Holistically-Nested Edge Detection with OpenCV and Deep Learning

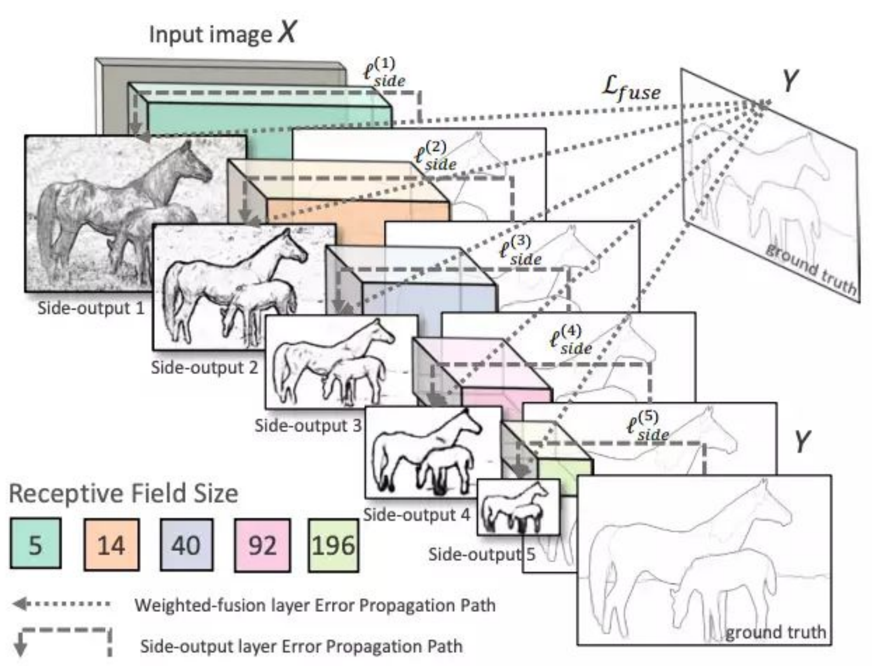
Project Overview:

In this project, we apply Holistically-Nested Edge Detection (HED) with OpenCV and Deep Learning on images and videos.

1. **Explanation:**

    a. **Holistically-Nested Edge Detection (HED):**

    - This edge detection algorithm performs image-to-image prediction by means of a deep learning model that leverages fully convolutional neural networks and deeply-supervised nets.
    - HED is a learning-based end-to-end edge detection system that uses a trimmed VGG-like convolutional neural network for an image-to-image prediction task.



- HED method is not only more accurate than other deep learning-based methods but also much faster than them too.

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

b. **OpenCV:**

- OpenCV has integrated a deep learning-based edge detection technique in its new fancy DNN module.
- **Version:** OpenCV version 3.4.3 or higher

2. **Project Structure:**

```
hed_using_opencv_dnn
│
├──hed_model
│   ├──deploy.prototxt
│   └──hed_pretrained_bsds.caffemodel
│
├──images
│    ├──portrait_dinesh.jpg
│    ├──portrait_person_1.jpg
│    └──portrait_person_2.jpg
│
├──detect_edges_image.py
└──detect_edges_video.py
```

3. Source Code Explanation:

Code Snipper #1:

```python
# import the necessary packages
import argparse
import cv2
import os
import time

# construct the argument parser and parse the arguments
ap = argparse.ArgumentParser()
ap.add_argument("-d", "--edge-detector", type=str, required=True,
    help="path to OpenCV's deep learning edge detector")
ap.add_argument("-i", "--image", type=str, required=True,
    help="path to input image")
args = vars(ap.parse_args())
```

Explanation:

1. We'll be using **argparse** to parse command-line arguments.
2. OpenCV functions and methods are accessed through the **cv2** import.
3. Our **os** import will allow us to build file paths regardless of the operating system.
4. **Time** allows calculating the processing time for Canny & HED.

This script requires two command-line arguments:

- **--edge-detector:** The path to OpenCV's deep learning edge detector. The path contains two Caffe files that will be used to initialize our model later.
- **--image:** The path to the input image for testing. Like I said previously — I've provided a few images in the "Downloads", but you should try the script on your own images as well.

Code Snipper #2:

```python
class CropLayer(object):
    def __init__(self, params, blobs):
        # initialize our starting and ending (x, y)-coordinates of
        # the crop
        self.startX = 0
        self.startY = 0
        self.endX = 0
        self.endY = 0
```

Explanation: In the constructor of this class, we store the **starting and ending (x,y)-coordinates** of where the crop will start and end, respectively.

