Computer Vision 1 - Final Project Part 2
Convolutional Neural Networks for Image Classification
20 Points

March 14, 2017

All the files should be zipped and sent to **computervision1.uva(at)gmail.com** before **03-04-2017** , **23.59** (Amsterdam Time).

1 Introduction

This part of the assignment makes use of Convolutional Neural Networks(CNN) [7].Previous part makes use of hand-crafted features like SIFT to represent images, then trains a classifier on top of them. In this way, learning is a two-step procedure with image representation and learning. The method used here instead *learns* the features jointly with the classification.

In this Assignment, we first describe the installation procedure, then proceed to experiment and evaluation. You will be working on the skeleton code you are provided and fill in the blanks.

2 Installation and Setup

The researchers have proposed different tools to train and evaluate CNNs. These include Caffe [4], Tensorflow [1], Theano [2] and MatConvNet [11]. Among them, MatConvNet is developed in C/C++ with a Matlab interface. As throughout the course we make use of Matlab as the programming environment, we use MatConvNet in this part of the assignment.

MatConvNet: It is recommended to go and follow the installation instructions available on `http://www.vlfeat.org/matconvnet/install/` according to your own setting. Once the setup is complete, it is beneficial to go through the example scripts provided under **examples** directory of MatConvNet and try to run them, especially **cnn_cifar.m**.

Liblinear: We provide LIBLINEAR library in the directory. You need to extract the zip file and run *make.m* script under *liblinear/matlab* directory, so that it will be compiled and ready to use.

3 The Tasks

Training CNNs roughly consists of three parts: (i) Defining the network architecture, (ii) Preprocessing the data, (iii) Feeding the data to the network and updating the parameters.

Defining the network architecture consists of choosing an appropriate set of functions that are connected to each other in the network. A network generally has three parts, namely **input**, **mapping** and **output** parts. Input part takes the image, applies some transformation and gives it to the mapping part. Mapping part is responsible for transforming the data in a way to capture task specific features from the data. Finally, the output part performs the classification task.

Data preprocessing consists of converting the images to appropriate data type (single), resizing input images to the input size of the network input, and subtracting the mean of the dataset images from each image.

Data feeding is the main part where the training is achieved. In this part, a subset of images are applied on the network, the classification error is measured, and the parameters of the network is updated accordingly.

3.1 Understanding the Network Architecture

In this part of the assignment, we provide an already trained network for the initial experiments. The task is to understand the architecture of the given network named **pre_trained_model.mat** under **data** directory. The answers to those questions are expected:

1. Do you observe any pattern in the architecture of the network? If so, describe it in your own words.
2. Which part of the network has the most parameters and the biggest size?

Hint. It is suggested to use **vl_simplenn_display.m** to visualize the network architecture.

3.2 Preparing the Input Data

After understanding the network architecture, the next step is to prepare the input data for the network. Every toolkit accepts the input data in a certain structure.

Let **imdb** denote the input struct to be given to MatConvNet. **imdb** has the following data members:

- **imdb.images**: This member includes all of the necessary information about the dataset: the images, their labels, and their splits. This should include three submembers, namely **data**, **labels** and **set**. Let D denote the dimension.
  - **imdb.images.data** is a $4D$ matrix with size: $(image\_height, image\_width, num\_channels, num\_images)$ where num denotes the number.

  - **imdb.images.labels** is a $1D$ vector where each label indicates which class the image belongs to, in range $[1, 4]$ in our case for *airplanes(1), cars(2), faces (3), motorbikes(4).* Make sure you assign proper class indices to the images.
  - **imdb.images.set** is a $1D$ vector indicating whether the image is from the training ($== 1$) or the testing ($== 2$) set.

- **imdb.meta** includes all necessary side information about the training classes and splits. It can have two submembers where each is a cell, named **sets** and **classes** respectively. **sets** includes the name of the splits, which is **train** and **validation** in our case. **classes** includes the name of the classes, in the same order provided in **imdb.images.labels**, which is *1. airplanes, 2. cars, 3. faces, 4. motorbikes.*

The task is to implement **getCaltechIMDB** function under **finetunecnn.m** script. This function reads images, their labels (*1. airplanes, 2. cars, 3. faces, 4. motorbikes*), and their splits (either *training* or *validation*) from the **data** directory and creates **imdb** struct described above.

3.3 Updating the Network Architecture

The provided network is trained on a different dataset named CIFAR-10 [6], which contains the images of 10 different object categories, each of which has $32 * 32 * 3$ dimensions. The dataset we use throughout the assignment is a subset of Caltech-101 [3] with larger sizes and different object classes. So, there is a discrepancy between the dataset we use to train (CIFAR-10) and test (Caltech-101) our network. One solution would be to train the whole network from scratch. However, the number of parameters are too large to be trained properly with such few number of images provided. One solution is to shift the learned weights in a way to perform well on the test set, while preserving as much information as necessary from the training class. This procedure is called transfer learning [5, 9] and have been widely used in the literature. This is also called **fine-tuning**, where the weights of the pretrained network changes gradually. To perform this task, one way is to use the same architectures in all layers except the output layer, as the number of output classes changes (from 10 to 4).

In this part of the assignment, you are expected to define the network structure. The architecture of the pre-trained network is provided in **update_model.m** script. Specifically, it is expected to update the output layer in a way that it is trained for classifying images of 4 different classes (rather than 10). You need to set NEW_INPUT_SIZE and NEW_OUTPUT_SIZE accordingly in Block-5 of the network architecture.

