UNIVERSITY of DEBRECEN

Dr. József Sütő
Department of IT Systems and Networks

Embedded Systems Laboratory

Lab 10: Image classification and object detection

Project overview

After OpenCV 3.3 the library includes an efficient deep neural network module (**dnn**). The module provides support to:

- Pre-process input image
- Loading pre-trained model from disk

This module supports several deep learning frameworks like Caffe and Tensorflow. In this laboratory you will see how can you use pre-trained deep learning models (on famous databases) on your input images.

The following topics will be covered in this lab:

- Image classification
- Object detection

Technical requirements

The following components are required to complete this lab:

- Raspberry Pi 3 Model B+ (MicroSD card, power supply, keyboard, mouse)
- Pi camera module

OR

- PC
- USB/built-in camera

Deep learning with OpenCV

Deep Learning is a rapidly growing domain of machine learning and if you are working in the field of computer vision or image processing, it will be very important for you. With OpenCV 3.3, we can utilize pre-trained networks with popular deep learning frameworks. Since the

models are pre-trained, you do not need to spend many hours or days to training the network. Popular network architectures compatible with OpenCV 3.3 include:

- Inception (GoogleLeNet)
- AlexNet
- SqueezeNet
- VGGNet
- ResNet

1. Image classification with the Inception model

In this exercise we will create a Python code which can be used for image classification. The classifier model is the *Inception model* trained on more million images from the ImageNet database (http://www.image-net.org/). The model to this exercise can be downloaded from the laboratory's website.

1.1. Import necessary libraries:

```
import numpy as np
import time
import cv2
```

1.2. Define paths to the test image, to the model source files, and to the labels file. The labels file includes all 1000 classes from the ImageNet database (out model was trained):

```
image_path = "…/your_test_image.jpg"
label_path = "…/imagenet_labels.txt"
prototxt_path = "…/googlenet.prototxt"
model_path = "…/googlenet.caffemodel"
```

1.3. Read image from disk. Read all lines from the labels file into a list. Extract relevant class names from lines:

```
image = cv2.imread(image_path)
rows = open(label_path).read().strip().split("\n")
classes = [r.split(",")[0] for r in rows]
```

1.4. The used classifier model requires fixed spatial dimensions for the input image. We use the *blobFromImage()* function to resize and normalize the image. At the end of the command, the output shape is (1, 3, 224, 224):

```
blob = cv2.dnn.blobFromImage(image, 1, (224, 224), (104, 117, 123))
```

1.6. Load model from disk:

```
print("Loading model...")

net = cv2.dnn.readNetFromCaffe(prototxt_path, model_path)
```

1.7. Send the test image (blob) through the model to obtain classification result:

```
net.setInput(blob)

start = time.time()

preds = net.forward()

end = time.time()

print("Classification time: {:.5} seconds".format(end - start))
```

1.8. Sort predictions (class probabilities) descending order and extract the top 5 predictions:

```
idxs = np.argsort(preds[0])[::-1][:5]
```

1.9. Loop over the top 5 predictions. Print out the higher prediction's class name to the image:

```
for (i, idx) in enumerate(idxs):

    if i == 0:

        text = "Label: {}, {:.2f}%".format(classes[idx], preds[0][idx] *
100)

        cv2.putText(image, text, (5, 25),
cv2.FONT_HERSHEY_SIMPLEX, 0.7, (0, 0, 255), 2)
```

1.10. Print out predictions (class name and probability value) into the console:

```
print("{}. label: {}, probability: {:.5}".format(i + 1, classes[idx],
preds[0][idx]))
```

1.11. Show the output image (**task to be solved alone!**). Sample images can be seen below.

UNIVERSITY of DEBRECEN

Dr. József Sütő
Department of IT Systems and Networks

Task to be solved alone:

1.A. On the laboratory's website you can find another label file – *imagenet1000_labels.txt*. Use this file instead of the *imagenet_labels.txt*!

