Doggie Smile

16th October 2017
Alzbeta Vlachynska, vlachynskaa@gmail.com

Have you ever tried to take a picture of a puppies litter? It seems nearly impossible to make them look into the camera and then press the trigger in the right moment, right? This application can help you with both tasks. It makes sounds to attract attention of the dogs; simultaneously it detects their pose in image and takes a picture automatically in the right moment. Try it!

(a) (b)

Figure 1 Example photo: (a) dog detection, (b) head pose detection.[1]

1 Basic steps of the application:

1. Real-time detection of dogs in the video stream.
2. Attract the dogs' attention.
3. Detection of dogs' head pose.
4. Picture evaluation – check if all the dogs in the image are looking into camera.
5. If the photo was accepted, save it and quit. Otherwise, repeat the process from beginning.

2 Realization

The code is written in C++ with the support of the OpenCV and Dlib libraries. I use a video stream from a webcam in this prototype application. This makes it accessible for testing. I describe each step of the application in more detail in next chapters. I always state the basic challenges of the part and then describe my solution.

2.1 Real-time detection of dogs in the video stream

The main challenge of this section is obvious – I need to find quite robust object detector that can run real-time. I used the MobileNet SSD model for this task. It is a Caffe version of the original TensorFlow implementation by Howard et al. [2]. This version was trained by chuanqi305, for more information about this model please see his GitHub repository. [3] This model combines the MobileNet architecture with the Single Shot Detector (SSD) framework for fast, efficient deep-learning based object detection. The MobileNet SSD reach 72.7% mAP (mean average precision). It can detect twenty objects in images (+1 for the background class), including dogs.

2.1.1 Realization

I used deep neural network (dnn) module in OpenCV to build the object detector. Let me show the usage of the detector on pieces of code:

1. Load the model.

```
String pathNetDogTxt("MobileNetSSD_deploy.prototxt");
String pathNetDogBin("MobileNetSSD_deploy.caffemodel");
Net netDog;
//read caffe model
netDog = readNetFromCaffe(pathNetDogTxt, pathNetDogBin);
```

2. Grab the query image (= one frame from video stream) and prepare the blob, which we will feed-forward through the network and get the Mat object *detections* as output.

```
// grab first image and get its size
cap >> img;
Size imgSize = img.size();
// create input blob: resize image and convert Mat to dnn::Blob image batch
Mat img300;
resize(img, img300, Size(300, 300));
Mat inputBlob = blobFromImage(img300, 0.007843, Size(300, 300), Scalar(127.5));
// apply the blob on the input layer
net.setInput(inputBlob); //set the network input
// classify the image by applying the blob on the net
Mat detections = net.forward("detection_out"); //compute output
```

3. Loop over all detected objects in variable *detections*, choose only dog class with the confidence of detection higher than 30%. Count all successful detections of dog in variable *nrDog* and draw the bounding box into the query image.

```
// look what the detector found
int nrDog = 0;
for (int i = 0; i < detections.size[2]; i++) {
// detected class
int indxCls[4] = { 0, 0, i, 1 };
int cls = detections.at<float>(indxCls);
// confidence
int indxCnf[4] = { 0, 0, i, 2 };
float cnf = detections.at<float>(indxCnf);
// mark with bbox only dogs
if (cls == 12 && cnf > 0.3) {
     // count the dog
     nrDog = nrDog++;
     // bounding box
     int indxBx[4] = { 0, 0, i, 3 };
     int indxBy[4] = { 0, 0, i, 4 };
     int indxBw[4] = { 0, 0, i, 5 };
     int indxBh[4] = { 0, 0, i, 6 };
     int Bx = detections.at<float>(indxBx) * imgSize.width;
     int By = detections.at<float>(indxBy) * imgSize.height;
     int Bw = detections.at<float>(indxBw) * imgSize.width - Bx;
     int Bh = detections.at<float>(indxBh) * imgSize.height - By;
     // draw bounding box to image
     Rect bbox(Bx, By, Bw, Bh);
     Scalar color(255, 0, 255);
     cv::rectangle(img, bbox, color, 2, 8, 0);
     String text = classNames[cls] + ", conf: " + to_string(cnf * 100);
     putText(img, text, Point(Bx, By), FONT_HERSHEY_SIMPLEX, 0.75, color);
     }
}
```

2.1.2 Evaluation

I run my program in MS Visual studio 2015, on computer with 4GB RAM memory and Intel® Core™ i5-52200U CPU @ 2.20 GHz processor. The size of the input image from webcam is 640x480 pixels. The average time taken by dog detector in these conditions is approximately 150 milliseconds per one processed frame. Therefore I could theoretically process 66.66 frames per second. This is far enough for the real time processing of the video since I get only circa 30 fps on my webcam.

I did not test the MobileNet SSD successful detection rate. It should have the 72.7% mAP (mean average precision) according to the author. During the testing of the application, it seems reliable enough for good functionality of the application.

2.2 Attract the dogs' attention

I use different sounds to attract dogs' attention. I downloaded the sounds of a cat meow and a toy squeaks from server with free audio orangefreesounds.com. [4] I play the sound approximately 10s after the program starts. The exact audio file to play is chosen randomly.

