Visual AI Players for the Game of Go

Sai Prakash Pathuru[1], **Sherman Kettner**[1]

University of Colorado Colorado Springs
1420 Austin Bluffs Pkwy, Colorado Springs, CO 80918 USA
spathuru@uccs.edu, skettner@uccs.edu

Abstract

This project envisions a future where artificial intelligence (AI) transforms our lives, enhancing productivity, healthcare, and sustainability. Leveraging the complex game of Go as a testing ground for AI capabilities, we confront the unique challenges posed by its simple yet intricately strategic rules. As the stones can be placed on the board anywhere, this creates a dynamic and expansive search space, pushing traditional AI players to their limits. Inspired by the transformative potential of current Visual AI, we want to use Convolutional Neural Networks (CNN) with image classification and image infill, to enhance AI's visual problem-solving capabilities. Through this project, we hypothesize that training an AI to recognize patterns in Go visually will bridge the gap between AI and human problem-solving proficiency, redefining the limits of AI in solving strategic challenges visually.

Introduction

"I imagine a world in which AI is going to make us work more productively, live longer, and have cleaner energy." - Fei-Fei Li Artificial intelligence (AI) holds immense potential to revolutionize our lives, fostering advancements in productivity, healthcare, and sustainability. This project leverages the complex game of Go as a platform to explore AI's potential and push the boundaries of its capabilities.

Our true objective is to develop an AI that can visually solve Go, mimicking human intuition. While mastering the game itself is a testament to AI's prowess, our ultimate goal is not only to master the intricacies of Go but, more importantly, to glean insights that can be applied to real-world challenges. In delineating the problem statement, we recognize that solving Go in isolation may not directly impact productivity, life expectancy, or energy solutions. Instead, the aim is to harness Go as a training ground for developing AIs capable of addressing complex, real-world problems.

Similar to how advancements in AI for Go games like AlphaGo paved the way for breakthroughs in protein structure prediction (AlphaFold), we believe that mastering visual aspects of Go can yield valuable insights applicable to real-world problems(Jumper et al. 2021). Historically, Go's deceptively simple rules and vast strategic depth have made it a valuable training ground for AI. This project seeks to contribute to this legacy by developing and refining an AI capable of recognizing patterns and making strategic decisions based on visual information, similar to human players.

We hypothesize that by training an AI to recognize patterns in Go, similar to human intuition visually, we will not achieve superior performance and decision-making capabilities compared to traditional methods relying solely on brute force calculations. But by bridging the gap between AI's capabilities and human problem-solving proficiency, this project aims to pave the way for future AI advancements that can address complex real-world challenges.

This Game of Go was invented 2500 years ago in china(Chen 2016). There different grid boards like 8X8, 13X13, 17X17, and 19X19, but the standard board is 19X19. The rules are simple, They are:

- Two player(black and white) take turns placing a stone of their own color on an empty grid point.
- Adjacent stones of the same color along the vertical and horizontal grid lines form an area.
- When an area is surrounded by an opponent;s stones, it's "alive" if it has at least one empty point inside the area; otherwise, the stones in the area are captured and removed from the board.
- However, if the surrounded area has only one empty point left, the opponent is going to place the stone there and the area is captured.
- If it has two empty points in different places then it is alive and not captured and depending on the situation, there maybe chance of not getting captured.

Problem Statement

In this paper, we address the challenge of employing a visual approach to solve logic problems, particularly focusing on the game of Go. While humans often rely on visual intuition

to tackle problems, this method has not been extensively explored in the field of AI. Our goal is to explore the potential of utilizing algorithms traditionally used in photo analysis to play Go, departing from traditional logical problem-solving approaches.

Related Work

Related Work 1

Until the year 2016, there was no proper AI player that is good enough to defeat the top champion player like Fan Hui, Lee Sendol. The main reason for not having an AI player that could defeat a human was, it is complex to develop and AI player that requires vast search space; the reason it requires vast search space is because the rules are simple and the standard board to play this game is 19X19 grid board, where it has 361 positions to place the stone and the stones can be placed anywhere of all the available empty spots, and next coming turns can be placed at random which makes it a requirement for the vast search space. (Silver et al. 2016)

In the year 2017, Google's DeepMind company developed an AI player for the game of Go which is called AlphaGo. So this AlphaGo is developed with a combination of Monte Carlo Tree Search(MCTS), and policy network to suggest the next possible winning moves and value network to evaluate the board position. They also used a combination of supervised learning to mimic human play and reinforcement learning to optimize strategies through self-play. This AlphaGo was tested on 29.4 million moves from 160,000 games played by human players, and the authors had taken this dataset from KGS Go server. The authors later hosted a five formal game against AlphaGo and Fan Hui(European Go game champion) and AlphaGo won all the 5 games against Fan Hui. This where AlphaGo created history that an AI player for the game of Go can defeat human players. (Silver et al. 2016)

What is this Monte Carlo Tree Search? How does it work? Monte Carlo Tree Search(MCTS) is a heuristic search algorithm used primarily in the field of artificial intelligence(AI), especially for decision-making processes in games and other complex systems. It combines the precision of tree search with the generality of random sampling, making it particularly effective in situations where the search space is too large to explore fully. (Chaslot et al. 2008) How does MCTS works:

- Selection — Starting from the root node, the algorithm selects successive child nodes down to a leaf node using a specific policy that balances exploration and exploitation. A common strategy used is the Upper Confidence Bound (UCB1), particularly its variation known as the Upper Confidence bounds applied to Trees (UCT).
- Expansion — Unless the leaf node ends the game or the decision process, it expands by adding one or more child nodes and choosing one of them.
- Simulation — From the new node, the algorithm performs a random simulation or roll out. This simulation continues until a predetermined condition is met, such as reaching a terminal state of the game or a maximum number of steps.
- Back propagation — The results of the simulation are then propagated back up the tree. Each node visited during the Selection phase is updated with the new information gained from the simulation, adjusting their values based on the outcome.

