Detecção de Arestas: Canny x CNN’s

Tomás Ferranti

Motivação

Problema


CLIO
7606 GNV


7606 GNV

Algoritmo de Detecção de Arestas de Canny

- Desenvolvido por John F. Canny em 1983 pelo MIT
- Trata detecção de arestas como um problema de processamento de sinais
- Ideia principal: “a mudança de intensidade dos pixels na imagem possui um valor alto nos pixels correspondentes a arestas”
- Possui cinco passos principais

Primeiro passo - Filtro gaussiano

- Aplicar um filtro gaussiano para remover ruídos suavizando a imagem

$$H_{ij} = \frac{1}{2\pi\sigma^2} \exp\left(-\frac{(i-(k+1))^2 + (j-(k+1))^2}{2\sigma^2} \right); 1 \leq i, j \leq (2k+1)$$

$$\mathbf{B} = \frac{1}{159} \begin{bmatrix} 2 & 4 & 5 & 4 & 2 \\ 4 & 9 & 12 & 9 & 4 \\ 5 & 12 & 15 & 12 & 5 \\ 4 & 9 & 12 & 9 & 4 \\ 2 & 4 & 5 & 4 & 2 \end{bmatrix} * \mathbf{A}$$

Segundo passo - Cálculo dos gradientes

- Gx e Gy: Gradientes da direção horizontal e vertical, respectivamente, calculados por um determinado operador (temos abaixo Prewitt e Sobel, respectivamente)
- São armazenados a intensidade e direção (vertical, horizontal, duas diagonais) para cada pixel:

$$\mathbf{G_x} = \begin{bmatrix} +1 & 0 & -1 \\ +1 & 0 & -1 \\ +1 & 0 & -1 \end{bmatrix} * \mathbf{A} \quad \text{and} \quad \mathbf{G_y} = \begin{bmatrix} +1 & +1 & +1 \\ 0 & 0 & 0 \\ -1 & -1 & -1 \end{bmatrix} * \mathbf{A}$$

$$\mathbf{G}_x = \begin{bmatrix} +1 & 0 & -1 \\ +2 & 0 & -2 \\ +1 & 0 & -1 \end{bmatrix} * \mathbf{A} \quad \text{and} \quad \mathbf{G}_y = \begin{bmatrix} +1 & +2 & +1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} * \mathbf{A}$$

$$\mathbf{G} = \sqrt{\mathbf{G}_x{}^2 + \mathbf{G}_y{}^2}$$

$$\mathbf{\Theta} = \operatorname{atan2}(\mathbf{G}_y, \mathbf{G}_x)$$

Terceiro passo - Limite de magnitude de gradiente

- Aplicar um filtro onde a magnitude do gradiente do pixel é anulada caso seja menor que pelo menos um dos pixels vizinhos que se encontram na direção do gradiente

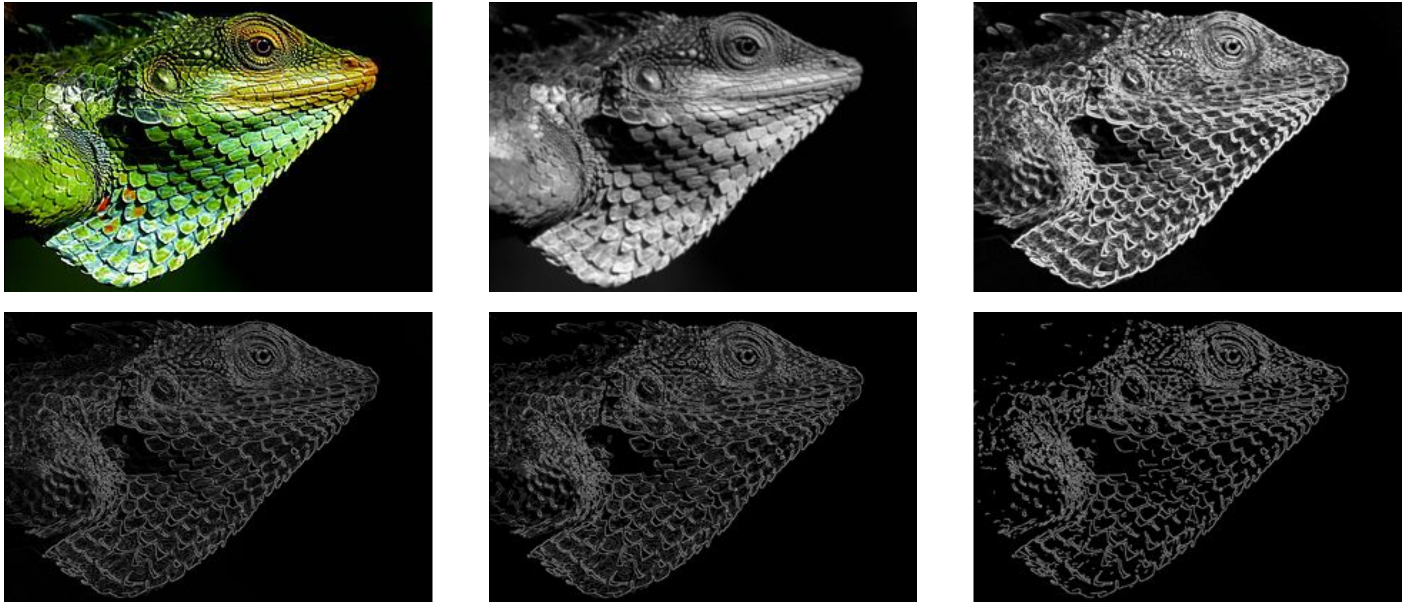
Quarto passo - Corte de dois limiares

- Dois limiares: baixo e alto
  - Pontos de arestas com intensidade acima do limiar alto são marcados como pontos de arestas fortes
  - Pontos de arestas com intensidade entre os dois limiares são marcados como pontos de arestas fracas
  - Pontos de arestas com intensidade menor que o limiar baixo são anulados

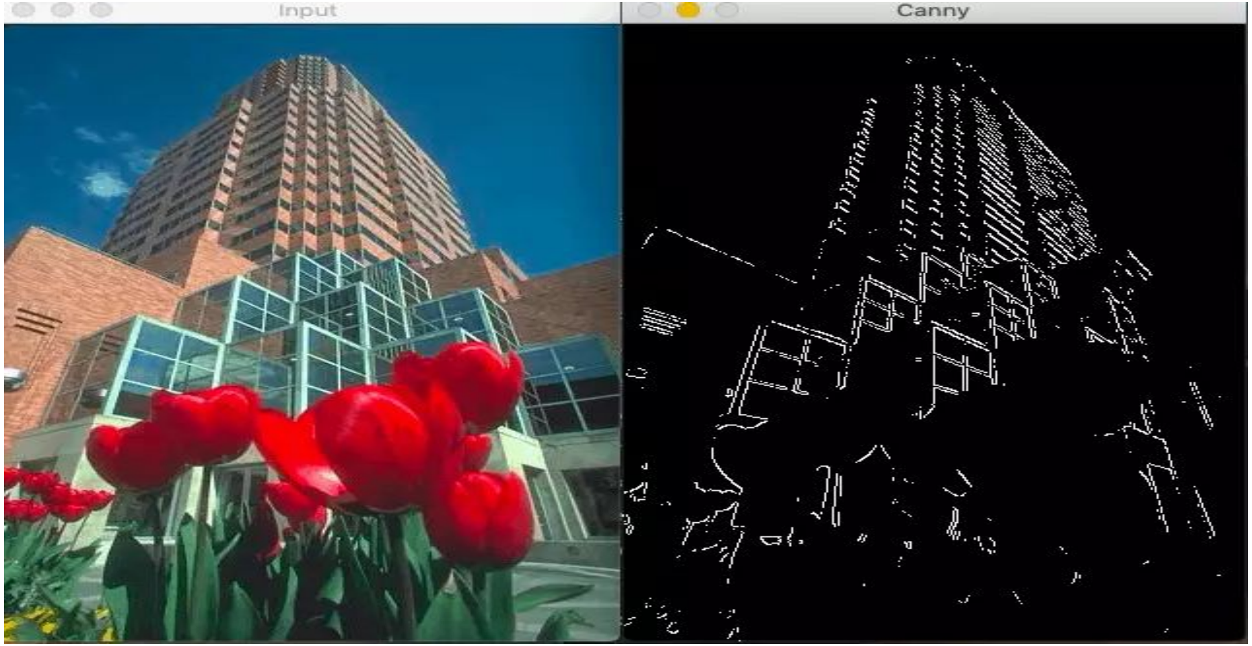
Quinto e último passo - *Hysteresis*

- Os pontos que fazem parte de arestas fortes vão para o resultado final
- Com relação aos que representam arestas fracas, eles são mantidos se e somente se há um ponto que representa uma aresta forte em sua vizinhança (3x3)