2.2.1 Realization

I use an audio function from windows standard library to play the sound.

```
// function to play the random sound
void playRndSound(std::vector<LPCWSTR> sounds)
     {
     // choose rondomly the sound and play it
     std::srand(time(NULL));
     int s = std::rand() % sounds.size();
     PlaySound(sounds[s], NULL, SND_ASYNC);
}
```

2.2.2 Evaluation

As far as I test it, the dogs loves the sounds. Sometimes evet too much. ☺

2.3 Detection of dogs' head pose

This task need to be split into two parts: dog's head detection and the landmark detection. I utilized the network based dog head detector and the landmark detector trained by David King [5] for this tasks. For more information about the models, see David King's Github.

2.3.1 Realization

I used dlib deep learning tools to detect dogs looking into the camera and the dlib shape predictor to identify the positions of the dog's eyes, nose, and top of the head.

1. We need to define special *net_type* to use the dog head detector.

```
// define net type for dogHeadDetector
template <long num_filters, typename SUBNET> using con5d = con<num_filters, 5, 5, 2, 2, SUBNET>;
template <long num_filters, typename SUBNET> using con5 = con<num_filters, 5, 5, 1, 1, SUBNET>;

template <typename SUBNET> using downsampler = relu<affine<con5d<32, relu<affine<con5d<32,
relu<affine<con5d<16, SUBNET>>>>>>>>>;
template <typename SUBNET> using rcon5 = relu<affine<con5<45, SUBNET>>>;

using net_type = loss_mmod<con<1, 9, 9, 1, 1,
rcon5<rcon5<rcon5<downsampler<input_rgb_image_pyramid<pyramid_down<6>>>>>>>>;
```

2. Load the models using the function *deserialize*.

```
// set inputs
String pathNetHead("dogHeadDetector.dat");
String pathLandmarkDetector("landmarkDetector.dat");
//read models
net_type netHead;
shape_predictor landmarkDetector;
// Load the dog head detector
deserialize(pathNetHead) >> netHead;
// Load landmark model
deserialize(pathLandmarkDetector) >> landmarkDetector;
```

3. Convert the openCV mat image to Dlib format and feed the network.

```
// convert OpenCV image to Dlib's cv_image object, then to Dlib's matrix object
Mat imRGB;
cv::cvtColor(img, imRGB, cv::COLOR_BGR2RGB);
dlib::matrix<dlib::rgb_pixel> imDlib(dlib::mat(dlib::cv_image<dlib::rgb_pixel>(imRGB)));
//detect dog heads in image
std::vector<dlib::mmod_rect>faceRects = netHead(imDlib);
```

4. Loop over all dog's faces and find the landmarks positions.

```
// Loop over all detected face rectangles
for (int i = 0; i < faceRects.size(); i++)
{
// For every face rectangle, run landmarkDetector
full_object_detection landmarks = landmarkDetector(imDlib, faceRects[i].rect);
}
```

5. Check that the dog is looking at the camera, implemented as the bool function *checkHeadPosition*. It verifies that the distances between left eye - nose and the right eye – nose and also the distances between left ear – nose and the right ear - nose are roughly the same. The code below shows only comparison of eye-nose distances, for ear-nose distance I do it likewise.

```
float maxError = 0.3;
//distance between left eye and nose and right eye and nose should be nearly the same
float diffLeye = euclideanDist(cv::Point(landmarks.part(3).x(), landmarks.part(3).y()),
cv::Point(landmarks.part(5).x(), landmarks.part(5).y()));
float diffReye = euclideanDist(cv::Point(landmarks.part(3).x(), landmarks.part(3).y()),
cv::Point(landmarks.part(2).x(), landmarks.part(2).y()));
float meanEyeDist = (diffReye + diffLeye) / 2;
// if the distance difference is bigger than 20% of its value return false
if ((diffLeye - diffReye) > meanEyeDist * maxError ||
    (diffLeye - diffReye) < -1 * meanEyeDist * maxError) {
     return false;
}
```

2.3.2 Evaluation

The dlib dog's head detector is not fast enough for real-time evaluation. The dog's head detection, the landmark detection and the evaluation of head position on the same computer and in the same conditions as were described in chapter 2.1.2 can take up to 9 seconds depending on number and size of the dog's head detected. Therefore, I chose the solution to take the picture short while after playing the sound and evaluate if all the dogs are looking into camera retroactively.

2.4 Picture evaluation

Since the dog head detector is not fast enough I evaluate the quality of the taken image retroactively. There is some time to direct the camera at dogs at the beginning of the application. Then I check if the number of detected dogs in the frames changed. When it stay stable for at least five frames in row I play the sound to attract dogs' attention. Then I wait for some short time and again check that the number of detected dogs did not changed in last five frames. If the conditions are fulfilled, I analyze the picture both by dog detector and by dog's head pose detector. The picture is accepted as output only if the number of detected dogs and number of correct head poses are the same. Otherwise, the process starts again from the beginning. Described process is illustrated in Diagram 1.

