Jigsaws and How to Solve Them

James Ngai *
Carnegie-Mellon University
jdngai@andrew.cmu.edu

Lawrence Feng
Carnegie-Mellon University
lawrencf@andrew.cmu.edu

Eray Can Elumar
Carnegie-Mellon University
eelumar@andrew.cmu.edu

Shreya Terupally
Carnegie-Mellon University
sterupal@andrew.cmu.edu

Abstract

The reassembly of fragmented images, akin to solving a jigsaw puzzle, presents a challenging problem in computer vision and machine learning due to its visual, semantic, and spatial nature. As the number of pieces increases, the permutations of a jigsaw puzzle undergo factorial growth, amplifying the complexity further. This problem has many applications such as reconstructing and restoring medical images, reconstructing damaged historical documents, and combining different camera views in autonomous perception systems. Prior approaches predominantly centered on the supervised learning of convolutional neural networks. This study builds upon these supervised learning techniques by integrating reinforcement learning approaches commonly used in game and puzzle solving. In this paper, we take inspiration from *AlphaGo's* variation of a Monte Carlo tree search. Our approach involves training a policy network through policy gradient methods, which assigns probabilities to state-action pairs. Additionally, we devised a novel supervised learning-based loss scheme to train our value network, enabling it to discern the semantic similarity between our images and the ground truth. Both the policy and value networks are based on deep convolutional neural networks. In addition, we leverage the episodic and exploratory nature of reinforcement learning and reframe the jigsaw puzzle as a sequential swapping game. The aim is to demonstrate the application of reinforcement learning to an image manipulation task.

1 Introduction

Image manipulation is a broad category of problems studied in computer vision and machine learning, including image denoising, translation, and transformation. Image manipulation can be used for solving problems in medicine, perception, and more. Here, we focus on the assembly of jigsaw puzzles as our image manipulation task. We wish to demonstrate the application of reinforcement learning to such an image manipulation task.

A jigsaw puzzle is a tiling puzzle that requires the assembly of interlocking and tessellating pieces. Each piece has a small part of a picture on it, and thus learnable, local features; when complete, a jigsaw puzzle produces a complete picture with emergent global properties. This problem is challenging because it requires the model to recognize local and global patterns, understand spatial relationships, make sense of fragmented information, and act with a tremendous search space. Solving the jigsaw puzzle problem using machine learning is an area of research that explores techniques for effectively integrating information from different parts of an input to improve overall understanding

*The link for our GitHub repo is https://github.com/james-ngai/11-785-Project

37th Conference on Neural Information Processing Systems (NeurIPS 2023).

and performance in tasks such as image segmentation, object recognition, and scene understanding. These applications are observed in reconstructing and restoring medical images, reconstructing damaged historical documents, and combining different camera views in autonomous driving. Earlier reassembly algorithms utilized shape [12], contour, or color [9] of the pieces to match adjacent pieces. With the rapid advancements in machine learning, numerous approaches have been used for jigsaw puzzle reassembly such as convolutional neural network (CNN) based methods [7], generative adversarial network-based methods [5], and reinforcement learning [3].

We generate 3×3 jigsaw puzzles from the MNIST dataset. There are $9! = 362880$ possible puzzle configurations which each correspond to a unique state. Though MNIST images are 28×28, we form our jigsaw puzzle using nine 9×9 pieces. These nine pieces are the input to our model. The digits in the MNIST dataset are centered, and so the loss of some edge pixels is irrelevant. CNNs underlie our puzzle-solving architecture. We are inspired by *AlphaGo's* Monte Carlo tree search (MCTS) variant, which utilizes value and policy networks to reduce the depth and breadth of the tree search, respectively.

2 Related Work

In the domain of reassembly of images altered by fragmentation and permutation, the current landscape predominantly manifests a convergence between Convolutional neural network (CNN) methodologies and Reinforcement Learning (RL) paradigms. Pioneering strides have been made by Le et al. [4] and Paumard et al. [6], wherein the utilization of CNNs has been instrumental in accurately discerning and repositioning image fragments, thereby effectuating image reassembly.

However, with the development of deep reinforcement learning, leveraging the seminal work of Sutton et al. [11], we have seen the application of reinforcement learning to countless tasks, particularly games. In 2016, *AlphaZero* established complete dominance over human and computer players in the game of Go using novel reinforcement learning techniques and deep CNNs.