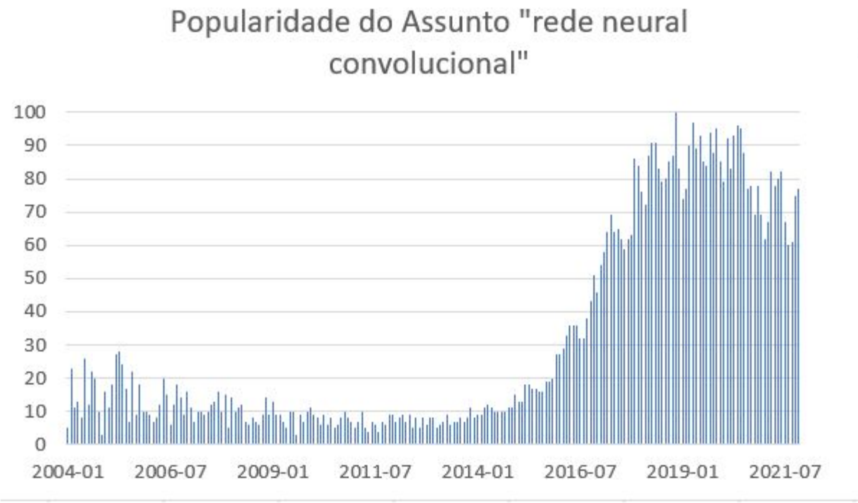
Exemplo



Nem sempre funciona...



Google Trends



An Introductory Review of Deep Learning for Prediction Models With Big Data

Frank Emmert-Streib[1,2], Zhen Yang[1], Han Feng[1,3], Shailesh Tripathi[1,3] and Matthias Dehmer[3,4,5]*

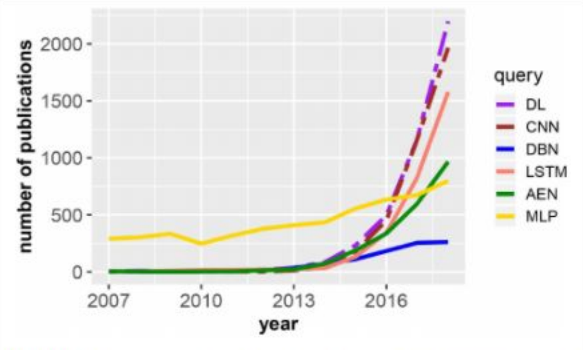
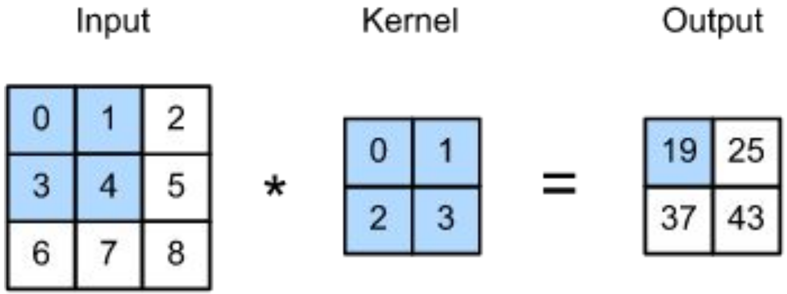
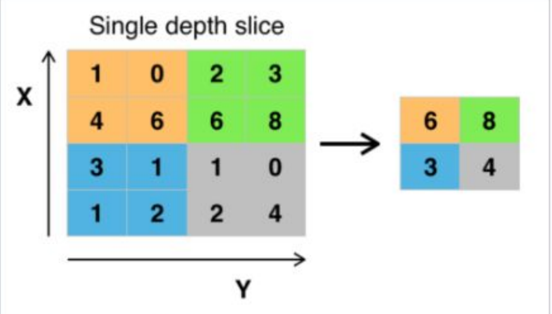


FIGURE 1 | Number of publications in dependence on the publication year for DL, deep learning; CNN, convolutional neural network; DBN, deep belief network; LSTM, long short-term memory; AEN, autoencoder; and MLP, multilayer perceptron. The legend shows the search terms used to query the Web of Science publication database. The two dashed lines are scaled by a factor of 5 (deep learning) and 3 (convolutional neural network).

Recapitulando CNN's

- Dados com uma estrutura padrão: áudios, fotos, imagens médicas ou vídeos
- Duas operações principais: convolução e *sub-sampling*
    - No caso de imagens, um kernel (geralmente 3x3 ou 5x5) convoluído nos pixels para gerar um determinado número de filtros que produzem os *feature maps*
    - Diferentes métodos de *sub-sampling*, com o principal sendo o de *pooling* (mínimo, máximo, médio)





Max pooling with a 2x2 filter and stride = 2

Formulação do Problema

- Receber diferentes imagens com dimensões de largura e altura fixas, com valores entre 0 (preto) e 255 (branco) indicando a escala cinza ou RGB
- Criar uma arquitetura para processar e treinar a rede neural
- Retornar como resultado para cada imagem uma de mesma dimensão onde os pixels são marcados com a probabilidade de pertencerem a uma aresta

$$MSE = \frac{1}{m\,n}\sum_{i=0}^{m-1}\sum_{j=0}^{n-1}[I(i,j) - K(i,j)]^2$$

Entendendo as Métricas

$$PSNR = 10 \cdot \log_{10}\left(\frac{MAX_I^2}{MSE}\right)$$

- Peak signal-to-noise ratio (PSNR)
- F-measure (F)
- Optimal dataset score (ODS) é o limiar no qual a F-measure é maximizada no dataset
- Optimal image score (OIS) é a média do limiar no qual a F-measure é maximizada em cada imagem
- Average precision (AP)

(IJACSA) International Journal of Advanced Computer Science and Applications,
Vol. 4, No. 10, 2013

Automated Edge Detection Using Convolutional Neural Network

Mohamed A. El-Sayed
Dept. of Math., Faculty of Science, University of Fayoum, Egypt.
Assistant prof. of CS, Taif University, KSA

Yarub A. Estaitia
Assistant prof. of Computer Science, College of computers and IT, Taif University, KSA

Mohamed A. Khafagy
Dept. of Mathematics, Faculty of Science, Sohag University, Egypt

