Projekts:

Valsts pētījumu programmas SOPHIS Projekts Nr.4. „Tehnoloģijas drošai un uzticamai gudrajai pilsētai”

Neironu tīkls kustīgu objektu skaitīšanai

Tehnoloģijas lietošanas apraksts

Autori:

Roberts Kadiķis, Kaspars Sudars

2017.g.

SATURS

Ievads

Šis dokuments apraksta tehnoloģiju un atbilstošo programmatūru kustīgu objektu skaitīšanai video, kas tika izstrādāta Valsts pētījumu programmas SOPHIS 4. projekta GUDPILS ietvaros. Izmantojot izstrādāto programmatūru, tā tika pielietota auto skaitīšanai, kas šķērso video atzīmētu virtuālu līniju (kā tas redzams Att. 1). Izstrādātā tehnoloģija balstīta uz mākslīgā neironu tīkla apmācīšanu un izmantošanu. Tālāk dokumentā aprakstīti rīcības soļi veiksmīgai neironu tīklu pielietošanai kustīgu objektu skaitīšanai. Galvenās darbības ir sekojošas: (1) video ieguve, (2) datu marķēšana, (3) neironu tīkla apmācība un (4) sistēmas testēšana un lietošana.

1. Video datu ieguve

Šī etapa ietvaros tiek nodrošināta video ieguve, kuros redzami skaitāmie objekti (skat. Att. 1). Šis etaps tipiski saistās ar IP video kameras uzstādīšanu un programmatūras nodrošinājumu, kas serverim ļauj piekļūt IP kameru video plūsmām, apstrādāt tās un saglabāt tās serverim pieejamā datu glabātuvē. Uzdevuma veikšanai rekomendējam izmantot Ubuntu (skat. https://www.ubuntu.com/) serveri, kurā instalēta OpenCV bibliotēka (https://opencv.org/). Projekta ietvaros ir lietota programma openRTSP, kas vāc video datus tālākai to apstrādei.

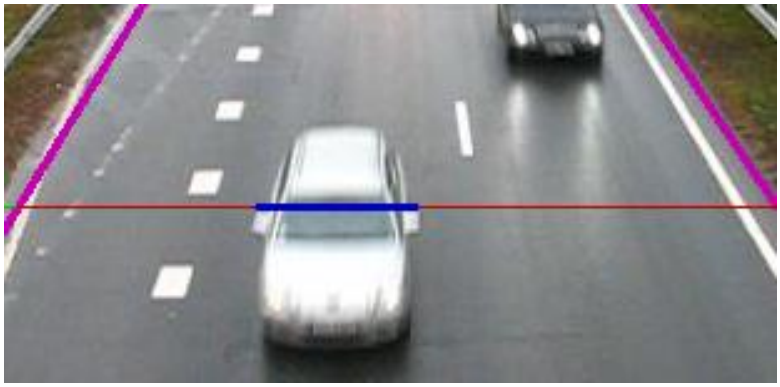
Datu ievākšanas programmas lietošana, palaižot komandu ar attiecīgajiem parametriem Ubuntu terminālī:

```
>> openRTSP -D 1 -c -B 10000000 -b 10000000 -q -Q -F
hdd/dati_ -d 2880000 -P 120 -t
rtsp://192.168.1.1:554/live1.sdp
```

Programma vāks datus no IP kameras ar adresi 192.168.1.1. Tas tiks darīts 2880000 sekundes, saglabājot 120 skundes garas video datnes mapē hdd/dati.

2. Datu marķēšana

Lai veiksmīgi darbinātu neironu tīklu apmācības programmu, nepieciešams liels apjoms marķētu video datu. Ar marķētiem datiem saprotam, ka katram video kadram ir piekārtota nepieciešamā neironu tīkla atbilde. Ja šādas atbildes ir pieejamas, tad tādā gadījumā saka, ka video dati ir marķēti. Realitāte ir tāda, ka bieži vien vajadzīgo atbildi ieejas datiem spēj piekārtot tikai cilvēks, tāpēc GUDPILS projektā ir izstrādāta programmatūra manuālai (jeb cilvēka vadītai) video datu marķēšanai, kur katrai reizei, kad auto/objekts pamet iestatītu līniju, marķētājs piekārto atbilstošu marķējumu, piemēram, nospiežot atbilstošo izvēles taustiņu, piemēram, “a” vai “b” tml.



Att. 1. Video kadrs, kur redzams kā auto šķērso virtuālu līniju

Izstrādātās marķēšanas programmas galvenās darbības ir sekojošas: (1) Vizualizēt attēlu marķētājam, (2) Saglabāt marķētāja izvēlēto atbildi. Marķēšanas procesā tiek iegūta marķētu vektoru kopa, kuru iespējams izmantot neironu tīklu apmācībai.

Marķēšanas programmas palaišanas piemērs:

```
>> python3 sa-labeling.py -vidName
/home/plocins/nextcloud/md001_104.avi
```

Programma video `/home/plocins/nextcloud/md001_104.avi` konvertē rindu laika attēlā (skat. Att. 2), kuru marķētājs tālāk marķē atbilstoši taustiņu nozīmei:

