# CONSTRUCTION OF A UNIVERSAL DISCONTINUITY DETECTOR USING CONVOLUTION NEURAL NETWORKS

SHUYI WANG*, ZIXU ZHOU†, LO-BIN CHANG*, AND DONGBIN XIU†

**Abstract.** We present a universal discontinuity detector constructed by deep neural networks. Using convolution neural network (CNN) structure, we design a comprehensive set of synthetic training data. The data consist of randomly generated piecewise smooth functions evaluated at equidistance grids, with labels denoting troubled cells where discontinuities are present. Upon successful training of the network, the CNN based detection network is capable of accurately identifying discontinuities in newly given function data by correctly labeling the troubled cells. Even though all of our training data have fixed size, the constructed detector can be applied to function data of arbitrary size, so long as they are on equidistance grids. To increase the detection efficiency in two- and three-dimensional cases, we propose a two-level detection procedure, where the detector is applied to a coarsened grid first and then to the fine grids only at the trouble cells identified at the coarse level. Through an extensive set of numerical tests, we demonstrate that the developed detectors possess strong generalization capabilities, in the sense that they are able to accurately detect discontinuity with structures much more complex than those in the training data.

**Key words.** Deep neural network, convolution neural network, discontinuity detection, troubled cell.

## 1. Introduction.

Discontinuity detection is a common numerical task in many applications including signal processing, shock-wave analyses for nonlinear PDEs, phase transitions in physical systems, change-point analyses in geology or biology to name a few. The task is to detect the discontinuity positions of a function given only the function values on a grid in the function domain. In recent decades, algorithms have been developed for this task, including wavelet-based algorithms [11, 3], filter-based algorithms [13, 6], high-dimensional discontinuity detection [16, 6], discontinuity in derivatives [2] and polynomial annihilation detection [1, 2].

Neural networks have become essential methodologies for many research fields related to artificial intelligence. In particular, convolutional neural networks (CNN) have become dominant in ImageNet competitions for computer vision as well as some other AI competitions since 2012, and therefore, many vision tasks have been investigated based on CNN models. Focusing on edge detection problems, various CNNs were recently trained to detect edges directly from images (e.g., [4, 12, 14, 10, 15]). Motivated by successful applications in edge detection, we develop discontinuity detection methods based on properly designed CNN models. In fact, the edge detection problem in computer vision can be viewed as a two-dimensional discontinuity detection problem where the differences are the inputs of detection algorithms and the training procedure. The observed data for edge detection are intensity values of image pixels, but the observed data for discontinuity detection are the function values at discrete locations (e.g., grid points). The training data for edge detection are often real images with labeled edges, while the training data for the discontinuity detection here are synthetic function values generated to provide sufficient properties of discontinuity to be learned by the training procedure. In addition, we propose a two-level detection procedure to reduce the computational cost for two- or higher-dimensional discontiuity detection problems. The two-level detection procedure is a coarse-to-fine

*Department of Statistics, The Ohio State University, Columbus, OH 43210. lobinchang@stat.osu.edu.

†Department of Mathematics, The Ohio State University, Columbus, OH 43210, USA. xiu.16@osu.edu. Funding: This work was partially supported by AFOSR FA9550-18-1-0102.

procedure with two detectors. The first detector is for coarse or preliminary detection over coarsened grids for fast detection of coarse trouble celles. The second detector is refined and detailed detection of troubled cells on the fine grids only within the detected coarse troubled cells.

The paper is organized as follows. The problem setup and preliminaries are discussed in the next section, where a brief review of CNN models and polynomial annihilation methods is included. This is followed by the constructions of our discontinuity detection procedures, CNN models and synthetic training data in section 3. Numerical experiments in section 4 are provided to demonstrate the utility of our approaches.

**2. Problem Setup and Preliminaries.** Consider a function $f : D \to \mathbb{R}$, where $D \subset \mathbb{R}^d$, $d \geq 1$, is a compact set. Obviously, $f$ is continuous at $x_0 \in D$ iff $\lim_{x \to x_0} f(x) = f(x_0)$. In this paper, we consider $f$, to be piecewise continuous, in the sense that there is a finite partition of $D$ on each coordinate such that $f$ is a continuous function on each polytope. Note that this definition includes the trivial case when $f$ is continuous on $D$.

Without loss of generality, we assume $D$ is a $d$-dimensional rectangle given by $I_1 \times I_2 \times \cdots \times I_d$ where $I_i = [a_i, b_i]$, $b_i = a_i + n_i \cdot \delta$, $n_i$ is a positive integer and $\delta > 0$. Consider a uniform grid over $D$ with $\prod_{i=1}^{d}(n_i + 1)$ grid points $S = \{(a_1 + i_1\delta, a_2 + i_2\delta, \cdots, a_d + i_d\delta) : i_k \in \{0, 1, 2, \cdots, n_k\}, \forall k = 1, \cdots, d\}$. The problem of discontinuity detection is to detect grid cells, called "trouble cells," that contain discontinuity points, based on the function values at the grid points.

**2.1. Polynomial annihilation detection.** Here we briefly review polynomial annihilation detection method ([1]), as it is an established method with solid mathematical foundation and will be used to compare with our proposed CNN detection method in this paper. The goal of polynomial annihilation is to construct a function $L_m f(x)$, $m \in \mathbb{N}$, such that, for $x$ away from discontinuity points of $f(\cdot)$, $L_m f(x) \approx 0$. Therefore, the detection of discontinuities is based on $|L_m f(x)| > t$ where $t$ is a threshold.

Specifically, suppose that $f$ is known only on the discrete set $S$. Let $\Pi_m$ be the space of all polynomials of degree $\leq m$ in $d$ variables. The value of $L_m f$ at $x \in D$ is determined by the function values of $f$ on a local set $S_x \subset S$ of $m_d = \begin{pmatrix} m+d \\ d \end{pmatrix}$ points around $x$ as follows. For simplicity, let $S_x := \{x_1, \cdots, x_{m_d}\}$ be the set of the $m_d$ nearest points to $x$. For polynomial annihilation up to degree $m-1$, one solves a linear system for the coefficients $\{c_j(x) : j = 1, \cdots, m_d\}$,

$$\sum\nolimits_{x_j \in S_x} c_j(x) p_i(x_j) = \sum_{|\alpha|_1 = m} p_i^{(\alpha)}(x), \tag{2.1}$$

where $\alpha = (\alpha_1, \cdots \alpha_d)$, $\alpha_1, \cdots, \alpha_d \in \mathbb{Z} \cup \{0\}$, $\{p_1, \cdots, p_{m_d}\}$ is a basis of $\Pi_m$, and

$$p_i^{(\alpha)}(x) = \frac{\partial^{\alpha_1 + \cdots + \alpha_d} p_i(x)}{\partial {x_{(1)}}^{\alpha_1} \cdots \partial {x_{(d)}}^{\alpha_d}}, \qquad x = (x_{(1)}, \cdots, x_{(d)}).$$

Since the solution of (2.1) exists and is unique, one can define

$$L_m f(x) = \frac{1}{q_{m,d}(x)} \sum\nolimits_{x_j \in S_x} c_j(x) f(x_j), \tag{2.2}$$

where $q_{m,d}(x)$ is a normalization factor. For the numerical experiments in Section 4 of this paper, we use the formulas of $\Pi_m$ and $q_{m,d}(x)$ on pp. 277 of [1].

