Project 2
Deep learning by PyTorch

Due date: 23:59 Sunday 10/11th (2020)

0. Acknowledgements

We would like to thank Svetlana Lazebnik at University of Illinois at Urbana-Champaign for letting us use their project material. Originally, Part 1 was developed by Medhini Narasimhan based on materials provided by Unnat Jain. Part 2 was developed by Medhini Narasimhan.

The goal of this assignment is to get hands-on experience designing and training deep convolutional neural networks using PyTorch. Starting from a baseline architecture we provided, you will design an improved deep net architecture to classify (small) images into 100 categories. You will evaluate the performance of your architecture by uploading your predictions to this Kaggle competition (https://www.kaggle.com/c/sfu-cmpt-image-classification-2020-fall). Kaggle allows one to create in-class competition website with auto ranking and evaluation (https://www.kaggle.com/c/about/inclass/overview).

1. Instructions

Most instructions are the same as before. Here we only describe different points.

1. Generate a zip or tgz package, and upload to coursys. The package must contain the following in the following layout. **data** folder is large for this project. Please do not include data folder.
    - {SFUID}/
        - {SFUID}.pdf (your write-up, the main document for us to look and grade)
        - lab2.ipynb
    - In addition, CSV file of your predicted test labels needs to be uploaded to Kaggle.
2. Project 2 has 15 points.

2. Overview

The goal of this assignment is to get hands-on experience designing and training deep convolutional neural networks using PyTorch. Starting from a baseline architecture we provided, you will design an improved deep net architecture to classify (small) images into 100 categories. You will evaluate the performance of your architecture by uploading your predictions to this Kaggle competition (https://www.kaggle.com/c/sfu-cmpt-image-classification-2020-fall). Note that the amount of coding in this assignment is a lot less again. We will not provide detailed instructions, and one is expected to search online, read additional documents referred in this hand-out, and/or reverse-engineer template code.

Deep Learning Framework: PyTorch

In this assignment you will use PyTorch, which is currently one of the most popular deep learning frameworks and is very easy to pick up. It has a lot of tutorials and an active community answering questions on its discussion forums. Part 1 has been adapted from a PyTorch tutorial on the CIFAR-10 dataset. Part 2 has been adapted from the PyTorch Transfer Learning tutorial.

Google Colab Setup

You will be using Google Colab, a free environment to run your experiments. If you have your own GPU and deep learning setup, you can also use your computers. If you choose Google Colab, here are instructions on how to get started:

1. Open Colab, click on 'File' in the top left corner and select upload 'New Python 3 Notebook'. Upload a notebook (.ipynb) file in the code package.
2. In your Google Drive, create a new folder, for example, “SFU_CMPT_CV_lab2”. This is the folder that will be mounted to Colab. All outputs generated by Colab Notebook will be saved here.
3. Within the folder, create a subfolder called 'data'. Upload data files (cifar100.tar.gz, train.tar.gz, test.tar.gz), which you can download here.
4. Follow the instructions in the notebook to finish the setup.

Keep in mind that you need to keep your browser window open while running Colab. Colab does not allow long-running jobs but it should be sufficient for the requirements of this assignment (expected training time is about 5 minutes for Part 1 and 20 minutes for Part 2 with 50 epochs).

Part 1: Improving BaseNet on CIFAR100 [10pts]

Dataset

For this part of the assignment, you will be working with the CIFAR100 dataset (already loaded above). This dataset consists of 60K 32x32 color images from 100 classes, with 600 images per class. There are 50K training images and 10K test images. The images in CIFAR100 are of size 3x32x32, i.e. 3-channel color images of 32x32 pixels.

We have modified the standard dataset to create our own CIFAR100 dataset which consists of 45K training images (450 of each class), 5K validation images (50 of each class), and 10K test images (100 of each class). The train and val datasets have labels while all the labels in the test set are set to 0. You can tune your model on the validation set and obtain your performance on the test set by uploading a CSV file to this Kaggle competition. Note that the number of submissions is limited to a few times per day, so try to tune your model before uploading CSV files. Also, you must make at least one submission for your final system. The best performance will be considered.