In the year 2017, authors came up with an evolution of AlphaGo, which is called AlphaGo Zero. The big difference between AlphaGo and AlphaGo Zero is, AlphaGo Zero develops it's own dataset from the scratch by playing the game against itself, unlike taking the dataset from KGS Go server as AlphaGo did. This development of a dataset from scratch is done using a combination of supervised learning and reinforcement learning. This dataset is generated by continuous self-play for 40 days and generated 29 million games of self-play. Later, AlphaGo Zero was played against AlphaGo, AlphaGo Lee, AlphaGo Fan, AlphaGo Master. Authors conducted an official 100 games between AlphaGo Zero and AlphaGo Master and AlphaGo Zero won 89 games out of 100. (Silver et al. 2017)

In the year 2018, authors came up with a paper called "Convolutional Neural Network for image detection and recognition". In this paper, authors applied CNN model to tackle image recognition and detection tasks, with a focus on exploring various architectural enhancements and optimization techniques to improve performance. This includes experimenting with different layers, filters, and training strategies to find the most effective configurations for processing and classifying image data accurately. some of the techniques that authors mentioned in the paper are Data Augmentation, Dropout. They performed their methodology on two different dataset, which are MNIST and CIFAR-10. MNIST contains handwritten digits and CIFAR-10 contains object detection dataset. MNIST contains 70,000 28X28 pixel images and CIFAR-10 contains 60,000 32X32 pixel images. When CNN model is applied on MNIST dataset they got 99.6 percent accuracy, and when CNN model was applied on CIFAR-10 dataset they got 80.17% accuracy. (Chauhan, Ghanshala, and Joshi 2018)

What is this CNN? How does it work? A Convolutional Neural Network (CNN) is a type of deep neural network that is particularly effective for processing data with a grid-like topology, such as images. CNNs are widely used in image and video recognition, image classification, medical image analysis, natural language processing, and other tasks involving large amounts of visual data. (Chauhan, Ghanshala, and Joshi 2018)

- Convolutional Layer — The core building block of a CNN is the convolutional layer. This layer applies a number of filters to the input. Each filter slides across the input image to produce a feature map or activation map. This process extracts important features from the input, such as edges, shapes, or textures.

- ReLU Layer — After each convolution operation, an activation function like the Rectified Linear Unit(ReLU) is applied. ReLU introduces non-linearity to the system, which allows the network to solve complex problems. It works by replacing all negative pixel values in the feature map with zero.
- Pooling Layer — Following the ReLU layer, a pooling layer is often applied. The most common type is max pooling, where the image is downsampled, reducing its dimensions by retaining only the maximum value in a local patch of pixels. This step reduces the computational complexity and helps in extracting robust features by providing a form of translation invariance.
- Fully Connected Layer — After several convolutional and pooling layers, the high-level reasoning in the neural network is done through fully connected layers. At this stage, the features extracted from the input image are used to determine the image's class.

In the year 2020, authors came up with as paper called "Automatically Designing CNN Architectures using the Genetic Algorithm for Image Classification" (Sun et al. 2020). In this paper, authors aim was to automatically design CNN architecture using genetic algorithm, which means the developers doesn't have to have expert knowledge on CNN and still design a good CNN architecture. Why do authors came up with automatically designing CNN architecture, they mentioned some of the challenges in manually designing CNN architecture. They are:

- Expertise requirement — Designing state-of-the-art CNN architectures typically requires deep expertise in CNNs and the specific problems being addressed, making it challenging for users without this specialized knowledge.
- Optimal architecture determination — Determining the optimal architecture for a CNN is a complex task that involves selecting the right depth, width, and types of layers, among other parameters. This complexity is difficult to navigate without substantial experience and understanding of CNNs.
- Time consuming process — The manual design of CNN architectures is a time-consuming iterative process that often involves a lot of trial and error, making it inefficient, especially for non-experts.
- Adaptability — Manually designed CNNs are often tailored to specific tasks or datasets and may not adapt well to different problems without significant modifications, requiring further expertise and effort.

This CNN architecture is build on skip layers and pooling layers. These components are integral to the CNN architectures that the genetic algorithm designs automatically. The skip layers help in building deeper networks without running into training difficulties, while pooling layers contribute to feature abstraction and dimensionality reduction.

- Skip layer — In a skip layer, the output of the first convolutional layer is passed through another convolutional layer and then added to the original input through a skip connection. This setup helps in training deeper networks by enabling the gradient to flow directly through the skip connections during backpropagation.
- Pooling layer — The pooling layer in this paper uses common settings where a 2x2 kernel is employed. The paper mentions two types of pooling: max pooling and mean pooling. Max pooling takes the maximum value from each sub-region of the input, while mean pooling calculates the average. The specific type of pooling is determined randomly during the initialization of the CNN architecture in the genetic algorithm process.

The authors had taken three categories: Manually designed CNN; Automatic + manually tuned CNN; and Automatically designed CNN (CNN-GA). They performed the above mentioned category CNNs on CIFAR-10 and CIFAR-100 datasets and calculated how many parameters needed, how many GPU days needed, and the manual assistance needed for each category of CNN. The results were really ground breaking, CNN-GA took less parameters, less GPU days, and without completely needing any assistance compared to categories of manually designed CNNs, and automatic + manual tuned CNNs.

What is this Genetic Algorithm? How does it work? A genetic algorithm is a type of optimization algorithm inspired by the process of natural selection, which is a fundamental mechanism of evolution in the biological world. The algorithm is used to solve optimization and search problems by mimicking the process of natural selection, where the fittest individuals are selected for reproduction to produce the offspring of the next generation. (Mirjalili and Mirjalili 2019) How does it work:

- Initialization — The process starts with a population of randomly generated individuals. Each individual, also known as a chromosome, represents a potential solution to the problem at hand.
- Fitness evaluation — Each individual in the population is evaluated to determine how fit or suitable it is as a solution to the problem.
- Selection — Based on their fitness, individuals are selected to breed a new generation. There are various selection methods, but the essence is that individuals with higher fitness have a higher chance of being selected for reproduction.
- Crossover(Recombination) — Pairs of individuals are selected based on their fitness, and parts of their genomes are exchanged to produce new offspring.
- Mutation — Mutation introduces genetic diversity into the population, allowing the algorithm to explore a broader range of solutions.
- The offspring are added to the population, and typically, some of the less fit individuals are removed. The population size usually remains constant from generation to generation. The algorithm terminates when either a maximum number of generations has been produced, or a satisfactory fitness level has been reached for the population. The best individual(s) from the final

population is considered as the solution to the problem.

In the year 2015, authors came up with a paper called "Training Deep Convolutional Neural Networks to Play Go" (Clark and Storkey 2015). The authors train deep convolutional neural networks (DCNNs) to predict moves by expert Go players, introducing novel techniques such as Network architecture, other Innovation techniques such as edge encoding, masked training, and reflection preservation.

