SAPIENZA
UNIVERSITY OF ROME

Advanced Machine Learning
Summer Semester 2020

Prof. Fabio Galasso <galasso@di.uniroma1.it>

Exercise 3: Convolutional Networks

In exercise 3, you will implement a convolutional neural network to perform image classification and explore methods to improve the training performance and generalization of these networks.

We will use the CIFAR-10 dataset as a benchmark for out networks similar to the previous exercise. This dataset consists of 50000 training images of 32x32 resolution with 10 object classes, namely airplanes, cars, birds, cats, deer, dogs, frogs, horses, ships, and trucks. The task is to implement convolutional network to classify these images using the PyTorch library. The four questions are

- Implement the ConvNet, train it and visualizing its weights(Question 1).
- Experiment with batch normalization and early stopping (Question 2).
- Data augmentation and dropout to improve generalization (Question 3).
- Implement transfer learning from a ImageNet pretrained model. (Question 4).

Questions 1-3 are based on the script `ex3_convnet.py` and question 4 is based on the script `ex3_pretrained.py`. To download the CIFAR-10 dataset execute the script `datasets/get datasets.sh` or set the `download` flag to `True` in the `torchvision.datasets.CIFAR10` function call.

Questions indicated by `report` should be answered in a separate report file - preferably a pdf format file.

The completed exercise should be handed in as a `zip`, `tar.gz` or `rar` format which compresses all the code + the `report`.

Submit it by sending an email to Prof. Fabio Galasso `galasso@di.uniroma1.it` including the report and the completed script files by Wednesday May 6th, 23:59.

Question 1: Implement a Convolutional Neural Network (10 points)

In this question we will implement a five-layered convolutional neural network architecture as well as the loss function to train it. Starting from the main file `ex3_convnet.py`. Refer to the comments in the code to the exact places where you need to fill in the code.

a) Our architecture is shown in Fig. 1. It has five convolution blocks. Each block consists of convolution, max pooling and ReLU operations, in this order. We will use 3x3 kernels in all convolutional layers. Set the padding and stride of the convolutional layers so that they maintain the spatial dimensions. Max pooling operations are done with 2x2 kernels, with a stride of 2, thereby halving the spatial resolution each time. Finally, stacking these five blocks leads to a 512 x 1 x 1 feature map. Classification is achieved by a fully connected layer. We will train convolutional neural networks on CIFAR-10 dataset. Implement a class `ConvNet` to define the model described. The ConvNet takes 32x32 color images as inputs and has 5 hidden layers with 128, 512, 512, 512, 512 filters, and produces 10-class classification. The code to train the model is already provided. Train the above model and `report` the training and validation accuracies. (`report`, 5 points)

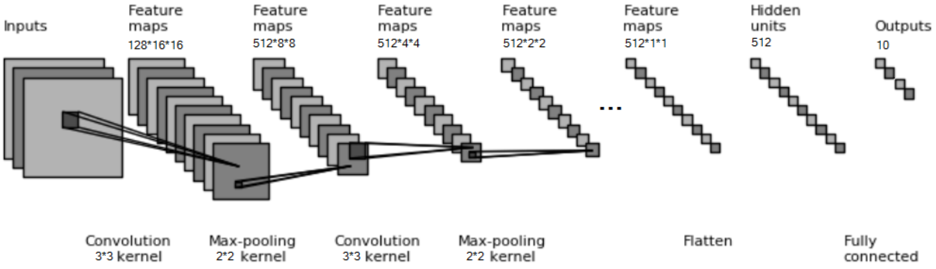

Figure 1: Figure depicting the convolutional neural networks.

