Core Mathematics of Edge Detection in Selected Traditional and Deep Learning Convolution Methods

Felicity Britt

Math 699

Spring 2026

Introduction

This report seeks to study the core mathematical concepts that underlie both traditional and deep learning convolutional neural network ("CNN") methods for edge detection in digital images. Edge detection is a critical process for object classification and segmentation in digital images. As such, an understanding of how the machine learning pipeline is structured, what the core underlying mathematical mechanics of each method are, and the relative performance of each method, all serve to motivate the selection of one CNN method over another. The goal of this report will be to compare the strengths and weaknesses of six methods, three traditional methods and three deep learning methods, and to run each of the models on a marginal image to detect its edges.

Data

The deep learning methods evaluated in this report were primarily trained on the Berkeley Segmentation Data Set 500 ("BSDS500"). Martin [5] developed a database of human segmented natural images that included 150 manually annotated ground truth images. This was the dataset used in their research, and subsequently the dataset was expanded to include 300 images, the BSDS300, and then 500 images, the BSDS500.[2]

The ground truth segments were created naively, in such a way that each observer could add low-, intermediate-, or high-level cues based on their own intuition. It was assumed that the annotators could segment based on their own familiarity with the objects and without any bias. Up to five different annotators per image were allowed to segment the images in order to capture a robust set of segments for each image.

For this report, the BSDS500 was accessed from Kaggle, an online machine learning project site with publicly available data, as well as many public projects with public code and models using the BSDS500 dataset to study and experiment with different CNN models.[1]

The 500 images used in the BDSD500 are broken down into the following sets for the machine learning process:

BSDS500 Data (Annotated)

- Test Images: 200
- Training Images: 200
- Validation Images: 100

Image Processing

As a necessary first step, we introduce the idea of how a digital image becomes operable for convolutions. An image can be converted into a three tensor having dimensions of pixel height, pixel width and RGB color channel. However, color channel as a dimension is flattened before any convolutions by converting the image to grayscale and expressing all colors in the shape of a logistic function taking values from 0 to 1, from black to white, respectively.

For example, throughout the project we will work with a simple image of a vizsla. This image is an 800x600 pixel color jpg image file. We will find the edges of this image using the six methods outlined in this report.

Figure 1: Baseline input image

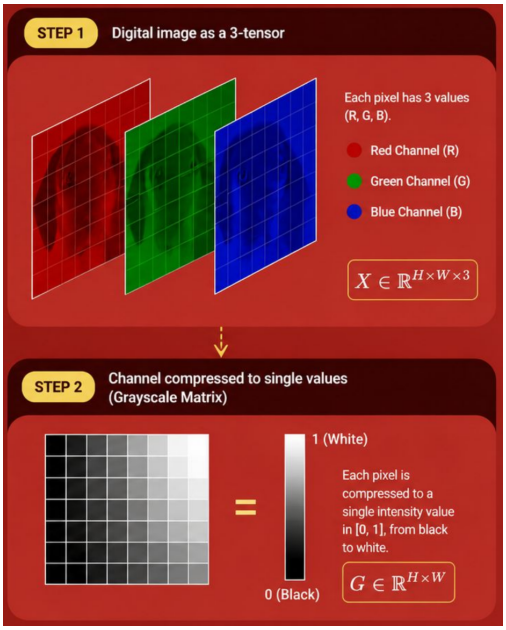


Figure 2: Converting vizsla.jpg into a 2D grayscale object

Traditional Methods and Deep Learning Algorithms

In this report, we will evaluate six different methods for edge detection. Three of the methods evaluated are considered "traditional methods". Traditional methods refers typically to algorithms for edge detection that were developed earlier in the history of machine learning and computer vision.

Generally speaking, traditional methods attempt to find gradient changes across image pixels where a large change in gradient from pixel to pixel indicates an edge. Traditional gradient change based methods use convolution operators, called kernels, that stride across the image pixel by pixel performing convolutions and generating output feature maps.

Popular traditional edge detection operators that were developed in the 1980's and are still useful today include the Sobel, Laplace, and Canny methods.

In contrast, "deep learning methods" started to emerge in the mid 2010's with many complex layers and they markedly improved segmentation performance. Deep learning research has yielded a number of new and improved algorithmic implementations and we will review three of them in this report. They are the Bi-Directional Cascade Network (BDCN), Dense Extreme Inception Network (Dexined), and Pixel Difference Network (PiDiNet).[11]

Traditional Methods

1. Sobel Operator

This algorithm uses a first derivative operator that computes the gradient intensity of a neighborhood of pixels. The algorithm uses two 3×3 kernels, also called filters, to perform convolutions across a neighborhood of pixels. One kernel seeks vertical intensity changes and one seeks horizontal intensity changes, then the results are combined into the gradient magnitude.[6][8]

$$G_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}$$

$$G_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}$$

$$G = \sqrt{G_x^2 + G_y^2}$$

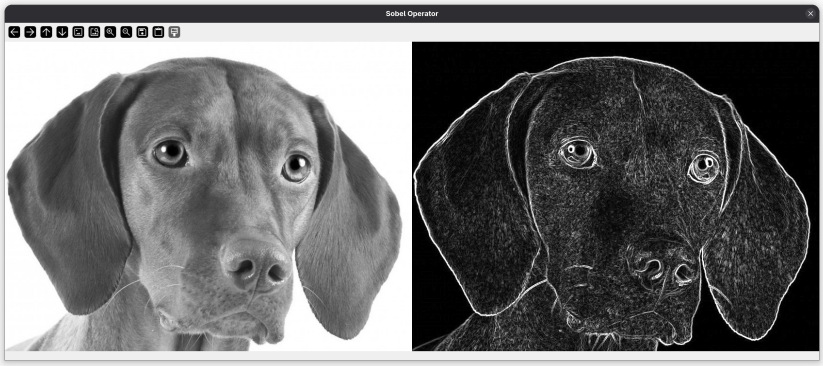


Figure 3: Grayscale input (left) and Sobel operator edges output (right)

Sobel is a very simple application with minimal operations and therefore is cheap for compute. Output features indicate boundary edges of the object as well as internal edges of the facial features. Even a fair amount of texture is returned.

