442 Final Project Report

442 Staff

1. Due Date

We will take the project reports until **5pm EST, April 26 2022**. This is an absurdly hard deadline and will require basically near-overnight turnaround on the project grading. We've been fairly flexible during the semester, but this is not a flexible deadline. Don't be late.

2. Overall structure of the report

Overall, your report should tell us about what you did. **The report should be four pages**, not including references. We will stop reading at the fourth page, so if you haven't introduced the experiments at the end of the fourth page then we'll assume you don't have experiments. This is harsh, but plenty of forms that you'll encounter in life have similar requirements.

This report should follow the CVPR format, for which there are LaTeXand MS Word templates. You can find the templates here (link to a .zip file that may auto-download). We use this format not due to any particular special attachment, but rather to standardize what 4 pages means. **You do not have to use LaTeX.** If you'd like to try, Overleaf is a great way to try it without installing it on your computer.

2.1. General overview

A good way to structure things would be the following sections (and their roles in the paper):

1. Abstract: You do not need an abstract. Please do not include one.
2. Introduction ($\leq$ 1 page): what did you do and why do I care?
3. Approach: what did you do in detail?
4. Experiments: how did you test that you did what you wanted to do?
5. Implementation ($\leq 1/4$ page): what stuff did you do, what stuff did you rely on others for.
6. References: refer to anything you used or any things you'd like to cite. References do not count for the page length, so cite away.

If you collected a lot of data yourself, you may want to add a section on data. If you think there's another section that would be important to have, feel free to include it. If you'd like to do a different format, that's maybe ok (although it's better to first understand a format before you ignore it, and the format must be understandable); if you deviate from the format, you'll need to cover all of the sections below in one way or another.

2.2. Sections of the report and requirements

Here are some descriptions of the sections and the questions you should address. You should always have an introduction, approach, experiments, and implementation.

Introduction ($\leq$ 1 page): A good introduction (at least for this report) should include:

1. Please motivate the problem you are solving and the method you are using.
2. Why should I care about the problem? Does it have an impact on life? That's not a rhetorical question – even people who study pure math have to motivate their problems (often via protein folding).
3. Why should I be able to solve the problem? Why should I be able to solve it with computer vision today?
4. What are some related pieces of work? Maybe mention 1 – 3. You may be building on these.
5. Finally, please summarize your method in at most three sentences. Less is more here.

Please don't address these in question form one after another – tell a story that addresses these questions!

Approach: A good description of an approach should let a knowledgeable expert reproduce things. Given the page length, that's going to be tricky. You'll need to decide what stays in and what does out. A common trick is to build on the work of others: the sentence "For finding the puppies, we train a maskRCNN detector on a Resnet-50 FPN backbone using ADAM" might require hundreds of pages to explain in a standalone fashion, but a handful of citations. Here are a few guidelines:

1. If it's a deep network, describe at least the architecture, the loss functions, and the training procedure.
2. If it's an algorithm, describe the steps in a coherent and simple way. If you have lots of steps, divide them up into sensible groups.
3. Please draw **at least one figure** that helps the reader

understand the method. I draw all of my figures in powerpoint. Google slides is also good. You are free to use powerpoint figures from the class slides, so long as you properly credit them. The same goes for other peoples' figures.

Experiments: Please describe experimental validation of your project. You must include both qualitative (i.e., outputs of the system for humans to look at, like pictures) and quantitative (i.e., numbers quantifying how well the system works). A section should probably include the following:

1. Data: What data did you test things on? Why does the data make sense?
2. Metrics: How do you measure success? Is it qualitative or quantitative? Why is it reasonable? What are some potential downsides? You should have SOME measure of quantitative evaluation, even if it's someone looking at results.
3. Qualitative results: please include some (i.e., at least one) figures showing your results.
4. Quantitative results: please include some quantitative numbers. You should compare against, if possible, some sort of simple baseline like random chance. Random chance is usually easy to compare with.

Implementation ($\leq 1/4$ page): If you are building on the code of others, please indicate what is yours and what was others' work. You almost certainly have built off of the work of others. That's expected! But it's important to identify what is yours and what is someone else's.

Data: If you've spent a lot of time collecting data, you may want to explain what you did.

References: You should cite stuff that you use. You can cite stuff to get out of explaining complicated stuff like the particulars of the deep network you used.