Object detection

Image classification gives information about the content of the image but it does not tell us where an object resides on the image. In order to obtain the bounding box coordinates for an object in the image, we need to apply **object detection**.
In deep learning-based object detection, there are three primary object detection methods at this time:

- **Faster R-CNNs**
- **You Only Look Once (YOLO)**
- **Single Shot Detectors (SSDs)**

Faster R-CNNs are likely the most popular method for object detection using deep learning. However, the technique can be difficult to understand, hard to implement, and challenging to train. Furthermore, even with the "faster" implementation of the algorithm can be quite slow. If we are looking for pure speed then we tend to use YOLO as this algorithm is much faster, capable of processing 40-90 FPS on a Titan X GPU. The simplified variant of YOLO can even get up to 155 FPS. The problem with YOLO is that it leaves much accuracy to be desired.
SSDs, originally developed by Google, are a balance between the two. The algorithm is more straightforward than Faster R-CNNs. SSDs also tend to be more accurate than YOLO.

2. Object detection with the MobileNetSSD model

The goal of this exercise is to use the pre-trained MobileNetSSD classifier for object detection. This model is rather fast and makes it possible real-time object detection on resource constrained devices such as the Raspberry Pi. The model to this exercise can be downloaded from the laboratory's website. To load the model, again the **OpenCV's dnn** module will be used.

2.1. Import necessary libraries:

```
import numpy as np

import cv2
```

2.2. Define paths to the model and the test image:

```
prototxt_path = "…/MobileNetSSD_deploy_prototxt.txt"

model_path = "…/MobileNetSSD_deploy.caffemodel"

image_path = "…/image.jpg"
```

2.3. Define a **confidence threshold** to filter out less certain predictions:

```
conf_limit = 0.25
```

2.4. Create a list with class labels our model was trained. Those labels will be used to image annotation.

```
CLASSES = ["background", "aeroplane", "bicycle", "bird", "boat",
"bottle", "bus", "car", "cat", "chair", "cow", "diningtable", "dog",
"horse", "motorbike", "person", "pottedplant", "sheep", "sofa", "train",
"tv/monitor"]
```

2.5. Generate as number of random colors as the number of classes. Each class will be marked with its own color on the test image:

```
COLORS = np.random.uniform(0, 255, size=(len(CLASSES), 3))
```

2.6. Load the pre-trained model from your disk:

```
print("Loading model...")

net = cv2.dnn.readNetFromCaffe(prototxt_path, model_path)
```

2.7. Load the image from your storage and construct an input blob. The *blobFromImage()* function is optionally resize, crop, subtract mean, scale, and swap channels in the input image:

```
image = cv2.imread(image_path)

(h, w) = image.shape[:2]

blob = cv2.dnn.blobFromImage(cv2.resize(image, (300, 300)), 0.007843,
(300, 300), 127.5)
```

2.8. Send the test image through the object detector model.

```
print("Sending image through the network...")

net.setInput(blob)

detections = net.forward()
```

2.9. Loop over all detections and extract confidence (probability) values

```
for i in np.arange(0, detections.shape[2]):

    # extract the confidence

    confidence = detections[0, 0, i, 2]
```

2.10. Filter out week detections with the earlier defined threshold value. If a confidence is higher than the threshold, extract the class index and the coordinates of the bounding box:

```
if confidence > conf_limit:

        idx = int(detections[0, 0, i, 1])

        box = detections[0, 0, i, 3:7] * np.array([w, h, w, h])

        (startX, startY, endX, endY) = box.astype("int")
```

2.11. Print out detected labels and annotate the image:

```
label = "{}: {:.2f}%".format(CLASSES[idx], confidence * 100)

print("{}".format(label))
```

cv2.rectangle(image, (startX, startY), (endX, endY), COLORS[idx], 2)

y = startY - 15 if startY - 15 > 15 else startY + 15

cv2.putText(image, label, (startX, y), cv2.FONT_HERSHEY_SIMPLEX, 0.5, COLORS[idx], 2)

2.12. Show the output image (same as 1.11). Some sample images can be shown below.

Task to be solved alone:

2.A. Extend the above code to work with real-time video stream!

References

[1] Rosebrock, A. https://www.pyimagesearch.com/2017/09/11/object-detection-with-deep-learning-and-opencv/. Accessed on 28/09/2020.

[2] Rosebrock, A. https://www.pyimagesearch.com/2017/08/21/deep-learning-with-opencv/. Accessed on 28/09/2020.

This work was supported by the construction EFOP-3.4.3-16-2016-00021. The project was supported by the European Union, co-financed by the European Social Fund.

UNIVERSITY of DEBRECEN

Dr. József Sütő
Department of IT Systems and Networks