# Image generation using Monte-Carlo Tree Search and Convolutional Neural Networks

Nymisha Bandi & Abhilash Chenreddy

**Abstract**

This research integrates CNN algorithm as a reward mechanism to Monte Carlo Tree Search (MCTS) to genetate images in real time. We investigate a drawing environment in which an autonomous agent sketches images. We combine Monte Carlo Tree Search with image classifiers such as deep Convolutional Neural Networks. We found that the agent is able to estimate after each iteration and paint the number correctly. However, limited computatinal power resulted in lesser number of epochs and iterations, making the images shaky. Also, the MNIST dataset used has multiple images for a single number, which has confused the agent at times. This can be resolved by using a dataset with single image for each number and use higher number of epochs and iterations.

*Keywords:* MCTS, CNN, Image generation

## 1. INTRODUCTION

Convolutional Neural Networks have proved to be the go to method when it comes to image classification. Pre-trained models are available on keras and TensorFlows , which can be modified to suit our purpose. MCTS is a heuristic search algorithm which plays a vital role in decision process. The application of MCTS is based on multiple actions. At each state, the game is played out to the very end by selecting moves at random. The final game result of each action is then used to weight the states in the game so that nodes with better scores are more likely to be chosen in future actions. The classifier evaluates how much the agents trajectory resembles the desired class, and that score is used as the reward backpropagated by MCTS to choose the final action.

### *1.1. Motivation*

The primary motivation behind this project is to integrate Monte Carlo Tree Search (MCTS) with Convolutional Neural Networks (CNNs) to create an autonomous agent which can perform desired function in real time. We would want to design the agent such that it considers random actions at a time stamp and choose the one that gives the highest reward. This process continues until a specified number of iterations is reached. Our goal is to produce a result that is the closest resemblance of the desired object class. The entire system acts as a real time collaborative environment between a human and a creative autonomous agent. MCTS is used to visualize the random actions whereas CNN is employed to compare the result image with target image and get a score, which can be back propagated to MCTS as reward.

### *1.2. Business goals*

This project has multiple business implications. So far, MCTS has been majorly used in the area of Atari games but has limited implementation in real time business applications. Here, by coupling MCTS with CNN, we have built an agent which learns to draw on its own by trying out various options with no human inputs, which can be a very useful application in the area of arts.

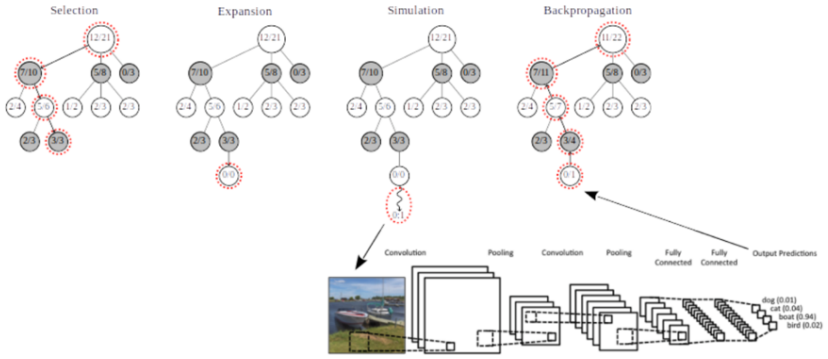
### *1.3. Applications and significance*

By coupling a learning algorithm with deep learning classifying algorithm, we have built an agent which learns to perform an action by exploring a set of actions greedily and picking the one that maximises the chance of classifying the resultant image correctly. By replacing CNN with other reward agents, this model can be applied to various areas like, autonomous text and music generation.

## 2. Background & Literature

The context of this work it to build computational models of creativity. For this purpose, we focus on exploring the opportunities provided by Deep artificial

Figure 1: Overview of MCTS and CNN agent




Neural Networks, especially Convolutional Neural Networks (CNN) in unison with with agent-based Artificial Intelligence methods such as Monte Carlo Tree Search (MCTS) in order to drive the autonomous drawing agents.

Deep convolutional architectures gained traction in image classification in the late 1980s, and they emerged to become the state of the art model when to comes to image classification . Besides this, CNNs has been successfully integrated to drive AI agents in areas such as those playing Atari games. In such cases, the CNN model has effectively acted as the agents eyesgiving them a reward for every action it takes, which can be used by the agent. MCTS has proved to be a successful algorithm which can be used the agent which makes optimical decisions based on the information provided by the CNN.

There is a very limited literature which dealt with the kind of models we are trying to build. However, one[2] by the author,Memo Akten is in the similar lines. it has used models like multiple logistic regression and CNN as reward agents. We have evaluated multiple models but decided to use CNN as it is one of the best models out there when it comes to image classification.