**2.2. Convolution Neural Network.** The Convolutional Neural Network(CNN) is a special type of neural networks. It was first introduced for solving the problem of document recognition [9], and becomes well-known thanks to the success of its applications on image classification [8]. A CNN contains an input layer, an output layer, and multiple hidden layers, where the hidden layers may consist of convolutional layers, pooling layers, fully connected layers and so on (see chapter 9 in [5] for more details).

**Convolutional Layers.** The convolutional layers are defined based on discrete convolutions. The discrete convolution of the input $u$ for a convolutional layer and a discrete kernel $w$, denoted by $u * w$, is given by

$$u * w(x) = \sum_{\tilde{x} \in I_x} u(\tilde{x}) w(x - \tilde{x}),$$

where $I_x$ is the neighborhood of $x$ determined by the kernel $w$. Therefore, the output of a convolutional layer of $k$ kernels is given by

$$\sigma(u * w_j + r_j), \text{ for } j = 1, 2, \cdots, k,$$

where $w_j$'s are the discrete kernels, $r_j$'s are the biases, and $\sigma(\cdot)$ is the activation function. The values of the discrete kernels and biases are the unknown parameters to be learned. The activation function used for our experiments is the rectified linear units (ReLU).

**Pooling Layers.** Pooling is a common down-sampling operation in CNNs, and it reduces the dimension by taking a summary statistic for each subregion of a partition of the input. In our 2-D experiments, we use the most common pooling method, called *max-pooling*. The output of a max-pooling layer is obtained by first partitioning the domain of its input $u$ into (non-overlapping) hyper-cubic subregions and then computing $\max_{x \in I} u(x)$ for each subregion $I$.

**Fully Connected Layers.** The fully connected layers are often constructed as the last layers in CNN models, based on which we make detection or classification decisions. If we represent or rewrite the input of a fully connected layer using a vector $u$, then the output can be obtained by a matrix multiplication with a bias offset, $Au + r$, where the entries of $A$ and $r$ are also unknown parameters to be learned.

**3. Construction of Discontinuity Detector.** Suppose we only observe the values of a function $f$ on the set of grid points $S$ over a uniform grid in $\mathbb{R}^d$. The discontinuity detectors constructed in this section are to detect the trouble cells, in which discontinuities exist.

**3.1. One-dimensional detector.** Let $D = [a, b]$ and $S = \{x_i = a + (i-1)\delta : i = 1, \ldots, N+1\}$ be a set of uniform grids, where $\delta = (b-a)/N$, $N \geq 1$. Therefore, the uniform grid has $N$ cells, denoted by intervals $[x_i, x_{i+1})$, $i = 1, \cdots, N$. Let

$$y = (y_1, \cdots, y_N) \in \{0, 1\}^N$$

be a binary vector indicating the ground truth trouble cells, where for each $i = 1, \ldots, N$, $y_i = 1$ if $i$-th cell is a trouble cell (i.e., there is at least one discontinuity point in the interval $[x_i, x_{i+1})$) and $y_i = 0$ otherwise. Let $v_f = \{f(x) : x \in S\}$ be the set of observed function values on $S$. Here we construct a detector that first

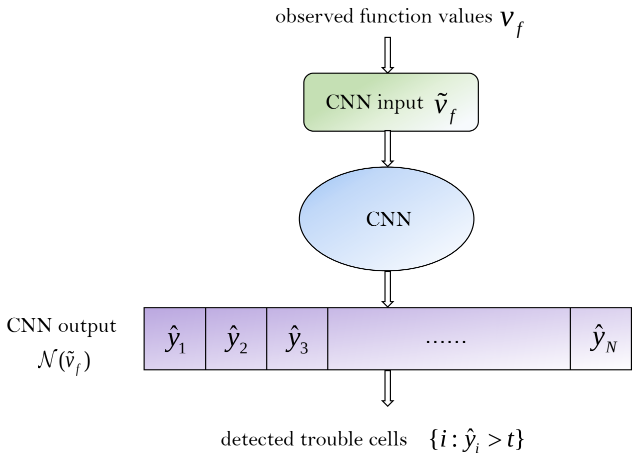

observed function values $v_f$
CNN input $\tilde{v}_f$
CNN
CNN output
$\mathcal{N}(\tilde{v}_f)$
$\hat{y}_1$
$\hat{y}_2$
$\hat{y}_3$
......
$\hat{y}_N$
detected trouble cells $\{i : \hat{y}_i > t\}$

Fig. 3.1: Diagram for the one-dimensional detection.

standardizes the observed function values and feeds them into a CNN to obtain an output vector of $N$ real values,

$$\hat{y} = (\hat{y}_1, \cdots, \hat{y}_N) = \mathcal{N}(\tilde{v}_f),$$

as an estimate of the ground truth $y$, where

$$\tilde{v}_f = \left\{ \frac{f(x) - \mu_f}{\sigma_f} : x \in S \right\},$$

denotes the set of standardized function values, with $\mu_f$ and $\sigma_f$ denote the mean and standard deviation of $v_f$, respectively. Then, for a chosen threshold $t > 0$, the detector labels each of $i$-th cell a trouble cell if $\hat{y}_i > t$. See Figure 3.1 for an illustration of discontinuity detection algorithm.

**CNN architecture.** For our experiments in Section 4, we consider a one-dimensional detector of size $N = 200$, where the CNN model contains five convolutional layers and one fully connected layer, summarized in the following table:

| Layer | input size | kernel size | num of kernel | stride | output size |
|---|---|---|---|---|---|
| conv1 | 202 | 2 | 24 | 1 | 201×24 |
| conv2 | 201×24 | 2×24 | 24 | 1 | 200×24 |
| conv3 | 200×24 | 2×24 | 24 | 1 | 199×24 |
| conv4 | 199×24 | 2×24 | 24 | 1 | 198×24 |
| conv5 | 198×24 | 2×24 | 24 | 2 | 99×24 |
| fully connected | 99×24 | | | | 201 |

**Training the CNN with synthetic training data.** We train the parameters of the CNN detector model using a synthetic dataset of $n = 1,000,000$ piecewise smooth functions on domain $D$. Each piecewise smooth function is generated as follows: (1) randomly select an integer $N_d$ from $\{0, \dots, M\}$ using discrete uniform distribution as the number of discontinuities. (In all most tests we set $M = 3$); (2) using uniform distribution on $D$, generate $N_d$ random numbers as the loations of the discontinuities. This partitions the domain $D$ into $(N_d + 1)$ subdomains; and (3) inside each subdomain, we create a random Fourier series

$$\tilde{a}_0 + \sum_{n=1}^{N_F} (\tilde{a}_n \cos nx + \tilde{b}_n \sin nx),$$

where $\tilde{a}_n$'s and $\tilde{b}_n$'s are i.i.d. Gaussian random variables $N(0, 1)$, $N_F$ is set to be 15 in all our trainings.

The network is trained with Keras API (https://keras.io/), by using mean squared loss function

$$\frac{1}{n} \sum_{i=1}^{n} \|y_i - \mathcal{N}(\tilde{v}_{f_i})\|_{l_2}^2$$

. We use 90% of the synthetic data for training and the remaining 10% for validation. The training is based on the Adam optimization [7] with the mini-batch size of 5,000 and for 1,000 epochs.