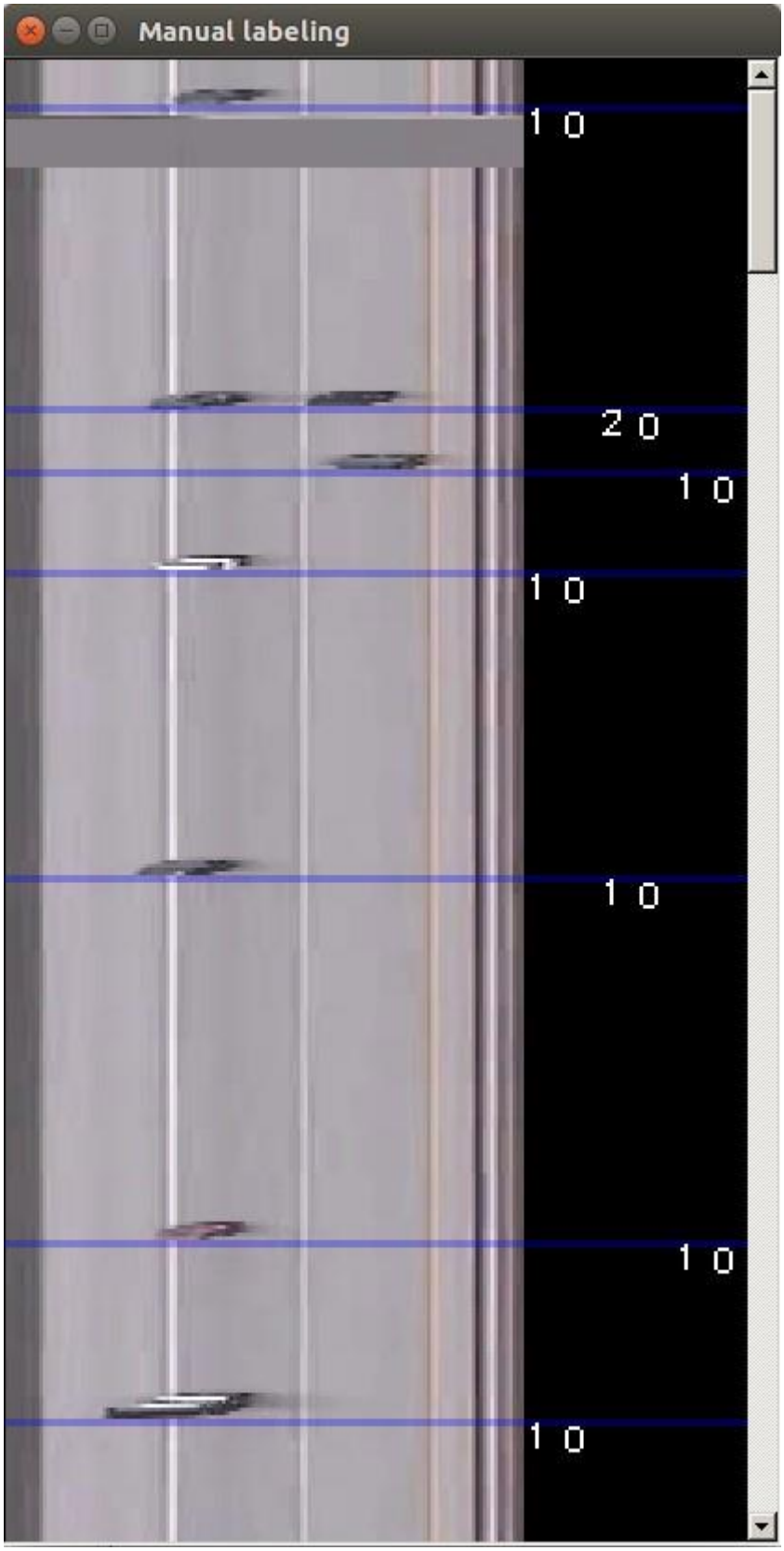
```
e – sākt marķēšanu
space – pārslēgt režīmu uz 2
cipars - pārslēgt režīmu uz 1

Režīms 1:
Pele L: Likt šeit pēdējo nospiesto ciparu
Pele R: augšējo līniju velkam uz kursoru / apakšējo dzēšam

Režīms 2:
Pele L: automātiski +1 marķējuma objekta robežai
Pele R: augšējo izdzēst (marķējums-1) / apakšējo līniju
pievilkt pa vienu kadru

Esc - iziet un saglabāt

[X] - iziet bez saglabāšanas
```



Att. 2. Video datu marķēšana.

3. Neironu tīkla apmācība

Kad ir iegūti apmācības dati, tad ir iespējams veikt neironu tīkla apmācību, kas veiks kustīgu objektu skaitīšanu video. Izstrādātais neironu tīkls šāda uzdevuma veikšanai ir aprakstīts zinātniskajā publikācijā un prezentēts konferencē: R. Kadikis, *Recurrent neural network based virtual detection line*, The 10th International Conference on Machine Vision ICMV 2017.

Projekta ietvaros neironu tīkls tika apmācīts uz Ubuntu servera ar Nvidia K40 GPU karti un 4 stundu video marķētiem datiem. Tālāk apmācīto neironu tīklu var integrēt un lietot objektu skaitīšanas sistēmā. Tehnoloģija tika aprobēta Biznesa augstskolā "Turība".

Neirona tīkla apmācības programmas palaišanas piemērs:

```
>> python3 linedetect-train.py
```

Programmā jādefinē ceļš uz apmācības datiem un ceļš, kur uz servera tiks glabāts apmācītais jeb iegūtais neironu tīkla modelis.

4. Sistēmas testēšana un lietošana

Pēdējā etapā apmācītais neironu tīkls tiek darbināts reālos apstākļos, kādos dati tika iegūti.

Projekta ietvaros tehnoloģija tika pārbaudīta Biznesa Augstskolas "Turība" automašīnu stāvlaukumā, kur tā skatīja iebraucošās un izbraucošās automašīnas, izmantojot pie stāvlaukuma vārtiem uzstādītu video kameru. Tehnoloģija ir novērtēta reālos apstākļos, kam ir saņemts dokumentāls apliecinājums.

Neirona tīkla programmas palaišanas piemērs objektu skaitīšanai:

```
>> python3 linedetect-realtime_vp_ks.py
```

Lai to veiksmīgi darbinātu, tajā jādefinē IP kameras adrese, kur programma var gūt apstrādājamos reālā laika video datus, un ceļš līdz neironu tīkla modelim, kas iegūts sistēmas apmācības procesā.