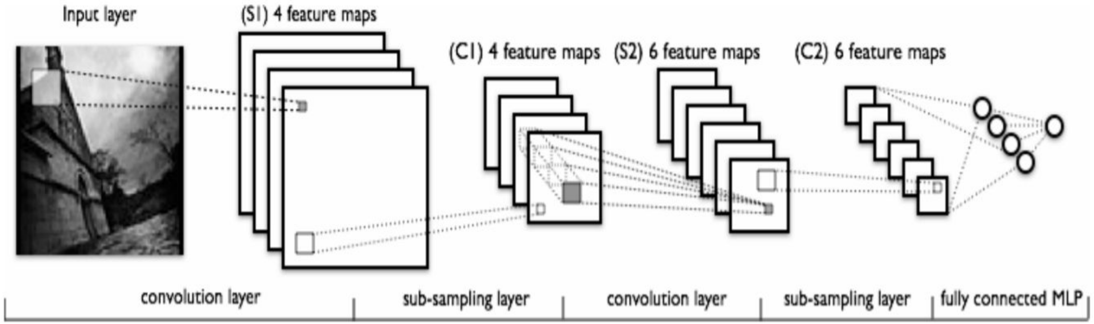


Fig. 6. Full Model of Convolutional Neural network

Fig. 7. Output of randomly initialized network

- Função de Perda: MSE
- # de parâmetros: 1.7M

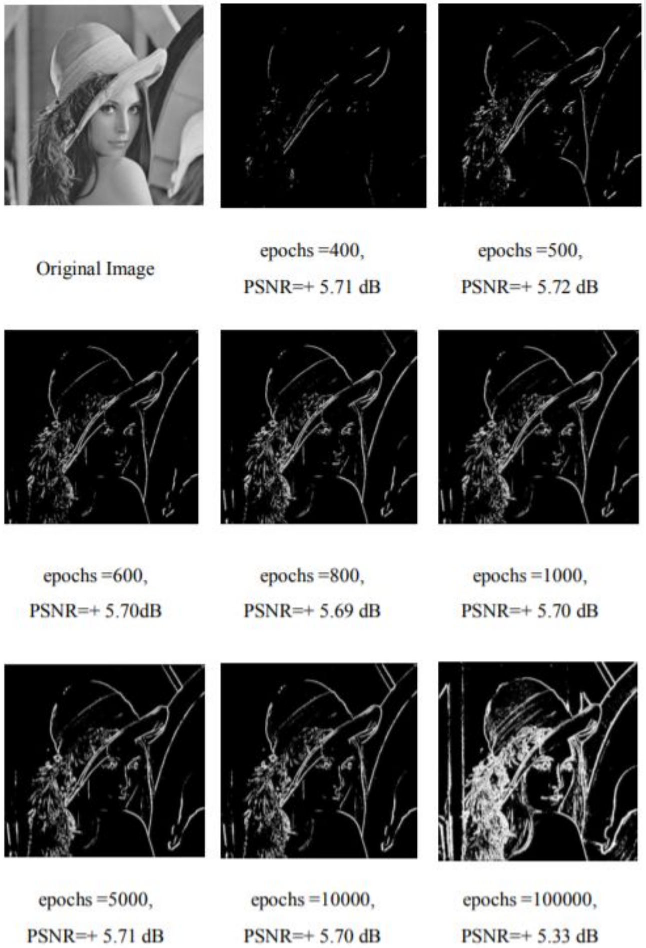


Fig. 9. output and PSNR values for different network statues

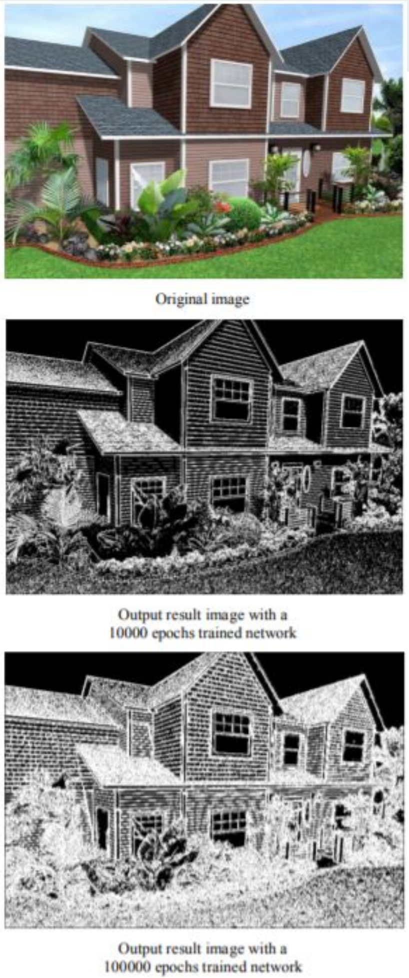


Fig. 13. output result for modern house image

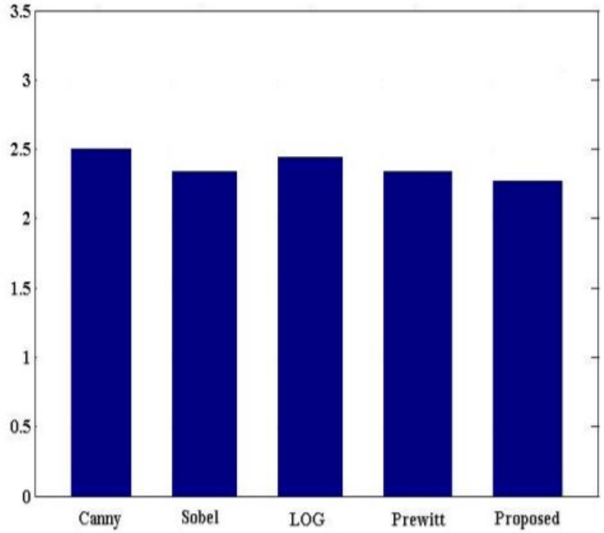


Fig. 12. PSNR values compared.

Holistically-nested Edge Detection (HED) - 2015

Holistically-Nested Edge Detection

Saining Xie
Dept. of CSE and Dept. of CogSci
University of California, San Diego
9500 Gilman Drive, La Jolla, CA 92093
s9xie@eng.ucsd.edu

Zhuowen Tu
Dept. of CogSci and Dept. of CSE
University of California, San Diego
9500 Gilman Drive, La Jolla, CA 92093
ztu@ucsd.edu

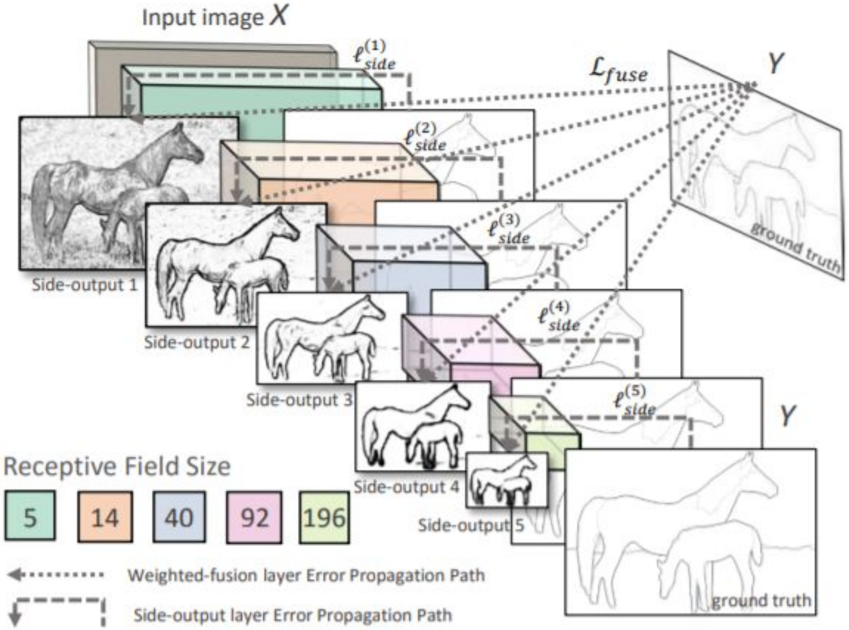


Figure 3. Illustration of our network architecture for edge detection, highlighting the error backpropagation paths. Side-output layers are inserted after convolutional layers. Deep supervision is imposed at each side-output layer, guiding the side-outputs towards edge predictions with the characteristics we desire. The outputs of HED are multi-scale and multi-level, with the side-output-plane size becoming smaller and the receptive field size becoming larger. One weighted-fusion layer is added to automatically learn how to combine outputs from multiple scales. The entire network is trained with multiple error propagation paths (dashed lines).

