Machine Learning for Computer Vision, MVA 2015-2016

Instructor: Iasonas Kokkinos, iasonas.kokkinos[at]ecp.fr

Derivarable - 10 Points/20 Deliverable: 22/4/16

Please use [ML4CV-Deliverable] in your email subject when submitting your report

In this assignment you will get to run and evaluate a system for face detection that relies on the deformable part model (DPM) paradigm.

The assignment is broken into four parts.

- Part 1 (3 Grades/10) asks you to train detectors for the individual parts of a face; these serve as 'unary', apperance-based terms for a DPM. There you will get to benchmark the different classifiers learned in the first part of the class (Lectures 1-4), and also assess the different feature design choices.
- Part 2 (3 Grades/10) asks you to perform inference with Deformable Part Models (DPMs). For this you will have to implement the Max-Product algorithm.
- Part 3 (2 Grades/10) asks you to write code that will allow you to perform mutli-scale detection and benchmark the performance of your system.
- Part 4 (3 Grades/10) asks you to use hard negative mining to learn the score function that drives DPM-based detection so as to improve the detection performance of your system.

The sum of the grades is more than 10, but you cannot get more than 10 grades out of the assignment - you are rather supposed to choose the parts that you will do.

The deliverable is designed so that you *do not need to solve* a part in order to advance to the next one. In particular solving Part 1 is not a prerequisite for Part 2, and Part 2 is not a prerequisite for Part 3. But you will need all of Parts 1-3 to get Part 4. Details about skipping parts are provided at the end of the respective sections; use these freely to start with the part which seems easiest/most interesting to you (and then potentially fill in things that you omitted at first).

You are asked to package your code together with your report in a single zip file. *Do not include* the 'data' folder. Other than that, running a 'main' file should deliver the results you provide in your report.

1 Part detector training

The following subsections describe the features, data and code that you will use in this assignment. You can see how these tie together by looking at `main_parts.m` . You have to fill in the pieces that are missing.

1.1 Features

The gateway function for extracting features in this assignment is `get_features.m`. It allows to extract both sparse (around a point) and dense (everywhere in the image) features. You can see a demonstration of the computed features in `demo_dense.m`. The computed features are Dense SIFT filters [3].

- Install the VLFeat library. Follow the installation instructions. Run and go through `vl_demo_sift_basic.m`, `vl_demo_sift_or.m`, and `vl_demo_dsift.m`.
- In `get_features.m` you are provided with code for computing dense features over the image domain. Make sure it works on your computer.
- To extract CNN features you have to install MatConvNet, following the instructions in the website. You can extract features using the standard pre-trained Alexnet model [6].

1.2 Data

The `load_im.m` function provides you with a grayscale image and a set of points. The second argument to the function indicates if the image is positive (face) or negative (background). For faces the points are the coordinates of 5 landmarks corresponding to the eyes, the nose, and the mouth corners. For background images the points are sampled from a regular grid on the image. See the first lines of `main_parts.m` for a demonstration of this function.

1.3 Tasks

Repeat the following steps for all five parts, for both SIFT and CNN features. During development feel free to use a smaller subset of the data (e.g. 30 positive and 10 negative images), to get things up and running quickly. Once you make sure everything is working, then go ahead and use the whole dataset (once).

0/3 Construct the training and test sets for your classifiers. (Done, you need to see how the relevant code in `main_parts.m` works)

2.0/3 Train a Logistic regression and a Linear SVM classifier using the training set.

Use cross validation to choose the best regularization parameter (different values will be needed for different part/classifier/feature combinations). Once the optimal parameters are chosen, retrain with the full training set.

.5/3 Write the code that will construct a precision -recall curve on the test set. This requires for each threshold value t estimating the precision $p(t)$, and recall $r(t)$ of the classifier on the test set. Or else, figure out how to use the `voc_pr.m` function of the `vlfeat` library.

.5/3 Display the results of your classifier as a dense score function, along the lines of the code at the end of `main_parts.m` .

Hint 1 (optional): for a linear SVM classifier you can recover the solution in terms of the inner product between the input and a weight vector. This makes the evaluation of the linear svm classifier much faster (a single inner product, instead of multiple ones).