Pielikums: programmu kodi

sa-labeling.py

```
'''
 A semi-automatic labeling of data for object detection in video
 using virtual detection line.

    Example use:
        python3 sa-labelling_2.py -vidname test_videos/night.avi -labelMode
onTheLine

    First, difine the ending points of the detectin line in the frame using two
    right-clicks. It is important to begini with the left ending point.

    The automatic part of the labeling is done by acquiring spatio-temporal image
from
    a user specified line in the video. Then a background is acquired and subtracted
    from every row (frame) in the spatio-temporal image.
    The manual labeling is done to correct the errors of automatic labeling.

    Press 'q' to close the stop the visualisation of input if enabled (-vizInput).

    Two labeling modes ar implemented:
        1) The label of current frame is equal to the number of objects on detectro
line.
            * Right-click: if closest label line is upwards, moves it to the
              cursor postition.
            * Right-click: if closest line is downwards, deletes it.
            * Left-click: adds a new label line at the cursor position. The label
              is the last number pressed on the keyboard.
        2) The label of current frame is equal to the number of objects that have left
the
        detection line at the current frame.
            * Righ-click: automatically finds the closest edge of the object
              upwards from cursor position. Puts a new label (1) line there, or
              increases existing label by 1.
            * Left-click: if closest label line is downwards, move this line
              one row up.
            * Left-click: if closest label line is upwards, decrease its label
              by 1.

    Use mouse weel to scroll trough the spatio-temporal image.

    Press 'Space' to switch between the labeling modes.

    Press 'Esc' when finished to save the labels and data to hdd.
    (Resulting files are saved in the directory of input video)
```

```
'''

import sys
import cv2
import numpy as np
import argparse
from PyQt5 import QtGui
from PyQt5 import QtCore
from PyQt5 import QtWidgets


# Input arguments
parser = argparse.ArgumentParser()
parser.add_argument('-vidName', '--vidName',
                    required = False, default = 'test_videos/night.avi')
parser.add_argument('-nClasses', '--nClasses',
                    required = False, default = 10, type = int)
parser.add_argument('-vizInput', '--vizInput',
                    required = False, default = False)
parser.add_argument('-labelMode', '--labelMode',
                    required = False, default = 'leftTheLine')
parser.add_argument('-autoMode', '--autoMode',
                    required = False, default = 'subtraction')
parser.add_argument('-subThr', '--subThr',
                    required = False, default = 10, type = int)
args = parser.parse_args()

VIDEO_NAME = args.vidName
N_CLASSES = args.nClasses
VIZ_INPUT = args.vizInput
LABEL_MODE = args.labelMode   # 'leftTheLine' or 'onTheLine'
AUTO_METHOD = args.autoMode   # 'subtraction' or 'mog2'
SUBTRACTION_THR = args.subThr

print("viz", VIZ_INPUT)


print()
print("Preparing the input..")
print()

cap = cv2.VideoCapture(VIDEO_NAME)
frames_count = int( cap.get(cv2.CAP_PROP_FRAME_COUNT) )
frame_height = int( cap.get(cv2.CAP_PROP_FRAME_HEIGHT) )
fps = int( cap.get(cv2.CAP_PROP_FPS) )
if (frames_count == 0):
    print("ERROR: could not determime the frame count!")
    print("Counting frames manualy..")
    while(True):
        ret, frame = cap.read()
```

```
        if not ret:
            break
        frames_count += 1

    print("  frame count", frames_count)
    print()



# Set the position and length of detection line with two mouse clicks
lineSet = False
click1 = False
click2 = False
def set_the_line(event, x, y, flags, param):
    global LINE_R, LINE_C_BEGIN, LINE_C_END, lineSet, click1, click2
    if event == cv2.EVENT_LBUTTONDOWN:

        if (click2 == False and click1 == True):
            LINE_C_END = x
            cv2.line(frame, (LINE_C_BEGIN, LINE_R), (LINE_C_END, LINE_R),
                (255, 0, 0), thickness=1, lineType=8, shift=0)
            click2 = True
            lineSet = True

        if (click1 == False):
            LINE_R = y
            LINE_C_BEGIN = x
            cv2.circle(frame, (LINE_C_BEGIN, LINE_R), radius=1,
                color=(255, 0, 0), thickness=3, lineType=8, shift=0)
            click1 = True

        cv2.imshow("input", frame)



# Create dislpay window for visualisation purposes
cv2.namedWindow("input")
cv2.setMouseCallback("input", set_the_line)


# Progress bar function
def progress_bar(name, val, end_val, bar_length=20):
    percent = float(val) / end_val
    hashes = '#' * int(round(percent * bar_length))
    spaces = ' ' * (bar_length - len(hashes))
    sys.stdout.write("\r" + name + ": [{0}] {1}%".format(hashes + spaces,
        int(round(percent * 100))))
    sys.stdout.flush()
```

```
# Initialize
current_label = 0
matricesDefined = False
backgroundHalfReady = False
background_ready = False
fgbg_mog = cv2.createBackgroundSubtractorMOG2()


# Process frames (create thesholded spatio-temporal image)
i = 0
while i < frames_count-1:
    ret, frame = cap.read()
    if ret == True:

        if (lineSet):

            if (matricesDefined == False):

                # Define the matrices (sizes)
                line_spatio_temporal = np.zeros((frames_count, LINE_C_END - LINE_C_BEGIN))
                spatio_temporal = np.zeros((frames_count, LINE_C_END-LINE_C_BEGIN, 3))
                spatio_temporal_bw = np.zeros((frames_count, LINE_C_END-LINE_C_BEGIN))
                if VIZ_INPUT:
                    vizbuf_diff = np.zeros((frame_height, LINE_C_END-LINE_C_BEGIN))
                    vizbuf_thr = np.zeros((frame_height, LINE_C_END-LINE_C_BEGIN))
                matricesDefined = True

            line_color = frame[LINE_R : LINE_R + 1,
                    LINE_C_BEGIN : LINE_C_END].copy()
            line = cv2.cvtColor(line_color, cv2.COLOR_BGR2GRAY)
            line_spatio_temporal[i,:] = line.copy()


            cv2.line(frame, (LINE_C_BEGIN, LINE_R), (LINE_C_END, LINE_R),
                    (255, 0, 0), thickness=1, lineType=8, shift=0)


        if (backgroundHalfReady and background_ready == False):
            background_ready = True
            background = line.copy()

        if (background_ready == False):
            cv2.imshow("input", frame)
            key = cv2.waitKey(0)
            key_char = chr(key & 255)
```

```
            if key_char == 'q':
                print("The line was not ok!")
                quit()
            elif key_char == 'e':
                i -= 1
                backgroundHalfReady = True


        # subtracting the background
        if (background_ready):

            if AUTO_METHOD == "subtraction":