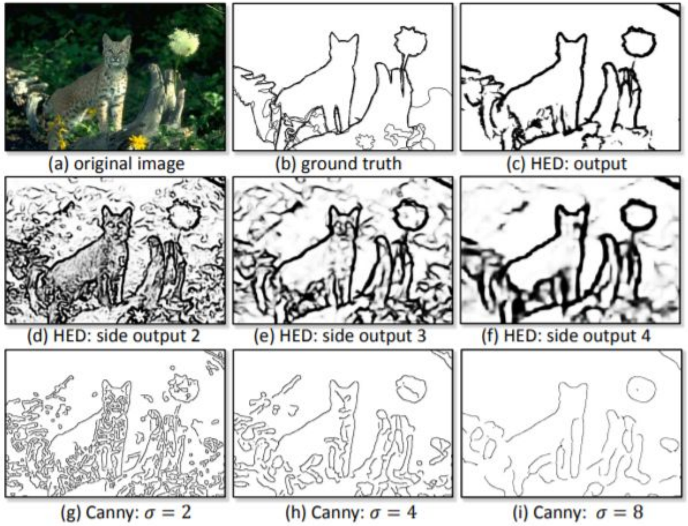


Figure 1. Illustration of the proposed HED algorithm. In the first row: (a) shows an example test image in the BSD500 dataset [28]; (b) shows its corresponding edges as annotated by human subjects; (c) displays the HED results. In the second row: (d), (e), and (f), respectively, show side edge responses from layers 2, 3, and 4 of our convolutional neural networks. In the third row: (g), (h), and (i), respectively, show edge responses from the Canny detector [4] at the scales $\sigma = 2.0$, $\sigma = 4.0$, and $\sigma = 8.0$. HED shows a clear advantage in consistency over Canny.

- Função de Perda: class-balanced cross-entropy
- # de parâmetros: 14.7M

Table 4. Results on BSDS500. *BSDS300 results,†GPU time

	ODS	OIS	AP	FPS
Human	.80	.80	-	-
Canny	.600	.640	.580	15
Felz-Hutt [9]	.610	.640	.560	10
BEL [5]	.660*	-	-	1/10
gPb-owt-ucm [1]	.726	.757	.696	1/240
Sketch Tokens [24]	.727	.746	.780	1
SCG [31]	.739	.758	.773	1/280
SE-Var [6]	.746	.767	.803	2.5
OEF [13]	.749	.772	.817	-
DeepNets [21]	.738	.759	.758	1/5†
N4-Fields [10]	.753	.769	.784	1/6†
DeepEdge [2]	.753	.772	.807	$1/10^3$†
CSCNN [19]	.756	.775	.798	-
DeepContour [34]	.756	.773	.797	1/30†
HED (ours)	**.782**	**.804**	**.833**	2.5†, 1/12

Table 5. Results on the NYUD dataset [35] †GPU time

	ODS	OIS	AP	FPS
gPb-ucm	.632	.661	.562	1/360
Silberman [35]	.658	.661	-	<1/360
gPb+NG[11]	.687	.716	.629	1/375
SE[6]	.685	.699	.679	5
SE+NG+[12]	.710	.723	.738	1/15
HED-RGB	.720	.734	.734	2.5†
HED-HHA	.682	.695	.702	2.5†
HED-RGB-HHA	**.746**	**.761**	**.786**	1†

Convolutional Encoder-Decoder Network (CEDN) - 2016

Object Contour Detection with a Fully Convolutional Encoder-Decoder Network

Jimei Yang
Adobe Research
jimyang@adobe.com

Brian Price
Adobe Research
bprice@adobe.com

Scott Cohen
Adobe Research
scohen@adobe.com

Honglak Lee
University of Michigan, Ann Arbor
honglak@umich.edu

Ming-Hsuan Yang
UC Merced
mhyang@ucmerced.edu

Table 1. Decoder network setup.

name	deconv6	deconv5	deconv4	
setup	conv	unpool-conv	unpool-conv	
kernel	$1\times1\times512$	$5\times5\times512$	$5\times5\times256$	
acti	relu	relu	relu	
name	deconv3	deconv2	deconv1	pred
setup	unpool-conv	unpool-conv	unpool-conv	conv
kernel	$5\times5\times128$	$5\times5\times64$	$5\times5\times32$	$5\times5\times1$
activation	relu	relu	relu	sigmoid

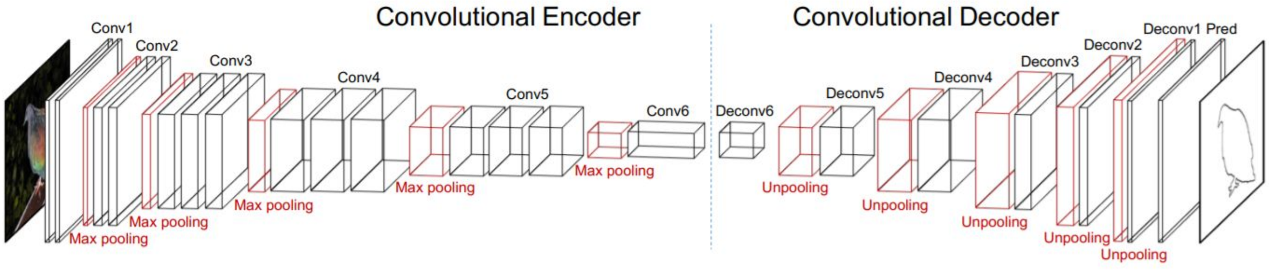
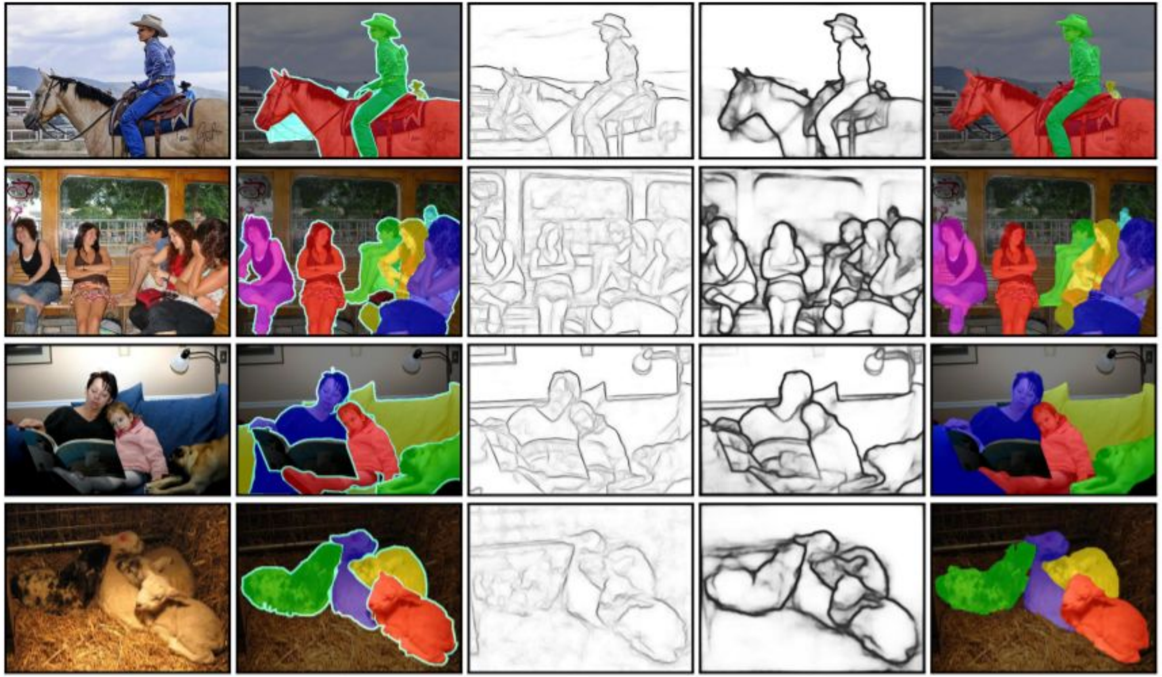


Figure 2. Architecture of the proposed fully convolutional encoder-decoder network.

