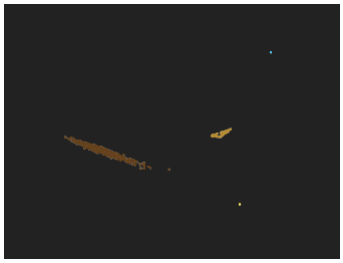


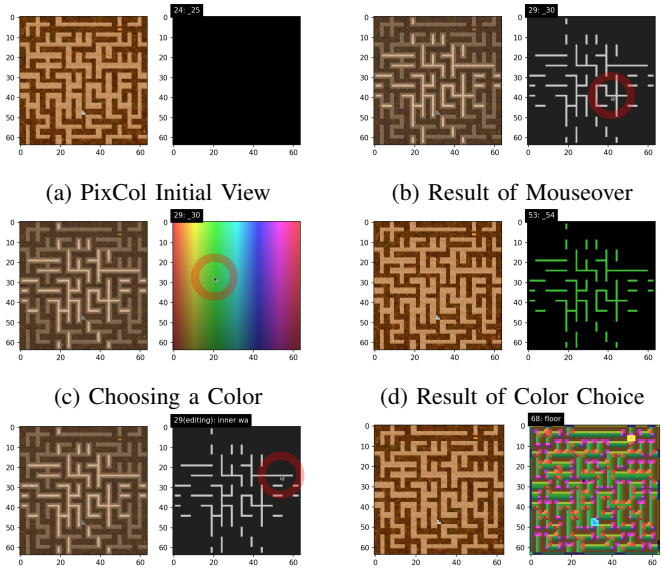
Fig. 12: The distribution of pixel colors found in the input. The dark brown floor varies the most. Both the cheese and mouse appear to be a single color, and thereby occupying a single point in red-green-blue-space.

1) Pixel Clustering: The output of any conv layer will be a tensor with shape = (number of dimensions, pixel width, pixel height). To cluster the pixels, the width and height are flattened together, and each pixel is treated as an independent datum living in $\mathbb{R}^{\#channels}$. The pixels can then be classified by any clustering method. We explored using hierarchical clustering, k-means, and manually clustering the points with NDSP.

Each of these methods seemed to have it's advantages. At the shallow layers, hierarchical clustering was fast and very accurate at giving clusters with meaningful semantics, while k-means tended to cut clusters, putting pixels into classes with other pixels seeming to represent dissimilar things. In both cases manual classification with NDSP took a great deal of effort, and gave better results with enough effort, however due to the nature of working with $\mathbb{R}^{64}$, human error often introduced different kinds of errors than seen in the clustering algorithms. Because of this, the best approach seems to be classifying with a computer algorithm and then manually editing the classification with NDSP to remove noticeable misclassification.

2) Pixel Coloring: Once we have a classification of pixels, we can use PixCol to visualize it. PixCol is an interactive plot created in jupyterlab with matplotlib that lets you easily create a colored image based on the information in the pixel classification and in the previous layers.

Fig 13a shows the view after initialization. On the left, the reference layer is shown. This can be the previous layer, an earlier layer, or any other image that may be useful to cross reference. On the right is the view of the current classification image. Since it has just been initialized, all the pixel classes have been assigned black, and the whole image appears black. When the user hovers their mouse over a pixel, all of the pixels in the same class are highlighted in white, as can be seen in Fig 13b. The user may click on the class to select a color for it. Selecting a color and the resulting change to the image is shown in Fig 13c and Fig 13d. While hovering over a class, the user may type in a label and press enter to assign it to the pixel class, Fig 13e. Finally, Fig 13f shows the results of fully coloring the classification for the output of the block1.conv layer. These results are discussed in more depth in **XXX LATER SECTION XXX**.



(a) PixCol Initial View

(b) Result of Mouseover

(c) Choosing a Color

(d) Result of Color Choice

(e) Typing Label

(f) Fully Colored

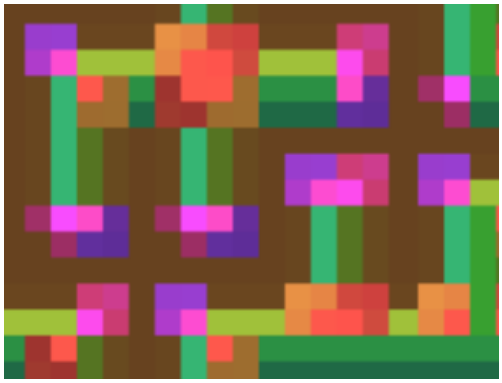
Fig. 13: PixCol usage.