                diff_image = cv2.absdiff(line,background)
                ret, diff_image_thr = cv2.threshold(diff_image,
                        SUBTRACTION_THR, 255, cv2.THRESH_BINARY)

                # update the background
                if current_label == 0:
                    background = cv2.addWeighted(background, 0.95, line, 0.05,
gamma=0)

            elif AUTO_METHOD == "mog2":
                diff_image = fgbg_mog.apply(line)
                diff_image_thr = diff_image


            # create the spatio-temporal image
            spatio_temporal[i,:] = line_color/255
            spatio_temporal_bw[i,:] = diff_image_thr

            # count the objects:
            if (sum(sum(diff_image_thr))/255 > 0.2 * line.shape[1]):
                current_label = 1
            else:
                current_label = 0

            # visualisation
            if VIZ_INPUT:
                # subtrraction buffer
                vizbuf_diff[0:frame_height-1,:] = vizbuf_diff[1:,:]
                vizbuf_diff[frame_height-1,:] = diff_image / 255
                vizbuf_thr[0:frame_height-1,:] = vizbuf_thr[1:,:]
                vizbuf_thr[frame_height-1,:] = diff_image_thr

                input_vizualisation = np.zeros((frame.shape[0],
                        frame.shape[1] + vizbuf_diff.shape[1]*2, 3))
                input_vizualisation[0:frame.shape[0], 0:frame.shape[1], :] = frame/255

                buffer_1 = frame.shape[1] + vizbuf_diff.shape[1]
                input_vizualisation[:, frame.shape[1]:buffer_1, 1] = vizbuf_diff
```

```
            input_vizualisation[:, buffer_1:buffer_1 + vizbuf_diff.shape[1], 1] =
vizbuf_thr

            cv2.imshow("input", input_vizualisation)
            keyr = cv2.waitKey(1)
            key_charr = chr(keyr & 255)
            if key_charr == 'q':
               break
         else:
            cv2.destroyWindow("input")

         progress_bar("Background subtraction:", i, frames_count, bar_length=20)
   i += 1
cv2.destroyWindow("input")



print()
print("Processing spatio-temporal image..")

# Morphological processing
spatio_temporal_bw_raw = spatio_temporal_bw.copy()
kernel = np.ones((3,3), np.uint8)
kernel2 = np.ones((5,5), np.uint8)
spatio_temporal_bw = spatio_temporal_bw.astype(np.uint8)
spatio_temporal_bw = cv2.morphologyEx(spatio_temporal_bw,
cv2.MORPH_CLOSE, kernel)
spatio_temporal_bw = cv2.morphologyEx(spatio_temporal_bw,
cv2.MORPH_OPEN, kernel2)


# Fill the holes in white blobs
im_floodfill = spatio_temporal_bw.copy().astype(np.uint8)
h, w = im_floodfill.shape[:2]
mask = np.zeros((h+2, w+2), np.uint8)
cv2.floodFill(im_floodfill, mask, (0,0), 255)
im_floodfill_inv = cv2.bitwise_not(im_floodfill)
spatio_temporal_bw = spatio_temporal_bw | im_floodfill_inv



def count_objects_on_the_line(lineThr, objectWidth):
   '''
   Counts how many objects are on a pixel line.
   Objects are connected white pixels, that are wider than
   arg[1] * widthOfTheLine
   '''
   line_width = lineThr.shape[0]
   j = 0
   curr_px = 0
```

```
    objects = 0
    px_counter = 0

    while j < line_width:
        prev_px = curr_px
        curr_px = lineThr[j]
        if curr_px > prev_px:
            px_counter = 0
        elif curr_px < prev_px:
            if px_counter > objectWidth * line_width:
                objects += 1
            px_counter = 0
        else:
            px_counter += 1
        j += 1

    return objects



def label_currently_on_the_line(imSpaTempThr):
    print()
    print("Using the 'currently on the line' labeling mode")

    labels = np.zeros((imSpaTempThr.shape[0], 1))

    i = 0
    while i < imSpaTempThr.shape[0]:
        lineThr = imSpaTempThr[i,:]
        labels[i] = count_objects_on_the_line(lineThr, 0.1)

        progress_bar("Automatic labelling:", i, spatio_temporal_bw.shape[0],
bar_length=20)
        i += 1

    return labels



def label_currently_left(imSpaTempThr):
    print()
    print("Using the 'currently left the detection line' labeling mode")

    labels = np.zeros((imSpaTempThr.shape[0], 1))
    i = 0
    line_width = imSpaTempThr.shape[1]
    objectsCurrent = 0
    while i < imSpaTempThr.shape[0]:
        lineThr = imSpaTempThr[i,:]
        objectsPrevious = objectsCurrent
```

```
        objectsCurrent = count_objects_on_the_line(lineThr, 0.1)
        if objectsPrevious - objectsCurrent > 0:
            labels[i] = objectsPrevious - objectsCurrent
        else:
            labels[i] = 0

        progress_bar("Automatic labelling:", i, spatio_temporal_bw.shape[0],
bar_length=20)
        i += 1

    return labels



# Choose the method
if LABEL_MODE == 'onTheLine':
    labels = label_currently_on_the_line(spatio_temporal_bw)
if LABEL_MODE == 'leftTheLine':
    labels = label_currently_left(spatio_temporal_bw)



def draw_gui_image():
    '''
    Draws labels to the spatio-temporal GUI image
    '''
    gui_image = np.zeros((spatio_temporal.shape[0], spatio_temporal.shape[1] + 60,
3))
    gui_image[0:spatio_temporal.shape[0],          0:spatio_temporal.shape[1]]          =
spatio_temporal
    current_label = 0
    change_delay = 0
    label_coord_x = spatio_temporal.shape[1]

    # Put lines
    for i in range(labels.shape[0]):
        previous_label = current_label
        current_label = labels[i,0]
        if current_label != previous_label:

            alpha = float(0.3)
            frame_overlay = gui_image.copy()
            cv2.line(frame_overlay, (0, i), (gui_image.shape[1], i),

                  (1, 0, 0), thickness=1, lineType=8, shift=0)
            cv2.addWeighted(frame_overlay, alpha, gui_image, 1 - alpha, 0,
gui_image)

    # Put text
    for i in range(labels.shape[0]):
```

```
        previous_label = current_label
        current_label = labels[i,0]

        label_coord_x == spatio_temporal.shape[1]
        if current_label != previous_label:

            cv2.putText(gui_image, str(int(current_label)), (label_coord_x, i+7),
                  cv2.FONT_HERSHEY_SIMPLEX, 0.3, (1,1,1))

            if label_coord_x == spatio_temporal.shape[1]+50:
               label_coord_x = spatio_temporal.shape[1]
            else:
               label_coord_x += 10

    # Make the image Qt ready
    gui_image = cv2.cvtColor(np.asarray(gui_image*255, dtype=np.uint8),
          cv2.COLOR_BGR2RGB).copy()
    return gui_image



def find_closest_object(mousePosX, mousePosY):
    ''' Finds the closest edge of an object upwards the current cursor
    position.'''

    regionHeight = 20
    regionWidth = 10
    if mousePosY < regionHeight:
       regionHeight = mousePosY

    region = spatio_temporal[mousePosY-regionHeight:mousePosY,
          int(mousePosX-regionWidth / 2): int(mousePosX+regionWidth / 2), :]
    region = region.astype(np.float32)
    region = cv2.cvtColor(region, cv2.COLOR_BGR2GRAY)

    originalLine = region[regionHeight-1, :]

    jump = regionHeight-1
    maxDif = 0
    firstDif = 10000

    imax = regionHeight-2