- Função de Perda: pixel-wise logistic
- # de parâmetros: ?



(a) (b) (c) (d) (e)

Figure 5. Example results on PASCAL VOC val2012. In each row from left to right we present (a) input image, (b) ground truth annotation, (c) edge detection [12], (d) our object contour detection and (e) our best object proposals.

Table 2. Contour detection results on BSDS500.

	ODS	OIS	AP
Human	.80	.80	-
SCG [4]	.739	.758	.773
SE [12]	.746	.767	.803
DeepEdge [6]	.753	.772	.807
DeepContour [43]	.756	.773	.797
HED [47]	.782	.804	.833
HED-new [1]	**.788**	**.808**	**.840**
CEDN-pretrain	.610	.635	.580
CEDN	**.788**	.804	.821

Richer Convolutional Features (RCF) - 2019

Richer Convolutional Features for Edge Detection

Yun Liu, Ming-Ming Cheng, Xiaowei Hu, Jia-Wang Bian, Le Zhang, Xiang Bai, and Jinhui Tang

- Função de Perda: class-balanced cross-entropy
- # de parâmetros: ?

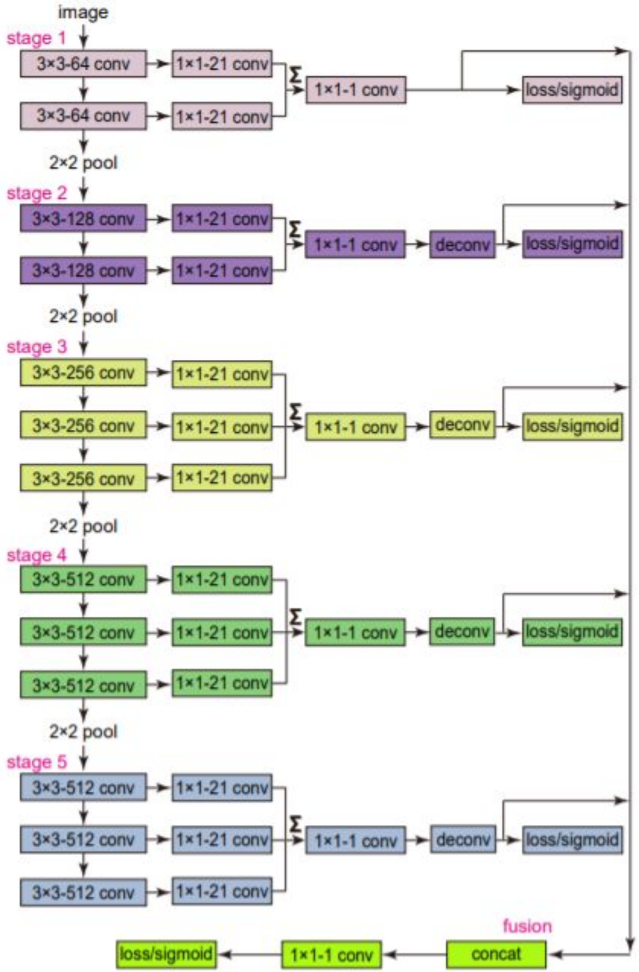


Fig. 2: Our RCF network architecture. The input is an image with arbitrary sizes, and our network outputs an edge possibility map in the same size.

Fig. 6: Some examples of RCF. **Top two rows**: BSDS500 [9]. **Bottom two rows**: NYUD [30]. From **Left** to **Right**: origin image, ground truth, RCF edge map, RCF UCM map.

Method	ODS	OIS	FPS
Canny [25]	0.611	0.676	28
Pb [8]	0.672	0.695	-
SE [10]	0.743	0.763	2.5
OEF [35]	0.746	0.770	2/3
DeepContour [15]	0.757	0.776	1/30†
DeepEdge [13]	0.753	0.772	1/1000†
HFL [36]	0.767	0.788	5/6†
N^4-Fields [14]	0.753	0.769	1/6†
HED [16]	0.788	0.808	**30**†
RDS [26]	0.792	0.810	**30**†
CEDN [32]	0.788	0.804	10†
MIL+G-DSN+VOC+MS +NCuts [33]	0.813	0.831	1†
CASENet [29]	0.767	0.784	18†
AMH-ResNet50 [28]	0.798	0.829	-
CED-VGG16 [27]	0.794	0.811	-
CED-ResNet50+VOC+MS [27]	0.817	0.834	-
RCF	0.806	0.823	**30**†
RCF-MS	0.811	0.830	8†
RCF-ResNet50	0.808	0.825	20†
RCF-ResNet50-MS	0.814	0.833	5.4†
RCF-ResNet101	0.812	0.829	12.2†
RCF-ResNet101-MS	**0.819**	**0.836**	3.6†

Fig. 4: The comparison with some competitors on the BSDS500 [9] dataset. † means GPU time.

Dense Extreme Inception Network (DexiNed) - 2020

Dense Extreme Inception Network: Towards a Robust CNN Model for Edge Detection

Xavier Soria[†] Edgar Riba[†] Angel Sappa[†,‡]
† Computer Vision Center - Universitat Autonoma de Barcelona, Barcelona, Spain
‡ Escuela Superior Politécnica del Litoral, Guayaquil, Ecuador

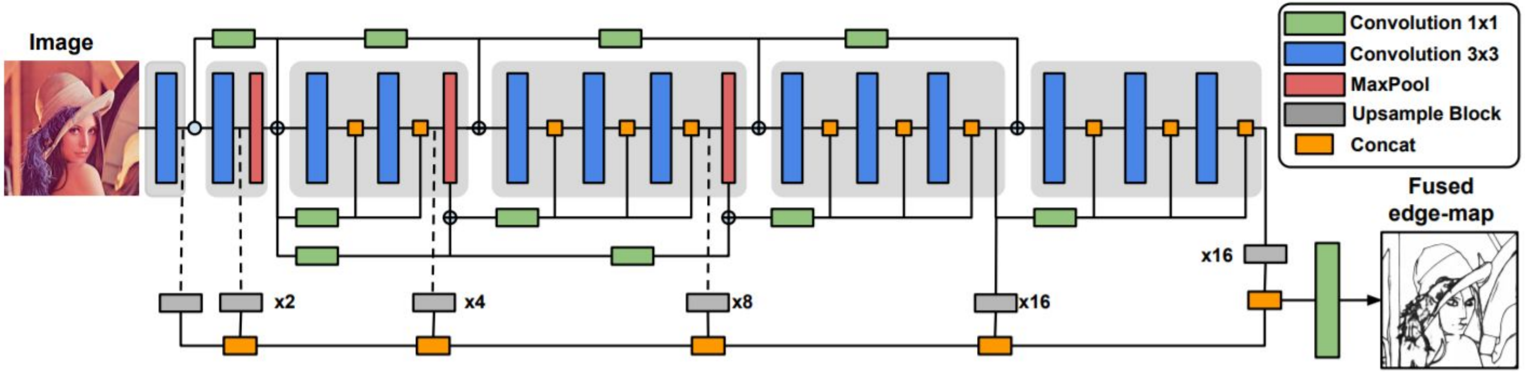


Figure 3. Proposed architecture: Dense Extreme Inception Network, consists of an encoder composed by six main blocks (showed in light gray). The main blocks are connected between them through 1x1 convolutional blocks. Each of the main blocks is composed by sub-blocks that are densely interconnected by the output of the previous main block. The output from each of the main blocks is fed to an upsampling block that produces an intermediate edge-map in order to build a Scale Space Volume, which is used to compose a final fused edge-map.