More recently, we have witnessed the assimilation of the *AlphaGo* algorithm [8] into the reassembly framework, as elucidated by Gras [3]. On the other hand, Song et al. [10] utilize Deep Q-Networks instead, another recent development in the application of reinforcement learning to image reassembly. These studies underscore the burgeoning interest in employing RL-based methodologies for the precise reconstruction of fragmented images.

In this vein, our current work draws significant inspiration from the foundational contributions of Sutton et al. [11] and Silver et al. [8], while also building upon the advancements put forth by Gras [3]. Our primary model is crafted by integrating key insights from these seminal works, striving to harness the collective potential of RL and CNNs for optimal image reassembly in fragmented and permuted scenarios.

We have identified two starting points for this project. We implemented a baseline that [1] utilizes a convolutional neural network architecture that represents the general approach that has been dominantly used for this task, and the second approach we have identified [3] uses reinforcement learning in its implementation that utilizes the recent advancements in this field. While classical image processing techniques and CNNs can solve the problem for 2×2 or 3×3 puzzles, the exponential size of possible rearrangements necessitates the use of reinforcement learning for larger grid sizes.

3 Baseline

3.1 Selection

The baseline model is a dual objective system that adds jigsaw puzzle solving as a secondary objective to improve accuracy on an image classification task. In this model, the original image and a shuffled image are fed as input, and the network aims to classify the image using the original image and tries to predict the permutation indices of the shuffled image. An overview of this network is given in Fig. 1. The backbone of this model is a pre-trained CNN such as ResNet or AlexNet whose last fully connected layer is removed. The output of this network is fed to a fully connected layer to classify the object and to another fully connected layer to predict the permutation indices of the shuffled image. The loss function is defined as the linear combination of the cross-entropy losses of both

objectives. In this framework, first MCTS randomly simulates different paths of actions and uses the

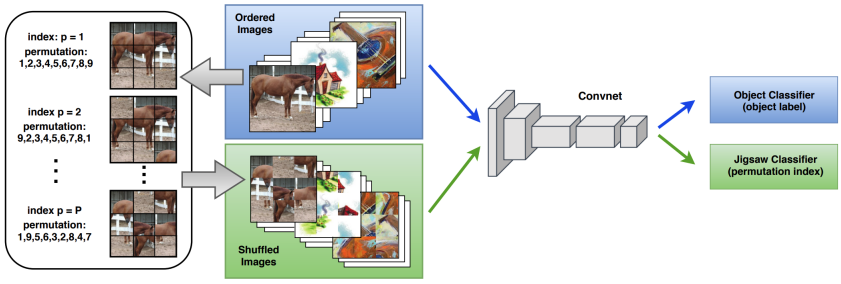

Figure 1: The model overview of the baseline 1

PN to estimate the value of these different paths, and the optimal action is estimated from these paths. Then, this optimal action is applied, and the same process starts again until assigning every fragment to a position. Also, after each simulation the final state value estimated by the VN is backpropagated. This lets the algorithm reassemble the puzzles without having the ground truth after the training phase.

3.2 Implementation

We implemented the baseline model with the following training parameters:

- Dataset: PACS [2] [13]
- Batch size: 128
- Pretrained network: ResNet-18
- Image Augmentations: color jitter, random horizontal flip, random grayscale
- Learning rate: 0.001

After training the network with these parameters, the accuracy we obtained is 82.14%. This was the dataset the original authors used, and we acknowledge the limitations this has on comparing our model, which was trained and tested on the MNIST dataset, with the baseline. In future work, we would like explore more about how this reinforcement learning method of solving image jigsaws generalizes to other domains and/or multiple domains.

4 Implemented Model

We chose to implement a different model than what we came across in our literature review as we found exhibit limitations. The first baseline model we have takes the shuffled images as input uses CNNs to predict the permutation needed to reassemble the image, but this approach is not accurate enough. The second baseline proposed by Gras [3] reassembles the puzzle piece by piece using the Monte Carlo tree search (MCTS), starting with an empty puzzle and putting one piece in every step. While this method demonstrates the application of reinforcement learning and graph-based search to puzzle reassembly, it is not robust to misplaced tiles, and cannot correct itself if a piece is misplaced. Another limitation we encountered was the mismatch between reinforcement learning and the non-sequential nature of this formulation. In response, we are proposing a substantial paradigm shift by modifying the fundamental gameplay to better align with RL's sequential learning strengths. Our approach involves adapting the RL agent to engage in an image reassembly game allowing the swapping of any puzzle piece with any other puzzle piece.

