CS 182 Vision Project - Generalizable Classifiers

The human vision system is remarkably robust, solving problems that existing computer vision systems are incapable of. We have covered some of these examples in class, namely adversarial images, however there are many different perturbations/shifts that occur in the real world that need to be planned for. In this project, we will be building robust computer vision classifiers for a small image dataset. Because you will be building a classifier which will be tested on an **fully unknown** test data-distribution, you will have to design a classifier which is robust to as many test-time perturbations as possible.

In this project you will be building a classifier on top of the Tiny ImageNet dataset, a classification dataset with 200 classes, each with 500 training images (100,000 images) that are 64x64 RGB images. The public validation dataset has 50 images per class (10,000 images). Your classifier should perform well on the public validation dataset, but it will also have to perform well on a hidden dataset which the graders will run. The hidden dataset will consist of images which your classifier might encounter in real world usage. Thus, you must:

1. Research what kinds of data perturbations could occur in real world models
2. Design and build a classifier which is robust to those perturbations

The performance of your model will be evaluated using the accuracy metric.

You are allowed to make any modifications to the model/training data in order to achieve the goal of the highest performance on the hidden dataset. **You will not necessarily be graded on your performance on the hidden dataset, but the steps that you took to improve your classifier's robustness**.

We will provide simple starter code in both Tensorflow 2.0 and Pytorch, however you are free to modify the code in any way, as long as it preserves the input/output requirements specified in the **"Requirements"** section.

Submission Deliverables

- A file "**test_submission.py [eval.csv]**" which takes the path to a CSV file as a command line argument, and returns a CSV file of predictions. The format of the CSVs is specified below in the "Requirements" section.
- A **"requirements.txt**" file in PIP format detailing the python packages which are required to run your project's **test_submission.py** file.
- A **README.txt** file which describes your project, and how to run the project if there are any requirements.

- A 4-8 page report detailing your approach to the problem, and any challenges that you faced along the way.

Model Training Code

You can download the starter code for Tensorflow 2 and PyTorch here:

- PyTorch**: CS182-Spring2020-Vision-Project-PyTorch.zip (https://bcourses.berkeley.edu/courses/1487769/files/76670821/download?wrap=1)**
- Tensorflow: **CS182-Spring2020-Vision-Project-Tensorflow.zip (https://bcourses.berkeley.edu/courses/1487769/files/76670822/download?wrap=1)**

This code provides boiler-plate code for loading the data, and evaluating a model. It is truly bare-bones, and serves only as an example to work from. Note that there is not even validation of the trained model, so at least some changes will have to be made. You do not have to use the included code, however, you must have file "**test_submission.py**" which takes the path to a CSV file as a command line parameter, which we can use for evaluation on the hidden test set.

Data

Tiny Imagenet has 200 classes. Each class has 500 training images, 50 validation images, and 50 test images. We have released the training and validation sets with images and annotations. We provide both class labels and bounding boxes as annotations; however, you are asked only to predict the class label of each image without localizing the objects.

You can download the images here **(https://tiny-imagenet.herokuapp.com/)** , or use the provided script in the boilerplate code.

The dataset zip file contains:

- train.images.zip - the training set (images distributed into class labeled folders)
- test.zip - the unlabeled 10,000 test images
- sample.txt - a sample submission file in the correct format (but needs to have 10,001 lines. One line per image in addition to the first header line)
- wnids.txt - list of the used ids from the original full set of ImageNet
- words.txt - description of all ids of ImageNet

Evaluation Metrics

The networks will be evaluated using the included submission script. The major metrics are:

- Accuracy (Number of correct samples / total samples)

- Accuracy @K (Accuracy in the top-k predictions)

Requirements

test_submission.py

This file should take the path to a CSV file as a command line argument. It will have the following format:

eval.csv

Image_id (int), image_path (str), image_height (int), image_width (int), image_channels (int)

The input image path will be an absolute path on the system that is running the code. The code will be run from whatever directory the *test_submission.py* file is in.