**3.2. Two-dimensional detector.** For two-dimensional detector, we consider a square domain $D = [a, b]^2$ with unform grid points $S = \{x_{ij} : 1 \le i, j \le N + 1\}$, where $x_{ij} = (a + (i-1)\delta,\ a + (j-1)\delta)$ for $i, j = 1, \cdots, N+1$ and $\delta = (b-a)/N$. This creates $N^2$ cells $C_{ij}$, which corresponds to the grid point $x_{ij}$, for $i, j = 1, \cdots, N$. Let

$$y = (y_{ij})_{i,j=1}^{N} \in \{0, 1\}^{N \times N} \tag{3.1}$$

be a binary matrix indicating the ground truth, where $y_{ij} = 1$ if $C_{ij}$ is a trouble cell and $y_{ij} = 0$ otherwise. Let $v_f = \{f(x) : x \in S\}$ be the set of observed function values on $S$. Similar to the one dimensional detector, we construct a detector that takes the observed function values $v_f$ and outputs a binary matrix to predict $y$.

**3.2.1. Synthetic data generation.** To generate synthetic training data, we create randomly generated piecewise smooth functions. More specifically, we first split the function domain $D$ into two subregions by a random curve and then generate two smooth functions for the subregions:

$$f_i(x, y) = \sum_{m+n \le N_p} a_{m,n}^{(i)} P_m(x) P_n(y), \qquad i = 1, 2, \tag{3.2}$$

where $P_n$'s are the standard Legendre polynomials, $a_{m,n}$ are coefficients randomly generated from Gaussian distribution $N(0, 10)$, $N_P$ is fixed at $N_p = 4$. For the random curve, which serves as the interface between the two subregions and is also the location of the discontinuity curve, we employ two cases:

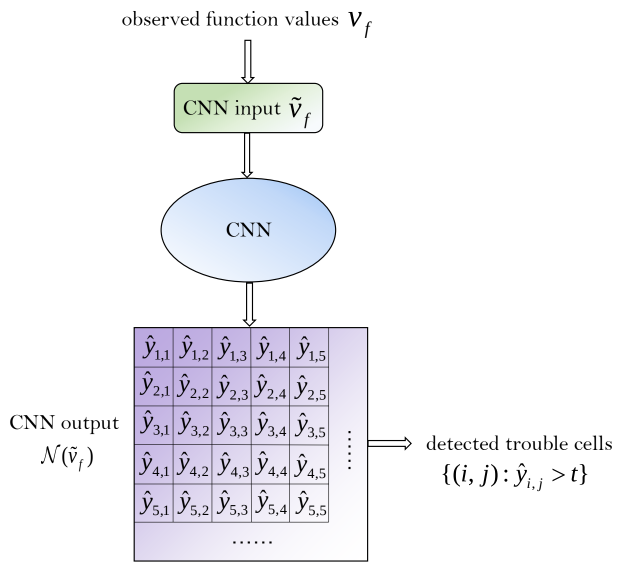
- Line cut. Defined by random straght line

$$\cos(\theta)(x - x_0) + \sin(\theta)(y - y_0) = 0,$$

  where $\theta \sim U(0, 2\pi)$ and $(x_0, y_0) \sim U(D)$;
- Circular cut. Defined as

$$(x - x_0)^2 + (y - y_0)^2 = r,$$

  where $r \sim U(0, 3)$ and $(x_0, y_0) \sim U(D)$.



observed function values $v_f$
CNN input $\tilde{v}_f$
CNN
CNN output $\mathcal{N}(\tilde{v}_f)$
$\hat{y}_{1,1}$ $\hat{y}_{1,2}$ $\hat{y}_{1,3}$ $\hat{y}_{1,4}$ $\hat{y}_{1,5}$
$\hat{y}_{2,1}$ $\hat{y}_{2,2}$ $\hat{y}_{2,3}$ $\hat{y}_{2,4}$ $\hat{y}_{2,5}$
$\hat{y}_{3,1}$ $\hat{y}_{3,2}$ $\hat{y}_{3,3}$ $\hat{y}_{3,4}$ $\hat{y}_{3,5}$
$\hat{y}_{4,1}$ $\hat{y}_{4,2}$ $\hat{y}_{4,3}$ $\hat{y}_{4,4}$ $\hat{y}_{4,5}$
$\hat{y}_{5,1}$ $\hat{y}_{5,2}$ $\hat{y}_{5,3}$ $\hat{y}_{5,4}$ $\hat{y}_{5,5}$
detected trouble cells
$\{(i,j): \hat{y}_{i,j} > t\}$

Fig. 3.2: Diagram for the one-step method for two-dimensional discontinuity detection.

Note that our synthetic training data cover very limited cases, which include only one discontinuity curve taking either straightline shape or circular shape. However, the resulting CNN detector model exhibits remarkable generalization capabilities and is able to detect discontintuities with much more complex nature. This will be shown in our examples in Section 4.

**3.2.2. One-level detection method.** Here we construct a detector that first standardizes the observed function values and produces an output $N \times N$ matrix

$$\hat{y} = \mathcal{N}(\tilde{v}_f) = (\hat{y}_{ij})_{i,j=1}^{N},$$

as an estimate of the ground truth $y$ in (3.1), where

$$\tilde{v}_f = \left\{ \frac{f(x) - \mu_f}{\sigma_f} : x \in S \right\}$$

is the set of standardized function values, with $\mu_f$ and $\sigma_f$ denote the mean and standard deviation of all values in $v_f$, respectively. With a prescribed threshold value $t > 0$, the cell $C_{ij}$ is labelled as a trouble cell if $\hat{y}_{ij} > t$. The diagram of the one-level detection algorithm is illustrated in Fig. 3.2.

For the experiments in Section 4, we consider a one-step detector of size $100 \times 100$ (i.e., $N = 100$), where the CNN model contains four convolutional layers, one max-pooling layer and one fully connected layer, as summarized in the following table:

| Layer | input size | kernel/pooling size | num of kernel | stride | output size |
|---|---|---|---|---|---|
| conv1 | 101×101 | 4×4 | 32 | 1 | 98×98×32 |
| maxpooling1 | 98×98×32 | 2×2 | | 2 | 49×49×32 |
| conv2 | 49×49×32 | 2×2×32 | 32 | 1 | 48×48×32 |
| conv3 | 48×48×32 | 2×2×32 | 32 | 1 | 47×47×32 |
| conv4 | 47×47×32 | 2×2×32 | 32 | 1 | 46×46×32 |
| fully connected | 46×46×32 | | | | 100 × 100 |

**3.2.3. Two-Step detection Method.** Due to the substantial increase in the number of parameters for the two-dimensional CNN detector, the computational cost becomes a major concern, especially for the extension to high-dimensional detection problems. Therefore, we propose a two-level detection approach that reduces the computational cost while retaining accuracy. The method will be more beneficial for higher dimensional problems. It consists of the following two steps.