2. Canny Operator

The Canny operator is a multi-stage edge detection algorithm designed to reduce noise while preserving strong edge structures. The method applies Gaussian smoothing before calculating image gradients. Non-maximum suppression is then used to thin edges, and hysteresis thresholding is used to preserve strong connected edge structures while eliminating weak noise responses.

Step 1: Gaussian Smoothing

Gaussian Kernel (2D)

$$G_\sigma(x, y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}}$$

Smoothed Image (Convolution)

$$I_s(x, y) = I(x, y) * G_\sigma(x, y)$$

Step 2: Gradient Computation Using Sobel Kernels

Sobel Kernel — Horizontal Gradient

$$G_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}$$

Sobel Kernel — Vertical Gradient

$$G_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}$$

Gradient Magnitude

$$G = \sqrt{G_x^2 + G_y^2}$$

Gradient Direction

$$\theta = \tan^{-1}\left(\frac{G_y}{G_x}\right)$$

$$\theta \in [-\pi, \pi)$$

Step 3: Non-Maximum Suppression

Quantized direction ranges, only the local maximum is kept, other directions are supressed to 0.

$$-22.5^\circ \leq \theta < 22.5^\circ$$

$$22.5^\circ \leq \theta < 67.5^\circ$$

$$67.5^\circ \leq \theta < 112.5^\circ$$

$$112.5^\circ \leq \theta < 157.5^\circ$$

Step 4: Double Thresholding

Classify strength of the edge based on upper and lower thresholds:
Strong Edge

$$G \geq T_H$$

Weak Edge

$$T_L \leq G < T_H$$

Non-Edge

$$G < T_L$$

Step 5: Hysteresis Edge Tracking

Keep weak edges connected to strong edges

Discard weak edges not connected to strong edges

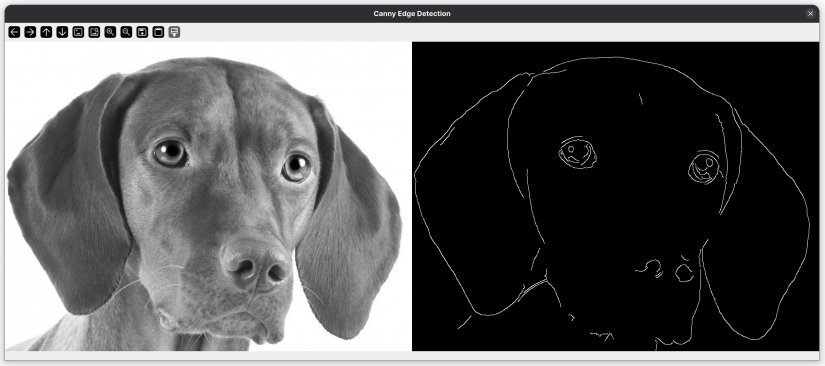


Figure 4: Grayscale input (left) and Canny edges output (right)

Because the Canny edge detection method uses both a Gaussian blur step at the beginning and the hysteresis step to conclude, notice that the ouput feature map has greatly reduced the interior noise that we got from the Sobel operator itself. This method returns much more distinct boundaries, both object boundary and interior boundaries. But, there is significant loss of texture, contour and other low- and intermediate-level features.

3. Laplacian Operator

The Laplacian operator uses second derivatives to detect rapid changes in image intensity. Unlike the Sobel operator, which detects directional gradients, the Laplacian operator is isotropic and responds equally to edges in all directions.

The Laplacian kernel we use for our output is the:

4-Neighbor (Cross) Laplacian Kernel

$$K = \begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$

The Laplacian is highly sensitive to noise, and therefore Gaussian smoothing is often performed prior to applying the operator.

Other Laplacian kernels include different versions of the matrix:

8-Neighbor (All Directions) Laplacian Kernel

$$K = \begin{bmatrix} 1 & 1 & 1 \\ 1 & -8 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

Isotropic (Normalized) Laplacian Kernel

$$K = \begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$$

How the Kernel Works: The center pixel is compared to its neighbors.
If it is very different from the surrounding pixels, the output will have a large magnitude. For example:

a	b	c
d	e	f
g	h	i

$$R = (b + d + f + h) - 4e$$

Large $|R|$ means strong intensity change (edge).