The python file should create a CSV file called *eval_classified.csv* in the current working directory which has the following fields:

Image_id (int), image_class (str)

The image class should be the corresponding wnid from the wnids.txt file available in the dataset. There is an example CSV in the boilerplate code.

requirements.txt

This is a requirements.txt file in PIP format containing all of the required python packages to run your evaluation code. See https://medium.com/@boscacci/why-and-how-to-make-a-requirements-txt-f329c685181e **(https://medium.com/@boscacci/why-and-how-to-make-a-requirements-txt-f329c685181e)** for more details about why this is required, and how to create one.

Readme.txt

The readme.txt should discuss how to run your code, and anything the grader should know about running the project.

Project Report

Your project report can be either styled as a research paper or as a blog post. Paper format reports are like research papers and will be 6-10 pages in length. Blog posts should have equivalent length, but may be more graphics heavy. Here's the example Blog from the "best

practices" section: http://distill.pub/2016/augmented-rnns/ **(http://distill.pub/2016/augmented-rnns/)** .

The goal of your report is to fill in all the details of your project, so that someone else could fully reproduce your results. There are different styles for writing research papers, and even more freedom in writing blog posts. We give some suggestions below on content to include. These items don't need to be separate sections, but your report should contain this content somewhere.

Problem Statement and Background

Give a clear and complete statement of the problem. Don't describe methods or tools yet. Where does the data come from, what are its characteristics? Include informal success measures (e.g. accuracy on cross-validated data, without specifying ROC or precision/recall etc) that you planned to use. Include background material as appropriate: who cares about this problem, what impact it has, what implications better solutions might have. Included references to any related work you know about.

Approach

Start with data preparation. If you used an off-the-shelf dataset, you don't need to say much. But if you did any non-standard preparation, or if you spent any time iterating on your data prep., please describe it thoroughly. Describe your baseline model. Give a graphical representation of it. Describe your final model. Explain the evolutionary process to get to it. What other models did you try? What worked and what didn't?

Results

Give detailed results for baseline and final models. Make sure you're very clear about what dataset was used, how train/validate/test partitioning was done, what measures were used. Explore surprises. Whenever a method didn't behave as expected, explore and try to figure out why. Please use visualizations whenever possible. Include links to interactive visualizations if you built them.

Tools

Describe the tools that you used and the reasons for their choice. Justify them in terms of the problem itself and the methods you want to use.

Lessons Learned

Give a high-level summary of your results. Make sure you summarize your results at the beginning and also at the end of your report. You can refer back to the "Results" section for elaborations. In particular emphasize any results that were surprising.

Team Contributions

This information must be in a section called Team Contributions at the end of your report. Please give a percentage breakdown of the effort from each team member, and what they worked on.

Please discuss this within your team to make sure every member agrees with the breakdown.

Extra Credit Challenges

We have several challenges which you can explore for extra credit opportunities.

Explainable AI

Most current methods for classification take inputs, and produce a class without any decipherable explanation or context for the results. You can earn some extra credit by tackling explainable AI, and producing an additional explanation for your classification in addition to the output class. The methods should be detailed in the report, and the generated explanations should be produced in a separate file from the main classification results.

Grading Rubric

The following sections will be used as an outline for determining your grade on the project:

Write-Up:

- Clarity, structure, language, references
- Background literature survey, clear understanding of the problem
- Insight and discussion of methodology, analysis, results, etc.

Technical Work:

- Correctness
- Technical depth
- Innovation

Evaluation & Results:

- Thoroughness in analysis and experimentation
- Results & performance (including on the hidden set)

Challenges:

- Performance on extra credit challenges
- Methods used for extra credit

Questions

For questions on this project, contact:

- David Chan: **davidchan@berkeley.edu (mailto:davidchan@berkeley.edu)**
- Haozhi Qi: **hqi@eecs.berkeley.edu (mailto:hqi@eecs.berkeley.edu)**