CVPR 2023, Projects assignments

Felice Andrea Pellegrino

January 2, 2024

Contents

Introduction

You are required to complete one of the following projects. You can work in group (up to three members); in that case, however, each group's member is responsible of the whole project and is expected to be aware of all the details of the work done by the group. The choice of the environment/language is free. A report with problem statement, description of the approach, implementation choices, results and references must be written and sent to the instructor (*fapellegrino@units.it*) along with a compressed folder containing the code (or, preferably, a reference to a repository where the code can be accessed).

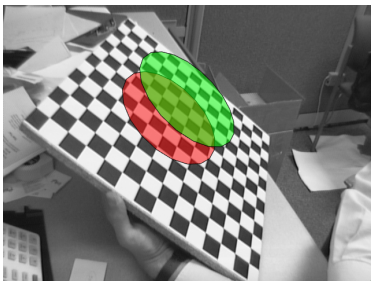

Figure 1: Superimposing a cylinder to a calibration image.

Project 1 (Calibration)

This project is based on the first lab lecture (see the Python starter code and images at the URL: `http://tinyurl.com/CVPR2023PA`). The starter code estimates homographies. Things to do:

1. calibrate using Zhang procedure, i.e., find the intrinsic parameters K and, for each image, the pair of R, t (extrinsic);
2. choose one of the calibration images and compute the total reprojection error (Lecture 9, page 45) for all the grid points (adding a figure with the reprojected points);
3. superimpose an object (for instance, a cylinder as in Fig. 1), to the calibration plane, in **all** the images employed for the calibration;
4. optionally, you could print a checkerboard and apply the previous steps to a set of images acquired with your own smartphone, webcam or digital camera (thus, calibrating your device);
5. optionally, you could add radial distortion compensation to the basic Zhang's calibration procedure;
6. optionally, you could compute the total reprojection error with radial distortion compensation and make a comparison to the one without compensation.

Notes

Camera calibration libraries are not allowed. Basically, you must implement everything by yourself. You can however, employ functions for detecting checkerboard corners such as OpenCV's `findChessboardCorners()`.

Figure 2: Examples of images from each of the 15 categories of the provided dataset (the same as [Lazebnik et al., 2006]).

Project 2 (Bag-of-words classifier)

This project requires the implementation of an image classifier based on the bag-of-words approach. The provided dataset (from [Lazebnik et al., 2006]), contains 15 categories (office, kitchen, living room, bedroom, store, industrial, tall building, inside city, street, highway, coast, open country, mountain, forest, suburb), and is already divided in training set and test set. Samples are shown in Fig. 2.
You are expected to:

1. build a visual vocabulary:
   - sample many (10K to 100K) SIFT descriptors from the images of the training set (you either use a detector or sample on a grid in the scale-space);
   - cluster them using k-means (the choice of the number of clusters is up to you, and you should experiment with different values, but you could start with a few dozens);
   - collect (and save for future use) the clusters' centroids which represent the k 128-dimensional visual words.
2. Represent each image of the training set as a normalized histogram having k bins, each corresponding to a visual word; a possibility is to perform a rather dense sampling in space and scale; another possibility is to use the

SIFT *detector* to find the points in scale-space where the descriptor is computed. In any case, each computed descriptor will increase the value of the bin corresponding to the closest visual word.

3. Employ a nearest neighbor classifier and evaluate its performance:
   - compute the normalized histogram for the test image to be classified;
   - assign to the image the class corresponding to the training image having the closest histogram.
   - repeat for all the test images and build a confusion matrix.
4. Train a multiclass linear Support Vector Machine, using the one-vs-rest approach (you will need to train 15 binary classifiers having the normalized histograms as the input vectors and positive labels for the "one" class and negative for the "rest.")
5. Evaluate the multiclass SVM:
   - compute the normalized histogram for the test image to be classified;
   - compute the real-valued output of each of the SVMs, using that histogram as input;
   - assign to the image the class corresponding to the SVM having the greatest real-valued output.
   - repeat for all the test images and build a confusion matrix.
6. Optionally, you could train the SVM using a generalized Gaussian kernel (Lecture 7) based on the χ^2 distance;
7. optionally, you could train the SVM using a generalized Gaussian kernel based on the earth mover's distance (EMD);
8. optionally, you could implement the multiclass SVM using the Error Correcting Output Code approach [Dietterich and Bakiri, 1994, James and Hastie, 1998];
9. optionally, you could use soft assignment to assign descriptors to histogram bins. Each descriptor will contribute to multiple bins, using a distance-based weighting scheme (see [Van Gemert et al., 2008]);
10. optionally, you could add some spatial information by taking inspiration from the spatial pyramid feature representation of [Lazebnik et al., 2006] (a simple approach could be that of building an *extended descriptor*, by stacking the histogram of the whole image, the histograms of the 4 quadrants and so on, up to a desired level).