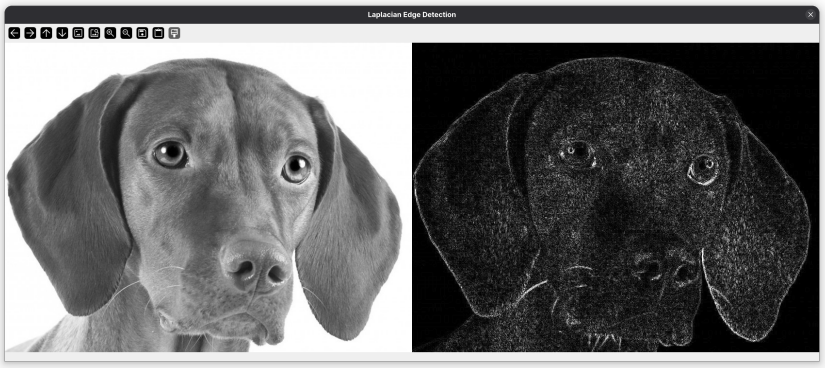


Figure 5: Grayscale input (left) and Laplacian edges output (right)

The Laplacian method picks up gradient changes with great sensitivity, but perhaps sacrifices finding more objective semantic borders for the object's major features. However, depending on the use case, this method may have its pros and cons.

Deep Learning Methods

Before we jump into the three specific deep learning methods that will be evaluated in this report, it is important to take into consideration the foundation on which deep learning is built.

Many of the newer, state of the art algorithms leverage the older methods as either pieces of their process or as ways to "pre-process" data, such that their algorithms can run more efficiently and yield more precise edges, or both ideally.

One of these foundational steps in the computer vision space is called VGG16, which is now colloquially referred to in many research papers as a "backbone". VGG16 was one of the original state of the art deep learning processes that leveraged a pipeline of convolutions layers and was developed at Oxford. Subsequent algorithms often use the backbone as a starting point and then refine and build from it to improve performance for specific use cases.[7]

4. Bi-Directional Cascade Network for Perceptual Edge Detection

This deep learning algorithm, commonly referred to as "BDCN", is a recent cutting edge algorithm developed in 2019 that was designed to improve edge detection by training each network layer with layer specific supervision.

Additionally, the algorithm requires far fewer parameters than other deep learning algorithms, making it more computationally efficient. Like many other deep learning processes, this method leverages the VGG16 backbone.

The novel approach of this method involves what the researchers call a Scale Enhancement Model, which consists of a set of parallel convolutions with different dilation rates. Dilated convolution increases the receptive field of network neurons, which allows better capture of multi-scale features. The bi-directional approach causes each layer to focus on a specific scale. One direction evaluates features deep-to-shallow and the other direction evaluates features from shallow-to-deep.[3]

BDCN Architecture

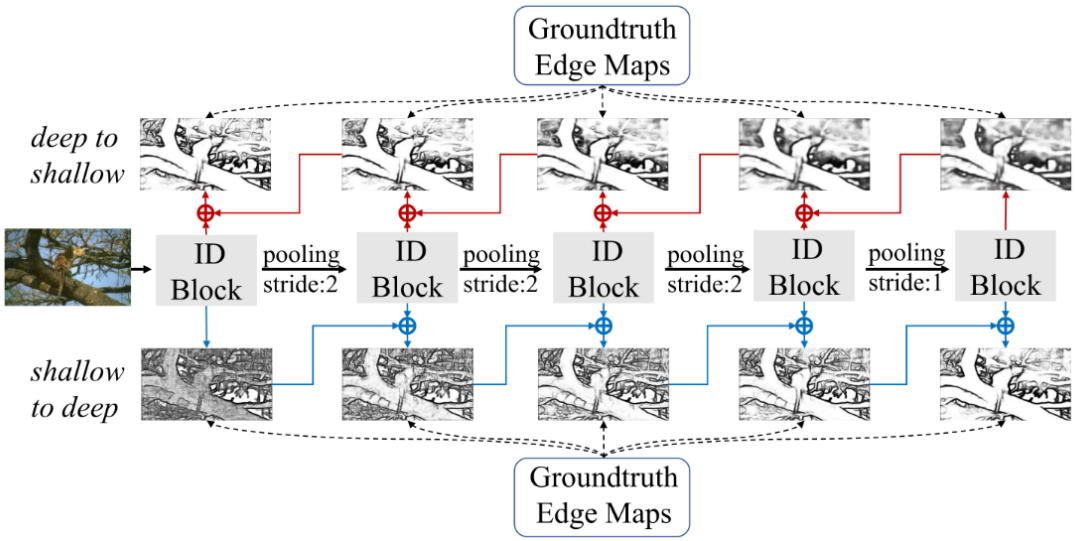


Figure 6: Illustrating the bi-directional ID block process

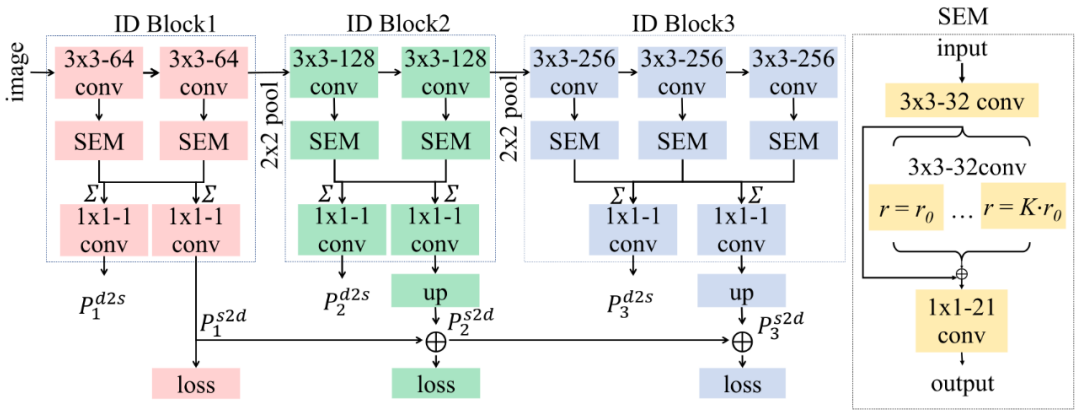


Figure 7: ID blocks and SEM blocks structure

BDCN Technical Steps

1. Approximate edges in both directions, $s2d$ and $d2s$, which implies two complementary supervisions.
2. Layer-specific loss functions.
3. SEM includes dilated convolution.
4. Overall loss function combines all intermediate and final predictions.
5. Shallow blocks identify local details while deeper blocks are sensitive to semantic edges.
6. SGD optimizer with learning momentum decay rates.
7. Non-Maximum Suppression for final edge maps.

