# Final Project Description

CS 482, Prof. Stein

Proposal Due: March 23, 2023 (or sooner!)

[Project proposal template adapted from GMU CS's Graduate NLP course.]

**Overview** The final project is an opportunity for you to explore in more depth one of the topics we have talked about in class. The goal of the project should be to either implement one of these approaches and/or propose a creative extension or application of one of these topics relying primarily on open source code. You are expected to show creativity on *something*: implementing an approach (or using open source code) and then applying it to an existing dataset, will not (in general) be sufficient for more than a grade of B on the project. It is also recommended that you focus more on the *substance* of the project, rather than simply tuning parameters; a project should avoid significant amounts of simply tuning parameters and simply seeing how well it does. When appropriate, you should try to come up with a quantitative metric to evaluate the approach; try to have a numerical value that says how well your approach worked so that you have a point of reference for discussion or to compare against something else. You may work alone or in teams of 2–3.

**Note:** I would like the format of the final project to be a *poster session* so that you can all see each other's work! However, I am having trouble securing a room and figuring out how you will all print the posters; if not a poster session, I will expect each team to produce a 10 minute recorded video presentation of their project and (possibly) upload those to Blackboard where you can see each others' work.[1]

[1] If you would, for whatever reason, not like to participate in either of those, come talk to me; different people feel comfortable with different forms of project and presentation and I'm sure we can work something out.

## 1 Requirements

Your final project will consist of:

- **A project proposal** Due: March 23, 2023
- **An 10 minute video submission** or **poster session** [Unsure which for now]. Due: [Final Exam Time, TBD] **no late days allowed!**)
- **A final project "tutorial"** (instead of a report) Due: [Final Exam Time, TBD] **no late days allowed!**

Students are free to complete the final project either individually or in a group (but with no more than 3 students). If working individually, the project can be less ambitious, but should not be less complete; if working in a group, all students should contribute equally, and all students will obtain the same grade from the project. Students are allowed (and, in fact, encouraged) to combine this project with their research, other course projects, or their hobby interests. However, the project should still touch upon concepts from this course and must be in the space of Computer Vision; you should think of the project as an opportunity for you to teach your classmates about something interesting you've been playing around with, so be sure to connect what you've done to what we discussed in class if possible. If the students are not sure whether their other projects could be combined with this course project, they should make a post on Piazza[2]. Students are recommended to discuss the project idea with the instructor (me) or create a post on Piazza.

Note that **any external resources used in this project must be clearly cited in your tutorial.** Also, **you are allowed (encouraged, even) to use ChatGPT in your project**. If you do so, please tell me how it went and what worked and did not. I'm excited by the potential of the technology to augment creativity and exploration, so try it out if you'd like!

Students are welcome to use open source implementations of code in their project, but if you choose to use a significant amount of open source code in your project, you must make sure that your project includes *some element of novelty or creativity.* The *novelty* must include either a new and/or creative extension of the approach under scrutiny or a new environment or dataset (including datasets or problems you have created yourself). The novelty does not need dramatic, but you need to show me that you *thought* about

[2]This post can be public on Piazza if they would like as well! I am happy for students to share their ideas on projects and even use Piazza to find potential collaborators.

the project, and did more than simply download an algorithm and data. As such, it is generally recommended that you use open source code as a starting point for the project and think of ways to extend it or apply it to a new domain or problem. Alternatively, you can spend a majority of your project implementing an existing algorithm—e.g., one of the algorithms we discussed in class and/or an extension of it. However, *some* element of creativity, especially an interesting or new dataset or application area or motivation, should also be included.

I am not looking for a crazy amount of work on this project and it is not my aim for it to take up a *ton* of your time. Try to come up with a solid plan that builds on what we talked about in class, and I can help you calibrate to make sure your plan is reasonable, not too much or too little work. You will have over a month to work on the project, so try to spend some time each week making progress towards it.

## 2 Project Proposal

The project proposal should include at least the following items:

- What problem you want to address, and the motivation (especially if it is a new or less-studied problem);
- What dataset(s), simulator, or problem setting you plan to use; if you plan to collect a new dataset, describe the procedure and the source data;
- A literature survey on existing work that studies the same/a similar problem and/or investigates the same/a similar method; note that students are expected to compare their proposed methods with existing work (or pick a reasonable baseline point of comparison), as a way of justifying the novelty of their work or to demonstrate that their implementation has succeeded in some way.

**A PDF proposal must be turned in through Blackboard by March 23 2022** (plus the usual 3 free late days if you feel like you should need it). The proposal will *not be graded*, but I will give you detailed comments on the quality of your plan and what I think you need to do to get a good grade (or if you plan to do too much). I will give you written comments within a couple days of you submitting the proposal. The proposal will be 10% of the final project grade. Please try to submit only one proposal per team!