    for i in reversed(range(regionHeight-2)):
       dif = abs(originalLine - region[i,:])
       sumDif = np.sum(dif)
       if i == regionHeight - 3:
          firstDif = sumDif
```

```
        if sumDif > maxDif:
            maxDif = sumDif
            imax = i

        if firstDif == 0:
            firstDif = sumDif

        else:
            if sumDif > 5 * firstDif:
                jump = i
                break

        if i == 0:
            jump = imax

        previousSumDif = sumDif

    jumpCoord = mousePosY - (regionHeight - jump)

    return(jumpCoord+1)



def change_label(event, mousePosX, mousePosY, newGuiLabel, LABEL_MODE):

    if newGuiLabel == 100:
        if event.button() == QtCore.Qt.LeftButton:

            jumpCoord = find_closest_object(mousePosX, mousePosY)

            # The LABEL_MODE must be 'lefTheLine'
            labels[jumpCoord, 0] += 1
            labels[jumpCoord + 1, 0] == 0
        if event.button() == QtCore.Qt.RightButton:
            for i in range(50):
                i_forward = mousePosY + i
                i_backward = mousePosY - i
                if i_backward < 0:
                    break
                if i_forward >= labels.shape[0]:
                    break
                else:
                    if labels[i_backward, 0] > 0:
                        labels[i_backward, 0] -= 1
                        break
                    if labels[i_forward, 0] > 0:
                        labels[i_forward, 0] -= 1
                        labels[i_forward - 1] += 1
                        break
```

```
else:
    # Add a new line with a label previously set by a keyboard key
    if event.button() == QtCore.Qt.LeftButton:
        oldLabel = labels[mousePosY, 0]
        for i in range(mousePosY, labels.shape[0]):
            if LABEL_MODE == 'onTheLine':
                if labels[i, 0] == oldLabel:
                    labels[i, 0] = newGuiLabel
                else:
                    break

            if LABEL_MODE == 'leftTheLine':
                if i == mousePosY:
                    labels[i, 0] = newGuiLabel
                else:
                    if labels[i, 0] == oldLabel:
                        labels[i, 0] = 0
                    else:
                        break


    # Find closest existing line
    if event.button() == QtCore.Qt.RightButton:
        i = 0
        currentLabel = labels[mousePosY, 0]
        while mousePosY + i < labels.shape[0] and mousePosY - i > 0:
            forward = mousePosY + i
            backward = mousePosY - i
            if (labels[forward, 0] != currentLabel):
                oldLabel = labels[forward,0]
                f = forward
                while (f < labels.shape[0]):
                    if labels[f, 0] == oldLabel:
                        labels[f, 0] = currentLabel
                        f += 1
                    else:
                        break
                break

            # If closest line is up
            if (labels[backward, 0] != currentLabel):
                if LABEL_MODE == 'onTheLine':
                    oldLabel = labels[backward,0]
                    b = backward
                    while (b != mousePosY):
                        b += 1
                        labels[b, 0] = oldLabel
                    break
```

```
                if LABEL_MODE == 'leftTheLine':
                    oldLabel = labels[backward,0]
                    b = backward
                    while b != mousePosY:
                        labels[b, 0] = 0
                        b += 1
                    labels[mousePosY] = oldLabel
                    break

            i += 1

    gui_image = draw_gui_image()
    return gui_image



def save_labels():
    ''' Converts labels to one-hot vectors and saves them to hdd'''

    print("Saving results..")
    labels_one_hot = np.zeros((labels.shape[0], N_CLASSES))
    for i in range(labels.shape[0]):
        labels_one_hot[i, int(labels[i,0])] = 1

    name = VIDEO_NAME[0:VIDEO_NAME.find('.')] + "_line_" + str(LINE_R) +
"_" + str(LINE_C_BEGIN) + "_" + str(LINE_C_END)

    np.asarray(line_spatio_temporal, dtype=np.int8)
    np.save(name + ".npy", line_spatio_temporal[:-1,:])
    np.save(name + "_labels.npy", labels_one_hot[:-1,:])


gui_image = draw_gui_image()

windowHeight = 800
bytesPerComponent = 3
imageHeight, imageWidth, bytesPerComponent = gui_image.shape
bytesPerLine = bytesPerComponent * imageWidth



# THE GUI
print()

class Window(QtWidgets.QMainWindow):

    def __init__(self):

        self.imSizeMult = 2 # 2
```

```
        super(Window, self).__init__()
        self.setGeometry(50,50, self.imSizeMult * imageWidth + 20, windowHeight)
        self.setWindowTitle("Manual labeling")
        self.newGuiLabel = 0
        self.home()


    def home(self):

        # Show image
        self.imageLabel = QtWidgets.QLabel(self)
        QI = QtGui.QImage(gui_image, imageWidth, imageHeight,
                          bytesPerLine, QtGui.QImage.Format_RGB888)

        imageSize = QI.size()
        imageSize.setWidth(imageSize.width() * self.imSizeMult)
        imageSize.setHeight(imageSize.height() * self.imSizeMult)
        QI = QI.scaled(imageSize, QtCore.Qt.KeepAspectRatioByExpanding)

        self.imageLabel.setPixmap(QtGui.QPixmap.fromImage(QI))
        self.imageLabel.setGeometry(QtCore.QRect(0, 0, 2*imageWidth,
2*imageHeight))
        self.imageLabel.mousePressEvent = self.mouse_press

        # Scroll
        scrollArea = QtWidgets.QScrollArea(self)
        scrollArea.setWidget(self.imageLabel)
        scrollArea.setGeometry(0, 0, 2*imageWidth + 20, windowHeight)
        scrollArea.setHorizontalScrollBarPolicy(QtCore.Qt.ScrollBarAlwaysOff)

        self.show()


    def mouse_press(self, event):
        mousePos = event.pos()
        mousePosX = int(mousePos.x() / self.imSizeMult)
        mousePosY = int(mousePos.y() / self.imSizeMult)

        gui_image = change_label(event, mousePosX, mousePosY, self.newGuiLabel,
LABEL_MODE)

        QI = QtGui.QImage(gui_image, imageWidth, imageHeight,
                          bytesPerLine, QtGui.QImage.Format_RGB888)
        imageSize = QI.size()
        imageSize.setWidth(imageSize.width() * self.imSizeMult)
        imageSize.setHeight(imageSize.height() * self.imSizeMult)
        QI = QI.scaled(imageSize, QtCore.Qt.KeepAspectRatioByExpanding)

        self.imageLabel.setPixmap(QtGui.QPixmap.fromImage(QI))
```

```
    def keyPressEvent(self, event):
        self.newGuiLabel = event.text()
        if event.key() == QtCore.Qt.Key_Space:
            self.newGuiLabel = 100
        if event.key() == QtCore.Qt.Key_Tab:
            self.newGuiLabel = 0
        if event.key() == QtCore.Qt.Key_Escape:
            save_labels()
            print("All done!")

            QtWidgets.QApplication.quit()


def run_gui():
    app = QtWidgets.QApplication(sys.argv)
    GUI = Window()
    sys.exit(app.exec_())


run_gui()
```

