

Homework 2 Part 2 (Simplified)

Image Recognition via Convolutional Neural Networks

11-785: Introduction to Deep Learning

OUT: June 1, 2019

DUE: NA

1 Introduction

Given an image of a mutually exclusive class, the task of classifying the type of the class is known as image classification. The data of interest is CIFAR-10. [1]

In this assignment, you will use convolutional neural networks (CNNs) to design an end-to-end system for image classification. For the classification task, your system will be given an image of a class as input and will output the ID class of the image.

To this end, you will train the CNN on a dataset with a few thousand images of labeled ID's (i.e., a set of images, each labeled by an ID that uniquely identifies the category). In doing so, you will gain experience with the concept of embeddings (in this case, embeddings for image information), optionally several relevant loss functions, and, of course, convolutional layers as effective shift-invariant feature extractors.

1.1 Overview

Convolutional Neural Networks (CNNs) are a class of deep learning neural networks. CNNs represents a huge breakthrough in image recognition. They're most commonly used to analyze visual imagery and are frequently working behind the scenes in image classification. They can be found at the core of everything from Facebook's photo tagging to self-driving cars. They're working hard behind the scenes in everything from healthcare to security.

In this exercise, your task is to create a Convolutional Neural Network(CNN) based image classifier for the CIFAR-10 dataset of images [4]. Feel free to experiment with different architectures, hyper-parameters, and training procedures to make yourself familiar with training and tuning CNN models. Please refer to the papers listed below for inspiration.

The dataset you will be using is the CIFAR-10 Dataset which consists of a collection of 60,000 images of 32x32 pixels with three 8-bit RGB channels. Note that, in some ML frameworks, this data set can be downloaded directly into data structures useful for processing in the CNN.

You will be evaluated based on the percentage of labels that you predicted correctly i.e. classification accuracy.

1.2 Key Concepts

1.2.1 Sources

Lecture 8: <http://deeplearning.cs.cmu.edu/slides.spring19/lec8.CNN.pdf>
Lecture 9: <http://deeplearning.cs.cmu.edu/slides.spring19/lec9.CNN.pdf>
Lecture 10: <http://deeplearning.cs.cmu.edu/slides.spring19/lec10.CNN.pdf>
Matt Zeiler: <https://youtu.be/ghEmQSxT6tw> (What the CNNs Sees)

1.2.2 Frequently Used Terms

Convolutional Layers: These layers consist of filters (see slide 213), strides (see slide 215), and channels. It is important to understand how these relate to each other, so the reader is encouraged to review Spring 19, Lecture 8 on CNNs.

Pooling Layers: Are used for dimensionality reduction by combining two or more parameters into one (see slide 31). The reader is encouraged to refer to slide Spring 19, lecture 10 on CNNs.

Embedding Layer: Are layers that function similar to fully-connected layers, which was discussed in the previous assignment on MLPs. Most CNNs are followed by an embedding layer.

Shift Invariance: In terms of how the information is processed, and for what we are trying to learn when applying a CNN to an image, the property of shift invariance (see slide 18) is certainly beneficial. This allows convolutional layers to make sense of images or other data types in a way that is not inherent or less intuitive for other networks.

Receptive Fields: When using convolutional layers, it helps to consider their receptive fields (see slide 213), since they determine the amount of information that is processed at any given scan of the filter. If the receptive field is too small, there might not be enough information to learn something meaningful, and if the receptive field is too large, the model might have trouble picking up on details. You are again encouraged to review Spring19, Lecture 8 on CNNs.

Feature Extraction: Fundamentally, CNNs are a feature extractor (see slide 161), which motivates their use not only in and of themselves but also in combination with other networks. You are again encouraged to review Spring19, Lecture 8 on CNNs. There is also an interesting dissertation on the subject by Matt Zeiler...