**For team proposals: list the name and email addresses of all team members. I will email you all my comments, since Blackboard makes it difficult to add comments for all of you. Only one of you should upload the proposal to Blackboard (so, one proposal per team).**

## 3 Project Presentation [if a video]

All teams will be required to submit a 10 minute presentation[3] in which they discuss their project. Presentations should generally have the following structure:

- [2 minute] Brief motivation and quick overview of problem setting.
- [2 minute] Relation to subject material from class; perhaps also include an equation or pseudocode block that shows in more depth what you have worked on.
- [6 minutes] Some results, what you did, anything interesting you found along the way, and what might not have worked as well as you expected.

The presentations are **short**, but the goal is to give both me and you classmates a high-level picture of what you did and both what worked and what didn't work as well as you'd hoped. The presentation will be 20% of your final project grade, though I intend to grade them pretty leniently: if you have put effort in the presentation, you should expect a good grade on this element; the technical content will be evaluated when I grade the tutorial. Finally, it does not matter much to me who presents, so it is acceptable if only one member of a team presents.

## 4 Project Tutorial Guidelines

In the past, I have asked for the final project to be in the form of a report, one that typically only I read; this feels like something of a waste: you all work hard on these, so it would be nice if they went somewhere!

[3]The exact length of the presentations will depend on how many teams there are to present.

This year, I would like the projects submitted in the form of a **tutorial**: instead of presenting what you've done, the final project *tutorial* will try to communicate to someone else (e.g., someone else in the class) how *they* would reproduce your results. The final project tutorial is typically expanded from the project proposal (except that all "plans" should be replaced by what have been actually done). Specifically, students should clearly describe:

- What problem you have addressed and the motivation;
- What datasets you have used or collected (with procedure) for the project;
- Existing work that studies the same/a similar problem and/or investigates the same/a similar method (including at least 5 references);
- The technical details of the proposed methods;
- Experimental results comparing the proposed methods with the baselines, plus analysis and discussion of the results;
- Possible future work extending from this project.

See more details about what I expect in the tutorial below.

Grading: Note that the final project will not be graded solely based on how well your approach works, as long as you can justify the novelty of your work (if applicable) and convincingly show that your system is doing something (e.g., with careful experimental designs, comprehensive quantitative/qualitative analysis on the results, thoughtful discussions, etc.). Good writing, e.g., being able to clearly describe what you have done and how they are compared with existing work, is also important.

- **Proposal: 10%**
- **Presentation: 20%**
- **Introduction (Motivation) and Related Work: 10%** Give a clear overview of why this problem is worth doing and why someone might want to follow this tutorial. In a (potentially separate) related work section, be sure to reference at least 5 papers for the approach itself or related.
- **Theory, Metric Definition, and Background Information: 15%** What is the metric you will be using to quantitatively evaluate the performance of your algorithms? Include an equation that defines this metric. How will I, the reader, know if your algorithm is implemented

correctly. Describe, using words and equations, what your implementation is doing. Be sure to include how it differs from what we derived and implemented in class. Also reference and briefly describe any datasets of simulated environments you use.

- **Approach, Implementation, and Results: 35%** The core of your tutorial. Go through how one might use your code and open source components to achieve your aim. Be sure to also include images for the sorts of results that can be expected (just as you would include results figures in a conference paper, etc.). Try also to include links or references to how to install any components upon which you rely. Also, be clear of how much code you used from elsewhere versus how much you implemented yourself; your projects should reflect a non-trivial effort, so it is not recommended that you pull a ton of code from other sources. Note: it is my aim that these tutorials may serve as starting points for future projects when this class is taught again; try to write the tutorial to help your peers understand what you've done and how to do it themselves.
- **Clarity, Figures, and Completeness of Tutorial: 10%** Hopefully this one speaks for itself. Try to be clear and to the point. I value clarity over length: please try to keep your tutorial short, yet complete. Wring a long tutorial is not the goal.

Submission format: I will leave this to you! I encourage submissions as (cleaned up) Jupyter notebooks[4], PDF documents, or HTML documents/sites: whatever format you think is appropriate.

Source code submission: while submitting the final project (through Blackboard), you should also submit your source code implementation (in a .zip file). The code does not need to be 100% reproducible, especially since I do not want to receive massive data buckets in your submission, but *I actively look at source code when grading!* I often use the source code to tell how much students implemented themselves versus leaned on available code and how they got certain things working; it is however strongly preferred that this information is all included in your tutorial. Also, you are welcome to put your project on public GitHub if you would like; I have had students in

[4]Note: the Jupyter notebook should be **organized** if that's what you decide to submit. It will be very hard for me to grade a large notebook with. If you submit a Jupyter notebook, I also recommend that you export it to a PDF or HTML so that you can be sure I can view it. It is unlikely that I will run the notebook (much), so that's not an issue.

the past who wanted to make their final projects publicly available.