3 Conclusion

The prototype application for the automatic dog photography was created. It is able to detect dogs in the real time video from a webcam. The program uses different sounds to attract dogs' attention. The quality of the image is automatically evaluated and accepted only if all the dogs in the picture are looking into camera. The full code of the application can be found on my GitHub [6]. The video on my YouTube channel [7] demonstrates the functionality of the program.

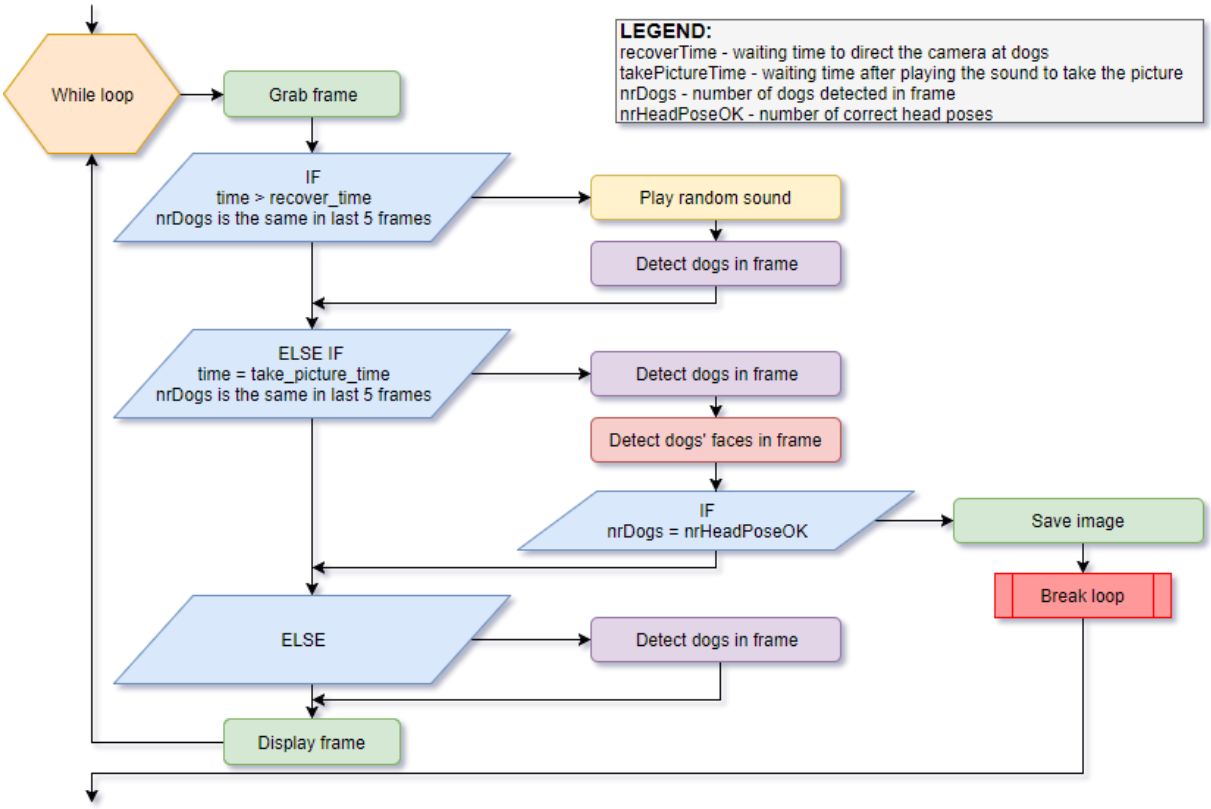

Diagram 1 Basic concept of the application

4 Future work

New dogs' head detector based on the MobileNet architecture could be trained to speed up the execution. If the head pose checking runs real time, the picture can be taken in the exact moment, when all the dogs are looking into camera. The program than may be implemented as mobile application using the mobile camera to shot the image.

5 References

[1] Vranova, I., The puppies' photography, Rajce.net photo gallery, September 3, 2015, URL: http://vranky.rajce.idnes.cz/STENATKA_PRTIKU_8_TYDNU_-_3.9.2015/#IMG_2940.jpg.

[2] Howard et al., MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications, Computer Vision and Pattern Recognition, April 17, 2017, URL: https://arxiv.org/abs/1704.04861.

[3] chuanqi305, MobileNet-SSD, GitHub repository, June 8, 2017, URL: https://github.com/chuanqi305/MobileNet-SSD.

[4] Orange Free Sounds © 2017, URL: http://www.orangefreesounds.com/.

[5] King, D., Hipsterize Your Dog with Deep Learning, Dlib blog, October 7, 2016, URL: http://blog.dlib.net/2016/10/hipsterize-your-dog-with-deep-learning.html.

[6] Vlachynska, A., Doggie-smile, GitHub repository, October 15, 2017, URL: https://github.com/vlachynska/Doggie-smile.

[7] Vlachynska, A., Application: Doggie Smile, YouTube channel: vlachynska, October 17, 2017, URL: https://youtu.be/g2TpWr6me1w.