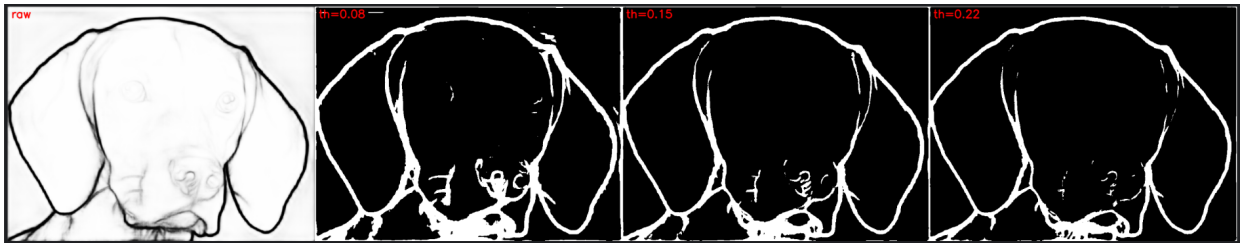


Figure 8: Raw BDCN output as well as varying threshold picks

BDCN does a great job at identifying semantic boundaries, the overall object, the interior larger features like ears, jaw and neck. This is a really good high-level cue result. Low- and intermediate-level cues have been supressed.

5. Dense Extreme Inception Network for Edge Detection

This deep learning algorithm, commonly referred to as "DexiNed", is a recent cutting edge algorithm developed in 2023 that was designed to focus heavily on edge detection performance and to carve out a special niche for algorithms that do not conflate edge detection with boundary or contour detection.

As such, the researchers built their own unique dataset, with their own ground truth annotations, in an effort to focus the end-to-end training process entirely on edge detection, without bias.

They argue that other datasets, including the benchmark BSDS500, do not properly isolate edges within the annotation process and are conflating boundaries and contours into the prediction process, thereby reducing fidelity in strictly identifying edges.

It is important to note that DexiNed does not train on BSDS500, but it does evaluate BSDS500 for performance benchmarking.

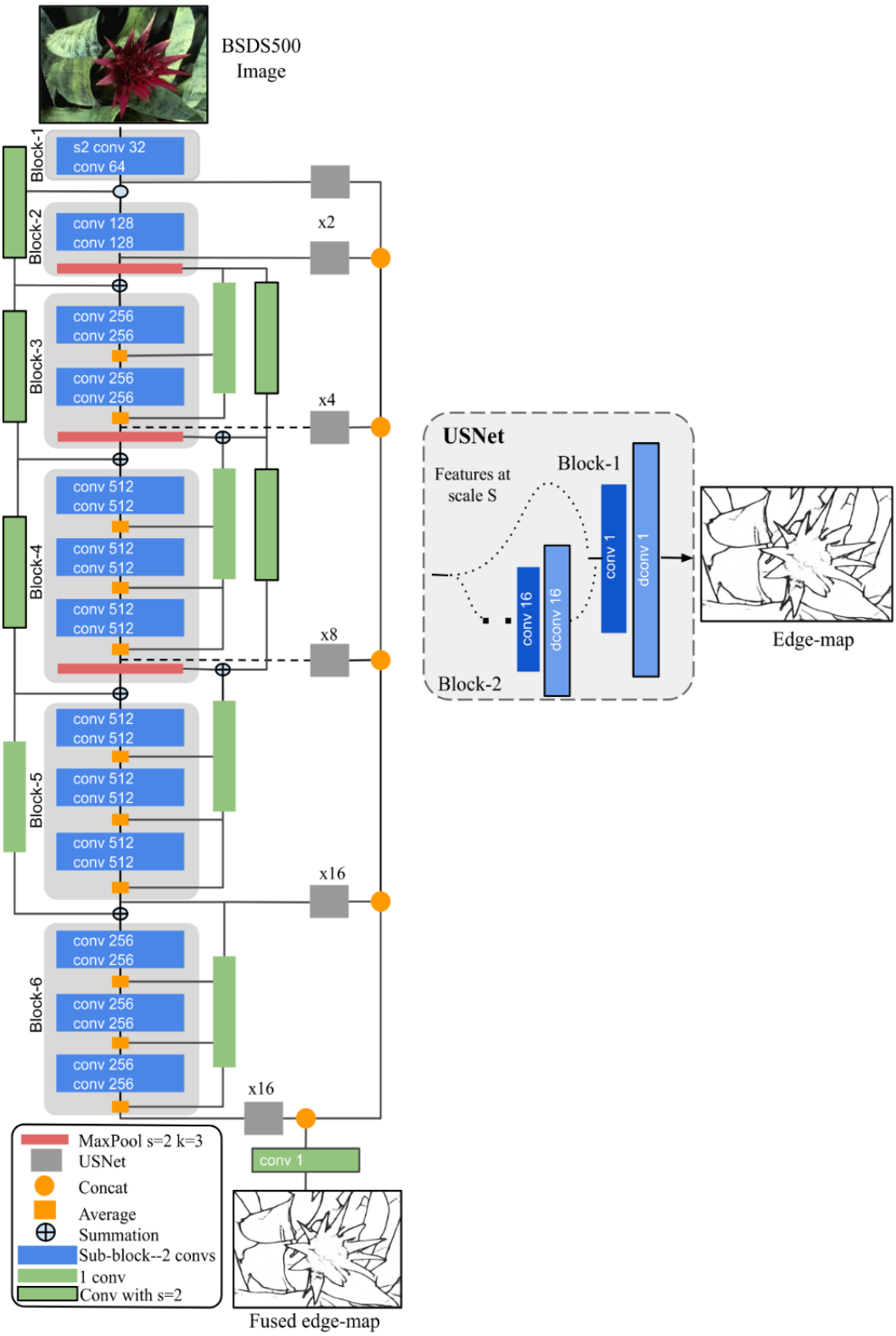
Some key differences between DexiNed and the other deep learning methods discussed in this report are that the DexiNed researchers designed their algorithm to train without prior weight initializations from pre-trained models, such as the commonly used backbone architecture of VGG16.[9][13]