- Network architecture — The DCNNs consist of several convolutional layers followed by a fully connected layer. The convolutional layers are designed to extract spatial hierarchies of features from the game board representations. Zero-padding is used to maintain the size of the feature maps throughout the convolutional layers. The final fully connected layer outputs a probability distribution over all possible next moves.
- Edge encoding — Edge encoding introduces an additional binary channel to the input that specifically denotes the edge of the board. This channel is padded with ones around the input, allowing the network to differentiate between the board's edge and empty points on the board. This approach helps the CNN recognize the significance of edge and corner positions in Go, which are crucial for effective gameplay.
- Masked training — Masked training addresses this by "masking" or zeroing out the outputs corresponding to illegal moves before applying the softmax layer during the network's training. This allows the network to focus on learning from only the legal moves, improving its ability to generalize and make valid predictions.
- Reflection preservation — Tying the weights of the network in a way that preserves symmetries, the number of unique parameters reduces, effectively compressing the model without losing its ability to generalize across symmetrically equivalent board states. This technique not only reduces the model's complexity but also imbues the network with a better understanding of the game's underlying symmetrical nature.

The authors represented the data in such a way, where binary matrices: 1 for current player's stones and another for another player; A third matrix indicates illegal moves due to the simple-ko rule. This representation is designed to capture the essential information about the game state without over complicating the input. They tested this approach on two datasets GoGoD and KGS. The training process involves minimizing the difference between the predicted move probabilities and the actual moves made by experts in historical game data. The authors' networks achieved move prediction accuracies of 41.06% and 44.37% on two Go datasets respectively.

In the year 2014, authors came up with a paper called "Move Evaluation in Go using Deep Convolutional Neural Networks" (Maddison et al.). This paper investigates the application of deep convolutional neural networks (CNNs) to predict expert human moves in the game of Go. The primary approach involves training a deep CNN with data from professional Go games to learn an evaluation function that can predict the next move in a given board state. This approach is notable because it directly tackles the challenge of constructing an evaluation function for Go, a task considered highly difficult due to the game's complexity and the vast number of possible positions. Their DCNN is built on:

- 12 Convolutional layers — DCNN with 12 weight matrices corresponding to 12 layers, utilizing rectified linear non-linearities. These layers include a mix of 5x5 and 3x3 filter sizes, operating on a 19x19 input space without pooling, with the outputs zero-padded back to 19x19. This deep structure allows the network to learn complex, hierarchical representations of the Go board, which are crucial for making accurate move predictions.
- Supervised learning — The DCNN is trained in a supervised learning, where it learns to predict expert human moves using a large database of professional 19x19 Go games. The goal is to match the input with the correct output, thereby learning to replicate expert decision-making. The document reports that the CNN achieved a prediction accuracy of 55%, which is comparable to a 6 dan human player.
- Probability distribution — The output layer of the DCNN is convolutional with position-dependent biases and two filters, producing two 19x19 planes that correspond to inputs for two softmax distributions of size 361 each. These distributions represent the network's predictions for the next move for black and white players, respectively. The action with the maximum probability from the softmax output is selected as the network's move when playing Go. This approach helps in picking the move with max probability.

They trained the CNN using KGS Go dataset. The network is trained using asynchronous stochastic gradient descent with 50 GPU replicas. The training involves 25 epochs with a batch size of 128 and a fixed learning rate, followed by fine-tuning on a single GPU with an annealed learning rate. Data augmentation is employed by including symmetric transformations of the board, enhancing the network's generalization capabilities. The DCNN achieved a move prediction accuracy of 55%. The DCNN won 97.2% of games against GnuGo and showed competitive performance against MCTS programs like MoGo and Pachi, even when these programs were allowed a large number of rollouts per move.

In the year 2008, authors came up with a paper called "Mimicking go experts with convolutional neural networks" (Sutskever and Nair 2008). This paper focuses on predicting moves made by expert Go players using convolutional neural networks (CNNs). Each Convolutional Neural Network has multiple convolutional kernels, where each CNN is focus on different parts of the training data to achieve better results. The size of the convolutional kernels of the first layer are 9X9 or 7X7, and the size of the hidden-to-output convolutional kernals are 5X5. They utilized a dataset known as the Gogod collection, which comprises

approximately 45,000 games played by Go experts. The primary objective was to enhance the performance of a computer Go player by accurately predicting professional moves. Their methodology involved using an ensemble of CNNs, with a notable result being that the ensemble predicted 36.9% of the moves correctly in the test expert Go games, a significant improvement over previous methods which predicted 34% correctly. Furthermore, the best single CNN from the ensemble achieved an accuracy of over 34%. These networks were trained on a subset of the data and learned rapidly, reaching 30% accuracy after learning from a small fraction of the training set, demonstrating the efficiency of using CNNs for this application.

In the year 2017, authors came up with a paper called "Bridging the gap between value and policy based reinforcement learning" (Nachum et al. 2017). This paper explore a new connection between value-based and policy-based reinforcement learning by focusing on the relationship between softmax temporal value consistency and policy optimality under entropy regularization. They introduce a novel reinforcement learning algorithm called Path Consistency Learning (PCL) which minimizes a notion of soft consistency error along multi-step action sequences extracted from both on-policy and off-policy traces. This algorithm is capable of generalizing both actor-critic and Q-learning approaches, and it notably allows for the unification of policy and state value models into a single entity, simplifying the architecture and potentially enhancing learning efficiency. In their experiments, the researchers utilized several benchmark tasks to compare PCL against conventional actor-critic and Q-learning algorithms. The results demonstrate that PCL not only meets but frequently surpasses these established baselines in terms of performance. Particularly, PCL achieves an impressive 36.9% prediction accuracy in certain test scenarios, which marks a substantial improvement over prior methods which had capped at around 34% accuracy. This outcome suggests that PCL is a robust method that leverages the benefits of both policy and value optimization techniques, making it a promising approach for complex reinforcement learning tasks.

In the year 2003, the authors came up with a paper called "Reinforcement learning as classification: Leveraging modern classifiers" (Lagoudakis and Parr 2003). This paper present a novel approach to reinforcement learning (RL) by integrating modern classification methods into the RL process. They propose a variant of approximate policy iteration, employing rollouts that facilitate the use of a pure classification learner, such as a Support Vector Machine (SVM), within the RL algorithm. This method addresses the complexities of feature engineering typically required in RL by utilizing the kernel trick of SVMs, potentially simplifying the application of RL as a more accessible "out-of-the-box" technique. The research experiments were conducted in the pendulum balancing and bicycle riding domains. The results demonstrate the effectiveness of using SVMs and neural networks as classifiers within the RL framework. In the pendulum balancing task, their approach achieved excellent balancing policies efficiently, often in one or two iterations. For the bicycle riding problem, the method showed promise, although it highlighted challenges related to parameter sensitivity and sample size requirements. The success in these tasks illustrates the viability of their method as an alternative strategy for tackling RL problems, reducing the reliance on extensive feature engineering and exploring the potential for broader applications of modern classification techniques in RL.