Notes

Libraries implementing the whole bag-of-words pipeline are not allowed. You can, however, employ off-the-shelf functions for keypoint detection/description, k-means clustering, SVMs, linear programming (for EMD computation).

Table 1: Layout of the CNN to be used in Project 3

#	type	size
1	Image Input	64×64×1 images
2	Convolution	8 3×3 convolutions with stride 1
3	ReLU	
4	Max Pooling	2×2 max pooling with stride 2
5	Convolution	16 3×3 convolutions with stride 1
6	ReLU	
7	Max Pooling	2×2 max pooling with stride 2
8	Convolution	32 3×3 convolutions with stride 1
9	ReLU	
10	Fully Connected	15
11	Softmax	softmax
12	Classification Output	crossentropyex

Project 3 (CNN classifier)

This project requires the implementation of an image classifier based on convolutional neural networks. The provided dataset (from [Lazebnik et al., 2006]), contains 15 categories (office, kitchen, living room, bedroom, store, industrial, tall building, inside city, street, highway, coast, open country, mountain, forest, suburb), and is already divided in training set and test set. Samples are shown in Fig. 2.

You are required to:

1. train a shallow network from scratch according to the following specifications:
   - use the network layout shown in Table 1;
   - since the input image is 64×64 you will need to resize the images in order to feed them to the network; follow the simple approach of rescaling the whole image independently along x and y to get the proper size (an example of such anisotropic rescaling is shown in Fig. 3)). Other approaches exist, see below about the optional improvement of data augmentation;
   - split the provided training set in 85% for actual training set and 15% to be used as validation set;
   - employ the *stochastic gradient descent with momentum* optimization algorithm, using the default parameters of the library you use, except for those specified in the following;
   - use minibatches of size 32;
   - set the initial bias values to 0 and use initial weights drawn from a Gaussian distribution having a mean of 0 and a standard deviation

of 0.01[1];

- by choosing an appropriate learning rate, and training for a few dozen epochs, you should be able to obtain an overall test accuracy of around 30% [2];
- report and discuss the plots of loss and accuracy during training, for both the training set and the validation set;
- comment on the criterion you choose for stopping the training;
- report the confusion matrix and the overall accuracy, both computed on the test set.

2. Improve the previous result, according to the following suggestions (not all the following will necessarily result in an increase of accuracy, but you should be able to obtain a test accuracy of about 60% by employing some of them):

   - data augmentation: given the small training set, data augmentation is likely to improve the performance. For the problem at hand, left-to-right reflections are a reasonable augmentation technique. By augmenting the train data using left-to-right reflections you should get an accuracy of about 40%;
   - batch normalization [Ioffe and Szegedy, 2015]: add batch normalization layers before the reLU layers;
   - change the size and/or the number of the convolutional filters, for instance try increasing their support as we move from input to output: 3×3, 5×5, 7×7;
   - play with the optimization parameters (learning rate, weights initialization, weights regularization, minibatch size ...); you can also switch to the *adam* optimizer;
   - dropout: add some dropout layer to improve regularization;
   - employ an *ensemble of networks* (five to ten), trained independently. Use the arithmetic average of the outputs to assign the class, as in [Szegedy et al., 2015].

   Comment on any significant change you notice after the application of the previous modifications.

3. Use transfer learning based on a pre-trained network, for instance AlexNet [Krizhevsky et al., 2012], in the following two manners (in both of the cases you should get a test accuracy above 85%):

   - freeze the weights of all the layers but the last fully connected layer and fine-tune the weights of the last layer based on the same train and validation sets employed before;

[1] note that such initialization may be inadequate if the input values do not fall within the range [0,255], thus make sure to avoid, for instance, normalization.