b) Implement a function `PrintModelSize`, which calculates and prints the number of parameters of a neural network. `PrintModelSize` takes a model as input, and returns the parameters of a model. This gives us a measure of the model capacity. `Report` the number of parameters for the described model. (`report`, 2 points)

c) Implement a function `VisualizeFilter`, which visualizes the filters of the first convolution layer implemented in Q1a. In other words, you need to show 128 filters with size 3 x 3 as color images (since each filter has 3 input channels). Stack these into 3x3 color images into one large image. You can use the imshow function from the matplotlib library to visualize the weights. See an example in Fig. 2. Compare the filters before and after training. Do you see any patterns? Discuss it in your `report`. (`report`, 3 points)

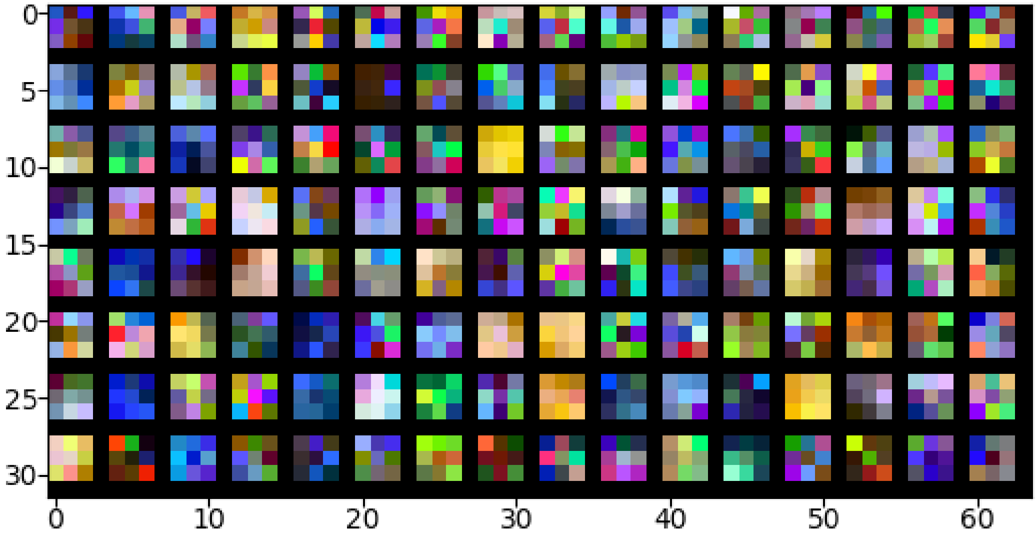

Figure 2: Example visualization of trained filters in the first conv layers.

Question 2: Improve training of Convolutional Networks (10 points)

a) Batch normalization is a widely used operation in neural networks, which will increase the speed of convergence, as well as reach higher performance. Read the paper "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift" for more theoretical details. In practise, this operation are implemented in most popular toolbox, such as PyTorch, TensorFlow. Add batch normalization in the model of Q1.a. Please keep other hyperparameters same, but only adding batch normalization. The ConvNet with batch normalization still uses the same class with Q1a, but different arguments. Check the code for details. In each block, the computations are in the order of convolution, batch normalization, pooling, ReLU. Compare the loss curves and accuracy using batch normalization to its counterpart in Q1a. (`report`, 6 points)

b) Early stopping is a method to alleviate overfitting. Instead of reporting the performance of final model, early stopping also saves the best model on the validation set during training. Increase the training epochs to 50 in Q1a and Q2a, and compare the best model and latest model on the training set. Due to the randomness, you can train multiple times to verify and observe overfitting and early stopping. (`report`, 4 points)

Question 3: Implementing the feedforward model (10 points)

We saw in Q2 that the model can start over-fitting to the training set, if we continue training for long. To prevent over-fitting, there are few paradigms we can adopt. A first approach is to get more training data. This might be a difficult and expensive process involving significant work, but it is generally the most effective way to learn more general models. A second more viable alternative is to perform data augmentation. Thirdly, we may regularize the model. In the following sub-questions we will experiment with each of the last two paradigms and measure their effectiveness on the model generalization.

