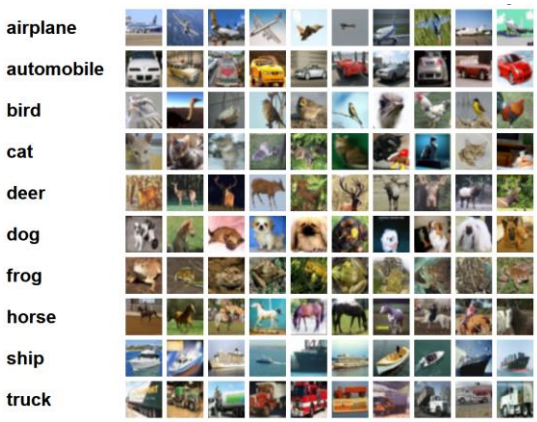
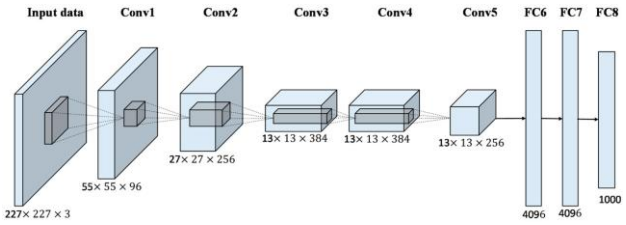


https://www.cs.toronto.edu/~kriz/cifar.html



https://www.mdpi.com/2072-4292/9/8/848

ELEC 475

Lab 3 – Image Classification

Lab 3 – Image Classification

Michael Greenspan

Contents

1. Introduction

In this lab you will implement an image classification model, with variations, and train and test it on the CIFAR 100 dataset.

Lab 2 contained an encoder, based on vgg. Starting with this (the *backend*), you will implement a classification *frontend*. The frontend takes the backend bottleneck as input, applies a number of layers, and outputs a 1xC tensor, where C is the number of class labels in the dataset.

Your task is as follows:

1.1 Step One - Vanilla Model

Implement a basic (*vanilla*) version of your model, comprising the vgg backend and your own custom frontend. Train and test it on the CIFAR100 dataset, using the *fine* labels of the train and test dataset partitions, respectively. When testing, calculate both the top-1 and top-5 error rates.

When training your network, you can start by loading the vgg backend pre-trained parameters that were distributed in Lab 2. Alternately, you can train both frontend and backend from scratch (i.e. with random initial weights).

1.2 Step Two - Architectural Mod

Implement an architectural modification to your vanilla model, in an effort to improve performance. This modification can draw from the various architectural concepts that we saw throughout the ILSVRC competition. Examples could include the addition of skip connections (*ResNet*), parallel branches (*GoogLeNet*), 1x1 convolutions (*SENet*), or many others.

The modifications must be at the architectural level (i.e. a change to the network structure), and can apply to either the frontend or the backend (or both).

As in Step 1, train and test your mod on CIFAR100 datasets, using the same hyperparameters that you used for the vanilla model.

Hint: Train and test on CIFAR10 first, and then move on to CIFAR100 once you've worked out all of the bugs.

2. Deliverables

The deliverable as described below comprises the following:

- The completed code, for both the vanilla model, and the mod;
- A report.

2.1 Completed Code

In addition to the code itself, provide the following four scripts to train and test your code (e.g. from the PyCharm terminal):

`train_vanilla.txt`

`test_vanilla.txt`

`train_mod.txt`

`test_mod.txt`

2.2 Report

You report should include the following sections:

- Section 1 - Vanilla Architecture: This section describes your vanilla model. Explain your reasoning behind the architecture of the frontend (i.e. why did you design it in this way, and what was the motivation behind the design.)

  Include a diagram of your network. You can draw the diagram manually (e.g. in PowerPoint, or by hand), or you can generate it using NN-SVG, or some other tool.

- Section 2 – Mod Architecture: This section describes your mod model, including a diagram. In addition to the architectural description, explain the reason for the mod (e.g. the motivation and concept behind the modification).

- Section 3 – Experiments: Describe the training and testing of your system, on CIFAR-100. Include enough detail to completely re-execute the experiments, such as all hyper-parameters used to train your system (initial learning rate, optimizer, batch size, etc.).

  Describe the hardware that was used to train, include the training plots for both systems, and the time that it took to train. Include final performance results (accuracies) for both models.

- Section 4 – Discussion: Discuss the performance of your system. Was it as expected? Did the mod improve accuracy, or not, and why? Where there any tradeoffs between the vanilla and mod (e.g. efficiency vs. accuracy)?

3. Submission

The submission should all of the elements described in Section 2.

All deliverables should be compressed into a single **<zzz>.zip** `file,` where filename **<zzz>** is replaced with your student number. If you are in a team of 2, then concatenate both student numbers, separated by an underscore. The zipped directory should include your code and scripts, your trained parameter files, and all output plots and images.

The report should include a title (e.g. minimally ELEC 475 Lab 3), your name and student number. If you are working in a team of two, then include the information for both partners in the report (but only make one submission in OnQ).

Your code should train by entering the syntax provided in your included scripts (e.g. **`train_vanilla.txt`**`, etc.)` on a PyCharm terminal.

The marking rubric is as follows:

Item	mark
Vanilla	2
Mod	2
Report	2
Correct submission format	1
Total:	**7**