## 3. Data Description

A dataset of numbers is needed so that the agent can compare at the end of each iteration. As we planned to sketch numbers, we decided to use MNIST dataset for this purpose. MNIST has 10 labels for numbers from 0 to 9 handwritten digit images. This database of handwritten digits is used to train our CNN to classify images of numbers. All images are size normalized to fit in a 20x20 pixel box and there are centered in a 28x28 image using the center of mass.

Once MCTS draws a image in Turtle, it is converted so as to make it readable by the CNN model and compare with the final resultant matrix of the MNIST dataset. For this, the .eps file resulted from Turtle is converted to a 4 dimensional vector of 28X28 size which can now be read by CNN model.

## 4. Methodology

The MCTS agent we used follows the directions and sketches the image using LOGO Turtle which comes in the Turtle package in python. Turtle is given a speed of 0(fastest) for the exploration. The angles we used for turtle to moves are -30° to +30°. The increment in the angle we considered is 10°. This would mean a total of 6 actions based only on the angle of rotaion or the degrees of freedom. It will be given a set of actions it is allowed to take, among which one will be chosen based on the instructions from MCTS. It will leave a trail as it moves. For this scenario, the LOGO is allowed to go forward, backward, turn clockwise or anti- clockwise and the various speed ranges. We use MCTS to explore a given number of alternative actions and paths at every timestep. The actions are chosen as a combination of the speed and the angle by which the turtle can turn. Every action in MCTS has 2 properties: Number of times the action is taken and the Reward associated with that action.

After each simulation, the whole trajectory of the agent so far will be considered as an image. This image is converted into a 28*28 image which is of grayscale and centered in order to fed into the image classifier, which is the CNN

here. The classifier now evaluates if the image drawn by the agent resembles the desired class and returns 1 or 0 as reward if it resembles or does not resemble respectively. This reward is then backpropagated to MCTS to decide on the next action. The next action is then picked using the accumulated results and the action with the highest Q-value is chosen for further exploration. This is continued till the entire image is obtained. The MCTS algorithm works as shown below:

**function** MCTS(s,d)
    *loop*:
    SIMULATE(s,d)
    **return** $argmax_a Q(s,a)$
**function** SIMULATE(s,d)
    **if** $d == 0$ **then**
        **return** 0
    **if** $s \notin T$ **then**
        $T = T \cup s$
        $N(s) \leftarrow 1$
        **return** ROLLOUT(s,d)
    $N(s) \leftarrow N(s) + 1$
    **if** $Untried\_actions(s)$ **then**
        $a \leftarrow random(Actions(s))$
        *update* $Untried\_actions(s)$
    **else**
        $a \leftarrow argmax_a(Q(s,a) + c\sqrt{\log \frac{N(s)}{N(s,a)}}$
    $s' \leftarrow next\_state(s,a)$
    $q \leftarrow Reward(s,a,s') + \gamma SIMULATE(s', d-1)$
    $N(s,a) \leftarrow N(s,a) + 1$
    $Q(s,a) \leftarrow Q(s,a) + \frac{q - Q(s,a)}{N(s,a)}$
    **return** q
**function** ROLLOUT(s,d)

24: **if** $d == 0$ **then**

25: **return** 0

26: **else**

27: $a \leftarrow random(Actions(s))$

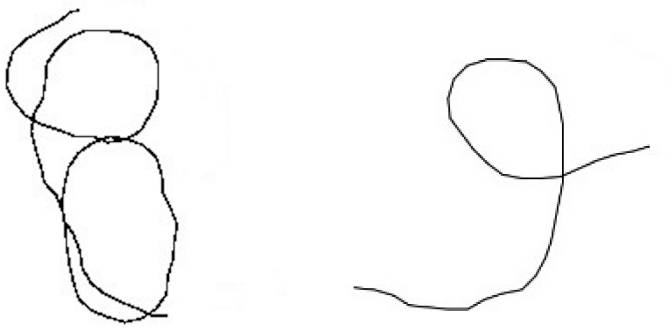
28: $s' \leftarrow next\_state(s, a)$

29: $r \leftarrow Reward(s, a, s')$

30: **return** $r + \gamma ROLLOUT(s', d - 1)$

Initially the user enters a number which the agent would later sketch. The MCTS agent would initially start at a predefined initial point and explores the arena to a given depth, 100 here for a given number of iterations, 1000 here. Each of these iterations will be compared with the MNIST dataset by CNN and the result will be given back to the MCTS as reward. After 1000 iterations at each node, the score for all the iterations will be used to decide the best action. The action with the best score is chosen and the same process is followed to pick the next action.