linedetect-train.py

```
##### PARAMETERS ###################################
#input_path = "/home/data/traffic/Begri_2013_002"

model_name = "TURIBA"
DATA_PATH = "/home/plocins/nextcloud/turiba/"
#"/home/roberts/data/traffic/all_ready/"
# /data/sa-labeling/
# /linedetect/toy_2017_v2/
CONTINUE_LEARNING = False
PRETRAINED_PATH = "models/detection_1/"
PRETRAINED_MODEL = "train0.962_val0.829.ckpt"
saveLogGraphic = True
PLOT_WIDTH = 50 #50 # 200
GPU = 0
BALANCE_SEQUENCES = False
ALLOW_SOFT_PLACEMENT = True

# Network parameters
n_input = 100 # 400
n_steps = 14 # people - 50; balls - 20
n_hidden = 100 # 128
n_hidden_2 = 100
n_classes = 5 # people - 10 ! balls 15 ?
```

```
#tf.get_variable_scope().reuse_variables()


# Training parameters
learning_rate = 2e-4
epochs = 5000 # same as training_iters
visualization = True
batch_size = 500 # 128
display_step = 1
validation = True
TRAIN_VAL_RATIO = 0.7



##### IMPORTS ######################################
import numpy as np
import matplotlib as mpl
mpl.use('Agg')

from shutil import copyfile
import time
from random import randint
import tensorflow as tf
#from tensorflow.models.rnn import rnn, rnn_cell
import pylab as pl
import pickle
import os
import data_preparation as dataprep
#import cv2
#scipy vajadzigs prieksh data_preparation
#visdom vajadzigs prieksh data_preparation

#pielikts
import matplotlib.pyplot as plt
#### LOADING AND PREPROCESSING THE DATA ##############

print()
print()
print("-------------------------------------------")
print("-------------LINEDETECT-TRAIN BEGINS--------")
print()
print("Preparing input data..")


#exec(open("data_preparation.py").read())
linesTrain, labelsTrain, linesVal, labelsVal = dataprep.load_input(DATA_PATH,
      TRAIN_VAL_RATIO, n_input)
# !!!!!!!!!!!!!! ATTENTION TODO
# needed for counting of moving objects
```

```
labelsTrain = dataprep.convert_label_format(labelsTrain, n_steps, n_classes)
labelsVal = dataprep.convert_label_format(labelsVal, n_steps, n_classes)



# SAVING THINGS TO HDD ############################################
os.makedirs("models/" + model_name, exist_ok=True)
#logGraphicName = "models/" + model_name + "/training_log.ps"

# Saving the parameters and hyperparameters
n_classes = labelsTrain[0].shape[1]
f = open("models/" + model_name + "/parameters.pickle", 'wb')
pickle.dump([n_input, n_steps, n_hidden, n_hidden_2, n_classes, learning_rate], f)
f.close()

os.environ["CUDA_VISIBLE_DEVICES"] = str(GPU)
gpu_used = "/gpu:" + str(GPU)


dataprep.print_data_stats("Training data", labelsTrain, n_classes)
dataprep.print_data_stats("Validation data", labelsVal, n_classes)



##### NETWORK ########################################
print()
print("Defining the network..")

if CONTINUE_LEARNING:
    exec(open(PRETRAINED_PATH + "linedetect-net.py").read())
    copyfile(PRETRAINED_PATH + "linedetect-net.py", "models/" + model_name +
"/linedetect-net.py")
else:
    exec(open("linedetect-net.py").read())
    copyfile("linedetect-net.py", "models/" + model_name + "/linedetect-net.py")



##### SEQUENCES AND BATCHES ##########################
print()
print("Preparing sequences and batches..")

sqPosition, nrSq, sqLabels = dataprep.sequence_positions(linesTrain, labelsTrain,
n_steps)
sqPositionVal, nrSqVali, sqLabelsVal = dataprep.sequence_positions(linesVal,
labelsVal, n_steps)

if BALANCE_SEQUENCES:
    balancer = dataprep.get_sequence_balancer(sqLabels)
    sqPosition = sqPosition + balancer
```

```
    balancerVal = dataprep.get_sequence_balancer(sqLabelsVal)
    sqPositionVal = sqPositionVal + balancerVal

    print()
    print("  Class relations after sequqnce balancing")
    classRelation = np.zeros(labelsTrain[0][0].shape)
    for position in sqPosition:
        classNow = np.argmax(labelsTrain[position[0]][[position[1]]])
        classRelation[classNow] += 1
    print("  ", classRelation)


# Number of batches to cover (almost) all training data
batches_total = np.floor(len(sqPosition) / batch_size)
batches_total_val = np.floor(len(sqPositionVal) / batch_size)

batch_x = np.zeros((batch_size, n_steps, n_input))
batch_y = np.zeros((batch_size, n_classes))


print()
print("    ", len(sqPosition), "train", len(sqPositionVal), "val sequences")
print ("    Total:", epochs, "epochs,", batch_size, "sequences of",
                                    n_steps, "time steps in a batch")
print ("    Training:", batches_total, "batches")
print ("    Validation:", batches_total_val, "batches")
print ()
#print("    Training data parameters")
#print("        min max", np.min(linesTrain), np.max(linesTrain))
