GenCNN: Neural Network with 100% Accuracy for Cycle Recognization Task

Jiaming Shan

Abstract

Currently, problems with strong logic are hard to tackle with most neural networks. We raise such a kind of problem by building a cycle dataset. We propose a new network architecture called GenCNN(Generalized Convolutional Neural Network) to handle this problem. We associate deterministic algorithms with uncertain neural networks, hoping to implement algorithms using neural networks. We change the convolutional kernel to GenKernel, a more general kernel, and introduce the global inputs mechanism. We show that GenCNN can achieve 100% accuracy on the cycle dataset. GenCNN has a strong connection with graph convolution networks, cellular automata, and algorithms. This work is an enlightening attempt to combine certainty and uncertainty. We hope it can be a good inspiration for people's future work.

1 Introduction

It is long known that it is hard for neural networks to tackle logical problems. Take the field of computer vision as an example, CNN-like [4] model architectures have common characteristics. They recognize particular patterns of data by convolutional layers. But the problem is that they just assume what the pictures look like, and can't identify them by the logic behind them. Also, if we have an algorithm that can recognize a pattern, we cannot transfer this algorithm to a neural network.

This paper trys to reveal and solve these problems by building a dataset and a new model architecture called GenCNN(Generalized Convolutional Neural Network).

Furthermore, we constructively proved that there exists a parameter selection on GenCNN that can achieve 100% accuracy on the dataset, while ResNet-50 [2] created by autogluon [1] will misclassify on some images.

Here is the **contribution** of our work:

- We use a deterministic algorithm to solve the problem, which is a new way.
- We expose the incapacity of the traditional neural networks to the logical problems.
- We use global inputs to tackle the oversmooth problem of GCN.
- We use a generalized kernel, and compare it to the cellular automata, expanding the possibilities of convolutional operations.

The code is available at `https://github.com/shanjiaming/GenCNN`.

2 Problem

The task is to classify whether there exists cycle in the images. The dataset is a set of images of size 28×28, with either 0 or 1 in every pixels. There are 60000 training images and 10000 test images.

Though there exists deterministic algorithm to solve this problem, it is a difficult problem to solve for neural networks. The difficulty lies in the fact that any single pixel change may lead to a change in the result. Also, for a graph with cycle, the cycle may exist in the corner of the picture, and without labeling, the neural network cannot know that this area is the key region that determines the label of the whole picture. Finally, the cycle may take various forms, as long as it can logically become a closed cycle.

