Exercise 4: Convolutional Neural Networks

Convolutional neural networks (CNNs) have outperformed classical hand-crafted features in a multitude of vision tasks over the last few years. In this exercise you will use a simple CNN to tackle the scene classification task. You will be asked to train a Toy CNN by yourself (Figure 1), adding a few commonly used techniques to further improve performance.

You will use the MatConvNet framework\(^1\) throughout the exercise to define, train, and test CNNs. To setup, run the base code /code/ex4.m after selecting show_init to true. This will evoke the setup for MatConvNet, as well as showing the initial performance of the Toy CNN (Figure 1).

We provide a benchmark dataset to work with\(^2\). The dataset is already contained in the /data/15SceneData folder. Note that the data is split into training and testing sets. The test data should never be used for training the CNN. The training set consists of 1500 images for 15 different scene categories.

![Figure 1: Toy CNN: our baseline system.](image)

As mentioned above, you will improve our Toy CNN by adding a few of common techniques. The first technique is the data augmentation via jittering (Question 1). The second technique is data normalization via mean subtraction (Question 2). You will then implement dropout, one of the widely used techniques for reducing overfitting of deep neural networks (Question 3). In Question 4, you will expand the capacity of our Toy CNN by adding more layers; you will also have to tweak some hyperparameters to maximize the performance.

The optional bonus question asks you to fine-tune a pretrained CNN. As training a deep neural network from scratch (random initialization) could take days and weeks, it is a common practice nowadays to fine-tune a network that has already been trained on either different data or different task – e.g. there exist many publicly available pretrained networks trained on ImageNet (ILSVRC 2012) for object classification task\(^3\).

**Quick Start Outline**

First make sure that MatConvNet is working. Running ex4.m or quickStartDemo.m will set up the framework automatically. This exercise is meant to be done by CPU only.

**Question 1: Data Augmentation (5 points)**

As you have learned during the lecture, representation learning (feature learning) typically needs a lot more images than the shallower counterparts (e.g. linear SVM) to reach its full potential. Although usually not as effective as feeding new training samples, data augmentation via jittering is known to help improve performance. Specifically, you synthetically increase our amount of training data by transforming (jittering) the image – e.g. left-right flip. Note that even if you left-right flip (mirror) an image of a scene, its category is unchanged.

---

\(^1\)http://www.vlfeat.org/matconvnet

\(^2\)http://www-cvr.ai.uiuc.edu/ponce_grp/data/scene_categories/scene_categories.zip

\(^3\)https://github.com/BVLC/caffe/wiki/Model-Zoo
You should implement the flipping in the `getBatch_1.m` function in the lines indicated by “Supplement Code Here”.

If you have implemented the flipping correctly, you should see about \( \sim 10\% \) increase in accuracy. You may notice that your training and test error don’t drop as quickly as before; you may need more training epochs, or you might have to try modifying the learning rate variable `opts.learningrate` (default value 0.0001) in `cnn_train.m`.

a) Implement left-right mirroring of training images during the learning process. (3 points)
b) Analyze the effect applying the left-right mirroring based on the training/validation loss and error plots. (2 points)

**Question 2: Mean Image Subtraction (5 points)**

A widely used pre-processing step for many machine learning algorithms is input normalization (zero mean, unit norm). In CNNs, a common practice is to subtract the “mean image” from all the training images, such that each input pixel has zero mean over the training set. As a result of mean image subtraction, you should see around \( \sim 15\% \) increase in accuracy.

a) Modify `setup_data_2.m` so that it first computes the mean image and then subtracts the mean from all images before returning the data `imdb`. Insert code at “Supplement Code Here”. (3 points)
b) Which data split should be used for computing the mean image? – training, test, or both? (1 point)
c) Analyze the effect of mean image subtraction. (1 point)

**Question 3: Dropout (5 points)**

Overfitting is a serious problem in deep neural networks due to the large number of parameters. Dropout is a simple yet powerful technique for addressing this problem. In this question, you will insert a dropout layer in the Toy CNN. As a result, you should see around 10% increase in test accuracy.

a) How does the dropout technique address the problem of overfitting? (1 point)
b) At test time, do we use the dropout layer and why? (1 point)
c) Insert a dropout layer in `cnn_init_3.m` at Supplement Code Here with the dropout rate of 0.5. Your test accuracy should increase by \( \sim 10\% \). Please try three different dropout rates 0.25, 0.5, 0.75 and analyze the plots. (3 points)

**Question 4: CNN with more Layers and Tricks (10 points)**

The Toy CNN that we are using has only 5 layers plus the input layer. Popular CNNs such as AlexNet\(^4\) or VGG-Net\(^5\) have more layers. In this question, you will expand the capacity of the Toy CNN by adding more layers as shown in Fig. 3. The corresponding file that you need to modify is `cnn_init_4.m`.

---


a) In the toy CNN, the convolution operation covers a window of 9x9 with a stride of 1, while the subsequent max pooling has a size of 7x7 and a stride 7. Change the receptive field of the convolution to 5x5 and of the max pooling operation to 3x3, while changing the stride of the max pool to 3 (like in Fig. 3).

b) Add an additional convolutional layer, max-pool layer and relu layer after the existing relu layer with the filter size and the stride size shown in Fig. 3 and comment on your findings. (5 points)

Note that the function `vl_simplenn_display(net, inputSize[64 64 1 50])` can help you check the structure of the network with its parameters. It is called automatically by running `cnn_init4.m` and you don’t have to change this function.

c) Replace the non-linear activation function by sigmoid in `cnn_init4.m` and report your observations. (2 points)

**Question 5: Fine-tuning a pre-trained deep network (10 Bonus Points)**

Training a deep neural network requires a lot of training data and training time. However, it is very often that one doesn’t have enough data to train a deep network from scratch (random initialization). There are two common practice to address this problem:

(1) A simple yet effective practice is to use some deep network that is trained using a large benchmark dataset (e.g. ImageNet) as a feature extractor. Then the extracted features can be fed into a classical classification method like SVM, logistic regression, etc.

(2) Fine-tuning is another common practice for training deep neural networks with few samples. It means that you initialize the weights of you network from a pre-trained model that are trained on either different data or different tasks. It is particularly effective than training from scratch (random initialization) when the training data is limited. In this question, you will implement the first method in part (a) and the second method in part (b).

a) In this part, we are going to use a pretrained VGG-F model. The VGG-F network has 4096 activations in the "fc7" layer. Extract those activations as your features and train a classifier (a linear SVM) to classify the scene categories. Implement your code in `ex4_bonus.m` as well as in `LinearSVMclassifier.m` at **Supplement Code Here**. (5 bonus points)

b) Alternatively, you can fine-tune the existing classification network (VGG-F model) for the scene classification task by changing the number of outputs of the last fully connected layers. You should be able to get the plot like Fig. 2 (a) which shows 90% accuracy on the test set. If you visualize the feature maps of the first convolutional layer, you will see an image like Fig. 2 (b). (5 bonus points)

Note that running the code `ex4_bonus.m` will automatically download the VGG-F model.

**Write up**

Write a summary of your observations and describe the results with/without each modification of the network. No points will be deducted if your accuracies are lower than the numbers mentioned in the exercise sheet.

*Please turn in your solution by sending an email (adding [hlcv-ex4] at the beginning of the subject) to Alina Dima <aldisma@mpi-inf.mpg.de> including all relevant m-files before Wednesday, June 14th, 23:59.*