The objective is to enable the RL agent to learn and execute optimal strategies in reconstructing fragmented images, thereby contributing to enhanced generalizability and efficiency in this domain.

We start by describing the problem in a more structured way.

[2]PACS is an image dataset for domain generalization. It consists of four domains, namely Photo (1,670 images), Art Painting (2,048 images), Cartoon (2,344 images) and Sketch (3,929 images).

4.1 Problem Description

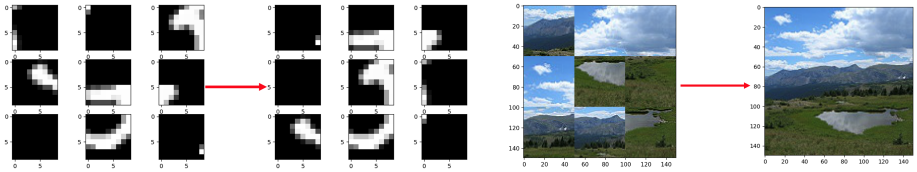

Figure 2: The jigsaw reassembly problem

In this paper, we assume that an image is divided into an $m \times m$ grid of same-sized tiles and each position of this grid is numbered from 1 to m^2. Letting S denote the original image, we use f_i to denote the i^{th} tile of the original image where $1 \leq i \leq m^2$, and we use $I(t)$ to denote the current state of the reassembled image. We also use $r_i(t)$ to denote the i^{th} tile of the reassembled image $I(t)$. We assume that as input we receive $r_i(t = 0)$, the tiles f_i in a shuffled order tiles where $1 \leq i \leq m^2$. We denote the permutation function used to shuffle the tiles as P, and using this P, the shuffled tiles can be expressed as $r_i(t = 0) = f_{P(i)}$. At time $t = 0$, we assume that the tiles are placed on the image in this shuffled order. In each round, an action $a_t = (i, j)$ can be taken. The action $a_t = (i, j)$ means swapping tiles at the i^{th} and j^{th} position. The final goal of this model is to reconstruct the original image by swapping tiles this way such that when the algorithm stops at round t_f, $I(t_f) = I$. An example shuffled image and its reconstruction is given in Fig. 2

4.2 Dataset

We use the MNIST [2] dataset in this project. The MNIST dataset is a dataset of images of handwritten digits and consists of $60,000$ training images and $10,000$ test images. The MNIST dataset is a standard, simple dataset we choose for the semantic and spatial properties of handwritten digits.

For each image in the dataset, we divide the image into nine 9×9 tiles, generate a random permutation, and reorder the tiles according to this permutation to get a shuffled image. This scrambled image is the input into our model. Our supervised learning value net was trained in batches but our policy network and the MCTS use instance sampling, considering just one puzzle at a time.

4.3 Model Description

In our jigsaw reassembly model, we start with a shuffled image, and at each round we select two tiles to swap their positions. To choose the two tiles that are swapped at each round, we use the Monte Carlo Tree Search (MCTS), a method that is also used in *AlphaGo*. Overall, MCTS is a search algorithm that uses two different network models, the Policy Network and the Value Network to generate the optimal action to take in each round. We discuss each component of this model in detail below.

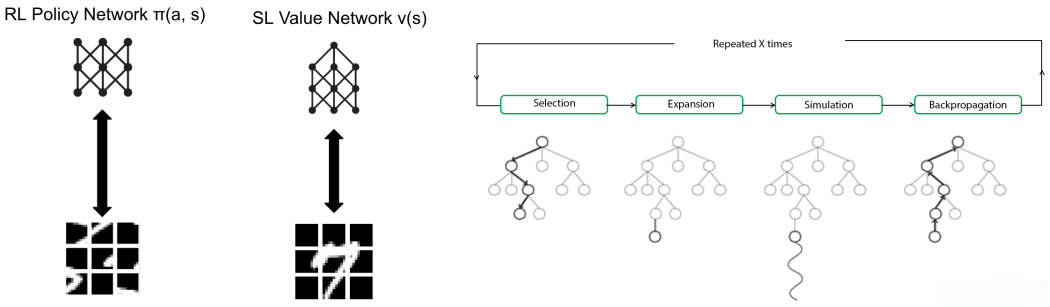

Figure 3: Overall view of the puzzle-solving architecture. The Policy Network is trained using policy gradient methods. The Value Network is trained using supervised learning. The Monte Carlo tree search utilizes the VN and PN to reduce the breadth and depth of search in four steps.