**Step 1.** Define a coarse uniform grid, where each cell is the union of several neighbouring cells of the original grid. The original cells are now subcells of the coarse cells. For the $101 \times 101$ original grid, we can choose an $11 \times 11$ coarse grid such that each coarse cell contains $10 \times 10 = 100$ subcells. We can then construct and train a one-step CNN detector as in Section 3.2.2 for detecting coarse cells that contain discontinuities. The detection threshold is chosen small enough so that ideally all coarse cells with discontinuity can be detected. Note that due to the small threshold, false detection is allowed and but would be eliminated in Step 2.

The CNN model in Step 1 contains four convolutional layers, one max-pooling layer and one fully connected layer, as shown in the table:

| Layer | input size | kernel/pooling size | num of kernel | stride | output size |
|---|---|---|---|---|---|
| conv1 | 101×101 | 4×4 | 32 | 1 | 98×98×32 |
| maxpooling1 | 98×98×32 | 2×2 | | 2 | 49×49×32 |
| conv2 | 49×49×32 | 2×2×32 | 32 | 1 | 48×48×32 |
| conv3 | 48×48×32 | 2×2×32 | 32 | 1 | 47×47×32 |
| conv4 | 47×47×32 | 2×2×32 | 32 | 1 | 46×46×32 |
| fully connected | 46×46×32 | | | | 10 × 10 |

**Step 2.** Next, we refine our detection by detecting fine trouble cells (subcells that contain discontinuities) within each detected coarse cell. The detection of fine trouble cells is operated on a very small grid – encompassing only the coarse celles. This can be efficiently done with the one-level CNN detector developed here, or the polynomial annihilation detector whenever the subcells are structured. See Figure 3.3 and Figure 3.4 for the diagrams of the two-step methods. The construction and training of the CNN detector for fine trouble cells are similar to the one-step CNN detector in Section 3.2.2 except each training sample is a matrix of the synthetic function values on a fine sub-grid over a detected coarse cell.

The CNN model used in Step 2 contains three convolutional layers and one fully connected layer, as tabulated below.

| Layer | input size | kernel size | num of kernel | stride | output size |
|---|---|---|---|---|---|
| conv1 | 100×100 | 2×2 | 32 | 1 | 10×10×32 |
| conv2 | 10×10×32 | 2×2×32 | 32 | 1 | 9×9×32 |
| conv3 | 9×9×32 | 2×2×32 | 32 | 1 | 8×8×32 |
| fully connected | 8×8×32 | | | | 10 × 10 |

**3.3. Generalization to three-dimensional detection.** For higher dimensional detection, the computation cost increases exponentially with the dimension. Therefore, the proposed two-level coarse-to-fine approaches are strongly preferred.

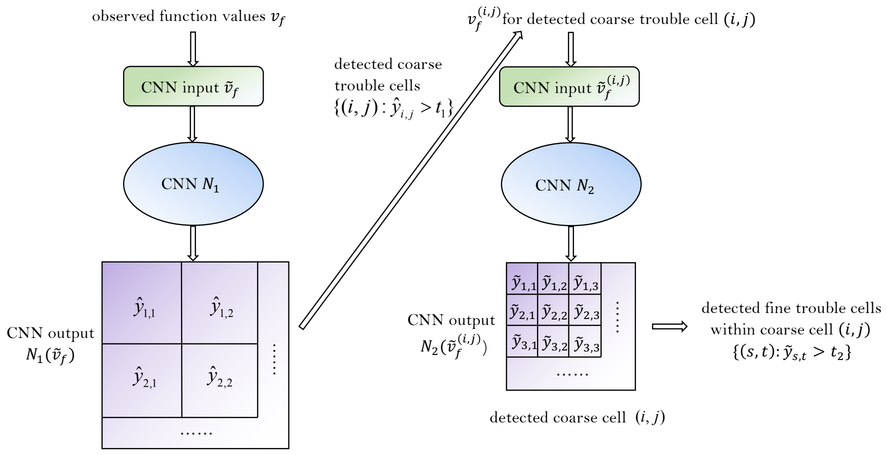

observed function values $v_f$
CNN input $\tilde{v}_f$
CNN $N_1$
CNN output $N_1(\tilde{v}_f)$
$\hat{y}_{1,1}$ $\hat{y}_{1,2}$ $\hat{y}_{2,1}$ $\hat{y}_{2,2}$
detected coarse trouble cells $\{(i,j): \hat{y}_{i,j} > t_1\}$
$v_f^{(i,j)}$ for detected coarse trouble cell $(i,j)$
CNN input $\tilde{v}_f^{(i,j)}$
CNN $N_2$
CNN output $N_2(\tilde{v}_f^{(i,j)})$
$\tilde{y}_{1,1}$ $\tilde{y}_{1,2}$ $\tilde{y}_{1,3}$ $\tilde{y}_{2,1}$ $\tilde{y}_{2,2}$ $\tilde{y}_{2,3}$ $\tilde{y}_{3,1}$ $\tilde{y}_{3,2}$ $\tilde{y}_{3,3}$
detected fine trouble cells within coarse cell $(i,j)$ $\{(s,t): \tilde{y}_{s,t} > t_2\}$
detected coarse cell $(i,j)$

Fig. 3.3: Diagram for the two-step method for two-dimensional detection where step 2 is based on a CNN detector.

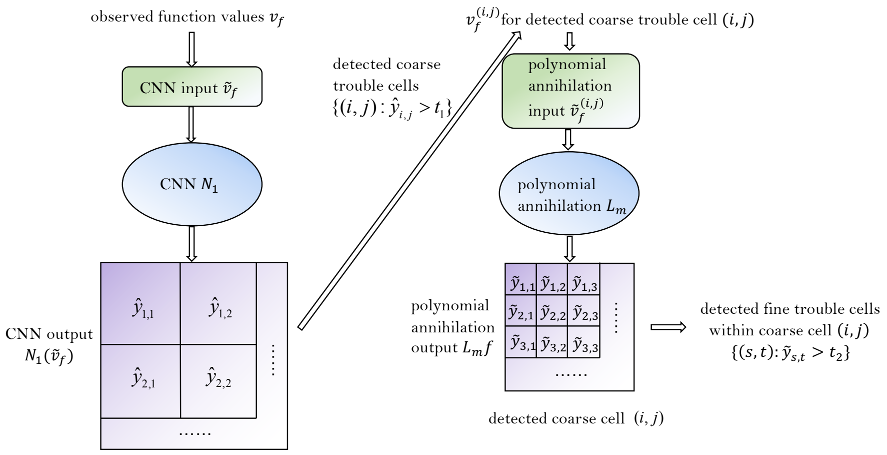

observed function values $v_f$
CNN input $\tilde{v}_f$
CNN $N_1$
CNN output $N_1(\tilde{v}_f)$
$\hat{y}_{1,1}$ $\hat{y}_{1,2}$ $\hat{y}_{2,1}$ $\hat{y}_{2,2}$
detected coarse trouble cells $\{(i,j): \hat{y}_{i,j} > t_1\}$
$v_f^{(i,j)}$ for detected coarse trouble cell $(i,j)$
polynomial annihilation input $\tilde{v}_f^{(i,j)}$
polynomial annihilation $L_m$
polynomial annihilation output $L_m f$
$\tilde{y}_{1,1}$ $\tilde{y}_{1,2}$ $\tilde{y}_{1,3}$ $\tilde{y}_{2,1}$ $\tilde{y}_{2,2}$ $\tilde{y}_{2,3}$ $\tilde{y}_{3,1}$ $\tilde{y}_{3,2}$ $\tilde{y}_{3,3}$
detected fine trouble cells within coarse cell $(i,j)$ $\{(s,t): \tilde{y}_{s,t} > t_2\}$
detected coarse cell $(i,j)$

Fig. 3.4: Diagram for the two-step method for two-dimensional detection where step 2 is based on the polynomial annihilation detector.