3) Pixel Distribution: Having put a color to every class and understood what conditions in the previous layer cause them may give the impression that we have the full picture. However, these are flat colors. The clusters they represent are not flat. They may be closer or further from any other nearby cluster, or they may have been misclassified. This motivates us to look at the clusters themselves. PCA, T-SNE, UMAP, or other more common statistical methods could be applied, however the interactivity of our approach fits well with "Visualizing Neural Networks with the Grand Tour" [7], and our project to extend it, started as a class project [8]. We have extended this into a tool we are calling n-dimensional scatter plot (NDSP), which is very helpful in making sense of and editing classifications in pixel distributions.

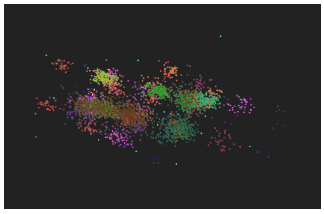
XXX could write a section here on features / implementation of NDSP

We show a case study to illustrate the usefulness of being able to interact with the classes in the pixel distribution. In Fig 14 an inconsistency is found in the pixel classification that is showing up in PixCol. The classification is exported from PixCol and imported into NDSP, where the clusters are interactively examined. The issue with the classification of the distribution is identified and corrected, and then the modified classification is imported back into PixCol to show that the correction has resolved the inconsistency.

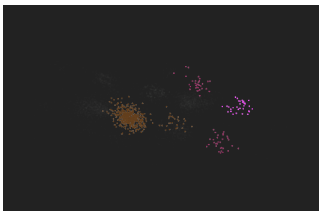
We used these tools and methodologies to examine the output from 3 layers of the policy network: The very first layer: block1.conv, the second conv layer: block1.res1.conv1, and the layer with channel 55 identified in [2]:block2.res1.resadd. Note that the first and second conv layers do have nonlinear, unparameterized layers block1.maxpool, and block1.res1.relu1 between them. These layers do output activations which could be interpreted using this method, however in the interest of time,



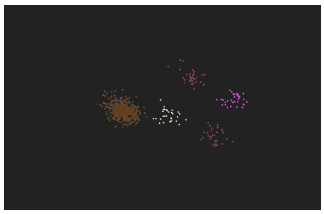
(a) Outer corners on the bottom left break the regularity established by the other inner and outer corners of having 4 class representations.



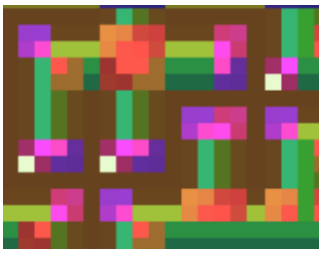
(b) Even with the coloring, it is too cluttered to see what's going on. Since the clusters are labeled, we can easily select the four clusters surrounding our interest and hide the rest.



(c) This makes it clear what happened. Basically, the clusters got too close together and got classified as one cluster.



(d) We can easily select the misclassified pixels and add them to a new pixel class.



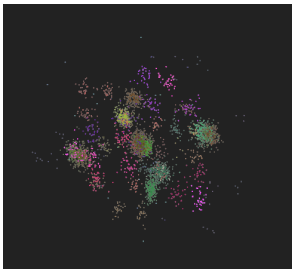
(e) Back in the image view we can confirm that we have corrected the misclassification by seeing that the new pixel class has completed the incomplete bottom left corners.

Fig. 14: Correcting pixel classification issue using NDSP.

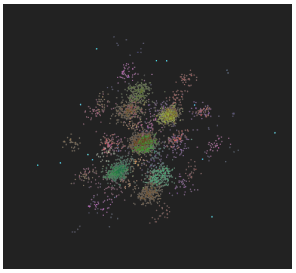
our focus was directed to conv layers. Also, block2.res1.resadd is about halfway through the network, much deeper than the other two. We briefly discuss our findings in the following sections.