Função de Perda: class-balanced cross-entropy

de parâmetros: ?

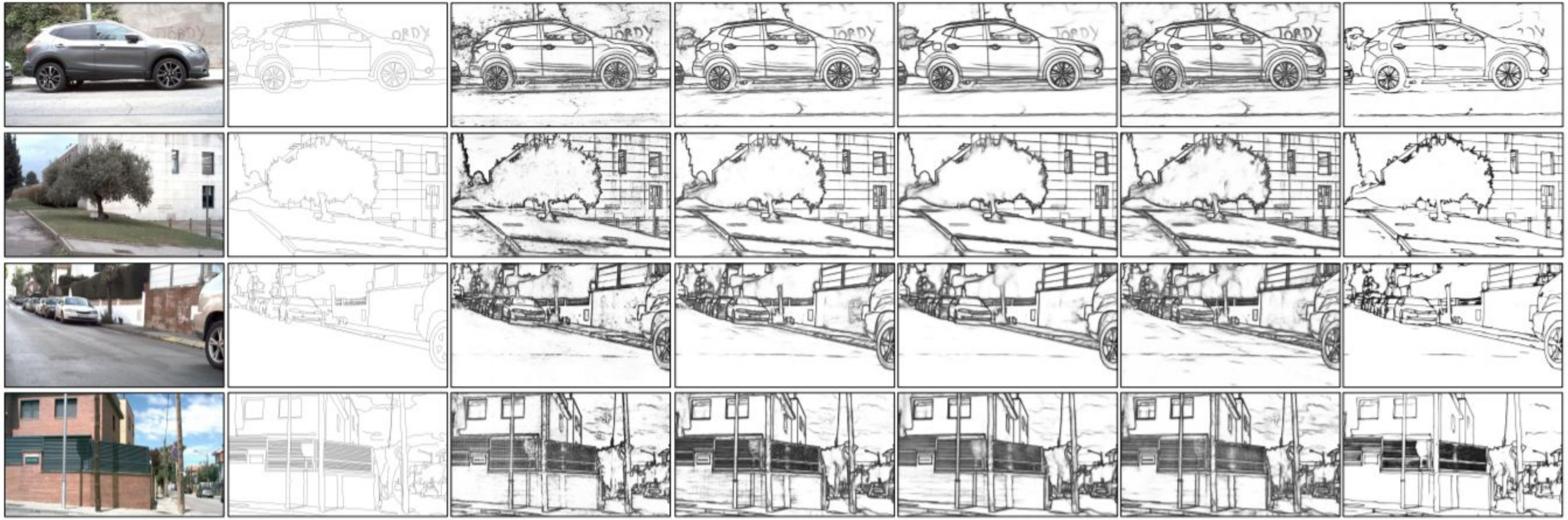


Image *GT* *CED [34]* *HED [36]* *RCF [20]* *BDCN [14]* *DexiNed*

Figure 7. Results from different edge detection algorithms trained and evaluated in BIPED dataset.

Resultados da DexiNed

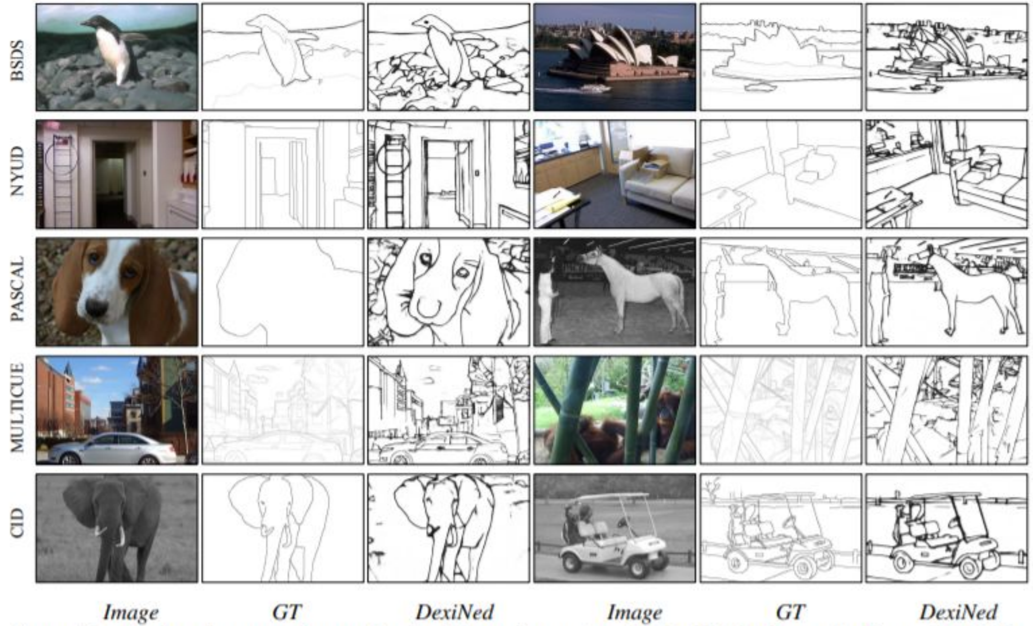


Figure 8. Results from the proposed approach using different datasets (note that DexiNed has been trained just with BIPED).

Outputs	ODS	OIS	AP
Output 1 ($\hat{y}_1$)	.741	.760	.162
Output 2 ($\hat{y}_2$)	.766	.803	.817
Output 3 ($\hat{y}_3$)	.828	.846	.838
Output 4 ($\hat{y}_4$)	.844	.858	.843
Output 5 ($\hat{y}_5$)	.841	.8530	.776
Output 6 ($\hat{y}_6$)	.842	.852	.805
Fused ($\hat{y}_f$)	.857	.861	.805
Averaged	**.859**	**.865**	**.905**

(a)

Methods	ODS	OIS	AP
SED[2]	.717	.731	.756
HED[36]	.829	.847	.869
CED[34]	.795	.815	.830
RCF[19]	.843	.859	.882
BDCN[14]	.839	.854	.887
DexiNed-f	.857	.861	.805
DexiNed-a	**.859**	**.867**	**.905**