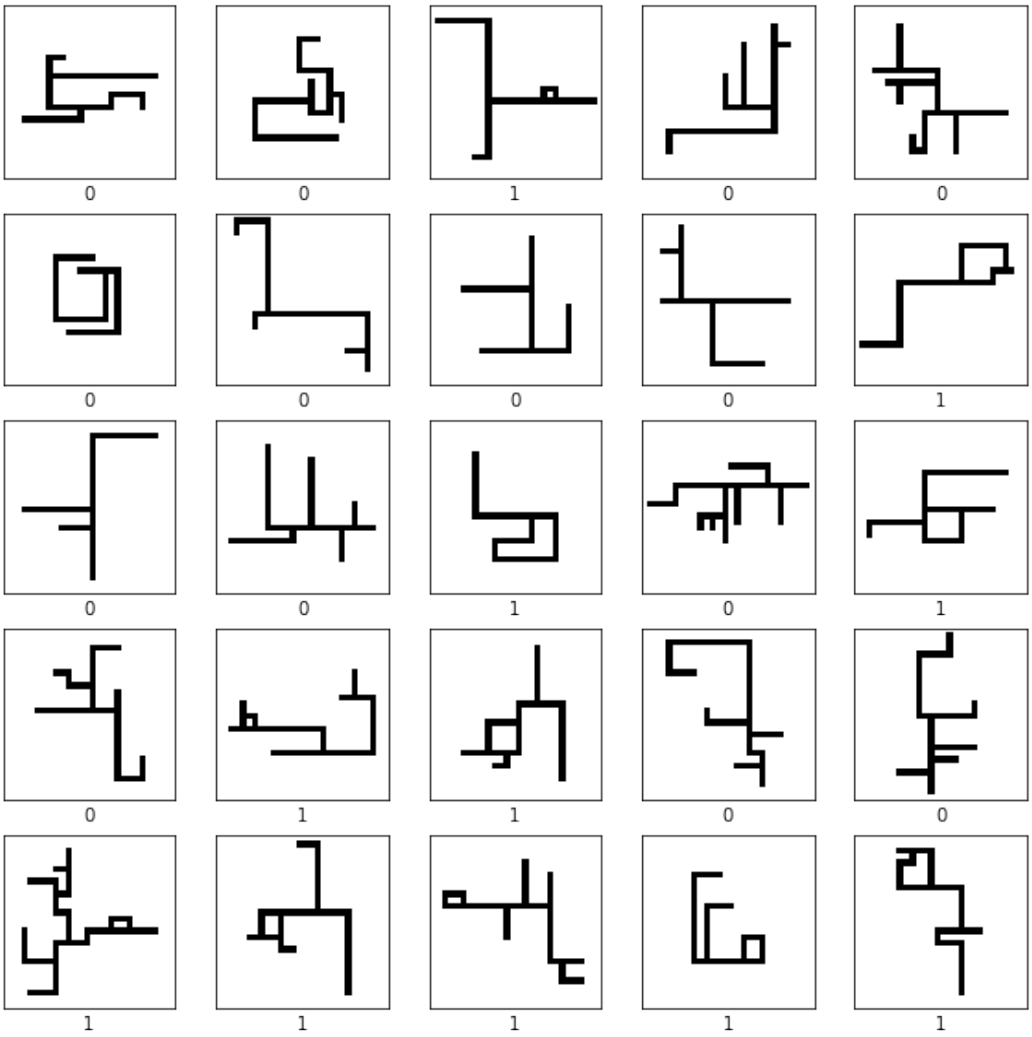

Figure 1: Cycle Dataset

3 Method

3.1 Deterministic algorithm of cycle detection problem

3.1.1 General Structure

Let us consider how we can solve the cycle-recognization problem with a deterministic algorithm (BFS). We call a pixel with value 1 a positive pixel and a pixel with value 0 a negative pixel. We only need to randomly choose a positive pixel, and know the distance of every pixel to the chosen pixel. Here, the distance means the minimum number of steps to reach the chosen positive pixel by only stepping to the neighbor **positive** pixels, and the neighbor pixels are up, down, left, and right

pixels, so at most four pixels. If we get the distance of every pixel, noticing that every cycle is an even cycle in this problem, so we only need to detect whether there are patterns a pixel with distance d has at least two neighbor pixels with distance $d-1$.

So we have three parts in this algorithm:

3.1.2 Initializer

Given the image, we will initialize the distance of every pixel to the chosen positive pixel by choosing a random positive pixel and setting its distance to 0, and setting all other pixels to $+\infty$.

3.1.3 Solver

Input the distance data output by the initializer, and do distance propagation operation repeatedly until all distance remains unchanged.

A distance propagation operation is to set the new distance of every pixel to the minimum of the old distance and the distance adding one of its neighbor positive pixels.

That is:

First, we let $d \mathrel{+}= inputs \cdot \infty$. That is using inputs as a mask. Here inputs is the original image that has $28 * 28$ 0 or 1.

Then, we let $d = padding(d, 1, 1, +\infty)$. That is to padding a $+\infty$ with width 1.

Finally, for all possible (i,j):

$$d_{new}[i,j] = \min(d[i,j], d[i+1,j]+1, d[i-1,j]+1, d[i,j+1]+1, d[i,j-1]+1)$$

Notice that we use the same raw inputs in each solver. We call this method as using global inputs.

3.1.4 Extractor

An extractor is to extract the distance data from the solver output.