a) Data augmentation is the process of creating more training data by applying certain transformations to the training set images. The underlying assumption is usually that the label of the image does not change under the applied transformations. This includes geometric transformations like translation, rotation, scaling, clipping, random cropping and color transformations like greyscaling, color-jittering. For every image in the training batch, a random transformation is sampled from the possible ones (e.g. random number of pixels to translate the image by) and is applied to the image. While designing the data input pipeline, we must choose the hyper-parameters for these transformations (e.g. limits of translation or rotation) based on things we expect to see in the test-set / real world. Your task in this question is to implement the data augmentation for the CIFAR-10 classification task.

Many of these transformations are implemented in the `torchvision.transforms` package. Familiarize yourself with the APIs of these transforms and the functions to compose multiple transforms or randomly sample them. Next, implement geometric and color space data-augmentations for the CIFAR-10 dataset, by choosing the right functions and order of application. Tune the hyper-parameters of these data-augmentations to improve the validation performance. You will need to train the model a bit longer (20-30 epochs) with data augmentation, as the training dataset is effectively larger now. Discuss which augmentations work well for you in the report. (`report`, 6 points)

b) Dropout is a popular scheme to regularize the model to improve generalization. A Dropout layer works by setting the input activations randomly to zero at the output. You can implement Dropout by adding the `torch.nn.Dropout` layer between the convolution blocks in your model. The layer has a single hyper-parameter p, which is the probability of dropping the input activations. High values of p regularize the model heavily and decrease model capacity, but with low values, the model might overfit. Find the right hyper-parameter for p by training the model for different values of p and comparing training validation and validation accuracy. You can use the same parameter p for all layers. You can also disable the data-augmentation from previous step while running this experiment, to clearly see the benefit of dropout. Show the plot of training and validation accuracies for different values of dropout (0.1 - 0.9) in the report. (`report`, 4 points)

Question 4: Implementing the feedforward model (10 points)

It has become standard practice in image-related computer vision tasks to use a pre-trained convolutional network as the backbone feature extraction network and train new layers on top for the target task. In this question we will implement such a strategy. We will use the `VGG_11_bn` network from the `torchvision.models` library as our backbone network. This model has been trained on ImageNet achieving top-5 error rate of 10.19%. It consists of 8 convolutional layers followed by adaptive average pooling and fully-connected layers to perform the classification. We will get rid of the average pooling and fully-connected layers from the `VGG_11_bn` model and attach our own fully connected layers to perform the CIFAR-10 classification.

a) Instantiate a pre-trained version of the `VGG_11_bn` model with ImageNet pre-trained weights. Add two fully connected layers on top, with BatchNorm and ReLU layers between them to build the CIFAR-10 10-class classifier. Note that you will need to set the correct mean and variance in the data-loader, to match the mean and variance which the data was normalized with when the `VGG_11_bn` was trained. Train only the newly added layers while disabling gradients for the rest of the network. Each parameter in PyTorch has a `requires_grad` flag, which can be turned off to disable gradient computation for it. Get familiar with this gradient control mechanism in PyTorch and train the above model. As a reference point you will see validation accuracies in the range 61-65% if implemented correctly. (`report`, 6 points)

b) We can see that while the ImageNet features are useful, just learning the new layers does not yield better performance than training our own network from scratch. This is due to the domain-shift between the ImageNet dataset (224x224 resolution images) and the CIFAR-10 dataset (32x32 images). To improve the performance we can fine-tune the whole network on the CIFAR-10 dataset, starting from the ImageNet initialization. To do this, enable gradient computation to the rest of the network, and update all the model parameters. Additionally train a baseline model where the same entire network is trained from scratch, without loading the ImageNet weights. Compare the two models training curves, validation and testing performance in the report. (`report`, 4 points)

Exercise credits: Prof. Bernt Schiele, Prof. Mario Fritz, Rakshith Shetty, Yang He, Dr. Seong Joon Oh.