[2] a random classifier for the same problem has an accuracy of less than 7%.

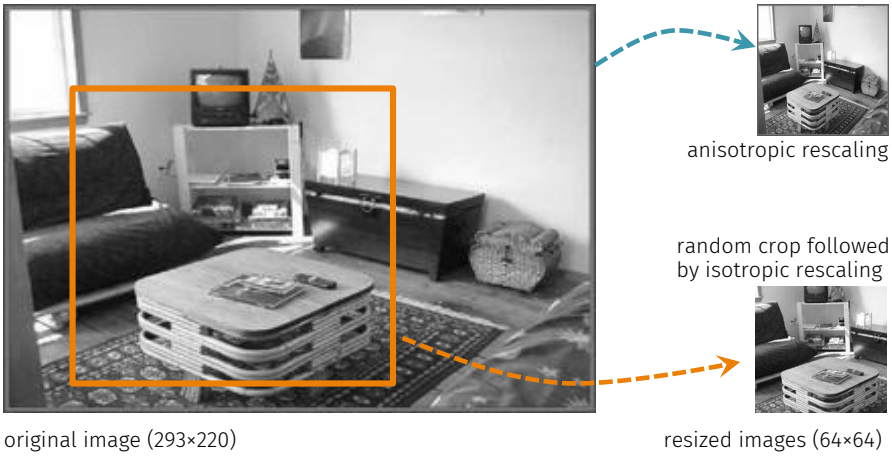

Figure 3: Two ways for resizing the images.

- employ the pre-trained network as a feature extractor, accessing the activation of an intermediate layer (for instance, one of the fully connected layers) and train a multiclass linear SVM. For implementing the multiclass SVM use any of the approaches seen in the lectures, for instance DAG.

4. Optionally, in tasks 2 and 3, you could improve data augmentation, adding to the left-right reflection a random cropping of a sufficiently large region, followed by small rotations and rescaling (an example of crop and rescaling is shown in Fig. 3);

5. optionally, in task 2, you could add more convolutional and/or fully-connected layers;

6. optionally, in task 3, you could employ nonlinear SVMs;

7. optionally, in task 3, you could implement the multiclass SVM using the *Error Correcting Output Code* approach [Dietterich and Bakiri, 1994, James and Hastie, 1998].

Notes

- Carry out task 1 exactly as required, in order to obtain a baseline for the subsequent tasks;

- if you use PyTorch, please note that `nn.CrossEntropyLoss()` already incorporates a softmax operation. Therefore, there's no need to employ a `nn.Softmax()` layer, as doing so would introduce redundant non-linearity

by stacking two softmax operations in sequence. While in many cases, this redundancy might not hinder learning, in the current scenario, it could compromise the effectiveness of the prescribed weight initialization, leading to ineffective learning;

- apply the data augmentation only to the train fraction of the training images. In other words, do not augment the validation and test sets;
- independent of the data augmentation techniques, for validation and test use only the anisotropic rescaling show on top of Fig. 3 (squeeze the whole image to the required size).

References

[Dietterich and Bakiri, 1994] Dietterich, T. G. and Bakiri, G. (1994). Solving multiclass learning problems via error-correcting output codes. *Journal of artificial intelligence research*, 2:263–286.

[Ioffe and Szegedy, 2015] Ioffe, S. and Szegedy, C. (2015). Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*.

[James and Hastie, 1998] James, G. and Hastie, T. (1998). The error coding method and picts. *Journal of Computational and Graphical Statistics*, 7(3):377–387.

[Krizhevsky et al., 2012] Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105.

[Lazebnik et al., 2006] Lazebnik, S., Schmid, C., and Ponce, J. (2006). Beyond bags of features: Spatial pyramid matching for recognizing natural scene categories. In *2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'06)*, volume 2, pages 2169–2178.

[Szegedy et al., 2015] Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V., and Rabinovich, A. (2015). Going deeper with convolutions. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

[Van Gemert et al., 2008] Van Gemert, J. C., Geusebroek, J.-M., Veenman, C. J., and Smeulders, A. W. (2008). Kernel codebooks for scene categorization. In *European conference on computer vision*, pages 696–709. Springer.