We get the distance of every pixel from the output of the solver, and the extractor detects whether there are patterns a pixel with distance d has at least two neighbor pixels with distance $d-1$. If there are, output 1, showing that the image contains a cycle, else output 0.

That is:

$$\begin{aligned} v &= d[i,j] \\ l &= [d[i,j-1], d[i-1,j], d[i+1,j], d[i,j+1]] - v \\ d[i,j] &= sum(l == -1) >= 2 \\ outputs &= Any(d) \end{aligned}$$

If any $d[i,j]$ outputs 1, then there is a cycle, and Extracor return 1, else return 0.

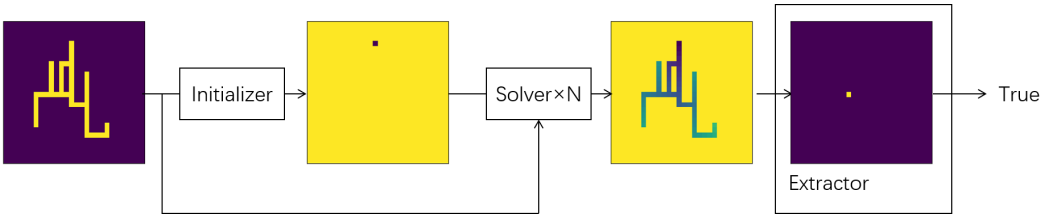

Figure 2: Data Flow of Deterministic Algorithm in Cycle Recognization

3.2 GenCNN on Cycle Dataset

3.2.1 General Structure

This model, GenCNN, is a neural network implementation of the above deterministic algorithm.

The model contains three parts: an initializer, a solver, and an extractor.

The initializer is used to initialize the initial state of a problem. The solver is used to solve the problem. The extractor is used to extract the result of the problem from the solver.

It is very similar to the deterministic algorithm, but we use pytorch's neural network and functional API to implement it.

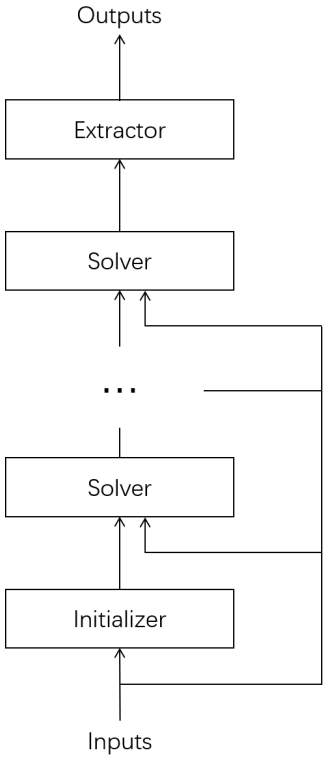

Figure 3: GenCNN Architecture

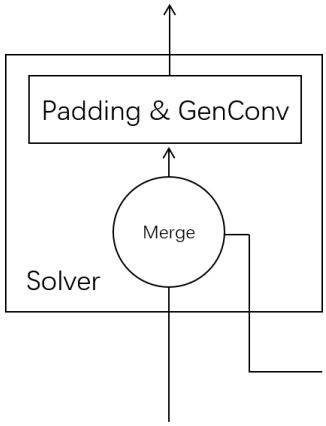

Figure 4: Solver Architecture

3.2.2 Initializer

The initializer is to randomly choose a positive pixel. That is not that easy to implement by pure neural network, so we use hard logic here.

3.2.3 Solver

As for the solver, we notice that the solver works like ConvNet, so we can call the solver a GenConv (Generalized Convolution Network).

ConvNet is $x_{new} = \sigma(\Sigma w_i x_i + b)$. x_i is a neighborhood point of x, while GenConv is $x_{new} = f(X)$.

We call f GenKernel, and this is how the name of GenCNN comes.

When handling the cycle dataset, since we want to have a finite representation of the infinite f, we use a MultiLayerPerceptron (MLP) as the GenKernel.