You can recover this weight vector from the solution delivered by `svmtrain_libsvm` as follows:

```
 alpha = model.sv_coef; % support vector coefficients
 SVs   = model.SVs; % features of the associated support vectors
  weight_vector = alpha'*SVs; % w = \sum_i a_i F_i
%remove rho (constant term) from the  weight of the constant component.
weight_vector(end) = weight_vector(end) - model.rho;
```

The last line is because of the way the features are constructed: namely, the features include the constant component in their last dimension. The SVM training routines 'append' the constant term (`model.rho`) automatically to the score function. You therefore need to add it to the last component of the weight vector.

1.4 Skipping part 1

The file `svm_linear.mat` contains pre-trained classifiers for face parts. These might not be the best possible in terms of performance, but are good enough for advancing to part Part 2.

2 Part combination: Max-Product for DPM detection

As discussed in class, if implemented naively, every message passing operation of Max-Product has complexity proportional to N^2, where N is the number of image pixels. For a star-shaped model with P parts, this amounts to $O(2PN^2)$ complexity. A substantial speedup can be obtained by using the Generalized Distance Transform technique of [1, 2] which implements every message passing operation with a complexity that is *linear* in the number of pixels.

You are provided with the code used to compute dense unary potentials, as well as almost all of the code for performing object detection using the GDT-based Max-Product algorithm. Your task is to implement Max-Product in the simple-but-slow way, and use the GDT-based implementation only to verify that your results are correct.

2.1 Max Product-GDT code

The code provided to you implements the 'leaf-to-root' stage of Max Product. In the inner loop of the first block of code (use the `main_dpms.m` file for reference) messages are sent from all parts to the root node.

Consider now the Max-Product operation involved in sending a message from the part to the root:

$$M_{p\to r}(X) = \max_{X_p} \underbrace{\Phi_p(X_p)}_{\text{unary}} \underbrace{\Psi_{p,r}(X_p, X_r)}_{\text{binary}} \tag{1}$$

The candidate object detections at the root are obtaiend by identifying local maxima of the root's 'belief' function:

$$B(X) = \Phi_r(X) \prod_{i\neq r} M_{i\to r}(X). \tag{2}$$

We can equivalently maximize $\log(B(X)) = \log \Phi_r(X) + \sum_{i\neq r} \log M_{i\to r}(X)$. We introduce the notation $m = \log M, \phi = \log \Phi, \psi = \log \Psi$ and the messages become:

$$m_{p\to r}(X) = \max_{X_p} \left[\phi_p(X_p) + \psi_{p,r}(X_p, X_r)\right] \tag{3}$$

The first term, $\phi_p(X_p)$, is what is referred to in the code as 'unary'. It is a function of image location, X_p, which indicates how well the individual part's model fits the image locally. As you have seen in the first deliverable, it can be constructed using SIFT features, filter responses, or anything else that describes the local appearance of the image.

Regarding the second term, in DPMs we typically assume that $\Psi_{p,r}$ is of the form:

$$\Psi_{p,r}(X_p, X_r) = \frac{1}{2\pi\sigma_1\sigma_2} \exp\left(-\left[\frac{(X_p(1) - X_r(1) - \mu_p(1))^2}{2\sigma_p(1)^2} + \frac{(X_p(2) - X_r(2) - \mu_p(2))^2}{2\sigma_p(2)^2}\right]\right). \quad (4)$$

Put in words, to evaluate the quality of a pair of points X_p, X_r we form the displacement vector, $X_p - X_r$ between the location of the part X_p and the location of the root X_r, take the difference from the 'nominal' displacement μ_p and then penalize any deviations according to the standard deviation along the different coordinates, $(\sigma_p(1), \sigma_p(2))$.

The parameters μ_p, σ_p of the Gaussians are estimated with maximum likelihood, based on the leaf-root location offsets of the training data. Note that we assume the Gaussian to have a diagonal covariance matrix - i.e. the product of the first and second dimensions does not show up in the likelihood expression.