As an example, consider a two-step method for three-dimensional detection on a uniform grid of size $101 \times 101 \times 101$, where the step 1 is based on a CNN detector for detection of coarse trouble cells on a coarse grid of size $11 \times 11 \times 11$ and the step 2 is to detect fine trouble cells with the polynomial annihilation method. In each sample, the total number of the coarse cells is $10 \times 10 \times 10 = 1,000$ and the total number of the fine cells is $100 \times 100 \times 100 = 1,000,000$.

For the experiment in Section 4, the CNN model contains two convolutional layers, one max-pooling layer and one fully connected layer, as follows.

| Layer | input size | kernel/pooling size | num of kernel | stride | output size |
|---|---|---|---|---|---|
| conv1 | 101×101×101 | 5×5×5 | 12 | 5 | 20×20×20×12 |
| maxpooling1 | 20×20×20×12 | 2×2 | | 2 | 10×10×10×12 |
| conv2 | 10×10×10×12 | 2×2×2×12 | 6 | 2 | 5×5×5×6 |
| dense | 5×5×5×6 | | | | 10 × 10 × 10 |

To train the model, we generate a data set of three-dimensional synthetic functions, similar to the two-dimensional training data. We first split the function domain $D$ into two subregions by a random discontinuity surface and then generate two smooth functions $f_1, f_2$ for the two subregions in term of Legendre polynomials:

$$f_i(x,y,z) = \sum_{m+n+k \leq N_p} a_{m,n,k}^{(i)} P_m(x) P_n(y) P_k(z), \qquad i = 1,2, \tag{3.3}$$

where $N_p = 3$, $a_{m,n,k}^{i)}$'s are i.i.d. $N(0,10)$, and the discontinuity surfacies are generated as random spherical surfaces defined by $(x-x_0)^2 + (y-y_0)^2 + (z-z_0)^2 = r$ with $r \sim U(0,3)$ and $(x_0, y_0, z_0) \sim U(D)$. Like the two-dimensional detection, the neural network detection exhibits remarkable generalizability, even though the train data consist of very limited classes of piecewise functions separated by very limited classes of discontituity surfaces.

**4. Numerical Experiments and Results.** In this section, we apply the proposed detection methods and the annihilation detection method on various test sets to compare the detection performances. Here, to detect trouble cells of a grid with the annihilation method, we claim a cell is a trouble cell if $|L_m f(x)| > t$ where $x$ is the center of the cell and $L_m f(x)$ defined in Section 2.1 is calculated using $\tilde{v}_f$. To study the robustness and applicability of the trained models, we also apply the methods on more general data that are not generated from the simulation approaches used to generate the synthetic training data. Notice that since the input of the (CNN) detectors only depends on the function values over the uniform grid with no reference to the grid location and the size of the grid cell, the detectors can be applied to any function domain as long as the grid size matches the size of the input. In fact, for the problem with a smaller grid, we can extend the grid and define the function value at each new grid point by extrapolation for a smooth extension to match the size of the detector input. For the problem with a larger grid, we use a "sliding-window" procedure in which the detection algorithm is implemented on subregions of the grid and combine the results of the subregions, where the subregions are determined by placing a sliding window with the size the same as the input size at different locations to cover the entire grid. Hence, the detection algorithms studied here can be applied to the problems with different function domains and uniform grids with different sizes.

**4.1. One-dimensional detection.** We first generate a test set of 10,000 functions using the same simulation approach as in section 3.1 for the training set. Note that the trouble cell detection for the CNN and annihilation methods are based on $\mathcal{N}(\tilde{v}_f)_i > t$ for cell $i$ and $|L_m f(x)| > t$ for cell center $x$, respectively. Thus, by varying the threshold $t$, we can plot the ROC curves to compare their overall performance. Figure 4.1 shows that the two ROC curves are nearly perfect and overlapped with each other. In figure 4.2, we visualize the detection performance by plotting the output of the CNN model (in orange) and the function values (in blue) together. The output values are close to one on the cells of discontinuities and zero elsewhere. Hence, the CNN detector results in an excellent performance, which is expected because the test data are generated in the same way as the training data.

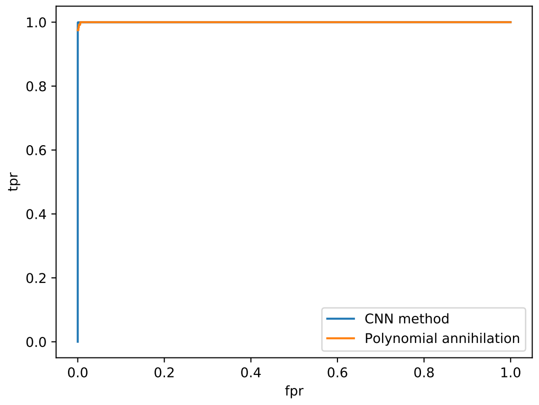


Fig. 4.1: **ROC curves of the CNN and annihilation methods.** The vertical coordinate represents the detection rate (true positive rate), and the horizontal coordinate represents the false alarm rate (false positive rate).

Recall that the synthetic functions in the training data have at most three discontinuities ($M = 3$). What if test function has more than three discontinuities? To answer this question, we apply the trained CNN detector on test functions with up to $M = 6$ discontinuities. Figure 4.3 shows that the CNN detector can accurately identify more than three discontinuities. The performance is consistent among test functions, even for the functions with two or three discontinuities close to each other as shown in the second plot of the figure. The experiment demonstrates the strong generalization capablity of the CNN deterctor – it "learns" the generic properties of discontinuity and is capable to handle scenarios different from the training samples.

The next example is to apply our detector on data with different resolutions, where the function domain is fixed but the grid size is different. As discussed in the first paragraph of Section 4, by extending smaller-size grids or implementing the sliding-window procedure onto larger-size grids, we can apply our trained CNN detector to detect trouble cells over grids of different sizes. Figure 4.4 shows that for the low-resolution data of grid size 52, the CNN output (orange curve) could result in more false detection, while the detection is significantly improved when we increase the grid size to 102. Perfect accuracy is achieved when the grid size is the same as the input size 202 or larger. This result shows that the trained model can be adapted to any data with enough resolution without retraining models with different hyper-parameter setups. The trained CNN model thus becomes a universal discontinuity detector on uniform grids of arbitrary size.

**4.2. Two-dimensional detection.** For two dimensional detection, we shal lable the one-level detetion method in Section 3.2.2 as "1CNN". For the two-level methods in Section 3.2.3, we shall use "2CNN" for the method with both CNN detection on the coarse and fine levels, and "CNNPoly" for the method with CNN on the coarse level and polynomial annihilation on the fine level.