Related Work 2

When the concept of visually solving the game of Go was conceived, our initial inquiry led us to explore existing endeavors with similar approaches. Our investigation uncovered the seminal work by Mnih et al. (2015) titled "Human-level control through deep reinforcement learning," which presented a breakthrough in employing a visual approach to solve games. Specifically, the authors utilized a deep Q-network (DQN) to successfully conquer 49 distinct Atari 2600 games. The significance of their achievement lay not solely in the number of games mastered but in the innovative manner in which the AI assimilated information.

What is this DQN? How does it work? Deep Q-Network (DQN) is a reinforcement learning algorithm that has proven particularly effective in solving complex decision-making problems, especially in domains like artificial intelligence (AI) gaming. DQN, rooted in the broader framework of Q-learning, introduces neural networks to approximate and optimize the Q-values of different state-action pairs efficiently. The algorithm employs a memory replay mechanism, allowing it to break correlations between consecutive experiences, thereby stabilizing learning and enhancing overall performance (Mnih et al. 2015). DQN operates through the following key steps:

- State Representation – DQN works with a neural network that takes the game state as input and outputs Q-values for each possible action. This network is trained to predict the expected cumulative reward for taking a particular action in a given state.
- Experience Replay – To mitigate the issue of correlated experiences, DQN utilizes a replay buffer to store and randomly sample past experiences. This helps break temporal correlations in the data, preventing the model from overfitting to recent experiences.
- Target Network – DQN employs two separate networks - the target network and the online network. The target network is periodically updated with the weights from the online network, stabilizing the learning process and improving convergence.
- Exploration vs. Exploitation – DQN balances exploration and exploitation by using an ε-greedy strategy. With probability ε, the algorithm selects a random action to explore the environment, while with probability $1 - \varepsilon$, it exploits the current knowledge by selecting the action with the highest Q-value.

- Loss Calculation and Backpropagation – The temporal difference error, calculated as the difference between the current Q-value prediction and the target Q-value, is used to compute the loss during each iteration. This loss is then backpropagated through the neural network to update its weights.

Going back to the paper in essence, the AI's perspective mirrored that of a human observer interacting with the Atari games - a visual interpretation of the screen and its contents constituted its entire knowledge base. Notably, prior to initiating the learning process, the AI was equipped only with information regarding the game inputs, without explicit knowledge of their functionalities (Mnih et al. 2015).

The outcomes of this visual-centric approach were compelling. Across the 49 Atari 2600 games, the AI either matched or surpassed the performance of professional human game testers in 23 instances. Among these, only one game had a prior linear learner that outperformed the DQN model. In the remaining 26 games, only three instances saw prior linear learners outperforming DQN. Remarkably, DQN scored below 50% of the professional human game tester's scores in merely 12 games, with 0% dictated by what random inputs achieved. Notably, there was only a single game where DQN failed to outperform random inputs (Mnih et al. 2015).

The paper's conclusion emphasized the potency of leveraging state-of-the-art machine learning techniques with biologically inspired mechanisms, creating agents capable of mastering a diverse array of challenging tasks (Mnih et al. 2015). Inspired by this success, we aim to extend and apply these concepts in the context of solving the intricate game of Go. Through this project, we aspire to showcase the continued effectiveness of integrating cutting-edge machine learning methodologies with biologically inspired mechanisms in conquering complex challenges, ultimately advancing the capabilities of artificial intelligence.

While the aforementioned paper (Mnih et al. 2015) showcased the feasibility of solving games through visual inputs, our curiosity extended to the algorithms commonly employed in contemporary image processing, a field pivotal to our exploration. While Convolutional Neural Networks (CNNs) emerged as the prevalent choice, we delved deeper into alternative methodologies. Notably, the paper titled "Generative Image Inpainting with Contextual Attention" by Yu et al. (2018) caught our attention, introducing a novel approach involving a unified feed-forward generative network with a contextual attention layer for image inpainting. This innovation aimed to address issues such as blurry textures and boundary artifacts inherent in traditional CNN-based approaches. While this wouldn't be a significant concern in our specific use case, the paper still provides fascinating insights into image inpainting.

While image inpainting might seem unrelated to the strategic complexities of the game of Go, an intriguing parallel emerged. The paper by Yu et al. (2018) highlights: "The core challenge of image inpainting lies in synthesizing visually realistic and semantically plausible pixels for the missing regions that are coherent with existing ones" (Yu et al. 2018). In the context of Go, the analogous challenge lies in playing stones in a manner that aligns with existing configurations while remaining semantically plausible for future moves.

The methodology in the paper by Yu et al. (2018) involved testing the algorithm on photos with randomly removed square areas, demonstrating its prowess in achieving high-quality inpainting across diverse datasets, including CelebA faces, CelebA-HQ faces, DTD textures, ImageNet, and Places2. Evaluation metrics like peak signal-to-noise ratio (PSNR) and total variation (TV) loss were employed, providing a quantitative measure of the inpainted image's proximity to the original (Yu et al. 2018). Simplifying this connection to Go, we envision a scenario where the AI, presented with a game board featuring strategically missing areas, could effectively predict and inpaint suitable moves based on its learned patterns and contextual understanding. This innovative approach opens avenues for enhancing the AI's decision-making processes in the intricate landscape of Go gameplay.

In the pursuit of implementing algorithms akin to those in Generative Image Inpainting, acquiring a robust dataset is imperative. The newest reputable dataset we found for Go is PGD: A Large-scale Professional Go Dataset for Data-driven Analytics . As described by the author, it is the "first professional Go dataset for sports analytics." The dataset contains a large amount of player, tournament, and in-game data, facilitating extensive performance analysis (Gao 2022).