Technical Steps

1. Sub-block convolution layers.
2. Parallel skip-connections to avoid vanishing edge features.
3. Intermediate edge maps via USNet.
4. Batch normalization and ReLU after convolution.
5. Upsampling process using transpose convolutions.

Figure 9: DexiNed edges output

DexiNed Architecture



DexiNed outputs very clean semantic boundaries for both object edges as well as internal edges. It picks up some countour edges as well and seems to capture well the main features of the dog without picking up superfluous textures or noise.

6. Pixel Difference Network

This deep learning algorithm, commonly referred to as "PiDiNet", is a recent cutting edge algorithm developed in 2023 that improved performance while remaining computationally efficient by cleverly using pixel difference convolution techniques.

The key to this algorithm comes from combining multiple types of pixel difference convolutions that capture gradient information in different directions. Pixel difference patterns include central differences, angular differences and radial differences, which each add signal into the final output feature map.[10][4]

Technical Steps

1. Build pixel difference convolutions (central, angular, radial).
2. Calculate pixel differences within a local patch, resulting in a pixel difference vector.
3. During training the kernels are encouraged to have higher inner products with important encodings to create higher activation response.
4. After convolution kernels are learned, they can be converted to vanilla layers by overwriting the kernel weights with the differences for efficiency.
5. Four stages and a max pooling layer are used for downsampling, residual path depth-wise convolution layer, ReLU layer, point-wise convolution layer, and CSAM to eliminate noise.
6. Final edge map comes from fusing four single-channel feature maps with concatenation, a convolutional layer and a sigmoid function.