BaseNet

We created a BaseNet that you can run and get a baseline accuracy (~23% on the test set). The starter code for this is in the BaseNet class. It uses the following neural network layers:

- Convolutional, i.e. nn.Conv2d
- Pooling, e.g. nn.MaxPool2d
- Fully-connected (linear), i.e. nn.Linear
- Non-linear activations, e.g. nn.ReLU
- Normalization, e.g. nn.batchnorm2d

BaseNet consists of two convolutional modules (conv-relu-maxpool) and two linear layers. The precise architecture is defined below (A typo in the input/output dimension for layer 6. It should be 10 | 5 instead of 5 | 5):

Layer No.	Layer Type	Kernel size (for conv layers)	Input | Output dimension	Input | Output Channels (for conv layers)
1	conv2d	5	32 | 28	3 | 6
2	relu	-	28 | 28	-
3	maxpool2d	2	28 | 14	-
4	conv2d	5	14 | 10	6 | 16
5	relu	-	10 | 10	-
6	maxpool2d	2	5 | 5	-
7	linear	-	400 | 50	-
8	relu	-	50 | 50	-
9	linear	-	50 | 100	-

Your goal is to edit the BaseNet class or make new classes for devising a more accurate deep net architecture. In your report, you will need to include a table similar to the one above to illustrate your final network.

Before you design your own architecture, you should start by getting familiar with the BaseNet architecture already provided, the meaning of hyper-parameters and the function of each layer. This tutorial by PyTorch is helpful for gearing up on using deep nets. Also, this lecture on CNN by Andrej Karpathy is a good resource for anyone starting with deep nets. It talks about architectural choices, output dimension of conv layers based on layer parameters, and regularization methods. For more information on learning rates and preventing overfitting, this lecture is a good additional read.

Improve your model

As stated above, your goal is to create an improved deep net by making judicious architecture and implementation choices. A reasonable combination of choices can get your accuracy above 50%. For improving the network, you should consider all of the following.

1. Data normalization. Normalizing input data makes training easier and more robust. Similar to normalized epipolar geometry estimation, data in this case too could be made zero mean and fixed standard deviation (sigma=1 is the to-go choice). Use transforms.Normalize() with the right parameters to make the data well conditioned (zero mean, std dev=1) for improved training. After your edits, make sure that test_transform has the same data normalization parameters as train_transform.

2. Data augmentation. Try using transforms.RandomCrop() and/or transforms.RandomHorizontalFlip() to augment training data. You shouldn't have any data

augmentation in test_transform (val or test data is never augmented). If you need a better understanding, try reading through PyTorch tutorial on transforms.

3. Deeper network. Following the guidelines laid out by this lecture on CNN, experiment by adding more convolutional and fully connected layers. Add more conv layers with increasing output channels and also add more linear (fc) layers. Do not put a maxpool layer after every conv layer in your deeper network as it leads to too much loss of information.

4. Normalization layers. Normalization layers help reduce overfitting and improve training of the model. Pytorch's normalization layers are an easy way of incorporating them in your model. Add normalization layers after conv layers (nn.BatchNorm2d). Add normalization layers after linear layers and experiment with inserting them before or after ReLU layers (nn.BatchNorm1d).

5. Early stopping. After how many epochs to stop training? This answer on stackexhange is a good summary of using train-val-test splits to reduce overfitting. This blog is also a good reference for early stopping. Remember, you should never use the test-set in anything but the final evaluation. Seeing the train loss and validation accuracy plot, decide for how many epochs to train your model. Not too many (as that leads to overfitting) and not too few (else your model hasn't learnt enough).

Finally, there are a lot of approaches to improve a model beyond what we listed above. Feel free to try out your own ideas, or interesting ML/CV approaches you read about. Since Colab makes only limited computational resources available, we encourage you to rationally limit training time and model size.