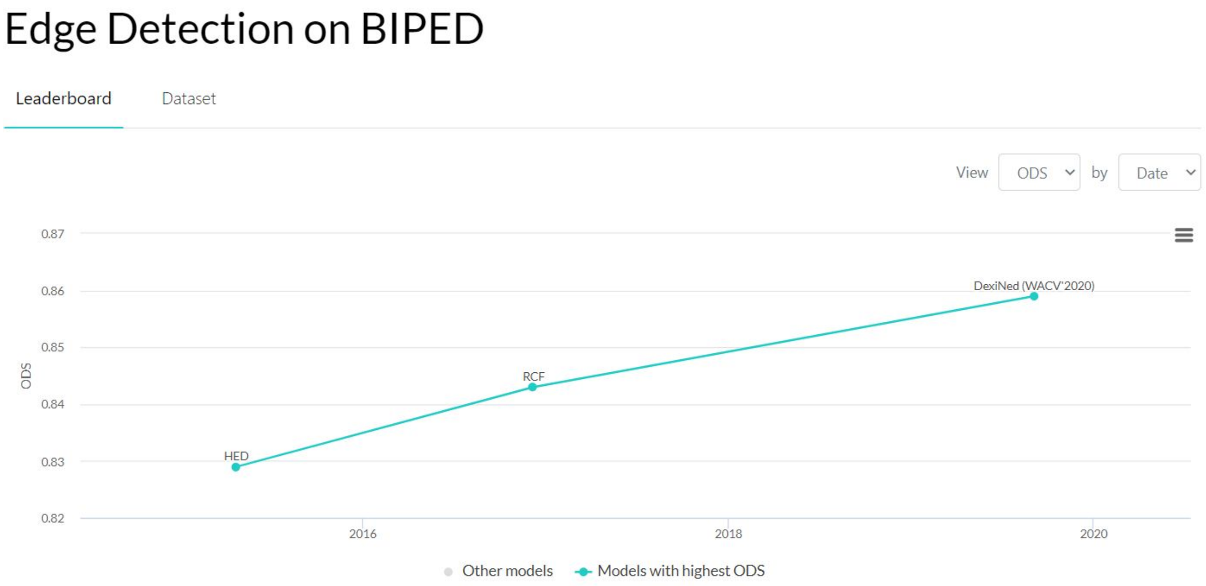
(b)

Table 1. (a) Quantitative evaluation of the 8 predictions of DexiNed on BIPED test dataset. (b) Comparisons between the state-of-the-art methods trained and evaluated with BIPED.

Dataset BIPED

- 250 imagens bases 1280x720x3 no total
- 250 imagens com detecção de arestas anotadas manualmente

https://paperswithcode.com/sota/edge-detection-on-biped-1



Pacotes e o opencv-python

In [1]:
```
import os
import numpy as np
import matplotlib.pyplot as plt
import cv2 as cv
```

In [2]:
```
train_path_img = "biped/BIPEDv2/BIPEDv2/BIPED/edges/imgs/train/rgbr/real/"
train_path_edges = "biped/BIPEDv2/BIPEDv2/BIPED/edges/edge_maps/train/rgbr/real/"
train_filenames = [name.strip(".jpg") for name in os.listdir(train_path_img)]

test_path_img = "biped/BIPEDv2/BIPEDv2/BIPED/edges/imgs/test/rgbr/"
test_path_edges = "biped/BIPEDv2/BIPEDv2/BIPED/edges/edge_maps/test/rgbr/"
test_filenames = [name.strip(".jpg") for name in os.listdir(test_path_img)]

X_train = [cv.imread(train_path_img+path+".jpg") for path in train_filenames]
y_train = [cv.imread(train_path_edges+path+".png") for path in train_filenames]
X_test = [cv.imread(test_path_img+path+".jpg") for path in test_filenames]
y_test = [cv.imread(test_path_edges+path+".png") for path in test_filenames]

y_train = [y[:,:,0] for y in y_train]
y_test = [y[:,:,0] for y in y_test]

print("Train dataset size: " + str(len(X_train)) + '/' + str(len(X_train)+len(X_test)) + '.')
print("Test dataset size: " + str(len(X_test)) + '/' + str(len(X_train)+len(X_test)) + '.')
print("X image shape: ",X_train[0].shape)
print("y image shape: ",y_train[0].shape)
```

```
Train dataset size: 200/250.
Test dataset size: 50/250.
X image shape:  (720, 1280, 3)
y image shape:  (720, 1280)
```

Como usar o *canny*

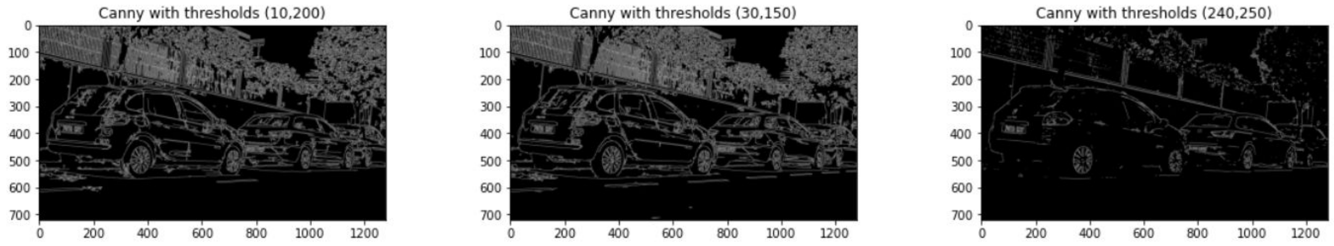
In [4]:

```
def toRGB(image):
    return np.concatenate(3*[image.reshape((image.shape[0],image.shape[1],1))], axis = 2)
```

In [9]:

```
# using the Canny edge detector
wide = cv.Canny(X_train[0], 10, 200)
mid = cv.Canny(X_train[0], 30, 150)
tight = cv.Canny(X_train[0], 240, 250)

# show the output Canny edge maps
fig, axs = plt.subplots(1, 3, figsize=(20,3))
axs[0].imshow(toRGB(wide), cmap = 'Greys')
axs[0].set_title('Canny with thresholds ('+str(10)+','+str(200)+')')
axs[1].imshow(toRGB(mid), cmap = 'Greys')
axs[1].set_title('Canny with thresholds ('+str(30)+','+str(150)+')')
axs[2].imshow(toRGB(tight), cmap = 'Greys')
axs[2].set_title('Canny with thresholds ('+str(240)+','+str(250)+')')
plt.show()
```



Carregando o HED

```
In [9]: # One of the layers dont belong to basic CV package, so we need to bind it
class CropLayer(object):
    def __init__(self, params, blobs):
        self.xstart = 0
        self.xend = 0
        self.ystart = 0
        self.yend = 0

    # Our layer receives two inputs. We need to crop the first input blob
    # to match a shape of the second one (keeping batch size and number of channels)
    def getMemoryShapes(self, inputs):
        inputShape, targetShape = inputs[0], inputs[1]
        batchSize, numChannels = inputShape[0], inputShape[1]
        height, width = targetShape[2], targetShape[3]

        self.ystart = (inputShape[2] - targetShape[2]) // 2
        self.xstart = (inputShape[3] - targetShape[3]) // 2
        self.yend = self.ystart + height
        self.xend = self.xstart + width

        return [[batchSize, numChannels, height, width]]

    def forward(self, inputs):
        return [inputs[0][:,:,self.ystart:self.yend,self.xstart:self.xend]]
```

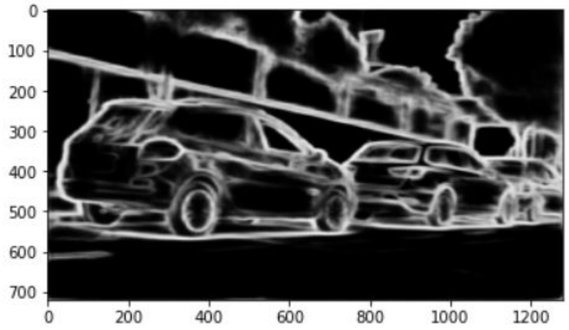
```
In [10]: # Binding it
cv.dnn_registerLayer('Crop', CropLayer)
```

```
In [11]: # Load the HED model. Download the deploy and the pretrained object
net = cv.dnn.readNet('deploy.prototxt.txt', 'hed_pretrained_bsds.caffemodel')
```