The inputs to the GenKernel is 5-D tensor, the outputs to the GenKernel is a 1-D tensor.

To represent the formula

$$d_{new}[i,j] = \min(d[i,j], d[i+1,j]+1, d[i-1,j]+1, d[i,j+1]+1, d[i,j-1]+1)$$

We use ReLU as the activation function of MLP.

Notice that $\min(a,b) = a - relu(a-b)$, and we can solve this problem.

The conclusion is that we can use a MLP whose width is 5-3-2-1 and choose ReLU as the activation function to build the solver part.

3.2.4 Extractor

The method we used to build the extractor part is similar to the solver part.

We still use MLP and ReLU to build the extractor part.

$$\begin{aligned} v &= d[i,j] \\ l &= [d[i,j-1], d[i-1,j], d[i+1,j], d[i,j+1]] - v \\ d[i,j] &= sum(l == -1) >= 2 \\ outputs &= Any(d) \end{aligned}$$

The only difficult part is the last two steps.

Notice that $1_{a<b} = min(relu(b-a) * 10000, 1)$, so we can compare two value using relu easily.

The third step (extract kernel step) can be rewrite as $d[i,j] = sum(-1.5 \leq l \leq -0.5) >= 2$, and we are done.

The last step (or step) $outputs = Any(d)$ can be represented as $outputs = \sum(relu(d)) > 0.001$, and we are done.

4 Analysis

4.1 In the viewpoint of Graph Convolution Network

Graph Convolution Network (GCN) is a generalization of Convolution Network (CNN).

GenCNN can also be used in a graph, and we can understand it as a variant of GCN [3] (Graph Convolutional Network).

It overcomes the depth restriction of GCN being too smooth when the depth is too deep [5], since the usage of global inputs and different GenKernel changed the fixed point.

4.2 In the viewpoint of Cellular Automata

GenKernel perfectly simulated evolutionary functions in cellular automata(CA), because in every evolution step of CA, the state of every cell is a function of the state of its neighborhood.

GenKernel f defined all the possible function.

So everything CA can done can be done by GenCNN theoretically.

4.3 More Generalized Model

4.3.1 Preparer

A more general GenCNN model has a preparer.

Notice that if in your dataset, each pixel does not contains much logical information but the visual type of perceptual information, you should first add a preparer, which is a CNN-like part to extract the logic information before the logic information goes to the GenCNN part. In the model we used in handling the cycle dataset, we do not use preparer, because every pixel has logical information to determine whether there is a cycle in the image.

Actually, when handling cycle dataset, the solvers try to solve the distance of every pixel to a certain pixel. This is exactly finding the fixed point of the distance regarding to the relaxation operation.

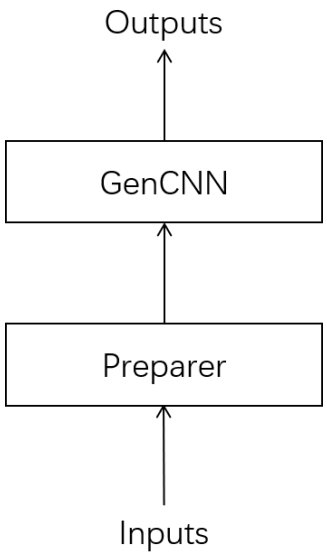

Figure 5: General GenCNN Architecture

4.3.2 MultiChannel

A more general GenCNN model has multi-channels just like traditional CNN, which is equivalent to holding a vector or tensor instead of a single value in $d[i, j]$. In this case, the GenKernel will be more complex, mapping from many tensors to a single tensor.

Using this method GenCNN can solve more problems. For example, $d[i, j]$ can record the pixels that has certain relation with the pixel i, j, like, belongs to the same connected component, though there will be more time cost.

The total time cost is $depth \times image\_size^2 \times kernel\_cost$.

4.3.3 Different Initializer, Solver and Extractor

The structure of the initializer, the solver, and the extractor can be changed.

For example, every solver can either use the same or different GenKernel.