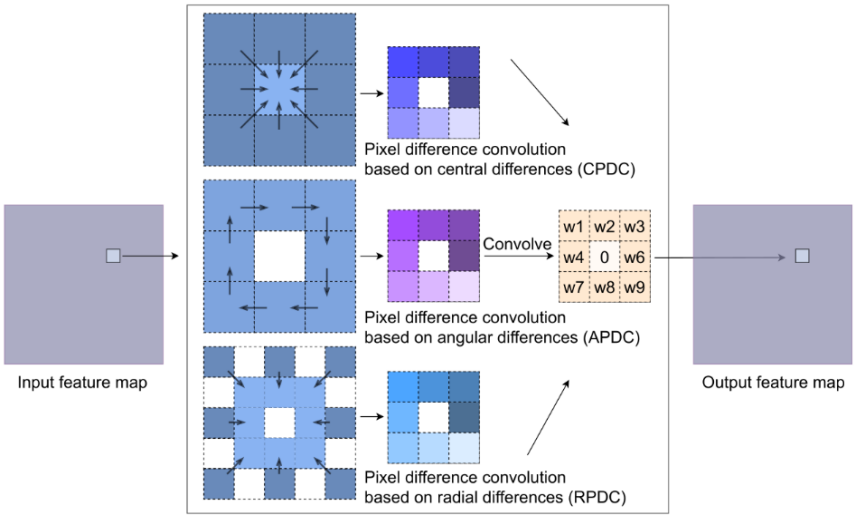


Figure 10: Pixel Difference Convolutions

PiDiNet Architecture

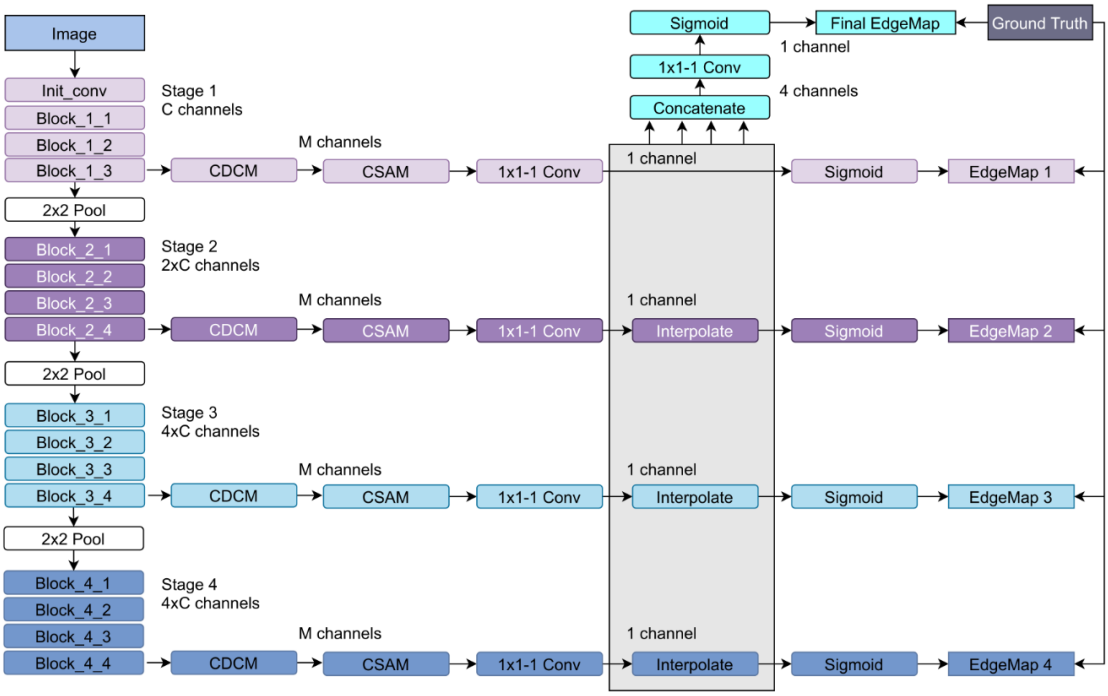


Figure 11: Full Architecture

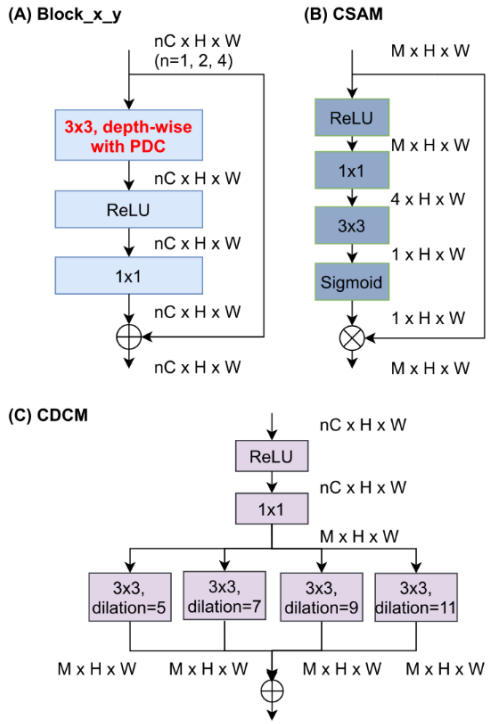


Figure 12: Sub-Architecture

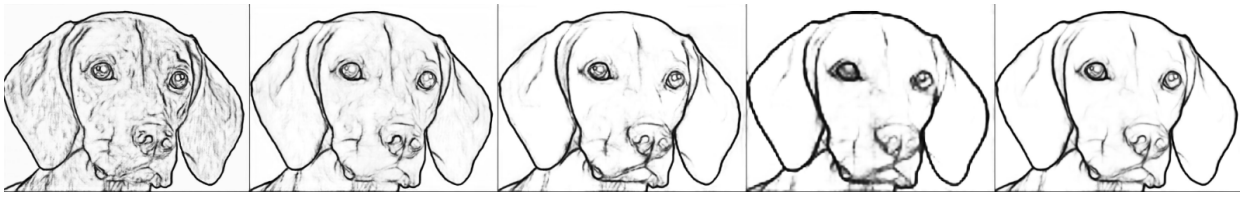


Figure 13: PiDiNet edge outputs, individual predictions (left 1-4) and final fused prediction (right)

PiDiNet fused output does a really good job at prodiving both global semantic boundaries, internal edges of intermediate-level features and even some textural elements. This method uses many intermediate predictions and fuses them into a final prediction.

Core Mathematical Concepts of Deep Learning CNN Algorithms

[12]

a. ReLU

Nonlinear activation functions in CNNs are used to introduce nonlinearity into the network, allowing the model to learn complex patterns and relationships that cannot be represented by linear operations alone.

Without nonlinear activations, a CNN composed only of convolutions and matrix operations would behave like a single linear transformation, regardless of how many layers it has.

$$f(x) = \max(0, x)$$

$$f(x) = \begin{cases} 0, & x \leq 0 \\ x, & x > 0 \end{cases}$$

b. Sigmoid and Softmax

The sigmoid and softmax functions are activation functions commonly used in the output layer of neural networks to convert raw outputs (logits) into interpretable probabilities.

The sigmoid function maps any real value into the range. Used for predicting the probability of a single class.

$$0 \leq \sigma(x) \leq 1$$

$$\sigma(x) = \frac{1}{1 + e^{-x}}$$

While, Softmax generalizes sigmoid to multiple classes. It converts a vector of raw scores into a probability distribution where:

$$\sum_i p_i = 1$$

$$\text{Softmax}(z_i) = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}$$

c. MaxPool

Max pooling is used in CNNs to reduce the spatial dimensions of feature maps while preserving the most important features.

$$y(i, j) = \max_{(m,n) \in \mathcal{R}} x(i + m,\ j + n)$$

$$\text{MaxPool}(X) = \max \begin{bmatrix} x_{11} & x_{12} \\ x_{21} & x_{22} \end{bmatrix}$$

$$\max \begin{bmatrix} 1 & 3 \\ 2 & 0 \end{bmatrix} = 3$$

d. Loss Function

Weighted Cross-Entropy Loss is used for classification problems where the dataset is imbalanced (for example, many more negative examples than positive ones).

$$\mathcal{L}(\hat{Y}, Y) = -\alpha \sum_{j \in Y_-} \log(1 - \hat{y}_j) - \beta \sum_{j \in Y_+} \log(\hat{y}_j)$$

where,

$$0 \leq \mathcal{L} < \infty$$

and a value closer to 0 is optimal.