#print()


# creating a single batch consisting of _batch_size randomly selected sequences
def next_batch(_batch_size, _batch, _perm, _lines, _labels, _sqPosition):
    for sequence_idx in range(_batch_size):
        currentSqPos = _sqPosition[_perm[sequence_idx + _batch_size * _batch]]
        batch_x[sequence_idx,:,:]          =          _lines[          currentSqPos[0]          ][
currentSqPos[1]:currentSqPos[1]+n_steps ]
        batch_y[sequence_idx,:] = _labels[ currentSqPos[0] ][currentSqPos[1] + n_steps
- 1, :]
    return batch_x, batch_y




##### TRAINING ####################################
print()
print("Training..")
```

```
epoch_viz = []
acc_train_viz = []
acc_val_viz = []
cost_viz = []
plotCounterBig = 0
perm = np.arange(len(sqPosition))
perm_val = np.arange(len(sqPositionVal))
acc_val = 0


# Launch the graph
with
tf.Session(config=tf.ConfigProto(allow_soft_placement=ALLOW_SOFT_PLACEM
ENT)) as sess:
    if CONTINUE_LEARNING:
        saver.restore(sess, PRETRAINED_PATH + PRETRAINED_MODEL)
    else:
        sess.run(init)

    # Keep training until max iterations reached
    for epoch in range(epochs):
        avg_cost = 0
        acc_train_vec = []
        acc_val_vec = []

        np.random.shuffle(perm)


        for batch in range(int(batches_total)):
            batch_xs, batch_ys = next_batch(batch_size, batch, perm,
                                    linesTrain, labelsTrain, sqPosition)
            sess.run(optimizer, feed_dict={x: batch_xs, y: batch_ys,
                                    istate: np.zeros((batch_size, 2*n_hidden))})

            if epoch % display_step == 0:
                # accuracy on the single batch
                acc_train_vec.append(sess.run(accuracy,     feed_dict={x:     batch_xs,     y:
batch_ys,
                                            istate: np.zeros((batch_size, 2*n_hidden))}))

                avg_cost += sess.run(cost, feed_dict={x: batch_xs, y: batch_ys,
                                        istate: np.zeros((batch_size, 2*n_hidden))})

        if epoch % display_step == 0:


            # Validation set
            if (validation):
                for batch in range(int(batches_total_val)):
                    batch_xs, batch_ys = next_batch(batch_size, batch, perm_val,
```

```
                                    linesVal, labelsVal, sqPositionVal)
            acc_val_vec.append(sess.run(accuracy, feed_dict={x: batch_xs, y: batch_ys,
                                                istate: np.zeros((batch_size, 2*n_hidden))}))
        acc_val = np.mean(acc_val_vec)


    # accuracy on the whole trraining dataset (mean from all batches)
    acc_train = np.mean(acc_train_vec)
    saver.save(sess, "models/" + model_name + "/train" + str(round(acc_train,3)) + "_val" + str(round(acc_val,3)) + ".ckpt")




    if visualization:
        try:
            epoch_viz.append(epoch)
            acc_train_viz.append(acc_train)
            cost_viz.append(avg_cost)
            if (validation):
                acc_val_viz.append(acc_val)

            if ( saveLogGraphic == True and epoch != 0 and epoch % PLOT_WIDTH == 0):

                plt.figure()
                pl.plot(epoch_viz, acc_train_viz, 'b')
                if (validation):
                    pl.plot(epoch_viz, acc_val_viz, 'g')

                plt.grid(True)
                plt.xlabel('epochs')
                plt.ylabel('accuracy')

                partLogName = ( "models/" + model_name + '/accuracy_'
                            + str(plotCounterBig) + '.ps' )
                plt.savefig(partLogName, bbox_inches='tight')


                plt.figure()
                plt.plot(epoch_viz, cost_viz, 'r')
                plt.grid(True)
                plt.xlabel('epochs')
                plt.ylabel('cost')
                partLogName = ( "models/" + model_name + '/cost_'
                            + str(plotCounterBig) + '.ps')
                plt.savefig(partLogName, bbox_inches='tight')

                plotCounterBig += 1
```

```
                    epoch_viz = []
                    acc_train_viz = []
                    acc_val_viz = []
                    cost_viz = []

            except KeyboardInterrupt:
                break


        print ("Epoch:", '%03d' % (epoch+1), "of", str(epochs),
              "cost=", "{:.4f}".format(avg_cost),
              " Train accuracy=", "{:.5f}".format(acc_train),
              " Val accuracy=", "{:.5f}".format(acc_val))


print ("Optimization Finished!")
##### IMPORTS ########################################
```