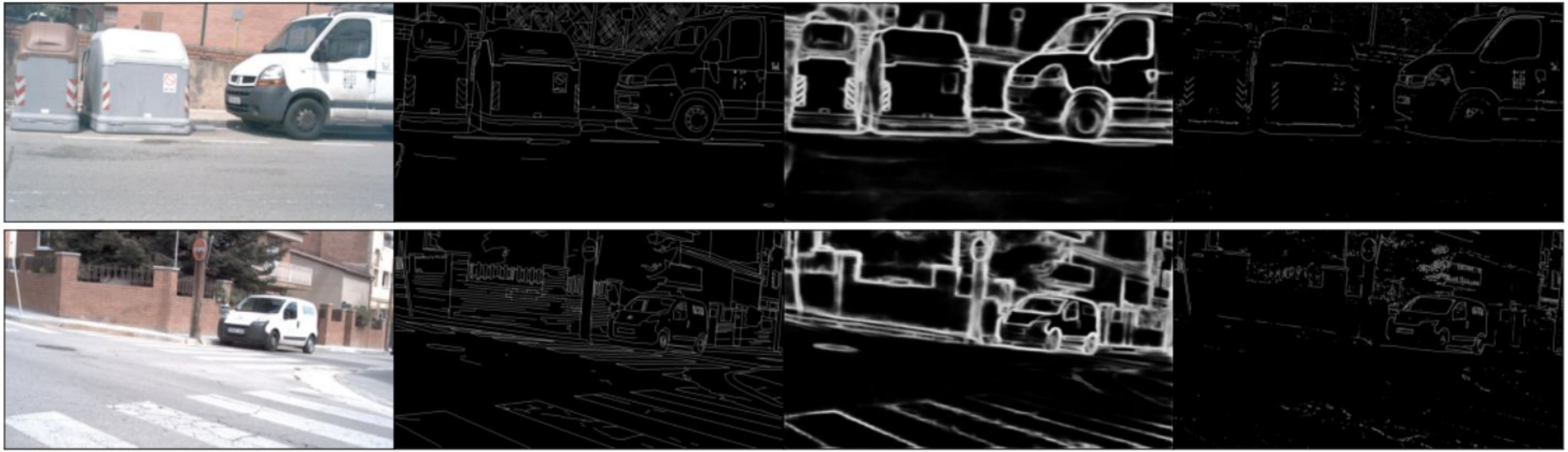
Gerando as predições

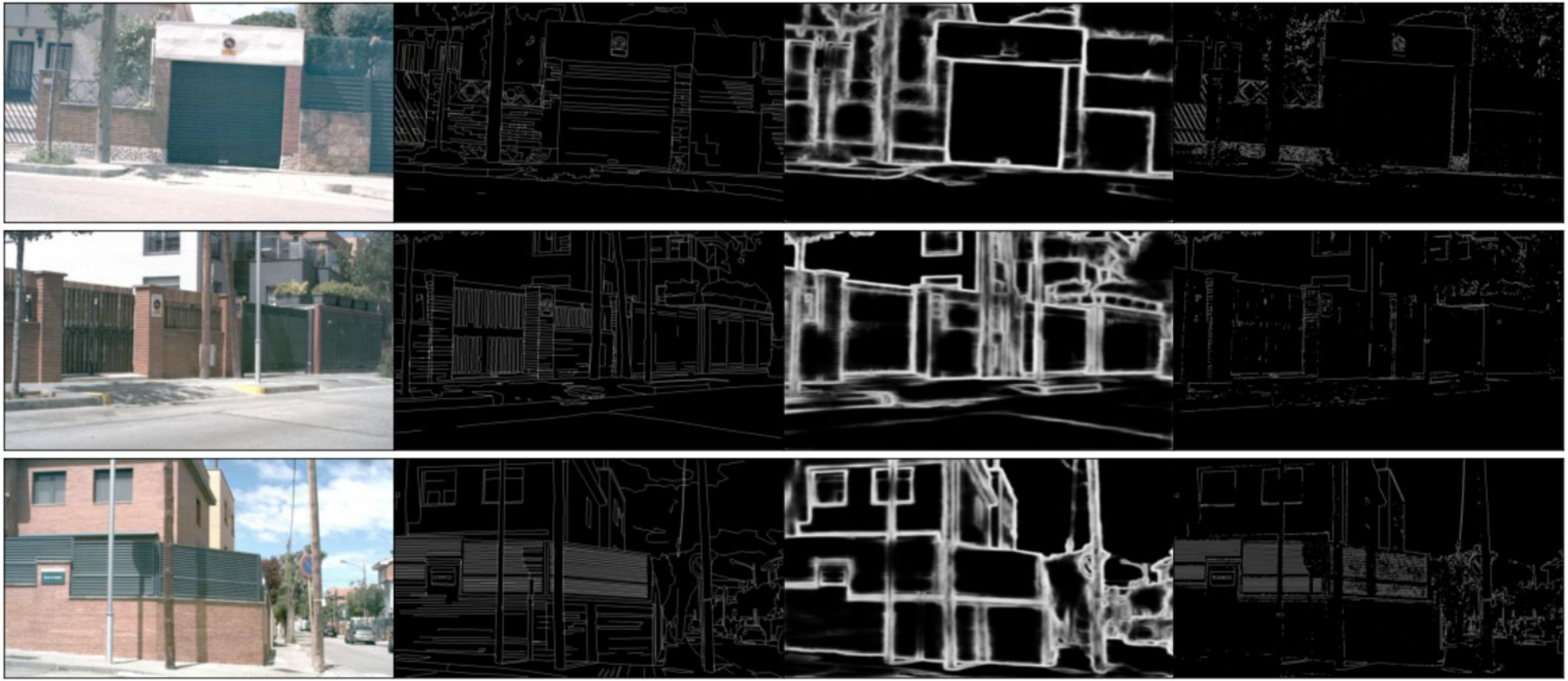
In [11]:
```
# Function of input of the net
def input_CNN(image):
    inp = cv.dnn.blobFromImage(image, scalefactor=1.0, size=(500, 500),
                               mean=(104.00698793, 116.66876762, 122.67891434),
                               swapRB=False, crop=False)
    net.setInput(inp)
    outG = net.forward()
    outG = outG[0, 0]
    outG = cv.resize(outG, (image.shape[1], image.shape[0]))
    outG = 255 * outG
    outG = outG.astype(np.uint8)
    return outG
```

In [12]:
```
# Test with an image
y1 = input_CNN(X_train[0])
plt.imshow(toRGB(y1),cmap='Greys');
```



Alguns resultados (imagem/anotação/HED/canny)





Referências

- Emmert-Streib, Frank & Yang, Zhen & Feng, Han & Tripathi, Shailesh & Dehmer, Matthias. (2020). An Introductory Review of Deep Learning for Prediction Models With Big Data. Frontiers in Artificial Intelligence. 3. 4. 10.3389/frai.2020.00004.
- El-Sayed, Mohamed & Estaitia, Yarub & Khafagy, Mohamed. (2013). Edge Detection Using Convolutional Neural Network. International Journal of Advanced Computer Science and Applications. 4. 11-17. 10.14569/IJACSA.2013.041003;
- Saining Xie, Zhuowen Tu: Holistically-Nested Edge Detection. CoRR abs/1504.06375 (2015);
- Jimei Yang, Brian L. Price, Scott Cohen, Honglak Lee, Ming-Hsuan Yang: Object Contour Detection with a Fully Convolutional Encoder-Decoder Network. CoRR abs/1603.04530 (2016);
- Y. Liu et al., "Richer Convolutional Features for Edge Detection," in IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 41, no. 8, pp. 1939-1946, 1 Aug. 2019, doi: 10.1109/TPAMI.2018.2878849;

Referências

- Xavier Soria, Edgar Riba, Angel Domingo Sappa: Dense Extreme Inception Network: Towards a Robust CNN Model for Edge Detection. CoRR abs/1909.01955 (2019);
- Deep Learning based Edge Detection in OpenCV: https://cv-tricks.com/opencv-dnn/edge-detection-hed/ ;