Figure 4.5 shows the detection performance for the three methods on test functions generated in the same way as our training data. The cells in red are the final

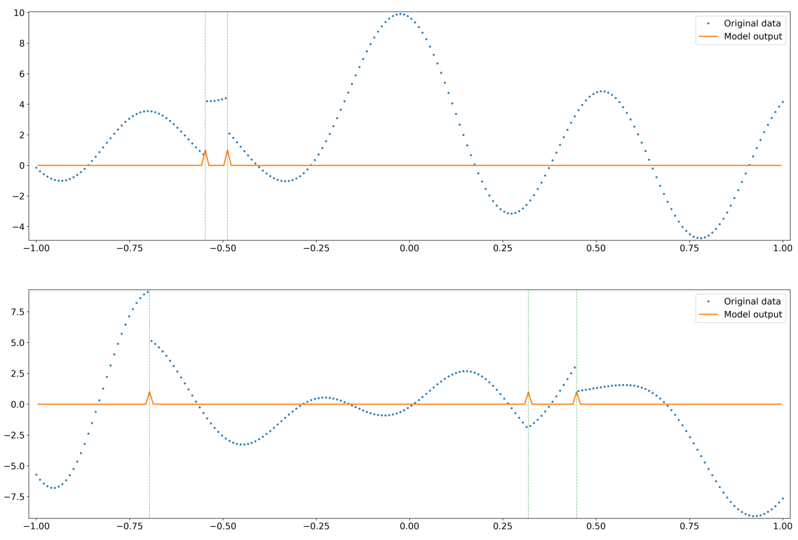


Fig. 4.2: **Detection with the trained CNN model.** The blue curve is the real function on 202 points in $[-1, 1]$, and the green vertical dashed lines indicate the true trouble cells. The orange curve is the output of the CNN model of size 201 corresponding to the 201 cells.

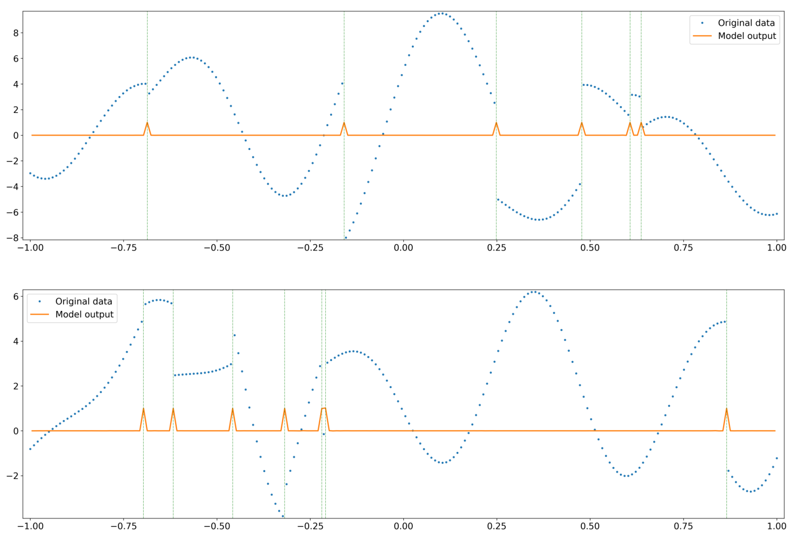


Fig. 4.3: **Detection on test functions with M = 6 discontinuities.** The blue curve is the real function on 202 points in $[-1, 1]$, and the green vertical dashed lines indicate the true trouble cells. The orange curve is the output of the CNN model of size 201 corresponding to the 201 cells.

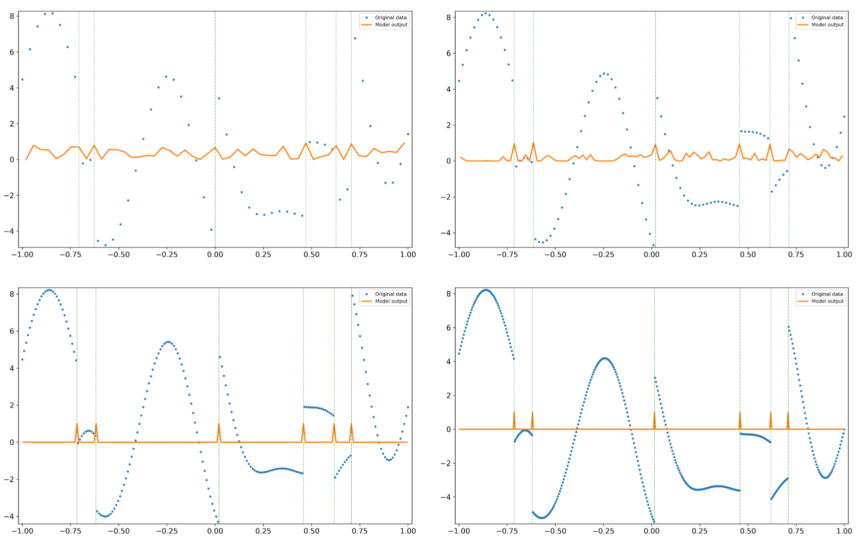


Fig. 4.4: **Detection on data with different resolutions.** Given a test function on $[-1, 1]$, the observed function values with different resolutions are generated by changing the grid size. From left to right and top to bottom, the grid sizes are 52,102,202, and 402.

detected (fine) cells, while the blue boxes are the detected coarse cells from Step 1 of the two-step methods. While the 1CNN model produces quite good results, it does identify some false detected trouble cells around the true trouble cells. This is visible in the top-left plot. On the other hand, both 2CNN and CNNPoly models produce highly accurate identification of the trouble cells. It must be remarked that for the first two examples, as shown in the left two columns of plots, the CNNPoly detection contains certain "gaps" in the discontinuity curves. In another words, the discontintuity curves appear to be "fragmented". Upon close examination of the three-dimensional surfaces of the original functions (shown in the first row of the plots), these "gaps" occur exactly at the locations when the two piecewise smooth surfaces cross over each other. Therefore, the discontinuity interfaces become "continuous" at the cross-over location. The CNNPoly model identifies this kind of singular locations as "continuous" and prduces a gap in the discontinuity interface. The 2CNN model, on the other hand, identifies the singularities as part of the discontinuities. This behavior further demonstrates the remarkable generalization capabilities of the trained CNN models. Even though as our training data are of very limited classes of simple cases, the trained CNN models are able to perform accuracetely for much more complex cases.