Code Snipper #3:

```
def getMemoryShapes(self, inputs):
    # the crop layer will receive two inputs -- we need to crop
    # the first input blob to match the shape of the second one,
    # keeping the batch size and number of channels
    (inputShape, targetShape) = (inputs[0], inputs[1])
    (batchSize, numChannels) = (inputShape[0], inputShape[1])
    (H, W) = (targetShape[2], targetShape[3])

    # compute the starting and ending crop coordinates
    self.startX = int((inputShape[3] - targetShape[3]) / 2)
    self.startY = int((inputShape[2] - targetShape[2]) / 2)
    self.endX = self.startX + W
    self.endY = self.startY + H

    # return the shape of the volume (we'll perform the actual
    # crop during the forward pass
    return [[batchSize, numChannels, H, W]]
```

Explanation:

This method is responsible for computing the volume size of the inputs and returns the **shape of the volume** to the calling function.

Code Snippet #4:

```
def forward(self, inputs):
    # use the derived (x, y)-coordinates to perform the crop
    return [inputs[0][:, :, self.startY:self.endY,
            self.startX:self.endX]]
```

Explanation:

This function is responsible for performing the **crop** during the **forward pass** (i.e., inference/edge prediction) of the network.

Code Snipper #5:

```
# load our serialized edge detector from disk
print("[INFO] loading edge detector...")
protoPath = os.path.sep.join([args["edge_detector"],
    "deploy.prototxt"])
modelPath = os.path.sep.join([args["edge_detector"],
    "hed_pretrained_bsds.caffemodel"])
net = cv2.dnn.readNetFromCaffe(protoPath, modelPath)

# register our new layer with the model
cv2.dnn_registerLayer("Crop", CropLayer)
```

Explanation:

Here, we load our **HED model** from disk and register **CropLayer** with the net.

Both the **protoPath** and **modelPath** are used to load and initialize our **Caffe model.**

Code Snippet #6:

```
# convert the image to grayscale, blur it, and perform Canny
# edge detection
# Canny Start time
startCanny = time.time()

print("[INFO] performing Canny edge detection...")
gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
blurred = cv2.GaussianBlur(gray, (5, 5), 0)
canny = cv2.Canny(blurred, 30, 150)

#Canny End Time
endCanny = time.time()

secondsCanny = endCanny - startCanny
print("Time taken for Canny: "+ str(secondsCanny) + " seconds.")
```

Explanation:

Our original image is loaded and spatial dimensions (width and height) are extracted.

We also compute the **Canny edge map** so we can **compare our edge detection results to HED**. Processing time is calculated using the start and end variables.

Code Snippet #7:

```
# HED Start time
startHED = time.time()

# construct a blob out of the input image for the Holistically-Nested
# Edge Detector
blob = cv2.dnn.blobFromImage(image, scalefactor=1.0, size=(W, H),
    mean=(104.00698793, 116.66876762, 122.67891434),
    swapRB=False, crop=False)

# set the blob as the input to the network and perform a forward pass
# to compute the edges
print("[INFO] performing holistically-nested edge detection...")
net.setInput(blob)
hed = net.forward()
hed = cv2.resize(hed[0, 0], (W, H))
hed = (255 * hed).astype("uint8")

#HED End Time
endHED = time.time()

secondsHED = endHED - startHED
print("Time taken for HED: " + str(secondsHED) + " seconds.")
# show the output edge detection results for Canny and
# Holistically-Nested Edge Detection
cv2.imshow("Input", image)
cv2.imshow("Canny", canny)
cv2.imshow("HED", hed)
cv2.waitKey(0)
```

Explanation:

To apply Holistically-Nested Edge Detection (HED) with OpenCV and deep learning, we:

a. Construct a **blob** from our image.
b. Pass the **blob** through the **HED net**, obtaining the **hed output**.
c. **Resize** the output to our **original image dimensions**.
d. Scale our image pixels back to the range [0, 255] and ensure the type is "uint8".
e. Processing time is calculated using the start and end variables.

Finally, we'll display:

a. The original input image
b. The Canny edge detection image
c. Our Holistically-Nested Edge detection result.

4. **Output:**

i. **Image #1:**

i. Input Image #1:

ii. Canny Image #1:

iii. HED Image #1:

iv. Processing Time:

```
C:\Users\kandi\Desktop\CDI\Projects\Edge Detection Project\hed_using_opencv_dnn>
s/portrait_person_1.jpg
[INFO] loading edge detector...
[INFO] performing Canny edge detection...
Time taken for Canny: 0.00697636604309082 seconds.
[INFO] performing holistically-nested edge detection...
Time taken for HED: 12.45242691040039 seconds.
```

Time Taken for:

Canny Edge Detection: **0.00697 seconds**

HED Edge Detection: **12.452 seconds**

c. **Image #2:**

i. Input Image #2:

ii. Canny Image #2:

iii. HED Image #2:

iv. Processing Time:

```
C:\Users\kandi\Desktop\CDI\Projects\Edge Detection Project\hed_using_opencv_dnn>
s/portrait_dinesh.jpg
[INFO] loading edge detector...
[INFO] performing Canny edge detection...
Time taken for Canny: 0.00797581672668457 seconds.
[INFO] performing holistically-nested edge detection...
Time taken for HED: 17.12980556488037 seconds.
```

Time Taken for:

Canny Edge Detection: **0.00797 seconds**

HED Edge Detection: **17.129 seconds**