Kaggle Submission

Running Part 1 in the Colab notebook creates a plot.png and submission_netid.csv file in your Google Drive. The plot needs to go into your report and the csv file needs to be uploaded to Kaggle.

Tips

- Do not lift existing code or torchvision models.
- All edits to BaseNet which lead to a significant accuracy improvement must be listed in the report. You must include at least one ablation study for such an edit (or a combination of edits), that is, submitting results with and without the edit to Kaggle, and reporting the performance improvement.

Grading Scheme

- Include a table illustrating your final network and describe what it is [2 pts]
- Include a plot.png from Colab notebook, illustrating the training loss and the validation accuracy. [1 pts]

- Include at least one ablation study, reporting the performance improvement. [1 pts]
- Base performance [2 pts]. One receives (2, 1, or 0 pts), if the best accuracy from your network is above or equal to (50, 45, or 40 %), respectively.
- Relative performance [4 pts]. Given N submissions, one receives (4, 3, 2, 1, or 0 pts), if your ranking is above or equal to (80, 60, 40, 20, or 0 percentile position), respectively. For example, given 80 submissions, the top 16 submissions will receive 4 points. The next 16 submissions will receive 3 points.

Part 2: Transfer Learning [5pts]

In this part, you will fine-tune a ResNet model pre-trained on ImageNet for classifying the Caltech-UCSD Birds dataset. This dataset consists of 200 categories of birds, with 3000 images in train and 3033 images in test. Follow the instructions in the notebook and complete the sections marked #TODO. Without changing the given hyperparameters, you should achieve a train accuracy of 15.5%. With slight tweaks to the hyperparameters, you should be able to get a train accuracy above 80%. Try whatever tricks to avoid overfitting and achieve a test accuracy above 55%. One of your architectures must achieve at least 80% training accuracy and at least 55% testing accuracy to get full marks. One must include screenshots of the outputs of the network, showing the accuracies.

Experiment with the following at minimum:

1. Vary the hyperparameters based on how your model performs on train in the current setting. You can increase the number of epochs to 50. This should take ~20 mins on Colab.

2. Augment the data similarly to Part 1.

3. **ResNet as a fixed feature extractor.** The current setting in the provided notebook allows you to use the ResNet pre-trained on ImageNet as a fixed feature extractor. We freeze the weights for all of the network except that of the final fully connected layer. This last fully connected layer is replaced with a new one with random weights and only this layer is trained.

4. **Fine-tuning the ResNet.** To fine-tune the entire ResNet, instead of only training the final layer, set the parameter RESNET_LAST_ONLY to False.

5. Try different learning rates in the following range: 0.0001 to 0.01.

6. Try any tricks or techniques you saw throughout DNN lectures such as drop-out, data augmentation, regularization, and etc.

7. If you're feeling adventurous, try loading other pre-trained networks available in Pytorch.

A few useful resources:

- This Kaggle tutorial is a helpful resource on using pre-trained models in pytorch.
- This post explains how to fine-tune a model in pytorch.
- https://arxiv.org/abs/1403.6382 - trains SVMs on features from ImageNet-trained ConvNet and reports several state of the art results.
- https://arxiv.org/abs/1310.1531 - reports similar findings.
- https://arxiv.org/abs/1411.1792 - studies transfer learning in detail.

Submission Checklist

- Part 1:
    1. CSV file of your predicted test labels needs to be uploaded to Kaggle.
    2. The report should include the following:
        1. The name under which you submitted on Kaggle.
        2. Best accuracy (should match your accuracy on Kaggle).
        3. Whatever was requested in the above ”Grading Scheme” section.
    3. lab2.ipynb
- Part 2:
    1. Report the train and test accuracy achieved by using the ResNet as a fixed feature extractor vs. fine-tuning the whole network. Include also the screenshots of the accuracies reported by the code.
    2. Report any hyperparameter settings you used (batch_size, learning_rate, resnet_last_only, num_epochs).
    3. lab2.ipynb