More numerical tests are performed on test data of vastly more complex nature than our training data. Figure 4.6 shows the detection results for data with triangular discontinuity curves. The cases include cross-over points along the discontinuous interfaces, shown in Column 3, as well as triangular discontinuities occupying a very small area, shown in Column 1. We observe that the trained CNN models are able to produce highly accurate detection, especially the two-level detection methods. And the 2CNN method is slightly mode accuracte than CNNPoly. This again suggests that the CNN model is able to extract generic features of the discontinuities from

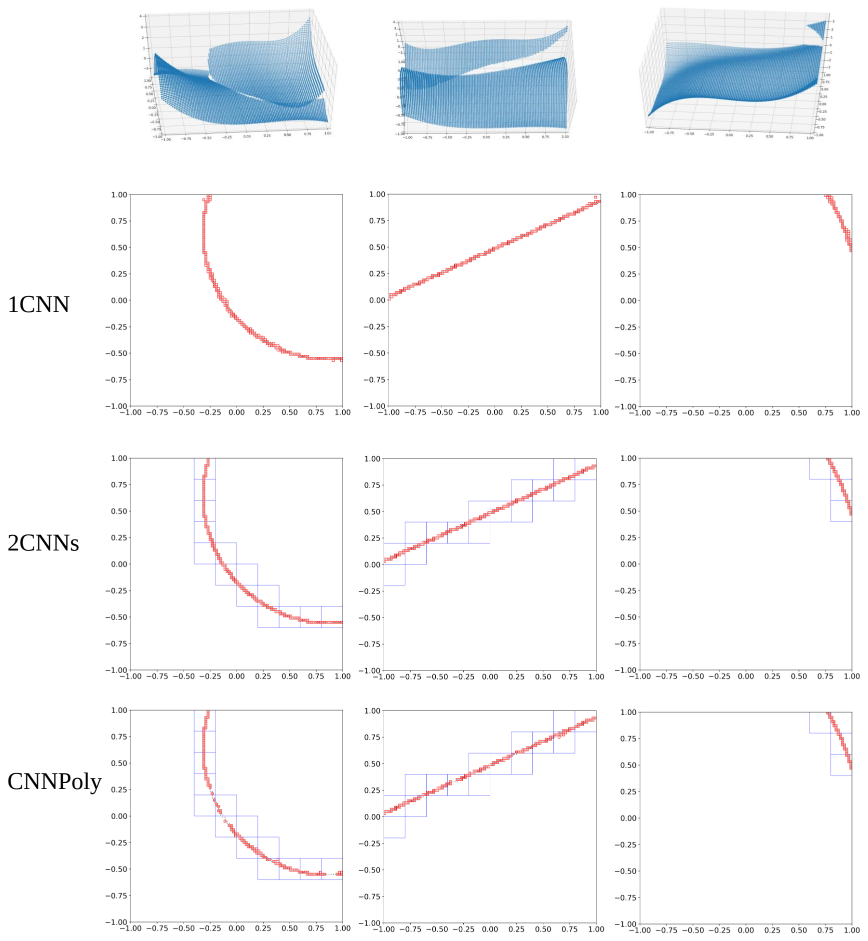


Fig. 4.5: **Two dimensional discontinuity curve detection**. Three test functions in the top row are generated for the detection experiment where the corresponding discontinuity curves are a circle, a line, and a circle. The performances of the three methods, 1CNN, 2CNN, and CNNPoly, are shown in the second, third and fourth rows, where blue boxes are coarse cells detected by the first step of the two-step methods and red cells are detected fine cells.

the simple cases of test data. In doing so, it is then able to detect in test data the discontinuities of much more complex nature.

As in Section 4.1, we apply our detectors on data with different resolutions by extending small-size grids or using the sliding-window procedure, where the function domain is fixed, but the grid size is varied for different resolutions. Figure 4.7 shows that the low-resolution data of grid size $61 \times 61$ may result in slightly more false detection, while the detection is improved when we increase the grid size to $101 \times 101$ or larger. For the two-step methods, 2CNN and CNNPoly, some boundary coarse

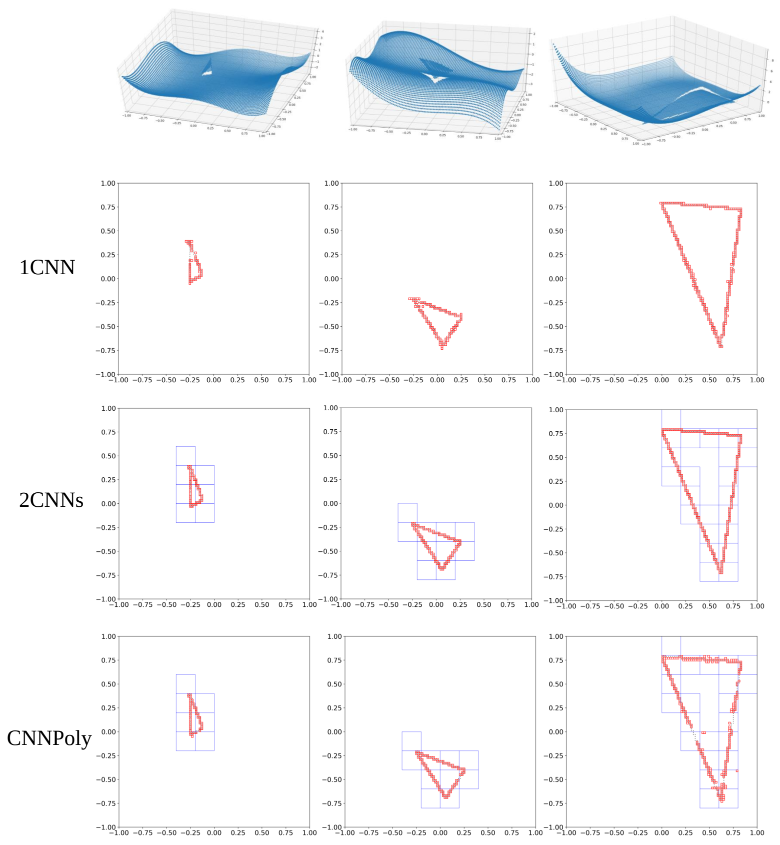


Fig. 4.6: **Detection on data with triangular discontinuity curves.** The models in the three detection methods are trained based on data with circular and linear discontinuity curves, but the test functions in this experiment are of triangular discontinuity curves. The trouble cells of the triangles are well detected. It demonstrates the ability of the trained model to detect new discontinuity curves that have not been seen in the training data.

cells are wrongly detected by the step-1 detector due to the artificial effects from the extension of the grid and function values. However, the step-2 detectors eliminate almost all false detection near the boundary. Therefore, the trained models can be adapted to any data with enough resolution without retraining models for different resolutions, similar to the conclusion in Section 4.1.

**4.3. Three-dimensional detection.** For three-dimensional detection, the results are similar to the two-dimensional problems. We only show the performance of the two-step methods discussed in Section 3.3. In Figure 4.8, blue cubes denote the coarse trouble cells detected by the Step-1 detection and the red cubes denote the fine trouble cells detected by the Step-2 detection. The detected cells well cover the true discontinuity surfaces with a small number of false detections. This is not unex-

