



Project 2

Prof. Adín Ramírez Rivera
adin@ic.unicamp.br

1 Description

Learning effective visual representations without human supervision is a long-standing problem. Most mainstream approaches fall into one of two classes: generative that learn to model the input distribution, or discriminative that learn splitting hyperplanes to distinguish the classes.

Inspired by recent contrastive learning algorithms, SimCLR [1] learns representations by maximizing agreement between differently augmented views of the same data example via a contrastive loss in the latent space (see Fig. 1).

2 Work

Your task is to reproduce the results of SimCLR [1] for CIFAR10. 🛠️ That means that you must implement their method and execute the experiments to compare

- different augmentation functions \mathcal{T} , and
- different similarity functions.

This work is the minimum expected. You could experiment with different configurations of the architecture (beyond the presented one in the paper) and compare those modifications too if you want to further explore the architectures.

Your work is to evaluate different transformation functions \mathcal{T} to augment the data [1, Section 3], and then compare their performance. As well, Chen et al. use the cosine similarity as their main similarity function. You need to evaluate other similarities and explore how they change the method's performance.

Regarding their architecture for the encoder $f(\cdot)$ and the projection head $g(\cdot)$, you can use their proposal or a reduced version so you can train it. (See Fig. 1 for a graphical representation.) You can decide to radically change their architectures too. 🛠️ Either way, you must explain your architecture design in the report.

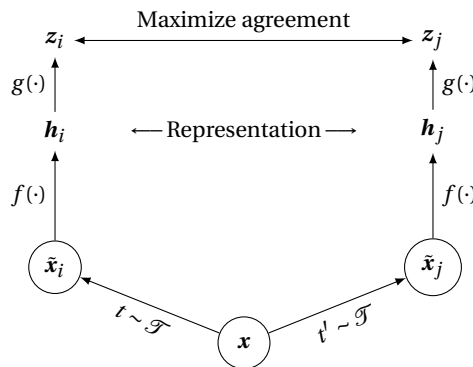



Figure 1. SimCLR [1, Fig. 2] framework. Two separate data augmentation operators are sampled from the same family of augmentations ($t \sim \mathcal{T}$ and $t' \sim \mathcal{T}$) and applied to each data example to obtain two correlated views. A base encoder network $f(\cdot)$ and a projection head $g(\cdot)$ are trained to maximize agreement using a contrastive loss. After training is completed, we throw away the projection head $g(\cdot)$ and use encoder $f(\cdot)$ and representation \mathbf{h} for downstream tasks.

Note that there is an implementation provided by the authors.¹ Nevertheless, you are expected to code your own solution. I recommend to check the implementation in case you get stuck, but not to do it from the beginning. That way you can get your own idea on how to implement your solution.

2.1 Report

 You must submit a report explaining the method and how you implemented it. Explain your results and whether your evaluation is equal, similar or different to the one in the original paper.

3 Evaluation

Your grade will be defined by the following aspects:

- | | |
|---|-----|
| 1. Explanation of SimCLR | 30% |
| 2. Explanation of how you translated their explanations into your code | 30% |
| 3. Experiments and discussion of the results | 40% |


Each item corresponds to the questions and requirements defined in the previous sections. The overall report, will be evaluated on the aspects regarding writing and requested plots or figures in the text. **Your language usage won't be graded**, but your ability to present your results, ideas, and how they are supported will be. Each point will be evaluated according to the completeness and correctness of the requested items.

Moreover, **the evaluation includes the contribution of each team member to the solution through the commits history and level of engagement of each team member through the repository**. Hence, do not commit all the work using one single student account. Each member should commit (using the email from the university that contains their student ID²) and handle their work on the repository.


4 Submission

Your submission must be through your assigned group on Gitlab. You must create a repository named `project - 2` and commit all the code and report there.

Your submission must have the following subfolders:

- `input`: a directory containing the input assets (images, videos or other data) needed to execute your experiments. Your generated data should be placed here.
- `output`: a directory where your application should produce all the generated files (otherwise stated in the problem).  This folder and all its contents must be added to the **artifacts path on the `.gitlab-ci.yml` setup**.

Name your outputs according to a convention that reflects the experimental setup.

- `src`: a directory containing all your source code. You only need to submit files that are not derived from other files or through compilation. In case some processing is needed, prefer to submit a script that does that instead of submitting the files.
- `Makefile`: a makefile (or equivalent) that executes your code through the docker image. You must provide the image for the project. It is strongly recommended that you use a vanilla version of your language of choice, for example for python use `python: latest`. The code **must** be executed through a standard call to `make` (or equivalent).  Moreover, note that your code **must** run on the Gitlab executor (pipeline) on the server itself. Hence, you need to use the `.gitlab-ci.yml` configuration. You can base your work on the **demo** that already provides an example of such functionality.

¹<https://github.com/google-research/simclr>

²Note that the repository information is extracted through scripts that use your student ID as an identifier for the committer. If you do not have the repository set correctly it won't find you. No effort will be made to find the missing information if you do not have your git set up correctly.

An exception to running the code on the Gitlab executor is if your code doesn't train or produce the results on a CPU. In that case, provide a link to a notebook where you run your code (e.g., Google Collaboratory) where the project can be executed directly.

- **report**: a directory containing the source files that produce your report. This directory must be only on its branch called **report**.

🔗 Your report **must** be written using \LaTeX (and friends) and compiled within the pipeline of the repository in a stage named **report**. Consequently, **the PDF must not be committed**. You can use the images from [adnrv/texlive](#) to build your PDFs. **Only the PDF of the report** must be added to the **artifacts path on the .gitlab-ci.yml setup**, and not other intermediary files of your report (e.g., images, log files, auxiliary files).

Your report must show all your work for the given project, including images (labeled appropriately, that is, following the convention given) and other outputs needed to explain and convey your work. When needed include explanations to the questions given in the project. **Your report should be constrained up to 4 pages in content (excluding images, tables, and references).**

🔗 The last commit that triggered the build must be before the deadline of the submission. Do not worry about the time executing the pipeline. However, **you must ensure that your code works as no attempt will be made to patch or run broken code.**

5 Notes

- ❗ All commits **must** come from the student's email (the one that includes the student ID). That is, you **must** configure your local git credentials in **every** machine that you will code with


```
git config --global user.email 123@unicamp.br
```
- ❗ **You are not allowed copy the code of the authors.** It is expected that you understand and produce the code to reproduce the method.
- ❗ All the submissions must be self-contained and must be executable in a Linux environment. Specifically, your code must execute in the docker image within the Gitlab pipeline, or within an automated environment that reproduce your findings (in case you need a GPU).
- ❗ Your report must be written in English.
- ❗ While there are several images available for \LaTeX , I recommend you to use one of the docker images [adnrv/texlive](#), available at docker hub (<https://hub.docker.com/r/adnrv/texlive/>). In particular the **basic** and **full** images are standard versions of **texlive**. However, their sizes vary considerably (consequently, the execution times on the pipelines). If you need particular packages it may be easier to take a **basic** image and just install the packages you need using **tlmgr**.
- ❗ It is your responsibility to make sure your code compiles and executes correctly. No effort will be made to run your code besides executing the pipeline within the Gitlab repository.

References

- [1] T. Chen, S. Kornblith, M. Norouzi, and G. Hinton, "A simple framework for contrastive learning of visual representations," *arXiv preprint arXiv:2002.05709*, 2020.