The utility of the PGD dataset was demonstrated in a paper where various machine learning algorithms, including Random Forest, XGBoost, LightGBM, CatBoost, ELO, TrueSkill, and WHR, were trained to predict game outcomes. Although all methods exhibited a success rate exceeding 50%, CatBoost emerged as the most proficient, boasting an impressive accuracy of 75.30% and an MSE of 0.1623. It is crucial to note that the primary objective of utilizing PGD was not to enhance these algorithms but to showcase the dataset's versatility across a diverse array of machine-learning techniques, an endeavor in which the paper succeeded admirably (Gao 2022).

PGD encapsulates a comprehensive dataset, comprising 98,043 games played by 2,148 professional players spanning from 1950 to 2021. The dataset offers detailed meta-information for each player, game, and tournament, along with analysis results for every move in the match, meticulously evaluated by an advanced AlphaZero-based AI. While certain aspects of this expansive dataset may not directly align with our specific application, such as player-specific meta-information, PGD stands out as one of the most meticulously curated datasets available for the

game of Go (Gao 2022). This meticulous curation proves particularly beneficial for our project, given our limited experience in playing Go and the challenges associated with creating our dataset.

As we started to experiment with using image infill on the PSG dataset to solve the game of Go, we realized that the "Generative Image Inpainting with Contextual Attention" by Yu et al (Yu et al. 2018), was an older paper. In fact, it is something of a starting point for the following three papers. "Image Inpainting Based on Generative Adversarial Networks,"(Jiang et al. 2020) "CR-Fill: Generative Image Inpainting with Auxiliary Contextual Reconstruction,"(Zeng et al. 2021) and "ITrans: generative image inpainting with transformers." (Miao et al. 2024) The following paragraphs will go over what was learned from the above papers and how it will apply to our project.

Starting with "Image Inpainting Based on Generative Adversarial Networks" it is trained to fill missing regions in CelebA dataset and LFW dataset, even irregular missing regions, which is important since we will most likely not have perfect square missing regions as in the older paper. They accomplish this with a CNN algorithm known as Wasserstein GAN.(Jiang et al. 2020)

What is this GAN? How does it work?

- Generative Adversarial Networks (GANs) were introduced by Ian Goodfellow in 2014. GANs operate as a two-player game between a generator and a discriminator, aiming to generate data that closely resembles real data. The generator learns to produce data samples from a random noise input, while the discriminator's role is to distinguish between real data and generated data. The ultimate objective of GAN training is to reach a Nash equilibrium, where the discriminator cannot differentiate between real and generated samples.(Jiang et al. 2020)
- Despite the promise of GANs, training them can be challenging, with difficulties in interpreting the losses of the generator and discriminator. To address these issues, Wasserstein GAN (WGAN) was proposed. WGAN modifies the training process by optimizing the Wasserstein distance, enhancing stability and effectiveness. The Wasserstein distance focuses on measuring the discrepancy between real and generated data distributions more accurately than traditional methods.(Jiang et al. 2020)

Overall while this paper improves on the first one it was not what we where looking for in this project as it relies very heavily on information found in the image to be infilled, rather than the test set.(Jiang et al. 2020) With that said, they did outperform the other state-of-the-art image infill generators on the datasets they trained on using the same comparison methods as used on "Generative Image Inpainting with Contextual Attention."(Yu et al. 2018)

While the last results were a step in the right direction, "CR-Fill: Generative Image Inpainting with Auxiliary Contextual Reconstruction," was arguably even less of what we were looking for as they were building on what the last paper did, but adding the Contextual Reconstruction. (Zeng et al. 2021) Again, what this means is they have the same general setup as described above with the generator and a discriminator, but they are using Contextual Reconstruction, which to extremely oversimplify takes what it assumes is the best match for the missing part of the image from the non missing part of the image and generate the infill using that.

Again, this is not what we necessarily want. Our generater will not need to care as much about what the infill looks like, it should be more concerned with making the current image look more like the general images it was trained on. With that being said, once again "CR-Fill: Generative Image Inpainting with Auxiliary Contextual Reconstruction," outperformed the new state-of-the-art algorithms when measured in the way explained with the first image inpainting paper.

Finally, we have come to the most recent and best match for our implementation: "ITrans: generative image inpainting with transformers."(Miao et al. 2024) This once more is using CNN, but this time, they use a global and local transformer where the global is focused on making it fit within the larger environment and the regional works on blending the individual particles. The main reason this one is a better fit is because of how irregular the shapes it can fill, which, as mentioned above, will be necessary for inpainting a Go board. The results were once again measured using the method explained in the first image infill paper. Once again, the results were exceptional for the time period.

The last papers we looked at concerned improvements we would like to make in the future. The first paper we looked at was "Pose2Trajectory: Using transformers on body pose to predict tennis player's trajectory."(AlShami, Boult, and Kalita 2023) The goal of this paper was to predict the future movement of a tennis player by feeding in the first part of a video and having it predict future movement. It is fed the trajectory information of the players, as well as all joints and ball positions. This achieved good results for 25 frames in advance for up to 500ms ahead. However, the best results were for 250ms ahead .

As for what we could use from this paper in our project is the idea of taking a series of images to predict the future. Currently, we only take one image to predict the future, but results would most likely improve if we used a series of images as they do in this paper.

Looking further into this idea, we read the paper "Deep Learning in Next-Frame Prediction: A Benchmark Review." (Zhou, Dong, and El Saddik 2020) This paper was an overview of two overarching approaches of predicting future frames.

- Sequence-to-One Architecture - previous frames on the

channel dimension as input images for networks, with chronological ordering, which generates possible future frames from a single image through specific techniques.

- Sequence-to-Sequence Architecture - predict one frame per time step, incorporating different weights for each time point to optimize temporal prediction, leveraging recurrent neural networks to capture temporal relationships alongside spatial features.

There were four algorithms for each approach. All of them were evaluated on four datasets, three of which are human movement datasets and one of which is a self-driving car dataset. Overall, the best performers were Sequence-to-Sequence Architecture using CNN with multilayer ConvLSTM, and Sequence-to-One Architecture using a structure of the pyramid autoencoder.(Zhou, Dong, and El Saddik 2020)

Both papers so far have mentioned LSTM, and the last paper cited it as one of the best-performing methods. So we decided to find a paper that looked more into LSTM. The one we found was "Evaluation of Deep Learning Models for Multi-Step Ahead Time Series Prediction." (Chandra, Goyal, and Gupta 2021) They where comparing Standard LSTM, bidirectional LSTM, encoder-decoder LSTM, and CNNs. All of these methods were used to predict the future of financial data sets. In particular, simulated time series Mackey-Glass, Lorenz, Henon, and Rossler. The real-world time series are Sunspot, Lazer and ACI-financial time series. The results indicated that encoder-decoder and bi-directional LSTM networks provide the best performance for both simulated and real-world time series problems.