The image obtained from the mcts-turtle agent is given as an input to our CNN model. The CNN model is defined as a sequential model. The first layer used in the model is a Conv2D layer. This layer creates a convolution kernel that is convolved with the layer input to produce a tensor of outputs. This is followed by another Conv2D layer. We then use a MaxPooling2D layer. This layer reduces the size of the image. The pool size used is 2*2. The next layer added to the CNN model is the Dropout. The fraction of the input units to be dropped is set to 0.25. We then Flatten the image using Flatten(). The next layer is a Dense layer with ReLU activation function and 128 units as the dimensionality of the output. After this, we use another set of Dropout with 0.5 as the rate and Dense layer with softmax as the activation function with the output dimensionality as the number of classes. We use the training data of 60000 observations to build the model and 10000 observations to test the accuracy of the CNN model built. After training for 8 epochs, the model obtained an accuracy 98.97%.



(a) Eight (b) Nine

Figure 2: Results of the numbers drawn by the agent

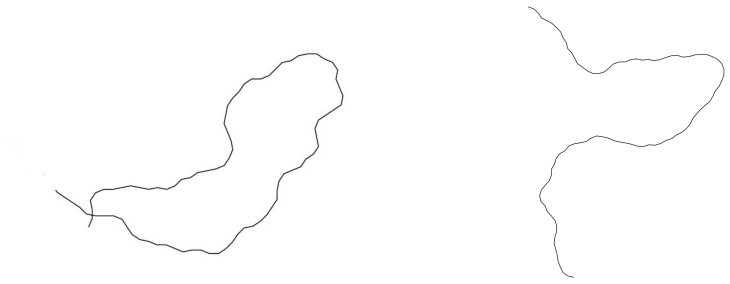
## 5. Results

The architecture built using MCTS and CNN seems to work. Few of the results are as shown in fig.2.

We built a multinomial logistic regression using MNIST data. Though the model built is shallow, the results obtained depended on the number of epochs the model was trained for. Fig.2(a) shows an image of eight which was drawn by the agent. We ran the MCTS + CNN(image classifier) setup for multiple inputs(numbers) and most of them performed decently. The starting position of the turtle pointer needs to be further worked on. Most of the numbers we tested the agent for, the starting point introduced some skewness to the resultant image.

## 6. Analysis & Inferences(Creative contribution)

After analysing the above results, it can be noticed that even though the sketching starts at a random point, it tends to alter its path to match the number the user wants to paint. The usage of MNIST dataset posed an issue here.



(a) Zero (b) Seven

Figure 3: Skewed results of the numbers drawn by the agent

The MNIST dataset has multiple images of each number, hence when CNN is built on it, there were way too many false positives which misguide the path taken. This is clearly visible in fig.3(b) where the detour while drawing a seven distorted it's appearance. Discriminative models are trained to classify natural images correctly with high accuracy, but unnatural images such as those which occur during the intermediate stages of the drawing are classified incorrectly with equal confidence. This is not very surprising because the training data doesn't have any noisy images.

Upon further analysis, we found that giving a 30° rotational degree of freedom works best for exploration especially in cases where the number of iterations are low. We have gradually increased it from 0° to 60° to come to the above conclusion. When the degree of rotation was as high as 60°, the image is very distorted and has a lot of noise.

Another enhancement which can be introduced is using probability outputs from the CNN as the reward instead of the flat 0, 1 reward being used now. This will help weigh an action classifying the image more correctly with more reward.

The number of steps taken for each iteration also plays a significant role. We varied the number of steps(depth) from 50 to 100. As you increase the depth, the image obtained reaches as close to original image as possible. The number of

iterations used were also varied from 100 to 1000. When we used 100 iterations, the performance was deteriorated. This can be seen in fig.3(a). The actions picked have not explored the possible right action. But, using 1000 iterations took more than 1 day to draw an image. Though the agent was drawing a good image, we had to discontinue the simulation due to hardware and time constraints.

Talking about the hardware constraints, we tried using google collab and crestle to run our code. But, the problem we faced is that none of these environments are compatible to use turtle. Which restricted us to using our laptop which had 8GB RAM, AMD A10 processor. So, using a GPU will obviously increase the speed of computation and training.

## 7. References


- Browne, Cameron B., Edward Powley, Daniel Whitehouse, Simon M. Lucas, Peter I. Cowling, Philipp Rohlfshagen, Stephen Tavener, Diego Perez, Spyridon Samothrakis, and Simon Colton. "A survey of monte carlo tree search methods." IEEE Transactions on Computational Intelligence and AI in games 4, no. 1 (2012): 1-43

- Akten, Memo, and Mick Grierson. "Collaborative creativity with Monte-Carlo Tree Search and Convolutional Neural Networks." arXiv preprint arXiv:1612.04876 (2016)

- Bertsimas, Dimitris, J. Daniel Griffith, Vishal Gupta, Mykel J. Kochenderfer, and Velibor V. Misic. "A comparison of Monte Carlo tree search and rolling horizon optimization for large-scale dynamic resource allocation problems." European Journal of Operational Research 263, no. 2 (2017): 664-678