- Y_- : set of negative class indices
- Y_+ : set of positive class indices
- α : weight applied to negative classes
- β : weight applied to positive classes
- $\hat{y}_j \approx 1 \quad \rightarrow \quad$ confidence the pixel is an edge
- $\hat{y}_j \approx 0 \quad \rightarrow \quad$ confidence the pixel is background

e. Backpropagation

Backpropagation is the algorithm used to train neural networks by computing how much each weight contributes to the prediction error and updating the weights to improve the loss function result.

Backpropagated Error Signal

$$\delta^{(l)} = \left(W^{(l+1)T} \delta^{(l+1)} \right) \odot f'(z^{(l)})$$

where:

$\delta^{(l)}$: error term (gradient signal) for layer l

$W^{(l+1)}$: weight matrix of the next layer

$W^{(l+1)T}$: transpose of the next layer's weight matrix

$f'(z^{(l)})$: derivative of the activation function

$z^{(l)}$: weighted input before activation

$\odot$: element-wise multiplication

Gradient Computation

$$\frac{\partial \mathcal{L}}{\partial W^{(l)}} = \delta^{(l)}(a^{(l-1)})^T$$

where,

$$\frac{\partial \mathcal{L}}{\partial W^{(l)}} \ : \text{ weight gradients}$$

$$a^{(l-1)} \ : \text{ activations from the previous layer}$$

$$\mathcal{L} \ : \text{ loss function}$$

Gradient Descent Weight Update

$$W^{(l)} \leftarrow W^{(l)} - \eta \frac{\partial \mathcal{L}}{\partial W^{(l)}}$$

where,

$$W^{(l)} \ : \text{ weight matrix at layer } l$$

$$\eta \ : \text{ learning rate}$$

$$\frac{\partial \mathcal{L}}{\partial W^{(l)}} \ : \text{ computed gradient}$$

f. ODS and OIS Performance Metrics

All of the deep learning algorithms reviewed in this paper use a common set of metrics for the performance of their models. Performance conceptually is how accurate the model produces its edges relative to the human ground truths from the original annotated training set. The higher the measure (max 1), the better.

ODS (Optimal Dataset Scale)
ODS measures the best F-measure using a single threshold across the entire dataset.

$$ODS = \max_{t \in T} F_1^{\text{dataset}}(t)$$

OIS (Optimal Image Scale)
For each image i, compute the best F-measure:

$$F_1^{i*} = \max_{t \in T} F_1^i(t)$$

Then average over all N images:

$$OIS = \frac{1}{N} \sum_{i=1}^{N} F_1^{i*}$$

$$OIS \geq ODS$$

F-measure
For a threshold t,

$$F_\beta(t) = \frac{(1+\beta^2)P(t)R(t)}{\beta^2 P(t) + R(t)}$$

Usually, $\beta = 1$, giving the standard F1-score:

$$F_1 = \frac{2PR}{P+R}$$

Precision and Recall

$$P(t) = \frac{TP(t)}{TP(t) + FP(t)}$$

$$R(t) = \frac{TP(t)}{TP(t) + FN(t)}$$

where:

- TP: True Positives
- FP: False Positives
- FN: False Negatives
- TN: True Negatives

Model Performance Results and Comparison

Each research team for the three deep learning algorithms studied in this report produce ODS and OIS metrics to benchmark the performance of their model to other well known models in the field. At the time of their research, each algorithm found best in class performance against a sample of its peers as shown below. An ODS or OIS with a perfect 1.0 score would reflect an exact replication of ground truth as an output of the model. Numbers closer to 1.0, therefore, are interpreted as better results.

BDCN Performance

Method	ODS	OIS	AP
Human	.803	.803	–
SCG [40]	.739	.758	.773
PMI [20]	.741	.769	.799
OEF [17]	.746	.770	.820
DeepContour [42]	.757	.776	.800
HFL [3]	.767	.788	.795
HED [49]	.788	.808	.840
CEDN [53] †	.788	.804	–
COB [32]	.793	.820	.859
DCD [27]	.799	.817	.849
AMH-Net [51]	.798	.829	.869
RCF [30]	.798	.815	–
RCF [30] †	.806	.823	–
RCF [30] ‡	.811	.830	–
Deep Boundary [23]	.789	.811	.789
Deep Boundary [23] ‡	.809	.827	.861
Deep Boundary [23] ‡ + Grouping	.813	.831	.866
CED [47]	.794	.811	.847
CED [47] ‡	.815	.833	.889
LPCB [9]	.800	.816	–
LPCB [9] †	.808	.824	–
LPCB [9] ‡	.815	.834	–
BDCN	**.806**	**.826**	.847
BDCN †	**.820**	**.838**	**.888**
BDCN ‡	**.828**	**.844**	**.890**