4.4 Monte Carlo Tree Search (MCTS)

The last component in our model is the The Monte Carlo Tree Search (MCTS) algorithm. MCTS is a heuristic search algorithm that was popularized by *AlphaGo*, though the top Go algorithms before *AlphaGo* also employed MCTS. The key idea behind MCTS is to simulate and statistically evaluate the outcomes of different trajectories to guide the decision-making process. It consists of four main steps:

1) Selection: The algorithm starts at the root of a tree and traverses down the branches, greedily selecting the node with the highest state-action potential $U(s, a)$, determined using the polynomial upper confidence bound for trees, which balances exploration (trying new moves) and exploitation (choosing moves that seem promising based on past simulations) with the parameter c_{PUCT}. We use the following formulas to perform this search:

$$N(s, a) = \sum_i \mathbb{1}(s, a, i) \tag{1}$$

$$Q(s, a) = \frac{1}{N(s, a)} \sum \mathbb{1}(s, a, i) V_\theta(s_i) \tag{2}$$

$$U(s, a) = Q(s, a) + c_{PUCT} \cdot P(s, a) \cdot \frac{\sqrt{\sum_b N(s, b)}}{N(s, a)} \tag{3}$$

where $Q(s, a)$ is the action value, V_θ is the parameterized value network, $P(s, a)$ is the prior probability of selecting action a in state s, as estimated by the policy network, $N(s, a)$ is the total visit count of state-action pair (s, a). $\sum_b N(s, b)$ is the visit count of the parent node of (s, a).

2) Expansion: Once a promising node is reached, the algorithm expands the tree by adding child nodes corresponding to possible moves from the promising node.

3) Simulation (or Rollout): Simulations or rollouts are performed from the expanded nodes. The value network is used to estimate the expected outcome of the game from each simulation, providing a more informed evaluation of potential moves. We do not estimate the expected outcome using rollouts in our design.

4) Backpropagation: The results of the simulation are then backpropagated through the tree, updating the statistics of each node along the path from the selected node to the root. This involves updating the number of visits and the cumulative reward.

5) Termination: The search is terminated and the final jigsaw reassembly is given as output when the score outputted from the value network is greater than some threshold, the maximum number of swaps has been reached, or we terminate the game manually when the puzzle has been solved.

The overview of these steps is given in Fig. 3.

4.5 Policy Network

The policy network is used to generate action probabilities for a given image reassembly state. It takes the current reassembed image at round $t - 1$ as input and produces a probability for each of the $N_A = \binom{m^2}{2}$ possible actions. The network is mainly composed of two blocks, first is the 3 Convolutional layers which takes the reassmembled image as input and outputs a vector of size 144. This vector is then passed through the fully connected layers block to produce a probability for each of the $\binom{m^2}{2}$ possible actions.

The architecture ablations of the policy network is given in Table 1.

The reward function ablations for the policy network are given in Table 2

The Policy Network is trained using policy gradient methods and the REINFORCE algorithm [11]. An overview in given in algorithm 1.

Model	Hidden Layers	Parameters	% Perfect Reassembly
1CNN_1Dense	$1 \times$ {CNN, Batchnorm}, 1 Dense	Kernel Size (3),	did not converge
2CNN_1Dense	$2 \times$ {CNN, Batchnorm}, 1 Dense	Kernel Size (3,5)	78.4%
3CNN_1Dense	$3 \times$ {CNN, Batchnorm}, 1 Dense	Kernel Size (3,5,9)	82.6%
1CNN_2Dense	$1 \times$ {CNN, Batchnorm}, 2 Dense	Kernel Size (3)	did not converge
2CNN_2Dense	$2 \times$ {CNN, Batchnorm}, 2 Dense	Kernel Size (3,5)	84.7%
3CNN_2Dense	$3 \times$ {CNN, Batchnorm}, 2 Dense	Kernel Size (3,5,9)	85.3%

Table 1: Policy Network Ablations (Intermediate Reward: 1, Reward Perfect Reassembly: 100)

