Computer Vision
CSCI-UA.0480-001 2020
Assignment 3

February 20, 2020

Overview

In the last assignment we used pre-trained AlexNet to perform image classification on cat and dog images. In this assignment we will **train** a Residual Neural Network (ResNet) to perform the exact same task. Besides training a neural network, this assignment also requires you to process data and validate performance.

PyTorch Tutorial

Before you start this assignment, read the PyTorch tutorial on how to train a classifier carefully (https://pytorch.org/tutorials/beginner/blitz/cifar10_tutorial.html#sphx-glr-beginner-blitz-cifar10-tutorial-py). If you think you still don't quite understand PyTorch, go through everything on the official PyTorch tutorial website. You should know how dataset works, what dataloader is, how to define a simple neural network model, and what training looks like in PyTorch.

ResNet

ResNet is a deep learning model that wins the ILSVRC 2015 with a **top-5 accuracy** of 96.4% and is used as the backbone for many other complex deep learning models. Just like AlexNet, there's already an existing implementation of ResNet in torchvision (https://pytorch.org/docs/stable/torchvision/models.html#torchvision.models.resnet18). There are different variants of ResNet (ResNet18, ResNet34, ResNet50, ResNet101 and ResNet152). The number indicates how many layers are in the ResNet architecture. For simplicity we will use ResNet18 in this assignment.

The original ResNet output a vector of 1000 numbers. The ith element of the output vector is interpreted as the probability that the input image belongs to the ith class. Therefore, the sum of all elements of the output vector is exactly 1. However, since we only have 2 classes (cat and dog), we need to modify ResNet so that it can perform 2-class image classification instead of 1000-class image classification.

- Create an empty nn module class
- Create a ResNet model inside the custom nn module using torchvision.models.resnet18
- Remove the last layer (hint: https://discuss.pytorch.org/t/resnet-pretrained-model-with-last-fc-layer-stripped-does-not-work/17951)

- Add a linear layer as the new last layer using torch.nn.Linear. The number of output features of this linear layer should be 2 because there are only 2 classes.
- Because the output size of this headless ResNet is (batch_size, 512, 1, 1), don't forget to resize the tensor to (batch_size, 512) after you pass the input data to the headless ResNet in the forward function.

Data Processing and Data Augmentation

First download the new imagenet_12.zip. There are two subsets inside, namely imagenet_12_train and imagenet_12_val. Before training we need to process our raw image data into a format that is more friendly to deep learning model. In addition to data processing, we also want to augment the data to boost the performance. We can define our data processing and data augmentation method in a Dataset class.

- Create a empty dataset class
- Define how you would like to load images and labels in the dataset class. You can use the starter code in assignment 2 as an example.
    - The first output element should be the probability of the class cat while the second output element should be the probability of the class dog. For example, the numeric label of cat is 0.
- Define the following transforms and use torchvision.transforms.Compose to chain them together. Again, if you are not familiar with torchvision.transforms, take a look at the starter code in assignment 2 or the PyTorch master documentation.
    - Resize images to 256x256
    - Randomly crop images into 224x224
    - Convert them into tensor
    - Normalize the tensor with transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225])

DO NOT use the data in the imagenet_12_val folder to train your resnet. Usually in a computer vision dataset there are three subsets: train, validation, test. Validation data is used to validate the performance after each epoch and test data is used to judge the final performance. Because the test data in ImageNet is not publicly available, we used validation data as our new test data. You should split the training data into 80% training data and 20% validation data using torch.utils.data.random_split. Since the split is random, use 0 as the fixed seed for reproducibility. (https://pytorch.org/docs/stable/notes/randomness.html)

Training and Evaluation

- Define a suitable loss function
- Define a suitable optimization algorithm (optimizer)
- Keep training the model until convergence. To tell if a model converges or not, evaluate the model with validation data after each epoch. If the validation accuracy doesn't go up after a few epoch, stop the training.

- Use GPU to do the training since it will be significantly faster. If you need access to GPU, you can use JupyterHub or Prince Cluster.
- Evaluate your trained resnet on the test data and print out the accuracy.

Please start early. Submit your work to nyu classes. Upload your python script and also report the test accuracy in the NYU Classes submission text box. This assignment is due on **Thursday March 5th**.

Grading

Deliverables	Points
Modifying ResNet	10
Data Processing and Data Augmentation	20
Training and Evaluation	20
Total	50