In order to integrate Eq. 4 in Eq. 3 we write:

$$\begin{aligned}
\psi_{p,r}(X_p, X_r) &= -\left[\frac{(X_p(1) - X_r(1) - \mu_p(1))^2}{2\sigma_p(1)^2} + \frac{(X_p(2) - X_r(2) - \mu_p(2))^2}{2\sigma_p(2)^2}\right] + c \\
&= -\frac{1}{2\sigma_p(1)^2}\left(X_p(1) - X_r(1)\right)^2 + 2\frac{\mu_p(1)}{2\sigma_p(1)^2}\left(X_p(1) - X_r(1)\right) \quad (5) \\
&\quad -\frac{1}{2\sigma_p(2)^2}\left(X_p(2) - X_r(2)\right)^2 + 2\frac{\mu_p(2)}{2\sigma_p(2)^2}\left(X_p(2) - X_r(2)\right) + c' \quad (6)
\end{aligned}$$

where $c = -\log(2\pi\sigma_1\sigma_2)$ and $c' = c - \frac{\mu_p(1))^2}{2\sigma_p(1)^2} - \frac{\mu_p(2))^2}{2\sigma_p(2)^2}$ are constants, independent of X_p, X_r. Observe that if $\sigma_p(1) = \sigma_p(2)$ the expression becomes proportional to the Euclidean distance between $X_p - X_r$ and μ_p; but in general $\sigma_p(1) \neq \sigma_p(2)$.

We can now get back to Eq. 3, and rewrite it as:

$$m_{p\to r}(X) = \max_{X_p} \phi_p(X_p) + a(X_p(1) - X_r(1)^2 + b(X_p(1) - X_r(1)) + c(X_p(2) - X_r(2))^2 + d(X_p(2) - X_r(2)) + e \quad (7)$$

where a, b, c, d, e can be determined in terms of μ_p, σ_p by identifying terms with Eq. 6. Generalized Distance Transforms [5] can perform the optimization in Eq. 7 efficiently by solving it in 'batch mode', namely for all values of X simultaneously. This is what the `dt` function performs.

Deliverables

0/3 Make the connection between what is described above and the code that you have been provided with.

1.5/3 Implement the message-passing computation along the lines of Eq. 3. Verify that the messages sent from the leaves (eyes, mouth corners) to the root (nose) are the same as the ones obtained using GDT.

- Show the messages sent from the parts to the root for the two implementations.
- Show the belief function computed at the root.

1/3 Implement root-to-leaves message passing to localize the leaf locations.

- Show the messages sent from the root to the parts.

.5/3 Figure out how to use the `ix,iy` outputs of the `dt` function to localize leaves.

Hint: use matlab's `imagesc` function to show the messages as heat maps.
Provide results on images [3,231,507].

2.2 Skipping Part 2

Just use the code provided in `main_dpms.m` for part combination.

3 Multi-scale Detection, Non-Maximum Suppression, Benchmarking [3/10]

So far your images were pre-processed so that faces would come at a normalized size; as you can see from the function `load_im.m`, if the `use_normalization` flag is on, the horizontal distance between two eyes is set to be constant.

This is meaningful during training, but will not be possible at test time; at that point, you will need to run your detector at multiple scales. The simplest way to do this is to rescale the given image at multiple resolutions and run the same detector on all of them. The response of your detector needs to be evaluated at all space and scale combinations.

1 (.5/3) Adapt the code developed in the the previous part so that you perform the (a) feature extraction, (b) unary score computation and (c) message passing operations at multiple scales. Use scales $[2^{-1}, 2^{-.5}, 1, 2^{.5}, 2^{1}]$.

2 (.5/3) Implement a function that takes as input an image and returns all object configurations scoring above a tunable threshold.

3 (.5/3) Visualize the bounding boxes corresponding to all objects scoring above a tunable threshold. See the code at the end of `second_deliverable` for a demonstration of how to visualize a bounding box.

4 (.5/3 Use function `nms.m` to implement nonmaximum suppression. Set the overlap threshold to 0.7. Show the bounding boxes before and after nonmaximum suppression.

Turning to making the evaluation more thorough, we need to assess the performance of a detector on a dataset of images in terms of precision-recall curves. Doing this will get you the third grade for this part.

On top of items [2-4] above, this requires identifying false-positives, true positives, and misses for a range of thresholds, forming the associated precision and recall values, and plotting the resulting PR curve.

This requires a criterion for deciding when a proposed bounding box is declared a false positive. The standard criterion is to measure the overlap between the candidate box and the ground truth box. If this is above a threshold the box is declared a positive, and a negative otherwise. The function `boxoverlap.m` implements this operation.

Furthermore, if two hypotheses are nearby and the overlap of both is above threshold, only one of them is declared positive, with the other being declared a false positive.

In specific, write a function, `compare.m` that takes as input the bounding boxes of a detector, their scores, a set of ground-truth bounding boxes, a number of thresholds on the score and returns, as a function of threshold, the number of

a True bounding boxes that were detected (true positives)

b True bounding boxes that were not detected (misses)

c Bounding boxes that were detected, but not true (false positives).