4.6 Policy Network Loss and Reward Functions

Let $J(\theta) = \mathbb{E}[\gamma^0 r_1 + \gamma^1 r_2 + \ldots + \gamma^{T-1} r_T | \pi_\theta]$ be the expected discounted reward under policy π_θ with discount factor γ. The policy gradient theorem states that

$$\nabla_\theta J(\theta) = \sum_{t=0}^{T-1} \nabla \log \pi_\theta(a_t|s_t) G_t \tag{4}$$

where $G_t = \sum_{t'=t+1}^{T} \gamma^{t'-t-1} r_{t'}$. Using this fact, we update our policy network parameters θ using the following:

Algorithm 1 REINFORCE

Random initialization of parameters θ
for each episode $(s_1, a_1, r_2, s_2, a_2, r_3, \ldots, s_{T-1}, a_{T-1}, r_T)$ **do**
  for $t = 1$ to $T - 1$ **do**
    if t is even **then**
      $\theta \leftarrow \theta + \alpha \nabla_\theta \log \pi_\theta(s_t, a_t) v_t$
    end if
  end for
end for

We perform gradient ascent on the expected outcome. The greater the expected reward, the greater closer a puzzle is to complete reassembly. This is our "loss" function for the policy network.

Intermediate Reward	Reward Perfect Reassembly	Normalized	% Perfect Reassembly
# of correct tiles 1	Perfect Assembly: 100	Y	85.4%
# of correct tiles - initial correct	Perfect Assembly: 100	Y	82.6%
# of correct tiles	Perfect Assembly:10	Y	did not converge
-1 per time step	Perfect Assembly:100	Y	did not converge

Table 2: Reward Ablations (Model: 3CNN_2Dense)

4.7 Value Network

The value network is used to score the final reassembled image. It gives a score in $[0, 1]$ based on how close it is to the reassembled image. The network is mainly composed of two blocks, first is the pretrained ResNet-18 block which takes the reassmembled image as input and outputs a vector of size 1000. This vector is then passed through the fully connected layers block to produce a scalar output.

This network is trained using shuffled images as input and the fraction of correctly placed tiles (number of tiles in correct position divided by the number of correct tiles) is used as the label. The loss function used is mean square error. The Stochastic Gradient Descent (SGD) optimizer with learning rate 0.001 is used as optimizer (since ResNet is pretrained the learning rate is small). The architecture of the value network is given in Fig. 4 and ablations in Table 3.

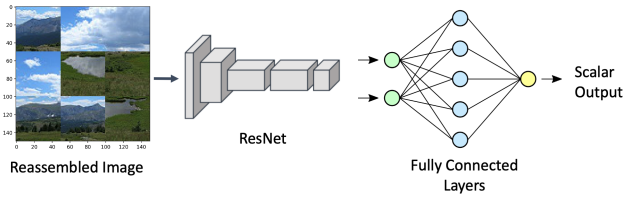

Figure 4: The value network

Model	Hidden Layers	Parameters
3layerCNN_2Dense	$3 \times$ {CNN, Maxpooling, Batchnorm}, 2 Dense	Kernel Size (3,5,9)
1Resnet18_2Dense	Resnet18, 2 Linear, Softmax	Resnet

Table 3: Value Network Architecture Ablations

4.8 Value Network Loss Function

The selection of the Mean Squared Error (MSE) loss function is ideal for regression tasks due to its ability to penalize larger errors quadratically. Let's denote the dataset as D, where 'n' represents the index of the nth image in the dataset. The permuted image as $P\left(I_n\right)$ with associated semantic score y_n and the value networks predicted semantic score as $\hat{y_n}$. This aligns with the objective of minimzing the average squared difference between predicted $\hat{y_n}$ and y_n to train the optimal value network for evaluating the state.

$$\text{MSE} = \frac{1}{|\mathcal{D}|} \sum_{n=1}^{|\mathcal{D}|} (y_n - \hat{y_n})^2 \tag{5}$$

4.9 Evaluation Metric

We have two different metrics to evaluate the performance of the model. The first metric M_p, is the fraction of perfectly reassembled images by the model. It can be written as:

$$M_p = \frac{\sum_{n=1}^{|\mathcal{D}|} \mathbb{1}\{I_n(t_f) = S_n\}}{|\mathcal{D}|} \tag{6}$$

where S_n is the n^{th} image in the dataset $\mathcal{D}$, $I_x(t_f)$ is the final reassembled image of S_n, and $\mathbb{1}\{\cdot\}$ is the indicator function.