Special instructions for special situations: Invariably, some people will fall into these categories:

1. Your system doesn't work because your idea just doesn't work and you now understand why: Identify why, and describe why. The best researchers will routinely pick apart their past ideas as missing the mark and that's fine. Most interesting projects don't work, and part of the point of doing a project is to identify if it will work.
2. Your system doesn't work because we can't get it to work: Identify some concrete subgoals that you have achieved and describe them.

Figures, Credits, etc.: You will build on the work of others. That's good! You just need to be precise and careful about what you claim credit for. See the later section on using open source code.

3. Showcase Video

Instead of a poster presentation, you will participate in a virtual poster session via 4 minute video of your group members presenting your project. With your permission, we'll share this with: (a) either just us, the course staff; (b) or the rest of the class.

The video should be approxiamtely 4 minutes. It's hard to hit four minutes to do in one go, so we're more flexible. But it absolutely must not be more than 5 minutes and if it's under 3 minutes, you likely need to add more material.

This video doesn't need to be fancy or overly produced; talking over a set of PowerPoint slides is perfectly acceptable. We will provide a google drive (and then share the sharable videos with the rest of the class).

The exact format and structure of the presentation can vary depending on the nature of your project. But you should clearly cover the following:

1. What is the problem you were solving?
2. Why is this problem important?
3. How have people tried to solve this problem before? Give a brief summary of 1-3 key pieces of related work.
4. How do you approach the problem? What is your key insight?
5. What are your main experimental results? You don't have to go into all details, but you should mention your main experiments.
6. What challenges remain? If you were to keep working on this project, what would you do in the future?

Showcase FAQ.

- **Is this synchronous?** No.
- **Do I need fancy video production?** No. Some of the best speakers I know of have genuinely quite terrible visuals. They're not flashy, but they are effective.

4. Writing advice for the report

It is true that 442 isn't a course on writing. But poor writing and communication can get in the way of you getting credit for what you've done. There's a somewhat infamous quote from AI pioneer Patrick Henry Winston that says "Your careers will be determined largely by how well you speak, by how well you write, and by the quality of your ideas, in that order."

Here are some general tips, accumulated from various feedback I've given on technical writing:

1. **(Structure) Tell a story:** Good technical writing convinces people to read. In the case of the final report, we're obliged to read the document. However, most of your audiences in life will be *apathetic*. You may want to pose questions. This way we'll read on to find the answers. You want to maintain a narrative so we know where things are going. You want to tell a story so we root for your system to succeed.
2. **(Structure) Your goal is to find the best paper in the page limit, not make the best paper fit in the page limit.** What you did obviously won't fit the page limit and so you have to lose information when you compress the idea. The question is not "can I make it fit" but "what are the most important parts of the story" and "what is the best 4 page story".
3. **(Structure) Abstraction is key:** Before you explain details, give the sketch: what are the inputs, what are the outputs, what's the main goal? If the writing flows across the sections and requires long chains of thought, refactor. You should want to have a single section for each thing. Don't have previews, don't bring things back
4. **(Structure) Be ordered:** Don't explain details for methods before you've introduced them.
5. **(Writing) Don't write:** Writing is often a terrible medium for communication. If you can use a figure, go for it. If you can refer to a paper to explain it, do it. If you don't need to mention it, don't mention it.
6. **(Writing) Don't be complicated:** The temptation is always to make things complicated. Don't! The highest praise for an idea is that it is clean and neat!
7. **(Writing) One point per paragraph:** Every paragraph should have exactly one point. Some paragraphs have more than one point and they end up being confusing. Some paragraphs have zero points and the reader is then sad.

   Moreover, that point should be summarized in the first sentence. This helps with people skimming. If your main point is halfway through the paragraph, people tend to miss it / get confused. The same logic applies to sentences and sections. However, people often don't have this issue because they use sentences a lot and because the sections are helpfully labeled.
8. **(Writing) Be clear:** obfuscation *does not* work. Rejection is the default in life. You need to positively prove that you've done good work, not hide the details and hope that "nobody takes points off".
9. **(Actually Writing) Use a spellchecker:** It's hard to catch typos, and I always have a few. If you use overleaf there's the classic red squiggle thing (and if you use word there's a spellchecker). If you edit .tex files on your computer, use aspell (aspell -c file.tex).
10. **(Actually Writing) Read it out loud:** You can't catch sentence fragments because you wrote them, but they jump out to casual readers. Reading the paper out loud helps. Writing then reading the next day helps even more. If you want to read out loud like a professional editor, I have been told that reading the sentences in reverse order is very useful.
11. **(Actually Writing) Rewrite:** Few people are blessed to write well the first time. For almost everyone it is far easier to write *something* and then edit until it looks better. Most of my good papers involve total rewrites.

