FINAL PROJECT
EE456 SECTION 001

Implementing and testing the Auto Encoder CNN to solve the Maze path finding problem.

Sahin, Efe

Penn State University | December 14

OVERVIEW

The objective of the project is to use Auto Encoder neural network design to solve a maze problem where the solution is a path from bottom left corner to the top right corner of an image. The neural net design choice has managed to figure out such solutions as output for given mazes as input. This project uses pytorch library for implementation, training, and testing. The results have shown the loss vs epoch graph and example solution images of the net during training and after testing. According to the results, the neural net has converged at a certain loss and results seem silhouettes of target images. Therefore, the neural net is observed to work as planned at giving outputs that solve the input problem.

However, the structure of the input target pairs for training has been generated randomly according to a certain rule where the target images are like each other. It is important to also mention that some of the input images are very complicated to the point it might not be beneficial for learning effectively. Despite these observations of the data generated for training, the neural net after 100 epochs, is able to predict solutions to the problem correctly.

TABLE OF CONTENTS

OUTLINE

This Project aims to solve a simple generated maze problem where the goals is to find a way from bottom left to the top right of an image by following a certain path. Finding a path inside a maze can be done many ways where one such example would be utilizing genetic algorithms. For this project, a type of convolutional neural net called Auto Encoder will be used which generates images based on given input images.

The net will accept a single channel 1 x 32 x 32 image of a maze problem and generate a silhouette of a solution which it manages to figure out. For effective learning to be accomplished, training will be done with 10000 input target pairs where the input is an image as mentioned before and the target is a 1 x 32 x 32 image of one path that solves the problem. The net is going to first train on the dataset and then be tested. Some of the objectives in this project will be plotting training loss vs epoch graph, and example solutions of the neural network with given problems and solutions. Some of the limitations of this project is not presenting accuracy of the neural network because auto Encoders don't classify inputs as they generate images only.

For this problem, I initially thought of using the self-organizing-maps (SOM) Kohonen neural network to solve the maze problem. However, as it will be discussed later in detail, this project instead uses Auto Encoder CNN with reasons to be explained.

It is important to mention that this report doesn't go detail on how such data is generated as it is not in the scope of achieving the objective.

Lastly the code uses python 3.8.15 with libraries like torch, torchvision, matplotlib and numpy inside of an Anaconda environment. The following are the versions used to complete the project.

- Torch: 1.14.0.dev20221130
- Torchvision: 0.15.0.dev20221130
- Matplotlib: (latest is preferred)
- Numpy: 1.24.0rc1 (latest is preferred)

METHODOLOGY

The initial approach to the project was using Kohonen SOM neural networks. Because the initial method was intuitive that SOM could solve the maze problem, the method didn't seem feasible later.

Research said that the commonly known Travelling Salesman Problem (TSP) could be solved with SOM which I thought could be reduced to solving the maze problem. Both problems simplify the input of coordinates to undirected graphs where TSP needs only coordinates, and maze problem need coordinates for turning points in the maze and which turning points are directly connected to other turning points. However unlike in the TSP problem, the length of the solution for a maze, or in other words, the number of clusters, wasn't clear to the neural network. The number of clusters is crucial information for the SOM in order to be able to reduce TSP to maze problem because in the TSP the number of clusters is known to be the number of all cities/points, whereas in the maze problem, it is unknown. This means that SOM cannot figure out such path in maze without knowing the length of the path/number of clusters for it to be able to converge at solution. Although TSP cannot be reduced to maze problem, there might be a way to implement SOM such that it can solve maze problem. But I instead decided to utilize CNNs, specifically an Auto Encoder.

ARCHITECTURE

DATA

Since Auto Encoders need image datasets to train on, this project uses a custom generated maze data set of size 10000 x 1 x 32 x 32 with target as the same size. It means that the training dataset has 10000, 1 channel 32 by 32 images of generated mazes with 10000, 1 channeled 32 by 32 images of a solution path. The Figure 1 shows example input target pair which the net trains on.