The M_p metric, measuring the ratio of perfectly reassembled images by the model, stands as a pivotal evaluation tool for assessing image reassembly performance. By quantifying the instances where the model accurately reconstructs input images, M_p offers a clear gauge of fidelity and accuracy in replication.

The second evaluation of metric that we use is M_e, the fraction of correctly placed tiles in the final reconstructed image. It can be written as:

$$M_e = \frac{\sum_{n=1}^{|\mathcal{D}|} \sum_{i=1}^{m^2} \mathbb{1}\{r_i(t_f) = f_i\}}{m^2|\mathcal{D}|} \tag{7}$$

The M_e metric, which measures the mean fraction of correctly placed tiles by the model, is crucial for understanding the model's performance. By quantifying the instances where the model accurately reconstructs input images with a nuanced measurement, as opposed to the binary metric of M_p, M_e offers a more informative gauge of accuracy in replication.

5 Results

5.1 Training Results of the Policy Network

The MNIST dataset was used to train the policy network. The MNIST dataset was divided into training and validation, and for each set the image is preprocessed in the following way. First, for image indexed as n in the dataset, $N_s(n) \sim x(n)$ represents the image $N_s(n)$ after random permutation of tiles. The target label for this image is a $m \times 1$ tensor $y(n) = \prod_{i=1}^{m}\{i\}$ where each tile is represented in its correct location. The network is trained on the MNIST dataset using these input images and output labels for 15 epochs. The training and validation metrics of the policy network during training in the MNIST dataset are given in Fig. 5 and Fig. 5. After training for 15 epochs, the policy network took on average 13.66 steps to fully reassemble the puzzle when reassembly occurred on the test set. The policy network was able to fully reassemble the puzzle $M_p = 92.95\%$ on the test set with $M_e = 85.2\%$.

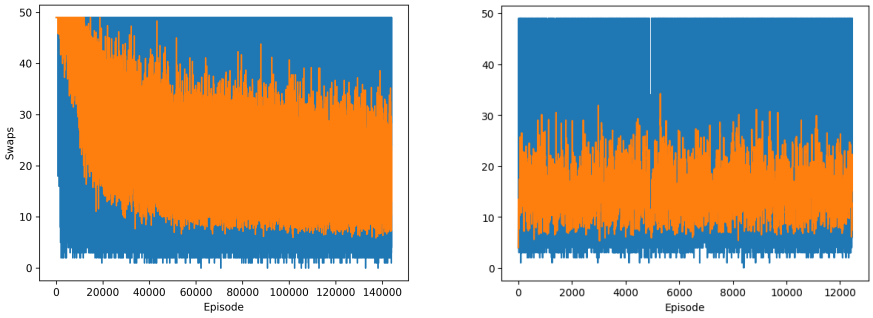

Figure 5: Training and Validation plot of the policy network. The orange line represents a running average of the number of swaps taken in the last 10 puzzles. The blue lines represent the number of swaps for each episode.

5.2 Training Results of the Value Network

MNIST dataset is used to train the value network. The MNIST dataset is divided into training, validation, and test sets, and for each set the image is preprocessed in the following way. First, for image indexed as n in the dataset, $N_s(n) \sim \text{unif}\{0, m^2\}$ that represents the number of tiles that will be shuffled is sampled uniformly from 0 to m^2. Then, $N_s(n)$ many tiles are selected randomly, and these tiles are shuffled to form the input image $x(n)$. The target label for this image is then calculated as $y(n) = \frac{1-N_s(n)}{m^2}$. The network is trained on the MNIST dataset using these input images and output labels for 50 epochs. The training and validation loss of the value network during training in the MNIST dataset is given in Fig. 6. After training for 50 epochs, the final validation loss is 0.0131, and the final training loss is 0.0133.