linedetect-realtime_vp_ks.py

```
import numpy as np
import tensorflow as tf
from scipy.misc import imresize
import pickle
import os
import cv2

import visdom
import time

VIDEO = "rtsp://IP"

LINE_R = 170
LINE_C_BEGIN = 1
LINE_C_END = 635

LINE_WIDTH = 1

model = "TURIBA"
model_weights = "train1.0_val0.999"
n_classes = 5 # 10 vehicles, people, 15 balls (balls3=10)

VISUALIZATION = True
```

```
# Load net parameters
f = open("models/" + model + "/parameters.pickle", 'rb')
[n_input, n_steps, n_hidden, n_hidden_2, n_classes, learning_rate] = pickle.load(f)
f.close()


n_seq = n_steps
#n_seq = int(40 / 2)
existance_margin = int(n_seq / 3)

#n_steps = 500


os.environ["CUDA_VISIBLE_DEVICES"] = "0"
#GPU_USED = '/gpu:2'
gpu_used = '/cpu:0'
ALLOW_SOFT_PLACEMENT = True


exec(open("models/" + model + "/linedetect-net.py").read())



cap = cv2.VideoCapture(VIDEO)
if VISUALIZATION:
    cv2.namedWindow("line detector")


batch_in = np.zeros((n_steps, n_input))
batch_yy = np.zeros((1, n_classes))

count = 0

time_vec = []



class Seqleft:

    def __init__(self, beginingThr, seqLength, n_classes):
        self.beginingThr = beginingThr
        self.seqLength = seqLength
        self.n_classes = n_classes
        self.baseLabel = 0
        self.beginingCounters = np.zeros((n_classes, 1))
        self.endingCounters = np.zeros((n_classes, 1))
        self.count = 0


    def add_current_label(self, currentLabel):
```

```
        previousBaseLabel = self.baseLabel
        for i in range(self.n_classes):
            if i < ( currentLabel - previousBaseLabel ):
                self.beginingCounters[i] += 1
            else:
                self.beginingCounters[i] = 0

            if self.beginingCounters[i] >= self.beginingThr:
                self.baseLabel += 1
                self.count += 1
                for j in range(self.endingCounters.shape[0]):
                    if self.endingCounters[j] == 0:
                        self.endingCounters[j] = 1
                        break
        # print()
        # print("current label", currentLabel)


    def check_counters(self):
        for i in range(len(self.endingCounters)):
            if self.endingCounters[i] > 0:
                self.endingCounters[i] += 1
            if self.endingCounters[i] > self.seqLength:
                self.endingCounters[i] = 0
                self.baseLabel -= 1




with
tf.Session(config=tf.ConfigProto(allow_soft_placement=ALLOW_SOFT_PLACEM
ENT)) as sess:

    saver.restore(sess, "models/" + model + "/" + model_weights + ".ckpt")

    f = 0
    seqleft = Seqleft(existance_margin, n_seq, n_classes)

    while(True):
        f += 1
        time_0 = time.time()


        ret, frame = cap.read()


        if f == 1:
            print()
            print()
```

```
    print("Input frame parameters:")
    print("  shape", frame.shape)
    print("  min max", np.min(frame), np.max(frame))
    print()



line = frame[LINE_R : LINE_R + LINE_WIDTH,
             LINE_C_BEGIN : LINE_C_END].copy()


cv2.line(frame, (LINE_C_BEGIN, LINE_R), (LINE_C_END, LINE_R),
         (255, 0, 0), thickness=1, lineType=8, shift=0)

line = cv2.cvtColor(line, cv2.COLOR_BGR2GRAY)

# TODO check if network input is same as for training


line = line.astype(float) / 255.


if (n_input != line.size):
    new_line = imresize(line, (1, n_input), interp='nearest', mode='F')
else:
    new_line = np.reshape(line, (1, n_input))



batch_in = np.append(batch_in[1:,:], new_line, axis=0)

if f >= n_seq:

    batch_xx = np.reshape(batch_in, (1, n_steps, n_input))


    result = sess.run(pred, feed_dict={x: batch_xx, y: batch_yy,
                                       istate: np.zeros((1, 2*n_hidden_2))})


    label_idx = np.argmax(result)

    seqleft.add_current_label(label_idx)
    seqleft.check_counters()
    previosuCount = count
    count = seqleft.count
```

```
time_no_viz = round(time.time() - time_0, 5)
if f >= n_seq:
    time_vec.append(time_no_viz)


if VISUALIZATION:
    cv2.putText(frame, str(count), (LINE_C_END-70, LINE_R),
                cv2.FONT_HERSHEY_SIMPLEX, fontScale = 1,
                color = (100, 255, 17), thickness = 2)

    cv2.imshow("line detector", frame)

    key = cv2.waitKey(1) # 1
    key_char = chr(key & 255)
    if key_char == 'q':
        break

time_all = round(time.time() - time_0, 5)

if len(time_vec) > 0:
    time_avg = sum(time_vec) / len(time_vec)

    print("\r frame", f, "    t_all:", time_all, "    t_nviz:", time_no_viz,
          "    avg:", time_avg, end="")
```