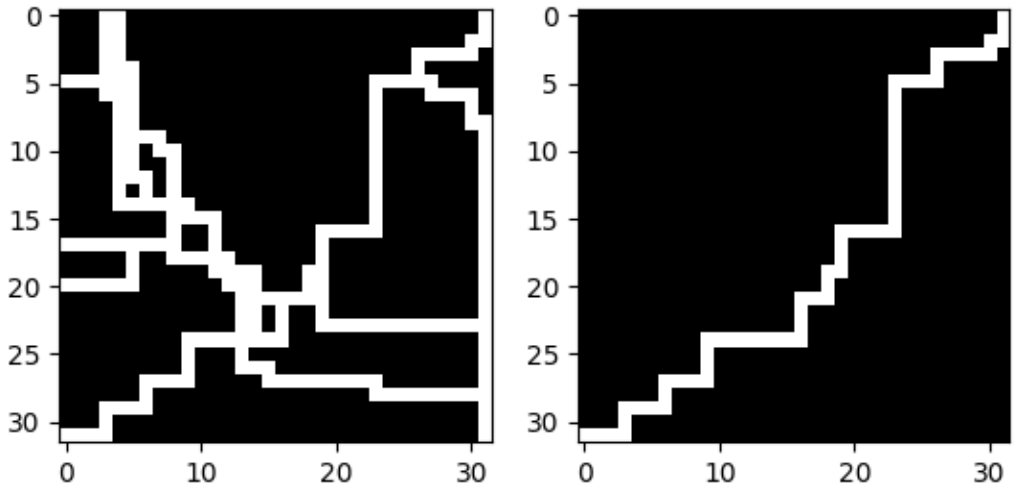


FIGURE 1: INPUT(LEFT), TARGET(RIGHT)

There are 10000 different pairs like the data in Figure 1 and they are generated by first creating a solution path(right) then adding noise to it by making the solution path branch to other paths(left). Although this randomized data has possible downsides like having simple mazes or very hard mazes, they might help the net learn better for the testing.

NEURAL NETWORK

Auto Encoders work in two parts, Encoder, Decoder. In the first part, the input image is compressed to smaller size with higher number of channels which can be said as the feature extraction phase. The Second part decodes the compressed image back to its original dimensional size. While doing encoding and decoding, it is trained to minimize the error between input and target images. As they are powerful at denoising and feature learning, they are ideal for solving maze problem.

The Auto Encoder implemented uses three convolutional layers for encoding, and three for decoding. In the middle of convolutional layers, gelu function is used followed by pooling layer. The decisions made at designing the net is a heuristic. In pytorch, it is possible to give a summary of the neural net architecture as can be seen in Figure 2. It suggests that there are 113,529 parameters that get trained. Higher the number of parameters, it is likelier the net will perform better. However, training time extends greatly. In Figure 3 can be seen a visual model of the neural net implemented.

```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1          [-1, 8, 32, 32]              80
         AvgPool2d-2          [-1, 8, 16, 16]               0
            Conv2d-3         [-1, 64, 16, 16]           4,672
         AvgPool2d-4           [-1, 64, 8, 8]               0
            Conv2d-5          [-1, 128, 8, 8]          73,856
         AvgPool2d-6          [-1, 128, 4, 4]               0
   ConvTranspose2d-7           [-1, 64, 8, 8]          32,832
   ConvTranspose2d-8          [-1, 8, 16, 16]           2,056
   ConvTranspose2d-9          [-1, 1, 32, 32]              33
================================================================
Total params: 113,529
Trainable params: 113,529
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.00
Forward/backward pass size (MB): 0.37
Params size (MB): 0.43
Estimated Total Size (MB): 0.80
----------------------------------------------------------------
```