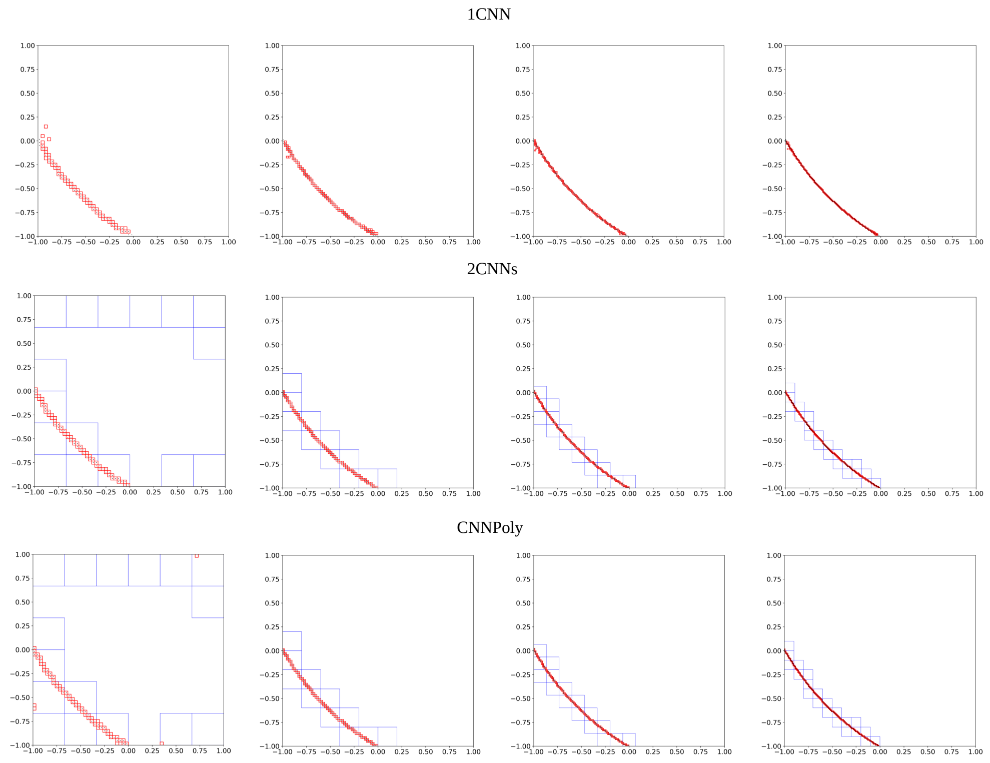


Fig. 4.7: **Detection on 2D data with different resolutions.** Given a test function on $[-1,1]^2$, the observed function values with different resolutions are generated by changing the grid size. From left to right, the grid sizes are $61\times61$,$101\times101$,$151\times151$ and $201\times201$.

pected, and even preferred in practice, for identitifiation of three-dimensional surfaces using cubes. Hence, the two-step method results in accurate detection similar to the result in the two-dimensional detection experiments. Other experiments in terms of resolutions and discontinuity surfaces can be likewise demonstrated as in Section 4.2 and therefore are skipped here.

**5. Conclusion.** This paper constructs discontinuity detectors using convolutional neural networks (CNN) and presents a two-step coarse-to-fine detection procedure for efficient detection of discontinuities of two- or three-dimensional functions. Generally, successful training of deep neural networks requires a very large training set that contains functions of a large variety of discontinuities, which should be dense enough in the space of functions with discontinuities. Since that space is infinite-dimensional, collecting such a dense training set is not feasible. Instead, we propose to generate sets of synthetic training data covering only basic scenarios of discontinuities, which provides enough information for the neural networks to learn the generic properties of discontinuities. Various detection experiments demonstrate that the trained models are able to provide accurate detection for functions with vastly different and more complex discontinuity structures unobserved from the training set.

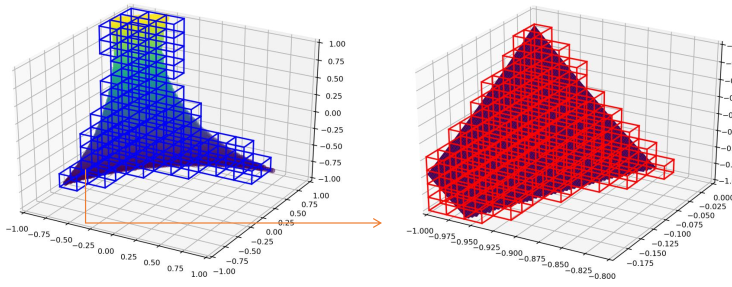


Fig. 4.8: **Three dimensional detection**. The three-dimensional plot on the left side shows the discontinuity surface on the 3-D domain of a three-dimensional test function. Blue cubes are detected coarse cells using the step-1 detector of CNNPoly. Red cubes in the right plot show the step-2 fine detection within a detected coarse cell.

The detection procedures can be adapted for function data with different function domains and different resolutions by the sliding window procedure or grid extension. In this sense, our proposed CNN models become univeral discontinuity detectors for uniform grids of arbitrary domains.

## REFERENCES

[1] R. Archibald, A. Gelb, and J. Yoon, *Polynomial fitting for edge detection in irregularly sampled signals and images*, SIAM Journal on Numerical Analysis, 43 (2005), pp. 259–279.

[2] R. Archibald, A. Gelb, and J. Yoon, *Determining the locations and discontinuities in the derivatives of functions*, Applied Numerical Mathematics, 58 (2008), pp. 577–592.

[3] M. Bozzini, F. De Tisi, and M. Rossini, *Irregularity detection from noisy data with wavelets*, in Wavelets, Images, and Surface Fitting, AK Peters/CRC Press, 1994, pp. 75–82.

[4] M. El-Sayed, Y. A. Estaitia, and M. A. Khafagy, *Automated edge detection using convolutional neural network*, International Journal of Advanced Computer Science and Applications, 4 (2013).

[5] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*, MIT Press, 2016. `http://www.deeplearningbook.org`.

[6] R. Jain, R. Kasturi, and B. G. Schunck, *Machine vision*, vol. 5, McGraw-Hill New York, 1995.

[7] D. P. Kingma and J. Ba, *Adam: A method for stochastic optimization*, arXiv preprint arXiv:1412.6980, (2014).

[8] A. Krizhevsky, I. Sutskever, and G. E. Hinton, *Imagenet classification with deep convolutional neural networks*, in Advances in Neural Information Processing Systems, 2012, pp. 1097–1105.

[9] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel, *Backpropagation applied to handwritten zip code recognition*, Neural Computation, 1 (1989), pp. 541–551.

[10] Y. Liu, M.-M. Cheng, X. Hu, J.-W. Bian, L. Zhang, X. Bai, and J. Tang, *Richer convolutional features for edge detection*, IEEE Transactions on Pattern Analysis and Machine Intelligence, 41 (2019), pp. 1939–1946.

[11] V. Suresh, S. Koteswarao Rao, G. Thiagarajan, and R. P. Das, *Denoising and detecting discontinuities using wavelets*, Indian Journal of Science and Technology, 9 (2016), p. 19.

[12] R. Wang, *Edge detection using convolutional neural network*, in Advances in Neural Networks – ISNN 2016, Cham, 2016, Springer International Publishing, pp. 12–20.

[13] M. Wei, A. R. De Pierro, and J. Yin, *Iterative methods based on polynomial interpolation filters to detect discontinuities and recover point values from fourier data*, IEEE Transac-

tions on Signal Processing, 53 (2004), pp. 136–146.

[14] C. Wen, P. Liu, W. Ma, Z. Jian, C. Lv, J. Hong, and X. Shi, *Edge detection with feature re-extraction deep convolutional neural network*, Journal of Visual Communication and Image Representation, 57 (2018), pp. 84–90.

[15] C. Xue, J. Zhang, J. Xing, Y. Lei, and Y. Sun, *Research on edge detection operator of a convolutional neural network*, in IEEE 8th Joint International Information Technology and Artificial Intelligence Conference, 2019, pp. 49–53.

[16] G. Zhang, C. G. Webster, M. Gunzburger, and J. Burkardt, *Hyperspherical sparse approximation techniques for high-dimensional discontinuity detection*, SIAM Review, 58 (2016), pp. 517–551.