Use the `overlap.m` function with overlap ratio .7 to decide if two bounding boxes are the same.

Provide the precision-recall curve of your multi-scale detection system, using positive images [501-600] and negative images [101-200]. Use a set of thresholds large enough so that you will cover both the high-recall and the high-precision regimes.

Hint: you may want to take a look at `MLCV_evaldet.m`, which is a simplified version of the `VOC_evaldet.m` file in `VOCdevkit.zip`. The first function largely provides the functionality you need, while the zipfile contains evaluation code for the Pascal benchmark `http://pascallin.ecs.soton.ac.uk/challenges/VOC/`.

4 Detecting faces using CNNs

In this part of the assignment, we depart from the DPM paradigm and we try to build a system that leverages convolutional neural networks to detect faces. We will base our system on a CNN that has been pre-trained for the task of image classification [6]. Instead of devising a network that directly outputs bounding boxes for faces, we treat this task as a pixel-wise labelling problem.

More specifically, given an input image, our goal is to assign a foreground (face) or background (non-face) label to each one of the pixels. This can be accomplished by replacing the last fully-connected layers, with convolutional layers, resulting in a a *fully convolutional* neural network (FCN), which produces a dense response map. Performance can be assessed in two ways: a) pixel-wise label accuracy; b) by extracting bounding boxes of connected components of labels in the image, and comparing their overlap to ground truth boxes (more information on that later).

To reduce computational demands, we *finetune* the network, keeping the weights of the first layers fixed and updating only the weights for the newly introduced convolutional layers. You are given a function `cnn_finetune.m` that you can use to train CNNs. All the extra functions that are needed can be included or completed in the same file.

1/3 Fill in the missing code in function `cnn_init` that determines the network architecture. Keep all Alexnet convolutional layers up to *conv3* and append new convolutional layers. The last conv layer should have a 1×1 kernel size, and produce a $H_s \times W_s \times L$ response, where H_s, W_s are the height and width of the (subsampled) output and L is the number of different class labels. For the purposes of this assignment, $L = 2$ (face and non-face).

Train networks in two different settings, changing the `backPropDepth` parameter accordingly:

a Keeping weights for layers up to *conv3* fixed, updating weights only for the newly introduced layers.

b Updating all weights, using a learning rate for the Alexnet-initialized layers that is $0.1\times$ the learning rate of the new, randomly initialized layers. This forces weights in the early layers to change at a slow rate, and makes the new layers responsible for adapting the network to the new task. You can also experiment with different architectures that you think will better suit the pixel-wise labelling task, training the full network using random initialization.

As mentioned, for a $H \times W \times 3$ RGB input image, the FCN produces a $H_s \times W_s \times L$ score output, where $H_s < H$ and $W_s < W$. This subsampled output can be upscaled into a dense score map for the original input image, through up-sampling with bilinear interpolation, and then turned into a $H \times W \times L$ probability

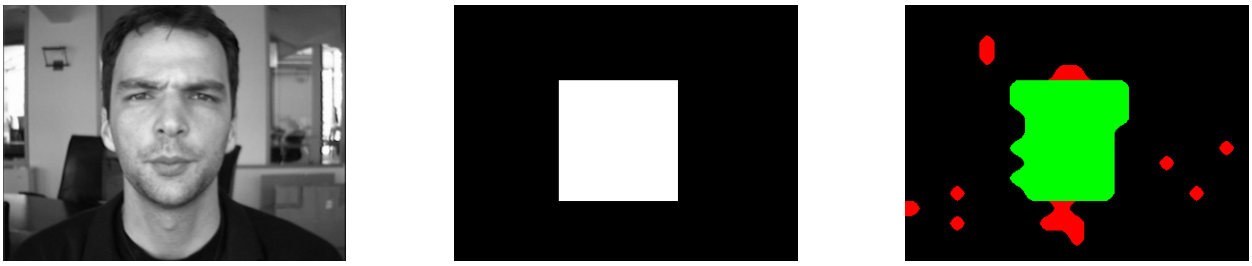

Figure 1: **Left:** Original image. **Middle:** Ground truth mask for "face", extracted from keypoint annotations. **Right:** Pixel-wise probability map for "face" produced by CNN (after thresholding). Green corresponds to true positives and red to false positives.