DexiNed Performance

Methods	Trained on	Tested on	ODS	OIS	AP	Tested on	ODS	OIS	AP
RCF[4] (2019)	MDBD [7]	MDBD [7]	.879	.888	.926	BIPED	.8140	.828	.871
BDCN[5] (2020)			.887	.891	.793		**.854**	**.863**	.768
CATS[6] (2021)			.891	.899	.809		.813	.831	.791
DexiNed-f (Ours)			.891	.896	.930		.787	.807	.844
DexiNed-a (Ours)			**.894**	**.902**	**.951**		.789	.813	**.875**
RCF[4] (2019)	BIPED	BIPED	.849	.861	.906	MDBD [7]	.839	.854	.865
BDCN[5] (2020)			.890	.899	.934		.855	.864	.692
CATS[6] (2021)			.887	.892	.817		.837	.840	.496
DexiNed-f (Ours)			**.895**	**.900**	.927		**.863**	.871	.867
DexiNed-a (Ours)			.893	.897	**.940**		.862	**.874**	**.919**

PiDiNet Performance

Method	ODS	OIS	FPS
Human	.803	.803	
Canny [5]	.611	.676	28
Pb [37]	.672	.695	-
SCG [59]	.739	.758	-
SE [9]	.743	.763	12.5
OEF [16]	.746	.770	2/3
DeepEdge [2]	.753	.772	1/1000†
DeepContour [47]	.757	.776	1/30†
HFL [3]	.767	.788	5/6†
CEDN [62]	.788	.804	10†
HED [60]	.788	.808	78‡
DeepBoundary [23]	.789	.811	-
COB [36]	.793	.820	-
CED [55]	.794	.811	-
AMH-Net [61]	.798	.829	-
RCF [31]	.806	.823	67‡
LPCB [8]	.808	.824	30†
BDCN [18]	.820	.838	47‡
FINED-Inf [56]	.788	.804	124†
FINED-Train [56]	.790	.808	99†
Baseline	.798	.816	96‡
PiDiNet	.807	.823	92‡,*
PiDiNet-L	.800	.815	128‡
PiDiNet-Small	.798	.814	148‡
PiDiNet-Small-L	.793	.809	212‡
PiDiNet-Tiny	.789	.806	152‡
PiDiNet-Tiny-L	.787	.804	215‡

Conclusion

This review of traditional and deep learning convolution algorithms was enlightening as it showcased how the original mathematical construct of the kernel convolution remains the backbone of even the most complex convolutional neural network algorithms. This simple mathematical operation has been used repetitively, in varied sizes, dilated, and supressed all to generate important signals that help identify edges.

The math of deep learning is complex and rigorous, so this survey is meant to be a primer for someone new to the field who seeks to understand the mathematical constructs that underlie the core components of deep learning convultional neural networks. We surveyed methods from the first wave of computer vision in the 1980's to modern best in class algorithms developed within the last five years and have measured their performance against each other and against peer models to see how far deep learning has taken us.

And to the purpose of our report, we have used the six methods to map edges from a marginal image, our vizsla, to see how well the simpler operators define edges as compared to the complex pre-trained models. And those outputs were shared herein for the reader to observe those differences. There is no clear winner, but rather we can see how specific use cases may warrant using one model over another, whether we have restricted compute power or compute time or if we seek to implement edge detection that focuses more on semantic boundaries or on finer intermediate- or low-level cues.

The online community is robust, with access to data and to the code of the researchers being widely available for others to study as cited below.

Bibliography

1. Ashwath, Balraj. “Berkeley Segmentation Dataset 500 (BSDS500).” Kaggle, October 12, 2020.
2. “Contour Detection and Image Segmentation Resources.” UC Berkeley Computer Vision Group.
3. He, Jianzhong, Shiliang Zhang, Ming Yang, Yanhu Shan, and Tiejun Huang. “Bi-Directional Cascade Network for Perceptual Edge Detection.” CVPR, 2019.
4. Hellozhuo. “PiDiNet GitHub Repository.”
5. Martin, D., C. Fowlkes, D. Tal, and J. Malik. “A Database of Human Segmented Natural Images and Its Application to Evaluating Segmentation Algorithms and Measuring Ecological Statistics.” Proceedings Eighth IEEE International Conference on Computer Vision, 2001.
6. moukthika. “Edge Detection Using OpenCV.” OpenCV, May 6, 2025.
7. Simonyan, Karen, and Andrew Zisserman. “Very Deep Convolutional Networks for Large-Scale Image Recognition.” arXiv, 2014.
8. Sobel, Irwin. “History and Definition of the So-Called Sobel Operator.” HP Labs, 1989.
9. Soria, Xavier, Angel Sappa, Patricio Humanante, and Arash Akbarinia. “Dense Extreme Inception Network for Edge Detection.” Pattern Recognition, 2023.
10. Su, Z., Zhang, J., Wang, L., Zhang, H., Liu, Z., Pietikäinen, M., and Liu, L. “Lightweight Pixel Difference Networks for Efficient Visual Representation Learning.” IEEE Transactions on Pattern Analysis and Machine Intelligence, 2023.
11. Sun, Rui, Tao Lei, Qi Chen, Zexuan Wang, Xiaogang Du, Weiqiang Zhao, and Asoke K. Nandi. “Survey of Image Edge Detection.” Frontiers in Signal Processing 2, March 9, 2022.
12. Szeliski, Richard. Computer vision: Algorithms and applications. Cham: Springer, 2023.
13. Xavysp. “DexiNed GitHub Repository.”