Overall going forward we think it would be interesting to implement an encoder-decoder and bi-directional LSTM network or some other form of LSTM and see if this gives us any improvement over our current implementation.

Methodology

As we have mention above, we are trying to solve the game of Go visually, why? because when two humans play this game they visually analyze the board and only think a head of 3-5 moves, also to push the limits of AI and see if it can solve the game of Go visually or not. We are taking the same idea which is solving the game of Go visually, and trying two different approaches using the same AI algorithm which is Convolutional Neural Networks (CNN).

Methodology 1

One of the approaches is implementing "Automatically Designing CNN Architectures Using the Genetic Algorithm for Image Classification". Designing CNN architecture using Genetic algorithm. Below is the algorithm overview of this paper:
During evolution, a population is randomly initialized with the predefined population size, using the proposed encoding strategy to encode the predefined building blocks. Then, a counter for the current generation is initialized to zero. During evolution, the fitness of each individual, which encodes a particular architecture of the CNN, is evaluated on the given dataset. After that, the parent individuals are selected based on the fitness, and then generated a new offspring by the genetic operators, including the crossover and mutation operators. Then, a population of individuals surviving into the next generation is selected by the environmental selection from the current population. Specifically, the current population consists of the parent population and the generated offspring population. Finally, the counter is increased by one, and the evolution continues until the counter exceeds the predefined maximal generation. The pseudo-code of proposed algorithm is:

- Input — A set of predefined building blocks, the population size, the maximal generation number, the image dataset for classification.
- Output — The discovered best architecture of CNN.
- $P_0 \leftarrow$ Initialize a population with the given population size using the proposed variable-length encoding strategy;
- $t_0 \leftarrow 0$;
- while $t <$ the maximal generation number do:
- Evaluate the fitness of each individual in P_t using the proposed acceleration components;
- $Q_t \leftarrow$Generate offspring from the selected parent individuals using the proposed mutation and the crossover operators;
- $P_{t+1} \leftarrow$Environmental selection from $P_t \cup Q_t$;
- $t \leftarrow t + 1$;
- end
- Return the individual having the best fitness in P_t.

Below is the breakdown of the proposed algorithm.
For Population initialization:

- Input — The population size is T.
- Output — The initialized population P_0.

For Fitness evaluation:

- Input — The population P_t of the individuals to be evaluated, the image dataset for classification.
- Output — The population P_t of the individuals with their fitness values.

For Individual fitness evaluation:

- Input — The individual *individual*, the available GPU, the number of training epochs, the global cache *Cache*, the training data D_{train} and the fitness evaluation data $D_{fitness}$ from the given image classification dataset.
- Output — The individual *individual* with its fitness.

For Offspring generating:

- Input — The population P_t containing individuals with fitness, the probability for crossover operation p_c, the probability for mutation operation p_m, the mutation operation list l_m, the probabilities of selecting different mutation operations p_l.
- Output — The offspring population Q_t.

For Environmental selection:

- Input — The parent population P_t, the offspring population Q_t.
- Output — The population for the next generation P_{t+1}.

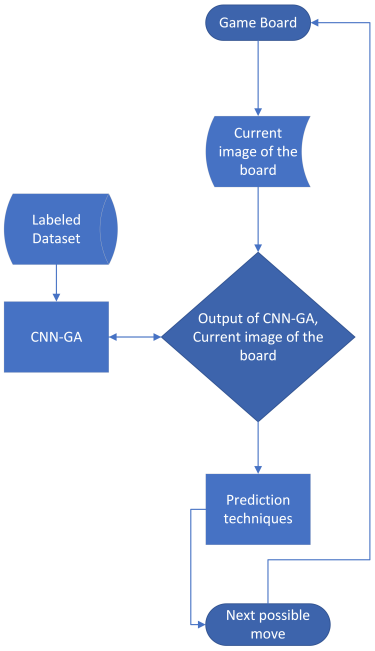

Figure 1: The flowchart of how Methodology 1 will work in its environment

We have a labeled dataset (black stones, white stones, grid) (Data 2023), we will take that dataset and represent it according to the papers: By taking three matrices: 1 for representation of black stones, another one for the representation of white stone, and the third for the representation of empty grid line (legal moves) (Clark and Storkey 2015) (Maddison et al.), feed the dataset to the above CNN-GA algorithm and train the model, and employ the prediction techniques from papers: probability distribution over the grid points at empty grid points (Clark and Storkey 2015), use a supervised learning to learn the expert moves from the dataset (Maddison et al.). Once the model is trained on the dataset, the trained model will be integrated with the board as one of the players.

If the above methodology fails, as a Plan B, we will integrate policy networks and value networks for reading the board positions and predicting the next possible move. The current board position is fed to the value networks and the output of the CNN-GA algorithm is fed to the policy network, and by implementing reinforcement learning (which helps the value and policy networks), we will predict the next possible move.

Methodology 2

The other approach we will try is currently to implement the "ITrans: generative image inpainting with transformers" program to train the AI in the flowchart shown below. Of course, besides the image infill AI, we also need the code that handles the conversion from SGF to an Image, code that will mask said image so that we can infill it, and finally, some way to read the infill and make a move back on the Go board. This is currently being designed to be handled with separate PowerShell and Python scripts. For visual aid on how all these reactions interact, please reference Figure 2.

This AI is trained on 120 move go games of 512*512px images we generated from (Redmon 20XX).

As for the algorithm used by ITrans, while I do not have to make it, I figure I should explain it anyway, at least in simple terms, since one of the reasons I am currently planning on implementing an already existing to image inpainting is because I do not think I could implement one that would work this well in the time left this semester. What this also means is that what they are doing is hard to explain in one paragraph, but here is my attempt (Miao et al. 2024).

- ITrans is made up of several sections. To start, the masked image is fed into an edge generator, which gives us an edge map. The edge map is fed into our first section.
  - Local Transformer Branch: This section takes the edge map + a grayscale version of the Image and 4 layers of CNN-based encoder/decoder network. The results of this are then fed into Local transformer.
    - Local transformer: the generates the final local attention map.

- While this is happening, the original image is also fed into another section:
  - Global Transformer Branch: Also uses 3 layers of CNN-based encoder/decoder network however, there is another section that connects each the encoder layers to its corresponding decoder layer called the Global transformer:
    - Global transformer: Does global self-attention on feature maps, which splits images into fixed-size particles before adding the class tokens that correspond to that particle. This contains info such as position.