Extractor can use MLP to extract the information like AlexNet [4].

There are still much to be done.

5 Experiments

We first build a cycle dataset.

Then we do the following experiments on this dataset:

- Build a deterministic algorithm to find the cycle.
- Build a GenCNN model to find the cycle.
- Use autogluon, an automl tool, to find the cycle.

Because the training time is too long (because of the cost of MLP), we did not train the GenCNN model.

The first and second methods can achieve 100% accuracy, while the third method cannot.

Though ResNet-50 uses the mechanism of residuals, which is equivalent to global inputs, thus also achieving good results, it still misclassified some images.

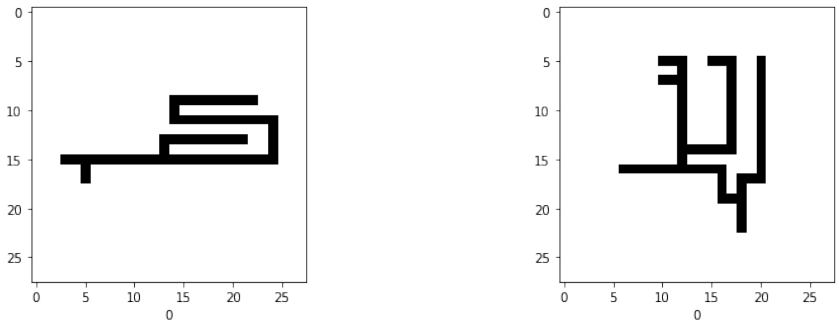

Figure 6: Two misclassified images by ResNet-50

6 Future Work

6.1 Dataset

The cycle dataset is too easy. We need more difficult deterministic datasets.

The question is whether we can generate more questions and answers with good distribution, but hard to solve by the classical algorithm. We know this kind of problems exist, that is integer decomposition, but it is not friendly to the neural network.

In short, we need more questions and data generation methods.

6.2 GenKernel

Further experiments can replace the MLP with a simpler GenKernel and increase its speed.

6.3 Loop Logic

We also noticed that current neural network have an incapacity of loop logic. That makes the GenCNN, RNN, and a lot of other models to be deep to simulate static loop unfolding, and that is not a good thing. Also, no current neural network can handle dynamic loop. This will lead to an unreasonable increase in time comparing to deterministic algorithm.

For example, what the initializer of GenCNN handling cycle dataset do now is to randomly choose a positive pixel. Let n be the number of pixels. This is easy to do in $O(n)$ in the deterministic algorithm. We can choose the first 1 we meet, and not choose other pixels. But in neural network, we need to determine whether to choose a pixel by $O(n)$, and the total time would be $O(n^2)$. That is not a good time complexity.

Our ultimate goal is to be able to implement all algorithmic logic with neural networks.

7 Conclusions

In this paper, we hope to draw attention to the general incompetence of neural networks for strong logic problems, and to arouse interest in solving it with cycle dataset. Our ultimate goal is to be able to implement all algorithmic logic with neural networks.

The new model GenCNN proposed in this paper is of value as an attempt to solve the problem, while changing the convolution and activation functions to arbitrary functions, extending the traditional form of convolution and greatly enhancing the capabilities of CNN, adding global inputs to solve the oversmooth problem of GCN.

Acknowledgements

Thanks my teacher Weinan Zhang for teaching me.

Thanks Jiaming Tang for the idea support.

Thanks my friends who give me courage to try new methods and ideas.

References

[1] Nick Erickson, Jonas Mueller, Alexander Shirkov, Hang Zhang, Pedro Larroy, Mu Li, and Alexander Smola. Autogluon-tabular: Robust and accurate automl for structured data. *arXiv preprint arXiv:2003.06505*, 2020.

[2] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[3] Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*, 2016.

[4] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, 25, 2012.

[5] Si Zhang, Hanghang Tong, Jiejun Xu, and Ross Maciejewski. Graph convolutional networks: a comprehensive review. *Computational Social Networks*, 6(1):1–23, 2019.