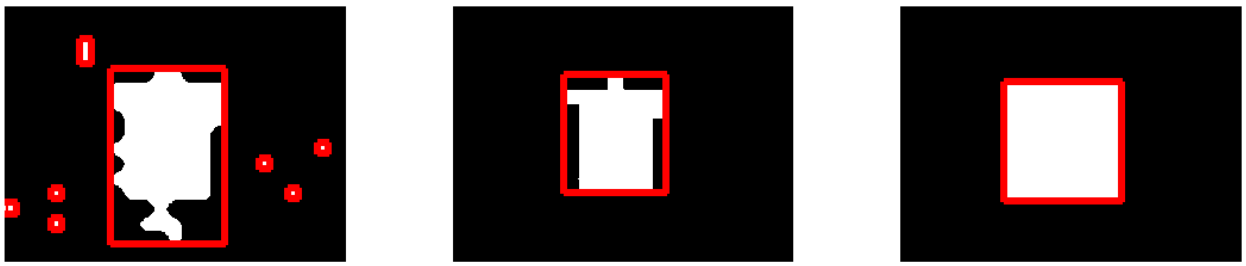

Figure 2: **Left:** Example of connected components at a low probability threshold and their respective bounding boxes. **Middle:** Connected components at a higher probability threshold and their respective bounding boxes. Note that detections with lower "face" probability have been eliminated. **Right:** Groundtruth mask and respective bounding box.

map using the *softmax* operation for three-dimensional inputs:

$$SM(X(i,j,k)) = \frac{e^{X(i,j,k)}}{\sum_{t=1}^{D} e^{X(i,j,k)}} \tag{8}$$

Usually the following, numerically stabler version of softmax is used:

$$SM(X(i,j,k)) = \frac{e^{X(i,j,k)-\max_d X(i,j,d)}}{\sum_{t=1}^{D} e^{X(i,j,k)-\max_d X(i,j,d)}} \tag{9}$$

Using Matlab notation, $SM(:,:,l)$ is the 2D probability map for class l. Softmax is particularly suited for multi-class problems but it can used in our two-class (face/non-face) problem as well.

2/3 Write a function `softmax.m` that computes the stable version of softmax. Complete the function `cnn_test` (contained in `cnn_finetune`). `cnn_test` should do the following:

a Compute output responses on all images in the test dataset and use `softmax` function to turn them into probabilities.

b Assess performance of the trained network in terms of pixel-wise labelling accuracy, using the precision-recall framework. Threshold the probability maps for "face" using different threshold values, compare to the ground-truth segmentation masks for each image, and plot precision-recall curves. A comparison example is given in Fig. 1.

c Assess performance of your trained network in terms of precision-recall, using the bounding box overlap criterion, as in Sec. 3. Since there will possibly be "face" positives at multiple locations in the image, you can extract a bounding box for each connected component, as shown in Fig. 2. Extract bounding boxes at different threshold values and plot precision-recall curves.

Hints:
- Use the matlab function `bwconncomp` to extract connected components on a thresholded probability map.
- Write a function `mask2bbox` that computes the endpoint coordinates $(x_{min}, y_{min}, x_{max}, y_{max})$ of the bounding box of a binary mask region.

References

[1] P. Felzenszwalb, D. Huttenlocher *Pictorial Structures for Object Recognition* International Journal of Computer Vision, Vol. 61, No. 1, January 2005.

[2] P. Felzenszwalb, R. Girshick, D. McAllester, D. Ramanan. *Object Detection with Discriminatively Trained Part Based Models.* IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol. 32, No. 9, September 2010

[3] W. T. Freeman and E H. Adelson, *The Design and Use of Steerable Filters.* IEEE Transactions on Pattern Analysis and Machine Intelligence, 1991.

[4] Navneet Dalal and Bill Triggs. *Histograms of Oriented Gradients for Human Detection.* IEEE Conference on Computer Vision and Pattern Recognition (CVPR), vol. II, pages 886-893, June 2005.

[5] Pedro F. Felzenszwalb and Daniel P. Huttenlocher *Distance Transforms of Sampled Functions.* Cornell Computing and Information Science TR2004-1963

[6] Alex Krizhevksy, Ilya Sutskever and Geoffrey Hinton *Imagenet classificati9on with deep convolutional neural networks.* Advances in neural information processing systems