    As we then rebuild the image with the 3 layers of CNN. The first layer also implements the result returned from the local transformer branch.

Finally, this outputs our inpainting result. For a much better explanation, please read "ITrans: generative image inpainting with transformers." But this is how I read it as working, simplified from 3 pages and 3 images (Miao et al. 2024).

Experiment Design

Our experiment design includes three parts: The game setup, AI player 1, and AI player 2. Coming to the game

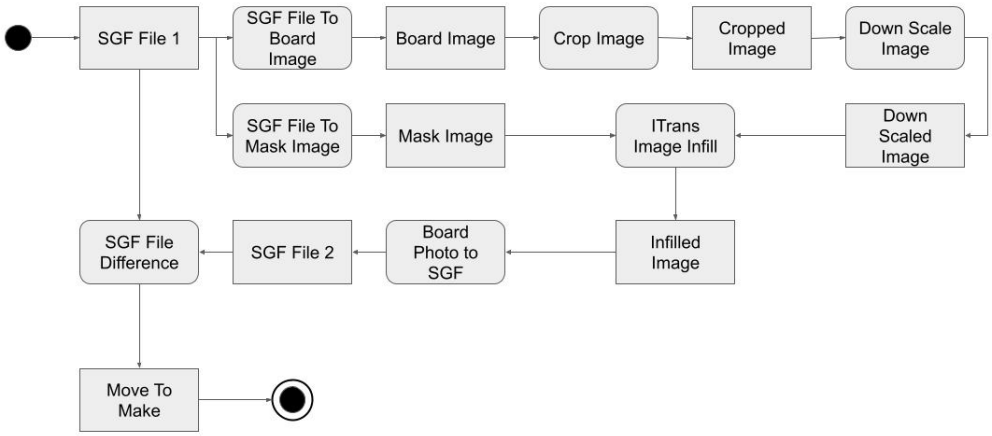

Figure 2: The above diagram is a flowchart for how Methodology 2 will work in its environment

setup, we did setup a board which is represented in 2D array, starting with the black move first and also an option called ”pass”, where you can skip your turn.

Player 1 will use CNN-GA (building CNN using genetic algorithm). The goal is to read the current board positions, detect the legal moves, and then the trained model should be able to predict the next possible move. As CNN-GA has skip layers, we do not have to reduce the learning time and size of the dataset.

Player 2 will use the image infill AI. While the final goal is for the AI to be able to make moves for any board state. To reduce learning time and size of the data set, currently, we will only be implementing it for move 119 to 120 and expanding in that area time allowing.

These two AI players will be integrated with the board setup and play the game against each other. Overall we will use the same method for testing its effectiveness. However, instead of comparing the entire game result, we will just be comparing the result of one move to what the current best, or other good AI players would recommend/not recommend playing.

Results

Currently, neither of the AI is fully working. Both AI has some training complete, but neither is correctly loading training data, and also have not trained enough to make a successful AI. That said, we do have the ITrans repo outputting testing values, but those are also not working. However, from what training we have completed, it seems that there is some promise of getting at least some successful results if we continue.

For the approach 1 (methodology 1), I worked on a database (Data 2023) which is labeled (blackstones, whitestones, and grid). I tried to feed the dataset to CNN-GA model and it ran into errors. After more exploration of internet, the code, and the dataset, I understand what the problem is and then I tried to mess around with libraries especially keras library, where I tried to give path to my dataset and it trained for like three hours and then ran into an error, which the error led to keras library. Before messing with the libraries, it warned me ”Automatically generated. Do not edit”, I did it anyways, I also tried to see if there is anyway to generate it automatically, and no luck. This is where my approach halted by the end of the semester.

The best visualization we have of the current training process is shown in Figure 3. Namely, as more training interactions are run, you can see ITrans beginning to understand what the edges should look like under the mask. However, you can also see the mask is not being loaded correctly since you should not see any empty spaces and should see all the stones. Overall, though, these are the results after half of the training pipeline run once on about 9000 images; as we generate more images and run the full training pipeline, possibly more than once, you can see the potential for this to work.

Discussion

While we have not had any success currently, we have started to see some progress that seems to indicate this may work. However, we still have to fix the data loading for both algorithms and while one is outputting results, both of them need work on that end as well. Going forward, we first need to get both algorithms working. From there, we will select the one that is working best and make improvements from there. The overall idea is if we can get any positive results with algorithms not meant for this problem, we should be able to get good results with one we make to solve this problem.

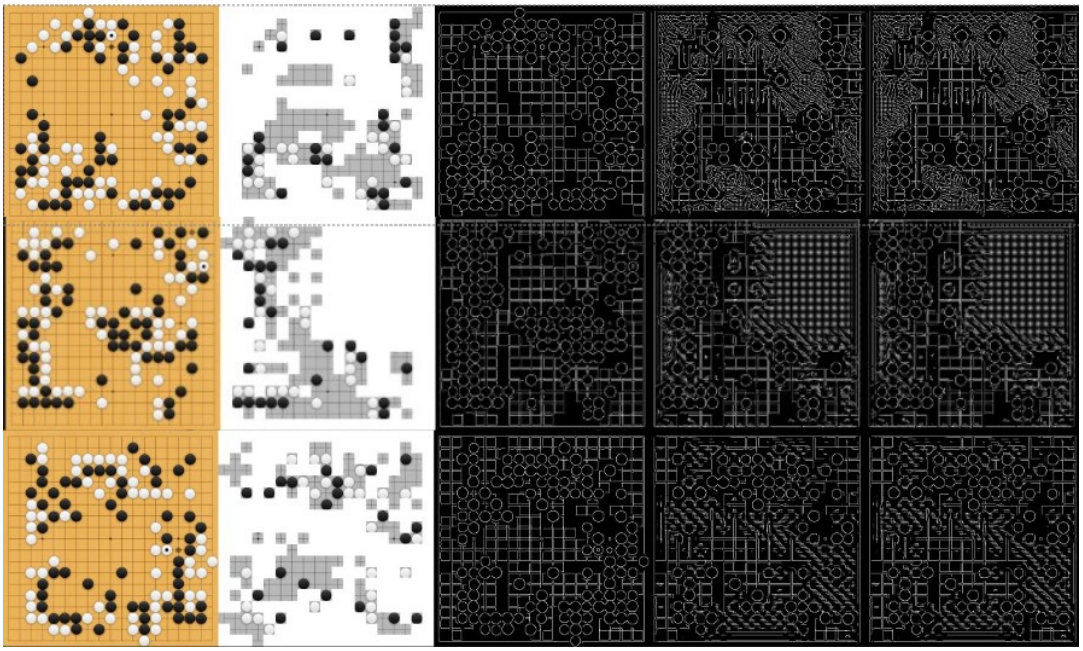

Figure 3: This is the output of the ITrans doing edge generation on training iterations 1, 3, 6