4) b1.conv: We are very impressed with the techniques ability to render the first layers activations interpretable. As could reasonably be predicted, the first layer is involved in basic edge detection tasks, but the exact nature of those edge detections is quite interesting. Each of the classes at this layer can be understood as relating to one of 4 concepts: the mouse, the cheese, straight paths, or corners.

The mouse is the most interesting concept when examined



(a) PCA view of corner clusters (pink and purple).



(b) PCA view of mouse pixels (light blue).

Fig. 15: PCA applied to different points. The is the difference because of how many more corner pixels than mouse pixels there are, or does it represent a difference in how the network conceptualizes?

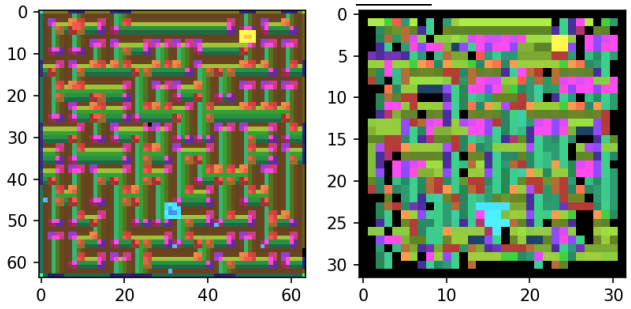


Fig. 16: Coloring of block1.conv as reference for coloring block1.res1.conv1

in NDSP. Unlike the others, it's points do not cluster together, but instead spread apart, moving outside of the cluster of all other classes. This may be caused by the different amounts of pixels, but may also represent differences in how the network will use the concepts. See Fig 15.

Because the pixels are so far out, hierarchical clustering assigns them all to separate categories from each other and everything else. K-means clustering on the other hand distributes them among other classes since they are in different directions and are far away from everything. It is only through examining their relation to the previous layer that we could identify them as cheese pixels. Manual clustering does seem to correctly identify them since even though they can be viewed as all around the other points, the view can also be rotated so they all cluster to one side. Further analysis is needed to give a good explanation of the actual structure that this represents.

The cheese is very similar to the mouse, although may not be as extremely dispersed as the mouse is. This may be because of it's relative importance being less than the mouse, since the mouse is the thing that moves, or it could be because the maze we examined had the mouse closer to more other concepts, while the cheese is near relatively fewer. It is totally surrounded by brown floor pixels. Further investigation of other mazes and the surrounding context of the pixels is recommended.

The consistency of the corners is quite impressive. Each type of corner across inner vs outer, top vs bottom, and left vs right has exactly 4 sub distributions. This may be a result of the 3x3 convolution kernel seeing the corner from exactly 4 spots. It may be that these different classes are to be merged into single corner concepts in the next layer, or it could be that the different corners will be extended to combine with other corners and walls into higher level concepts. It is interesting that the clusters in Fig 14d seems to have a geometric relationship with the shape that the pixels found in relation to one another. Further analysis on the pixel distribution of block1.res1.conv1 may clarify which is the case.

The walls and floor are the somewhat inconsistent. part of this seems to be because of aliasing. Some of the walls and floors are thicker or thinner than others. This was not something we had noticed until we started exploring the pixel class locations in PixCol. This seems to cause places where there is or is not a inner wall concept, or just left of wall or right of wall concepts. Some of the wall classes are quite small and inconsistent. There may be more to these representations than is obvious, although it may also be caused by the noise in the maze texture and wall thickness.

Also worth noting is that the outside edges and corners of the maze get their own sets of wall and corner classes, consistent along each edge, but not shared between edges or with the rest of the classes inside the maze. This may be because of the way the downstream layers use the outer edges, or, again, it may be an artifact of the convolution process.

Overall, the concepts of layer 1 are not terribly abstract. Basically all of them could be easily implemented manually. However we are curious to know whether the inconsistencies will be abstracted or made use of in deeper layers.

5) b1.res1.conv1: The layer b1.res1.conv1 seems to be moving to noticeably higher abstraction concepts. As can be seen in Fig 16, this seems in part forced by the reduction in number of pixels from the maxpool layer. However, it must be assumed that it is also transforming the representation of the data in a way that moves it closer to a decision on which direction the mouse will move.