FIGURE 2: SUMMARY BY PYTORCH

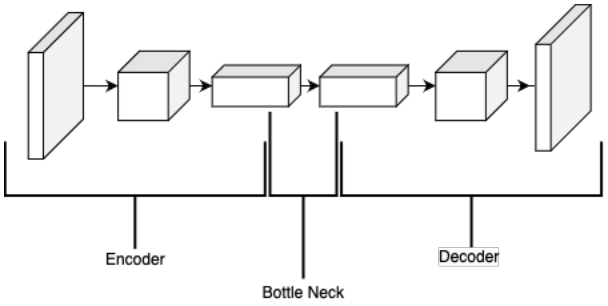


FIGURE 3: AUTO ENCODER MODEL

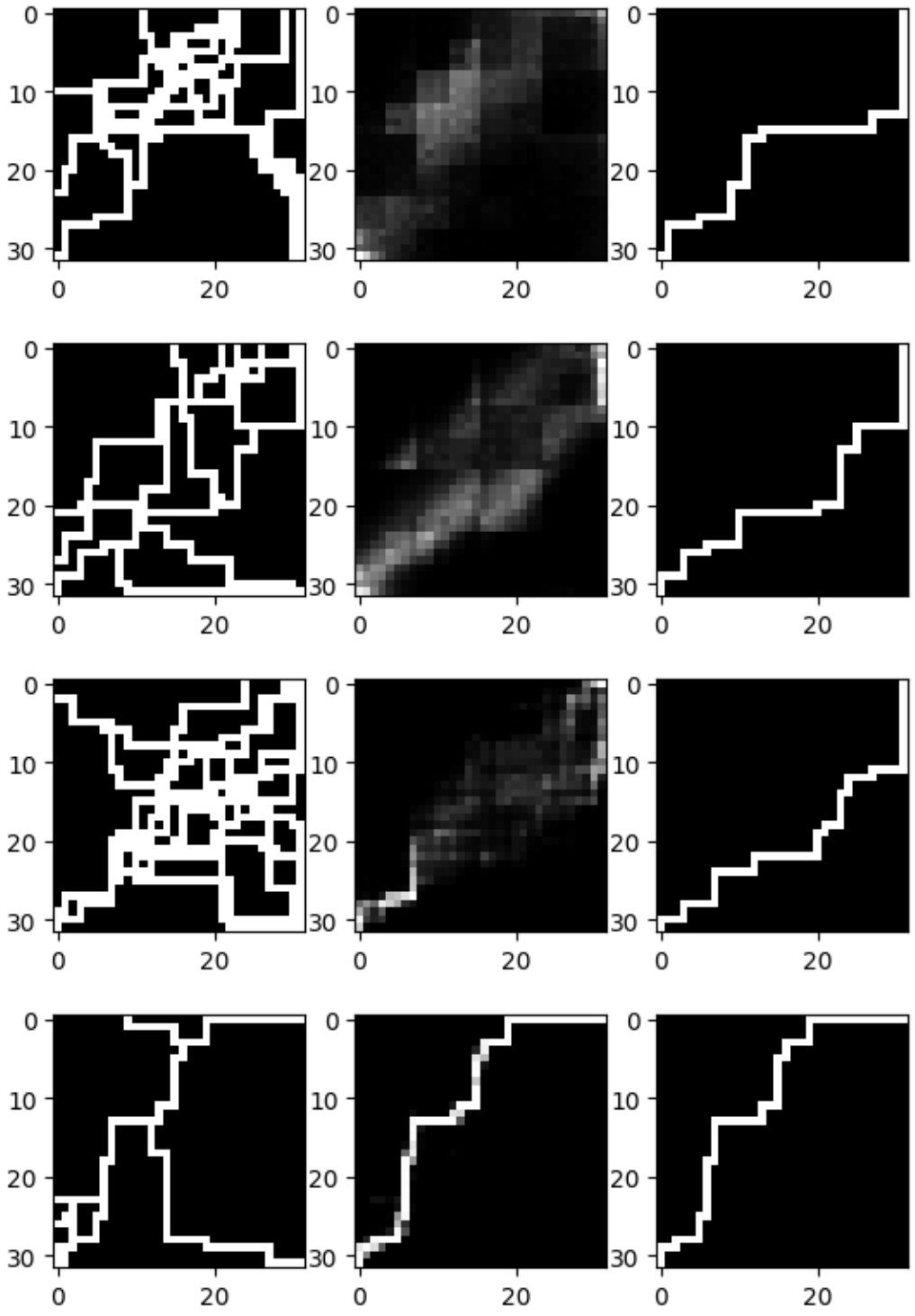
The neural net model in Figure 3 approximates the model used in this project where it decompresses the input 1x32x32 image with extracting features from it and reconstructing an image of same size based on most significant findings.

Throughout the learning process, the neural net calculates loss each epoch. Loss is very important to know as training happens on 10000 images each epoch, it uses the loss function called binary cross entropy loss (BCE) to find the difference between output and the target image and therefore optimize accordingly.

RESULTS

TRAINING

A run for 100 epochs, was sufficient at achieving satisfying outputs that were seeming correct. As mentioned before, accuracy cannot be calculated for Auto Encoder nets because the output is an image rather than a definitive class. To solve this issue, Figure 4 shows the output generated with its different input and target for given epochs in order to give a visual idea of how the neural network is learning.