2 Image Classification

An input to your system is an image and you have to predict the ID of the image. The true image ID will be present in the training data and so the network will be doing 10-way classification to

get the prediction. You are provided with the train set and have the option to split it into a training and validation set. With the validation set, you can fine-tune the model based on the accuracy you get.

3 Dataset

To download data for **Local** development:

```
import torch
import torchvision
from torchvision.datasets import CIFAR10

TRAIN_TRANSFORMS = torchvision.transforms.ToTensor()
TEST_TRANSFORMS = TRAIN_TRANSFORMS
BATCH_SIZE = 16

### DO NOT MODIFY ANY CODE STARTING HERE ###

cifar_train = CIFAR10('./data', download=True, train=True,
                      transform=TRAIN_TRANSFORMS)
cifar_test = CIFAR10('./data', download=True, train=False,
                     transform=TEST_TRANSFORMS)

train_dataloader = torch.utils.data.DataLoader(cifar_train,
                                                batch_size=BATCH_SIZE,
                                                shuffle=True)
test_dataloader = torch.utils.data.DataLoader(cifar_test,
                                              batch_size=BATCH_SIZE,
                                              shuffle=False)

del cifar_train, cifar_test

### END OF CODE NOT TO MODIFY ###
```

This dataset is not large, less than 1GB, and you may wish to take advantage of this by running multiple experiments at a time. As a kind reminder here, Colab is a free computing environment provided by Google, and the ease of importing this data would make that environment preferred. Make sure to save a copy of your code to your Drive, and you can open it from that same location. If, for whatever reason, you want to save data to the drive, then you must first mount the drive to your Colab notebook before specifying the path. In this regard, the following may be useful:

Google Colab Setup: <https://colab.research.google.com/notebooks/io.ipynb>

The data contains images of size 32 by 32, each with 3 channels. For classification, you will be given an object in an image. What you need to do is to learn to classify this image into correct image IDs.

It's very important to note that, for classification, the train, validation, and test contain the same set of images. what it means is that your network cannot classify images unless it has seen it before, and the dataset is set up that way. In other words, the exact images are not the same between the train, validation, and test set but rather they are all sampled from the same distribution.

3.1 Using the Data

You will have one folder that will download after you run the provided script. Further, two data loaders will be generated:

```
train_dataloader
test_dataloader
```

You are encouraged to follow this tutorial on data loaders to better understand what they are and why we are using them.

DataLoader Usage: https://pytorch.org/tutorials/beginner/data_loading_tutorial.html

Data loaders will become invaluable in this course.

4 Getting Started

The script provided will allow you to load subsets of the training data while prototyping and experimenting. Your job is to create the architecture to utilize the data loader for training with N-way classification on the 32x32 images.

4.1 Preprocessing

In this homework, there is not much of pre-processing to be done on the images. Though optional, you are encouraged to explore the torchvision transformations described here:

Torchvision Transforms: <https://pytorch.org/docs/stable/torchvision/transforms.html>

Here is an example of usage:

```
import torch
import torchvision
from torchvision.datasets import CIFAR10

TRAIN_TRANSFORMS = transforms.Compose([
    transforms.RandomHorizontalFlip(),
    transforms.ToTensor(),
    transforms.Normalize([0.45, 0.45, 0.45],
                        [0.25, 0.25, 0.25])
])

TEST_TRANSFORMS = torchvision.transforms.ToTensor()

BATCH_SIZE = 16

### DO NOT MODIFY ANY CODE STARTING HERE ###

. . .

### END OF CODE NOT TO MODIFY ###
```

4.2 Getting Up-and-running

Throughout the course, you will be introduced to a variety of complex and nuanced topics so it is often helpful to see them used end-to-end in an applied context. Knowing this, we strongly recommend you *first* reference the official PyTorch repositories to give you an idea of what is expected of you. They can be found at the following links:

Official Page: <https://pytorch.org/tutorials>
Repository: <https://github.com/pytorch/tutorials>
Documentation: <https://pytorch.org/docs/stable/index.html>

There are, however, many topics that we will be covering for which there is no official PyTorch documentation. If this is the case, you are encouraged to search and review other tutorials out there.