Conclusion

In conclusion, this project envisions a transformative future where artificial intelligence (AI) becomes a catalyst for improving productivity, healthcare, and sustainability. The unique challenges presented by the complex game of Go, with its deceptively simple yet strategically intricate rules, serve as a proving ground for advancing AI capabilities. The ability to place stones anywhere on the Go board introduces a dynamic and expansive search space, pushing traditional AI approaches to their limits. Inspired by the transformative potential of contemporary Visual AI, we propose the integration of Convolutional Neural Networks (CNN) with visual input feed through either image classification or image infill to elevate AI's problem-solving capacities. By integrating CNN with visual input, we aim to train AI to visually recognize patterns through image classification or image infill and improve problem-solving capabilities. This approach, if successful, could bridge the gap between AI and human expertise in strategic decision-making, not just in Go, but across various real-world applications. We hope this project contributes to the advancement of AI and unlocks new possibilities for AI-powered solutions.

References

AlShami, A.; Boult, T.; and Kalita, J. 2023. Pose2Trajectory: Using Transformers on Body Pose to Predict Tennis Player's Trajectory. *Journal of Visual Communication and Image Representation*, 97: 103954.

Chandra, R.; Goyal, S.; and Gupta, R. 2021. Evaluation of Deep Learning Models for Multi-Step Ahead Time Series Prediction. *IEEE Access*, 9: 83105–83123.

Chaslot, G.; Bakkes, S.; Szita, I.; and Spronck, P. 2008. Monte-carlo tree search: A new framework for game ai. In *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, volume 4, 216–217.

Chauhan, R.; Ghanshala, K. K.; and Joshi, R. C. 2018. Convolutional Neural Network (CNN) for Image Detection and Recognition. In *2018 First International Conference on Secure Cyber Computing and Communication (ICSCCC)*, 278–282. Jalandhar, India.

Chen, J. X. 2016. The evolution of computing: AlphaGo. *Computing in Science & Engineering*, 18(4): 4–7.

Clark, C.; and Storkey, A. 2015. Training deep convolutional neural networks to play go. In *International Conference on Machine Learning*, volume 38 of *Proceedings of Machine Learning Research*, 1766–1774. PMLR.

Data, S. 2023. Go Positions Dataset. https://universe.roboflow.com/synthetic-data-3ol2y/go-positions. Visited on 2024-03-24.

Gao, Y. 2022. PGD: A Large-scale Professional Go Dataset for Data-driven Analytics. In *2022 IEEE Conference on Games (CoG)*, 284–291. Beijing, China.

Jiang, Y.; Xu, J.; Yang, B.; Xu, J.; and Zhu, J. 2020. Image Inpainting Based on Generative Adversarial Networks. *IEEE Access*, 8: 22884–22892.

Jumper, J.; Evans, R.; Pritzel, A.; and et al. 2021. Highly accurate protein structure prediction with AlphaFold. volume 596, 583–589.

Lagoudakis, M. G.; and Parr, R. 2003. Reinforcement learning as classification: Leveraging modern classifiers. In *Proceedings of the 20th International Conference on Machine Learning (ICML-03)*, 424–431.

Maddison, C. J.; Huang, A.; Sutskever, I.; and Silver, D. ???? MOVE EVALUATION IN GO USING DEEP CONVOLUTIONAL NEURAL NETWORKS.

Miao, W.; Wang, L.; Lu, H.; and et al. 2024. ITrans: generative image inpainting with transformers. *Multimedia Systems*, 30(21): 1–11.

Mirjalili, S.; and Mirjalili, S. 2019. Genetic algorithm. *Evolutionary algorithms and neural networks: Theory and applications*, 43–55.

Mnih, V.; Kavukcuoglu, K.; Silver, D.; and et al. 2015. Human-level control through deep reinforcement learning. *Nature*, 518(7540): 529–533.

Nachum, O.; Norouzi, M.; Xu, K.; and Schuurmans, D. 2017. Bridging the gap between value and policy based reinforcement learning. *Advances in neural information processing systems*, 30.

Redmon, J. C. 20XX. Joe's Go Database.

Silver, D.; Huang, A.; Maddison, C. J.; Guez, A.; Sifre, L.; Van Den Driessche, G.; Schrittwieser, J.; Antonoglou, I.; Panneershelvam, V.; Lanctot, M.; et al. 2016. Mastering the game of Go with deep neural networks and tree search. *nature*, 529(7587): 484–489.

Silver, D.; Schrittwieser, J.; Simonyan, K.; Antonoglou, I.; Huang, A.; Guez, A.; Hubert, T.; Baker, L.; Lai, M.; Bolton, A.; et al. 2017. Mastering the game of go without human knowledge. *nature*, 550(7676): 354–359.

Sun, Y.; Xue, B.; Zhang, M.; Yen, G. G.; and Lv, J. 2020. Automatically designing CNN architectures using the genetic algorithm for image classification. *IEEE transactions on cybernetics*, 50(9): 3840–3854.

Sutskever, I.; and Nair, V. 2008. Mimicking go experts with convolutional neural networks. In *Artificial Neural Networks-ICANN 2008: 18th International Conference, Prague, Czech Republic, September 3-6, 2008, Proceedings, Part II 18*, 101–110. Springer.

Yu, J.; Lin, Z.; Yang, J.; Shen, X.; Lu, X.; and Huang, T. S. 2018. Generative Image Inpainting with Contextual Attention. In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 5505–5514. Salt Lake City, UT, USA.

Zeng, Y.; Lin, Z.; Lu, H.; and Patel, V. M. 2021. CR-Fill: Generative Image Inpainting with Auxiliary Contextual Reconstruction. In *2021 IEEE/CVF International Conference on Computer Vision (ICCV)*, 14144–14153. Montreal, QC, Canada.

Zhou, Y.; Dong, H.; and El Saddik, A. 2020. Deep Learning in Next-Frame Prediction: A Benchmark Review. *IEEE Access*, 8: 69273–69283.