The main evidence we can see of this is the expansion of the mouse concept. It appears it may be starting to perform some kind of flood fill algorithm, sending the concept of "mouse goes right" down one path, while sending the concept of "mouse goes up" up another path. There also is minor evidence that the mouse may be altering or resolving maze structure around it as seen by the strange formation of corners to the bottom left of the mouse, and it seeming to push the wall out of the way to the top left. However, the concepts are fuzzy so further investigation is needed.

Some evidence of the higher level of abstraction is seen in the loss of the "floor" and "inner wall" concepts, which seem to have been replaced only by walls. Additionally, the concept of "upper right corner" (purple) seems to have fallen a distance from it's original reference pixels. In some cases falling under a layer of the concept of "upper left corner" while presumably still representing a path between two corners that the agent

could travel. It seems that by following rules such as ”you may only move from a left wall to a right wall while traveling right, never while traveling left”, and ”You may only move from a right wall to an upper wall if an appropriate corner is also present” one can still make sense of and navigate this representation of the maze.

This makes the layer more difficult to interpret, but also more rewarding. Unfortunately we do not fully understand the concepts represented on this layer yet, but feel that with more time to cross reference between the PixCol and NDSP, good progress could be made understanding concepts deeper into the network.

6) b2.res1.resadd: As a final experiment with PixCol and NDSP we attempted to look at a much deeper layer, b2.res1.resadd, which is the residual block. This means we are half way through the second of 3 Impala blocks, past 8 conv layers. By this point the image has been reduced to a 16x16 image with 128 channels. The concepts must be much more abstracted and are possibly bleeding together in interesting ways.

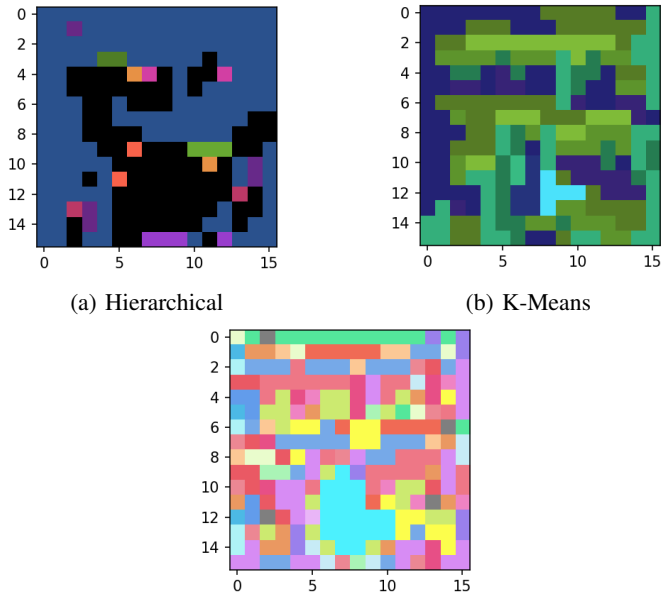
We first noticed that hierarchical clustering no longer seems to perform well. The blue shown in Fig 18a is all one classification, and every pixel left black was checked and belongs to a classification alone. Clearly the density of the points is quite variable, an idea that is reinforced by examining the pixel distribution in NDSP. Fig 18 shows the distribution. By looking at the manually colored and K-Means colored versions, it appears that the the mouse concept is pulling more pixels further from the rest of the distribution. The appearance of the colorings suggest there is noise or quite abstracted representations. Two other possabilities seem worth mentioning, that the network is ”throwing out” useless information as it goes, such as deleting the existence of short dead end paths from it’s representations of the maze, and so the representation at block2.res1.resadd may not be more abstract, but just contain less unnecessary information. It could also be the case that the network is making use of orthogonal subspaces, in which case, continuing to represent the output as an image may lose it’s sensability. This is possibly something that could be investigated using extensions to NDSP. Another possible avenue for understanding this layer is to work our way from the shallow layers building up interpretations of the concepts the network is working with at that each layer to inform the interpretation of the next. Indeed, this space seems to have many promising directions for further research.