5 Submission

If you are developing locally, simply compare your train and validation accuracy to that of your test accuracy. Be careful not to accidentally use the test data during development, and try to only compare to the test set infrequently. Verifying with respect to the test set too often can cause overfitting.

Though not required, we encourage you to practice writing a one-page write up for your own reference describing what model architecture, loss function, hyperparameters, any other interesting detail led to your best result for the above two competitions. Such a write-up should limit the content to one page.

6 **Kaggle**

Once you have finished testing locally, and you want a more realistic homework part 2 experience, then we have set up a Kaggle page for you. This can be your first time using Kaggle, at least in the context of this course! The page is here:

Kaggle Page: <https://www.kaggle.com/c/homework-2-part-2-simplified>

The only difference between local development and Kaggle development is that the former has access to the test labels while the latter does not.

The data for the assignment can be downloaded using a similar script, but you will have the testing labels removed. This is more realistic, because we will never provide you with the testing labels on the homework.

To download data for **Kaggle** development:

(see next page)

```

import torch
import torchvision
from torchvision.datasets import CIFAR10

TRAIN_TRANSFORMS = torchvision.transforms.ToTensor()
TEST_TRANSFORMS = TRAIN_TRANSFORMS
BATCH_SIZE = 16

### DO NOT MODIFY ANY CODE STARTING HERE ###

cifar_train = CIFAR10('./data', download=True, train=True,
                      transform=TRAIN_TRANSFORMS)
cifar_test = CIFAR10('./data', download=True, train=False,
                     transform=TEST_TRANSFORMS)

temp_dataloader = torch.utils.data.DataLoader(cifar_test,
                                              batch_size=10000,
                                              shuffle=False)
temp_data = next(iter(temp_dataloader))[0]

train_dataloader = torch.utils.data.DataLoader(cifar_train,
                                              batch_size=BATCH_SIZE,
                                              shuffle=False)
test_dataloader = torch.utils.data.DataLoader(temp_data,
                                              batch_size=BATCH_SIZE,
                                              shuffle=False)

del cifar_train, cifar_test, temp_dataloader, temp_data

### END OF CODE NOT TO MODIFY ###

```

The following tutorial might be helpful for those who are new and would like to see an example workflow from start to end using the Titanic dataset.

Example: <https://www.kaggle.com/jlawman/complete-beginner-your-first-titanic-submission>

7 Conclusion

That's all. As always, feel free to ask on Piazza if you have any questions.

Good luck and enjoy the challenge!

8 References

- [1] CIFAR10 Dataset: Learning Multiple Layers of Features from Tiny Images, Alex Krizhevsky, 2009. Images collected by Alex Krizhevsky, Vinod Nair, and Geoffrey Hinton of the Canadian Institute For Advanced Research, available at: <https://www.cs.toronto.edu/~kriz/cifar.html>.
- [2] Sandler, M.; Howard, A.; Zhu, M.; Zhmoginov, A.; Chen, L.C. Mobilenetv2: Inverted residuals and linear bottlenecks. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Salt Lake City, UT, USA, 18–22 June 2018; pp. 4510–4520.
- [3] Handwritten digit recognition with a back-propagation network, Y. LeCun, In Proc. Advances in Neural Information Processing Systems, (1990).
- [4] ImageNet Classification with Deep Convolutional Neural Networks, Krizhevsky, A., Sutskever, I. and Hinton, G. E., NIPS: Neural Information Processing Systems, (2012).
- [5] Receptive fields of single neurons in the cat's striate cortex, Hubel, D.H.; Wiesel, T.N., J Physiol. 148 (3), (1959).
- [6] Deep learning, LeCun, Y., Bengio, Y., and Hinton, G., Nature, 521, May, (2015).
- [7] Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position, K. Fukushima, Biological Cybernetics, 36, 4, April (1980).