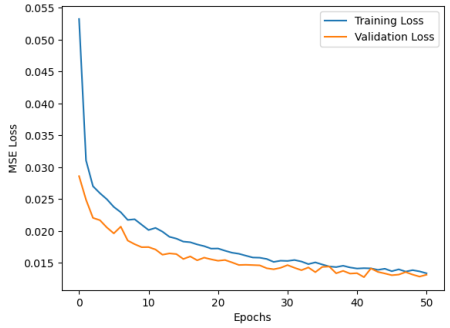

Figure 6: The MSE loss of the value network during training on the MNIST dataset

5.3 Results of the MCTS

The MCTS performance had varied results. During some runs, the MCTS achieved reassembly in significantly fewer swaps than a greedy use of the policy network and in other cases, the MCTS achieved reassembly at higher rates with the use of significant computational resources. More ablations and hyperparameter tuning and problem-specific tuning need to be done to apply the MCTS properly to this problem of puzzle reassembly.

6 Discussion

Jigsaw assembly is a computationally intensive task due to the large search space that needs to be considered, and as a result of this, even other competitive baseline models consider only 3×3 jigsaws. Hence, executing this pipeline with computational efficiency is key to the success and practical applicability of the model. In this work, we achieve computational efficiency through the use of both the policy network and value network in the MCTS search. The main intuition behind it is that a good policy network and a good value network would sufficiently constrain tree search for any size board, maintaining computational efficiency.

In terms of accuracy, as discussed in the results section, our model has higher accuracy compared to the baseline models. Our base MCTS model, though accurate, is sensitive to noise perturbations and not smooth training. This is because the hyperparameter for the Exploration/Explotation tradeoff in MCTS requires careful tuning for the specific dataset. This can be improved by using an ensemble of MCTS agents with different hyperparameters to ensure robustness. Additionally, we can also introduce prioritized experience replay to our MCTS for more stable learning, improving sample efficiency, and faster convergence.

For practical application, additional experimentation on more diverse datasets with varying colors is necessary. Without such experiments, it would be imprudent to extrapolate the performance of our architecture to real-world applications in autonomous vehicles and medical imaging.

7 Future Works

We used the pretrained ResNet model in our project to cut back on training time for our value network. In the future, a more fine-tuned convolutional neural network can be used in the value network to improve the accuracy of the model and the size of the model.

We can also improve the computational efficiency of our model by fine-tuning our model for increased efficiency, and also experiment using an ensemble of simpler MCTS agents. This way, we can also extend our results to more complex puzzles such as the 4×4 sized puzzles. Using our trained models in an image classification task using transfer learning is also one of the future works that we will consider.

More investigation needs to be done in the hyperparameter tuning of the MCTS, such as the number of simulations, exploration weight c_{PUCT}, exploration weight scheduling, etc. These hyperparameters are key for balancing accuracy and computational efficiency.

8 Conclusion

The reassembly of fragmented images remains a complex challenge within the realms of computer vision and machine learning. Our endeavor aimed to push the boundaries of conventional supervised learning by integrating reinforcement learning techniques inspired by AlphaGo's Monte Carlo tree search. Through the development of a policy network utilizing policy gradient methods and a unique loss scheme that utilizes supervised learning for semantic evaluation, we successfully ventured beyond the traditional approaches focused solely on convolutional neural networks.

Our study represents a important milestone in tackling the intricate nature of image reassembly. We not only demonstrated the efficacy of our AlphaGo-inspired approach in near-matching existing benchmarks but also showcased the adaptability of reinforcement learning methodologies in addressing visual, semantic, and spatial challenges inherent in fragmented image reconstruction.

The implications of our findings reverberate across diverse domains, ranging from medical imaging reconstruction to historical document restoration and autonomous perception systems. By leveraging the episodic and exploratory characteristics of reinforcement learning, we reframed the puzzle assembly process as a sequential swapping game, showcasing the versatility and applicability of RL paradigms in complex image manipulation tasks.

While we achieved considerable success in capturing semantic meaning within images and translating them into reassembly, our findings also open avenues for further exploration. Importantly, our MCTS algorithm needs to be fine tuned to balance both improve policy/action selection and computational resources. MCTS can be computationally costly, and the greater the computational cost the more MCTS is likely to succeed, though for sufficiently complex games, such a large tree search is feasible and the balance between search and utilization of the policy network and value network becomes crucial. In our simple 3×3 scenario, we were not able to sufficiently perform hyperparameter tuning and analysis. Future endeavors could delve deeper into refining the RL model's adaptability to varying puzzle complexities or explore novel ways to integrate multiple modalities beyond visual cues for a more holistic reassembly approach.

In essence, our most significant contribution is the success of our AlphaGo-inspired approach, offering a promising direction for the fusion of reinforcement learning and image manipulation tasks, laying the groundwork for more sophisticated applications in the realm of computer vision and machine learning.