5. Using Open Source Code

The question of "can I use existing code?" is complicated because it means lots of different things. We've answered a few questions and so we'll try to answer it once and only once. In general, open-source code is great. If we had to re-invent the wheel everytime we built something in science, we'd be in trouble. Hopefuly you build your system on top of open source code!

Before anything else, if you're worried, we understand. This is tricky. Please feel free to talk to us. We're happy to help you.

The key question is: **what value are you adding on top of the existing source?** This value-added needs to be clear to you and it needs to be clear in the report. Above all else, you can't just pass off other peoples' code as your own, and you can't just copy-paste stuff in.

Here are some examples along the spectrum, using colorization as an example.

1. Rewriting a deep learning library from scratch in assembly and making a colorization system with it. **Well that's something. Totally not necessary but interesting!**
2. Implementing colorization from scratch in PyTorch using standard libraries. **Great! That's exciting.**
3. Using someone else's super duper cool data loader for semantic segmentation, adapting it for colorization, and saying "we did colorization and we used X" for data loading. **Great! You explained what code**

is from where. Colorization's a different problem than semantic segmentation and so there's value added here.

4. Running colorization and evaluating your results using a standard library. **Great! Totally fine. Using someone else's evaluation is good since evaluation code is hard to write.**
5. Running someone else's code for colorization and saying "we did colorization by `git clone && python demo.py`". **No good, but honest. There's not much work in running git clone. Can you do some more work on top of this library?**
6. Reading someone else's code for colorization and then implementing it from scratch yourself. **Not ideal, especially if you're doing a side-by-side read. This is the real danger territory. Don't do this.**
7. Running someone else's code for colorization and saying "we implemented colorization". **No good. There's not a lot work and you're passing off someone else's code as your own.**

Here are different ways that you could interact with other code:

1. **Libraries/packages** enable you to use existing code in a nicely packaged API format. These are always safe so long as they do not trivialize the problem. Calling `yolo.detect()` does make for a real object detection project. But you can use YOLO if you're trying to track giraffes.
2. **Officialish tutorials** are probably fine if it's not a line-by-line description of code to do the entirety of a task.
3. **Course code** is always fine.
4. **Unrelated small (< 50 line) snippets** like data loaders, training loops, unusual losses, GPU magic. Totally fine if they enable you do something cool and so long as you do not pass it off as your own.
5. **Github repos that implement what you are doing** are not ok.

Unfortunately, the peril of open-ended stuff is going to be that it's hard to draw exact lines without forcing people into a few pre-made projects, and without sucking the fun out of the project.

6. Grading Scheme

Each component of the project grading will be based on $\approx 50\%$ apparent amount accomplished and $\approx 50\%$ quality of presentation. If peer ratings suggest that something has gone seriously wrong, there's at most a $\approx 25\%$ additive penalty based on optional peer ratings (if applicable – mostly this will be ignored).

Peer Review: We will also optionally invite the team to comment on the relative contributions of its members. Historically this has only been an issue for 1 or 2 groups a term. If there have been people who have been total free-riders or people who have carried the team. If your project has gone as well as any group effort, then you do not need to fill this out. Remember: group projects do not go away as you get older, they just get fancier titles.

We will ask you to rate all the team members' contributions from 1-3:

- 1 is $\frac{1}{3}\times$ the work of an average student (i.e., if the person contributed, it was probably negatively)
- 2 is the average student
- 3 is $3\times$ the work of an average student (i.e., the project would have died an ignominious death without this person).

Most team members are a 2 – this is not Uber or WeRateDogs – and we do not want to litigate differences between 25% and 35%. However, if someone refuses to show up to any meetings, they should not get credit for the work of others.

Are we ranking projects: No. We're not stack ranking people. Given the spread of experience coming into the course in computer vision, this is a terrible idea – then the project is a measurement of what you knew coming in rather than what you learned along the way. The lack of stack-ranking is especially important for projects that are pre-baked. There's not a finite number of high grades to be given out for depth prediction.