0
10
20
30
0
20

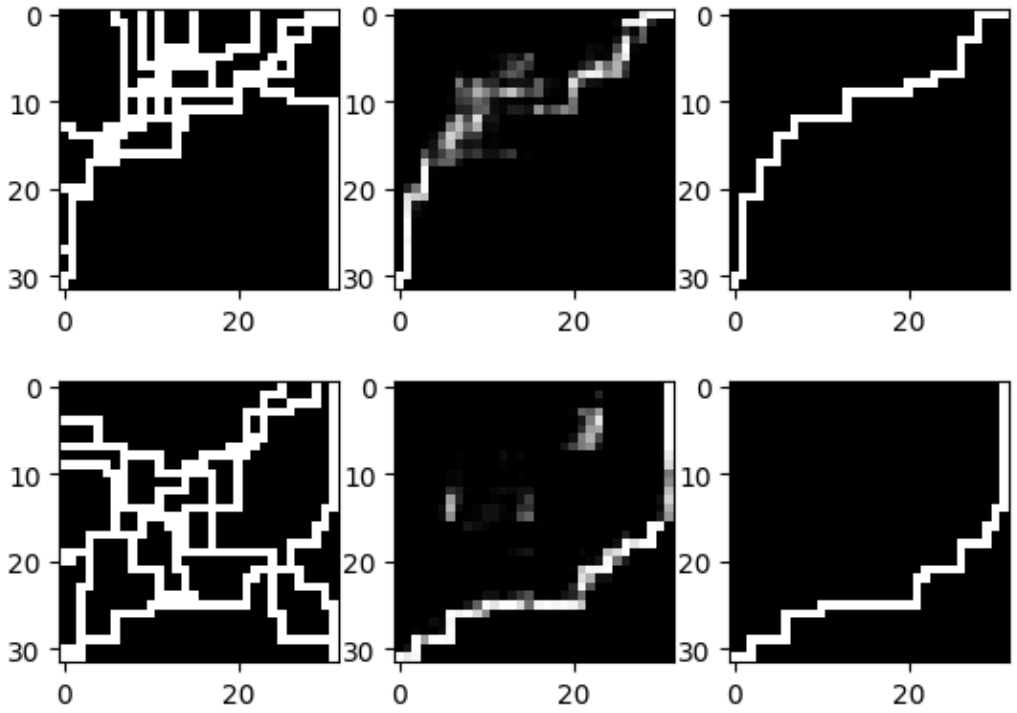


FIGURE 4: EPOCHS 1 3 23 63 93 100 DURING TRAINING (TOP TO DOWN)

It should be noted that the all the targets have one thing in common, by the definition of the maze problem in this project, the path is from bottom left to top right corner. This might be one of the reasons the net is succeeding with its learning.

As can be seen with the correctness of outputs from the neural network as epochs get higher, the loss vs epoch graph implies the visual observations from Figure 4 are infact not by chance. It can be said that 100 epochs is enough iterations for training as the curvature of the line seems to converge at the value 1.28.

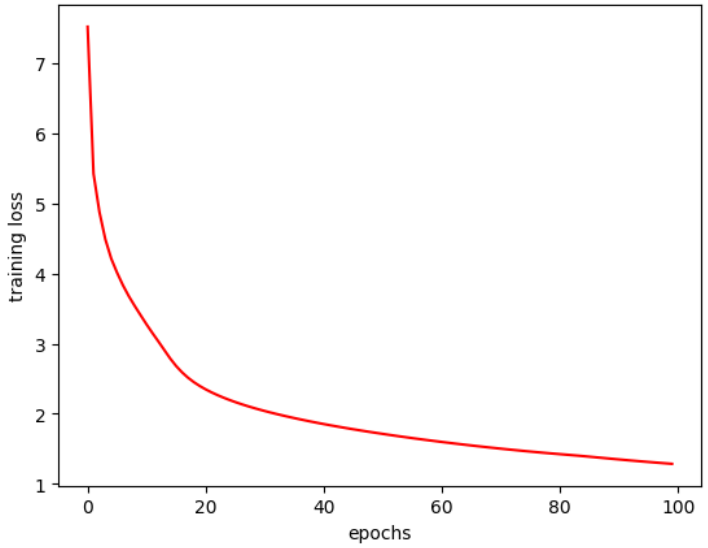


FIGURE 5: LOSS FROM TRAINING

TESTING

After training phase comes testing on brand new data which it has never learned from before. Since the accuracy cannot be calculated, the only metric is visually judging the result. Therefore, I just tested for 3 randomly generated mazes and here can be seen in figure 6.