9 Division of Work

- Eray: Baseline implementation, implementation and the training of the value network, and writing the report.
- Lawrence: Reformulated and implemented jigsaw puzzle as sequential swapping game, managed dataset and dataloader, implemented and ablated policy network, implemented MCTS and integrated value network and policy network, and helped create the report.
- James: Managed dataset and data loader, ablated policy network, helped implement MCTS, and helped write the report.

Acknowledgments and Disclosure of Funding

We thank our TAs Dheeraj Pai and Rucha Manoj Kulkarni for their guidance and support on this project.

References

[1] Fabio Maria Carlucci, Antonio D'Innocente, Silvia Bucci, Barbara Caputo, and Tatiana Tommasi. Domain generalization by solving jigsaw puzzles. In *CVPR*, 2019.

[2] Li Deng. The mnist database of handwritten digit images for machine learning research. *IEEE Signal Processing Magazine*, 29(6):141–142, 2012.

[3] Johan Gras. Puzzle reassembly using model based reinforcement learning. 2019. URL `https://johan-gras.github.io/projects/puzzlereassembly/`. `https://johan-gras.github.io/projects/puzzlereassembly/`.

[4] Canyu Le and Xin Li. Jigsawnet: Shredded image reassembly using convolutional neural network and loop-based composition. *arXiv preprint arXiv:1809.04137*, 2018.

[5] Ru Li, Shuaicheng Liu, Guangfu Wang, Guanghui Liu, and Bing Zeng. Jigsawgan: Auxiliary learning for solving jigsaw puzzles with generative adversarial networks. *IEEE Transactions on Image Processing*, 31:513–524, 2021.

[6] M. M. Paumard, D. Picard, and H. Tabia. Image reassembly combining deep learning and shortest path problem, 2018.

[7] Marie-Morgane Paumard, David Picard, and Hedi Tabia. Deepzzle: Solving visual jigsaw puzzles with deep learning and shortest path optimization. *IEEE Transactions on Image Processing*, 29:3569–3581, 2020.

[8] David Silver, Aja Huang, Chris J. Maddison, Arthur Guez, Laurent Sifre, George van den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, Sander Dieleman, Dominik Grewe, John Nham, Nal Kalchbrenner, Ilya Sutskever, Timothy Lillicrap, Madeleine Leach, Koray Kavukcuoglu, Thore Graepel, and Demis Hassabis. Mastering the game of Go with deep neural networks and tree search. *Nature*, 529(7587):484–489, January 2016. doi: 10.1038/nature16961.

[9] Kilho Son, James Hays, and David B Cooper. Solving square jigsaw puzzles with loop constraints. In *Computer Vision–ECCV 2014: 13th European Conference, Zurich, Switzerland, September 6-12, 2014, Proceedings, Part VI 13*, pages 32–46. Springer, 2014.

[10] Xingke Song, Jiahuan Jin, Chenglin Yao, Shihe Wang, Jianfeng Ren, and Ruibin Bai. Siamese-discriminant deep reinforcement learning for solving jigsaw puzzles with large eroded gaps. *Proceedings of the AAAI Conference on Artificial Intelligence*, 37(2):2303–2311, Jun. 2023. doi: 10.1609/aaai.v37i2.25325. URL `https://ojs.aaai.org/index.php/AAAI/article/view/25325`.

[11] Richard S Sutton, David McAllester, Satinder Singh, and Yishay Mansour. Policy gradient methods for reinforcement learning with function approximation. In S. Solla, T. Leen, and K. Müller, editors, *Advances in Neural Information Processing Systems*, volume 12. MIT Press, 1999. URL `https://proceedings.neurips.cc/paper_files/paper/1999/file/464d828b85b0bed98e80ade0a5c43b0f-Paper.pdf`.

[12] Kang Zhang, Wuyi Yu, Mary Manhein, Warren Waggenspack, and Xin Li. 3d fragment reassembly using integrated template guidance and fracture-region matching. In *Proceedings of the IEEE international conference on computer vision*, pages 2138–2146, 2015.

[13] Kaiyang Zhou, Yongxin Yang, Timothy M. Hospedales, and Tao Xiang. Deep domain-adversarial image generation for domain generalisation. *CoRR*, abs/2003.06054, 2020. URL `https://arxiv.org/abs/2003.06054`.