xxxxxxxxxxxxxxxxxxxx xxxxxxxxxxxxxxxxxxxx TODO : finish writing this section xxxxxxxxxxxxxxxxxxxx xxxxxxxxxxxxxxxxxxxx

xxxxxxxxxxxxxxxxxxxx TODO w: write this section

E. xxx–Authors and Affiliations

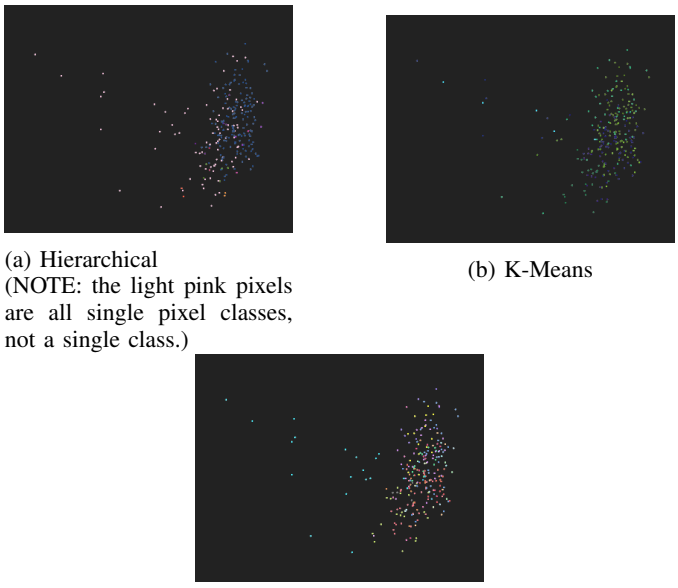
Figure Labels: Use 8 point Times New Roman for Figure labels. Use words rather than symbols or abbreviations when writing Figure axis labels to avoid confusing the reader. As an example, write the quantity “Magnetization”, or “Magnetization, M”, not just “M”. If including units in the label, present



(a) Hierarchical (b) K-Means

(c) Manual with NDSP

Fig. 17: PixCol colorings of block2.res1.conv1 pixel classifications by different clustering techniques.



(a) Hierarchical
(NOTE: the light pink pixels are all single pixel classes, not a single class.)

(b) K-Means

(c) Manual with NDSP

Fig. 18: NDSP distributions of block2.res1.conv1 pixel classifications by different clustering techniques.

them within parentheses. Do not label axes only with units. In the example, write “Magnetization (A/m)” or “Magnetization {A[m(1)]}”, not just “A/m”. Do not label axes with a ratio of quantities and units. For example, write “Temperature (K)”, not “Temperature/K”.

ACKNOWLEDGMENT

xxx The authors would like to thank each other. Thanks Triston. Thanks Tristan. xxx

The preferred spelling of the word “acknowledgment” in America is without an “e” after the “g”. Avoid the stilted expression “one of us (R. B. G.) thanks . . .”. Instead, try “R. B. G. thanks. . .”. Put sponsor acknowledgments in the unnumbered footnote on the first page.

REFERENCES

xxx TODO: once everything is written, go through and swap these around so they appear in order introduced.

REFERENCES

[1] Karl Cobbe, John Schulman et al, “Leveraging Procedural Generation to Benchmark Reinforcement Learning,” arxiv https://arxiv.org/abs/1912.01588

[2] Ulisse Mini, Alexander Turner et al, “Understanding and Controlling a Maze-Solving Policy Network,” arxiv https://arxiv.org/abs/2310.08043

[3] Lauro Langosco, David Krueger et al, “Goal Misgeneralization in Deep Reinforcement Learning,” arxiv https://arxiv.org/abs/2105.14111

[4] Klas Leino, Linyi Li et al, “Influence-Directed Explanations for Deep Convolutional Networks,” arxiv https://arxiv.org/abs/1802.03788

[5] Chris Olah, Ludwig Schubert, et al, “Feature Visualization,” distil.pub https://distill.pub/2017/feature-visualization/

[6] Ulisse Mini, “Integrated Gradients,” hackmd.io https://hackmd.io/@ulirocks/integrated-gradients

[7] Mingwei Li, Carlos Scheidegger, et al, “Visualizing Neural Networks with the Grand Tour,” distil.pub https://distill.pub/2020/grand-tour/

[8] T. Trim, W. Jones, J. Wright, ”AI Understanding” onlineacademiccommunity.uvic.ca. https://onlineacademiccommunity.uvic.ca/dimensionbros/ (accessed Aug 2, 2024).

[9] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, Oleg Klimov ”Proximal Policy Optimization Algorithms” https://arxiv.org/pdf/1707.06347