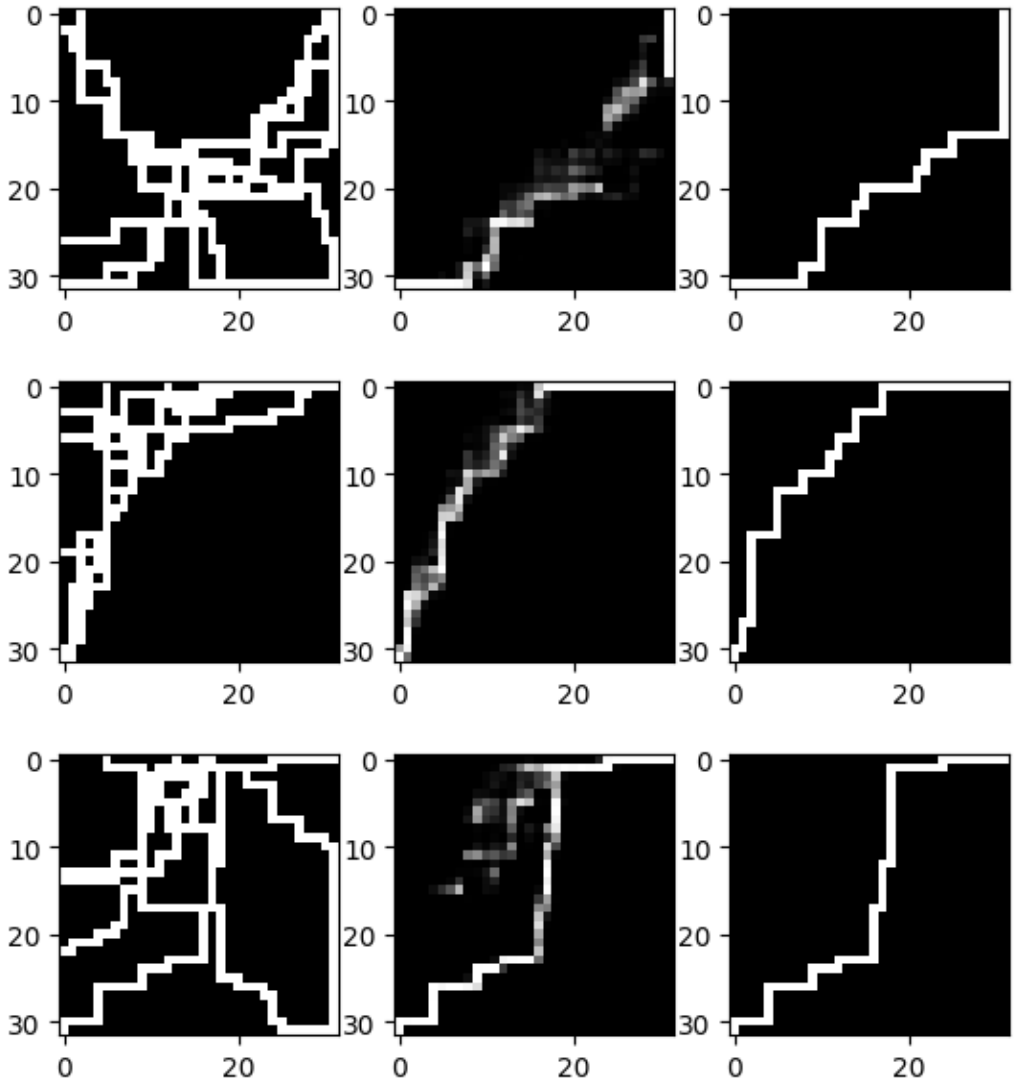


FIGURE 6: TESTING OF 3 INPUT TARGET PAIRS

As can be seen from the testing, the net can precisely choose the correct path. Although some of the outputs are not so confident such as the third one, they all manage to output a silhouette of the solution.

SUMMARY

The report has explained why Auto Encoder is a better solution compared to SOM which was the initial plan. The SOM neural nets are dimension reducing nets and couldn't be used since the very similar TSP problem couldn't be reduced to Maze problem. On the other hand, the Auto Encoder is a feature extractor neural net where it figures out the paths inside the input maze.

The Auto Encoder Neural Network is giving output images much resembling the target image. However, this project is lacking a deterministic approach to measure the overall effectiveness of the net i.e. there isn't a way to verify correctness of output images deterministically. The only mentioned measure of overall effectiveness is loss vs epoch graph and it suggests that the neural net is converging at a result.

Overall, the solution would seem feasible if the problem input could be scaled up for even bigger path finding problems. Additionally, if the data for training could be more similar to a commonly known maze structure and the target as a better representation of the intended answer, I think that Auto Encoder design would effectively find solutions.

BIBLIOGRAPHY

Dr. Vaibhav Kumar. 2021. How to implement convolutional Autoencoder in pytorch with Cuda. (August 2021). Retrieved December 16, 2022 from https://analyticsindiamag.com/how-to-implement-convolutional-autoencoder-in-pytorch-with-cuda/

Laurene V. Fausett. 1994. *Fundamentals of Neural Networks: Architectures, algorithms, and applications*, Upper Saddle River, NJ: Prentice-Hall.