3.4 Setting up the Hyperparameters

There are different parameters that must be set properly before training CNNs. These parameters shape the training procedure. They determine how many images to be processed at each step, how much the weights of the network will be updated, how many iterations will the network run until convergence. These parameters are called **hyperparameters** in the machine learning literature.

You are given a set of hyperparameters and default values to begin with, on top of **update_model.m** script. These include:

1. Learning rate for previous layers (**lr_prev_layers**)
2. Learning rate for updated layer (**lr_new_layers**)
3. Weight decay (**weightDecay**)
4. Batch size (**batchSize**)
5. Number of epochs (**numEpochs**)

In this part, the task is to experiment with different parameter settings and pick up the best setting. Specifically:

1. Set batch size to $[50, 100]$
2. Set number of epochs to $[40, 80, 120]$

and note all the results with all the parameter settings.

4 Experiments

Once the input data is prepared, the network architecture is updated, and best hyperparameter setting is found, you are ready to experiment with training and evaluation. This can be achieved via running the provided **main.m** script. This script updates the network, trains it on the dataset provided, extract features from a certain layer using both pretrained and fine-tuned network. Then, these features are used for evaluation.

4.1 Feature Space Visualization

Once the network is trained, it is a good practice to understand the feature space by visualization techniques. There are several techniques to visualize the feature space. In this part, you are expected to use **t-sne** [8], which is a dimensionality reduction method. **t-sne** takes as input features and labels, then reduces their dimensions to 2, where the structure of the feature and label space is preserved as much as possible. The Matlab script for computing **t-sne** can be obtained here: `https://lvdmaaten.github.io/software/#t-sne`

In this step, you need to provide the visualizations for:

1. Features of the pre-trained network on the provided dataset
2. Features of the fine-tuned network on the provided dataset

It is necessary to comment on the differences and similarities between the visualization of different settings described above.

4.2 Evaluating Accuracy

In this step, you are expected to measure the accuracy of 3 different configurations:

1. The classification accuracy of SVM using pretrained features
2. The classification accuracy of SVM using fine-tuned features
3. The classification accuracy of fine-tuned CNN

Remember that CNN can also be used for classification, where the output layer gives the predictions about the input images. The results obtained in this part should be compared against the results obtained with **Bag-of-Words** approach.

5 Bonus

You can experiment with the following ideas to get bonus points for this assignment:

1. **[Up to 2 pts.] Data augmentation**. CNNs tend to work well when there is more data available, however it is not always possible to get more labelled examples. So people try to augment the data by using different kinds of tricks like **rotation, scaling** etc. Up until now, you used a simple for of rotation (flip) using **getSimpleNNBatch** code we provide under **finetunecnn**. You can also define different operations to increase the number of instances you use using data augmentation tricks.
2. **[0.5 pts.] Freezing early layers**. When you look at the default setting and parameters we provide, you observe that learning rate is different for early layers and the layer we fine-tune and randomly initialize. One way to achieve fine-tuning is also freezing completely early layers and only training the last layer. You can achieve this by simply setting the learning rate for early layers to 0. You can report the accuracy you obtained.
3. **[Up to 2 pts.] Deeper networks**. The network we experiment with for this assignment has few number of layers. One trick with CNNs to get a better accuracy so far has been to increase the depth (of number of layers) to increase the expressivity of the network. You can look for already trained models like **VGG [10], AlexNet [6]** etc. to extract features and report results for the given task. You can find a list of pre-trained models here: `http://www.vlfeat.org/matconvnet/pretrained/`
4. **[Up to 5 pts.]** Any other thing that you can think of or find from the literature that can boost the results.

In order to get the bonus, your submission should first satisfy the previous steps. You need to implement and explain the method and justify your reasoning. You can get at most 5 points from bonus part.

References

[1] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, et al. Tensorflow: Large-scale machine learning on heterogeneous distributed systems. *arXiv preprint arXiv:1603.04467*, 2016.

[2] R. Al-Rfou, G. Alain, A. Almahairi, C. Angermueller, D. Bahdanau, N. Ballas, F. Bastien, J. Bayer, A. Belikov, A. Belopolsky, et al. Theano: A python framework for fast computation of mathematical expressions. *arXiv preprint arXiv:1605.02688*, 2016.

[3] G. Griffin, A. Holub, and P. Perona. Caltech-256 object category dataset. 2007.

[4] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell. Caffe: Convolutional architecture for fast feature embedding. In *Proceedings of the 22nd ACM international conference on Multimedia*, pages 675–678. ACM, 2014.

[5] S. Karayev, M. Trentacoste, H. Han, A. Agarwala, T. Darrell, A. Hertzmann, and H. Winnemoeller. Recognizing image style. *arXiv preprint arXiv:1311.3715*, 2013.

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

[7] Y. LeCun, Y. Bengio, and G. Hinton. Deep learning. *Nature*, 521(7553):436–444, 2015.

[8] L. v. d. Maaten and G. Hinton. Visualizing data using t-sne. *Journal of Machine Learning Research*, 9(Nov):2579–2605, 2008.

[9] A. Sharif Razavian, H. Azizpour, J. Sullivan, and S. Carlsson. Cnn features off-the-shelf: an astounding baseline for recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, pages 806–813, 2014.

[10] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[11] A. Vedaldi and K. Lenc. Matconvnet: Convolutional neural networks for matlab. In *Proceedings of the 23rd ACM international conference on Multimedia*, pages 689–692. ACM, 2015.