## 5 A Sample Project Proposal

**You are expected to do a proposal for your project so that I can ensure you are not being too ambitious (or perhaps not ambitious enough).**

Below is an example of a pretty solid project proposal: it is on the ambitious side, though includes a couple of places in which the project could be extended or shrunk if the baseline results take too long or are poor and a fallback option in case things do not work out. A "safe" project proposal might include more off-the-shelf code, but you should aim to replace off-the-shelf modules with modules you implement yourself as you have time. For example, in the proposal below, I suggest that I will use the GMM implementation built-into VLFeat; if I still felt that my project was lacking and I had time, I might consider implementing this myself. Consider including such optional/bonus implementation goals in your project proposal.

You may use this example as a guide for your own proposal, though I ask that if you would like to do this project yourself, you include a non-trivial change—e.g., implementing a different dense image descriptor, or implementing VLAD or Fisher Vectors for recognition—for your own project. I also note that this project is more about implementing an already understood algorithm using a number of off-the-shelf components on an interesting problem or as a tool to achieve some objective of your own design; as a result it's not particularly *creative*. Your projects could be more on the creative side, to try something a bit out of the box using well-understood or off-the-shelf tools, and would require less in the way of implementation to allow for more room to play around and experiment.

> **Title**: Object Detection using HoG and Bag of Words
>
> **Goal**: Detect images that contain my guitar in different places around my house using Bag of Words methods over a HoG-based image description.
>
> **Technical Background**: I plan to use the *Bag of Words* method we discussed in class to classify the images. As I have time, I will compare different similarity measures for the Bag of Words

classifier. I will compare my implementation against a pretrained implementation of AlexNet.

**Data**: I will generate "training images" by taking a few videos around my home. I will separate these images into a positive (containing the guitar) and negative (not containing the guitar) set of images. Collecting a few hundred images using 5–10 short videos recorded on my smartphone should be sufficient for data collection.

**Implementation Plan**

- One of the components of my implementation will be to implement HoG (Histogram of Gradient) features we discussed in class. I will implement a HoG descriptor and then use these features to represent the images.
- *Bonus Goal*: Also extract features using the *dense SIFT* descriptor provided by the VLFeat toolbox and compare performance.
- Compute a "vocabulary" using VLFeat's GMM routines.
- Compute the Bag of Words descriptor for my training images. (Intermediate result: I can compute a small vocabulary for debugging purposes, which would allow me to visualize some of the BoW descriptor of my training images. I will likely include a couple figures showing this in my writeup.)
- Train an SVM classifier, again using VLFeat, to classify images containing and lacking the guitar.
- I will evaluate performance—measured in terms of the number of correctly classified images and the area under the receiver operating characteristic—first on my training images and then on a set of held-out images, which I will also collect myself.

**Comparison Algorithm [bonus objective]**: *Acoustic Guitar* is one of the ImageNet classes, which means that an off-the-shelf deep neural network trained on the ImageNet dataset should be able to detect the guitar. I will compare performance of my classifier against a deep neural network by following this tutorial to get a pre-trained AlexNet.

**Failure Modes**: I suspect that I will not have enough data during the training phase to compute a very good vocabulary for BoW recognition. If my performance is poor and I have time, I may download the ImageNet pretrained vocabulary and see if I can get that to work.

## 6 Some Project Inspiration

Here are some thoughts I had for what a solid project might look like. I have not included what data you might use for each of these projects, and I'm hoping that if you choose to do one of them you will be creative with what data you decide to test on.

- *Object Detection using Bag of Words*: See the example proposal above.
- *Improving Stereo Depth*: In class, we used stereo patch-matching to compute dense depth from stereo images and the results were rather poor. We discussed a few techniques that one might use to remedy this. Belief Propagaion is a technique that would make for a good project (The Graph Cut algorithm might be another). Three-person teams could compare against deep learning algorithms like this stereo depth algorithm or even this monocular depth prediction code.
- *Feature Detection and Description*: Implement one (or probably two) of the other more complicated feature descriptors we discussed in class (e.g., SIFT, SURF). You could compare against a learned feature descriptor using code like LF-Net (warning: code requires an NVIDIA GPU and NVIDIA Docker). Note also that coming up with the "ground truth" feature matches may require some creativity; perhaps you can use a homography to transform an image and compute matches between those so that you know which feature matches are inliers vs. outliers.
- *Optical Flow & Object Tracking*: Though we haven't discussed Optical Flow very much in class, you should still feel free to implement the Lucas-Kanade method for object tracking or so-called "dense" optical flow (Gunnar-Farneback method) and compare it to (perhaps) a learned algorithm. There are learning-oriented approaches to this.

You should also consider exploring some of the image generation tools like DALL-E or Stable Diffusion as a component of your system. CLIP is also remarkable: it lets you turn images and language into low dimensional "embeddings" (think of them as being like a "learned descriptor" for each) and then relate the two. That capability alone is enough for some image understanding and some interesting applications.

Other final projects might explore some aspects of *multi-view stereo* or even *non-lambertian photometric stereo*, which you could implement using simulated data from Blender. Feel free to get creative and find other papers either online or as discussed in our course textbook that inspire new ideas.