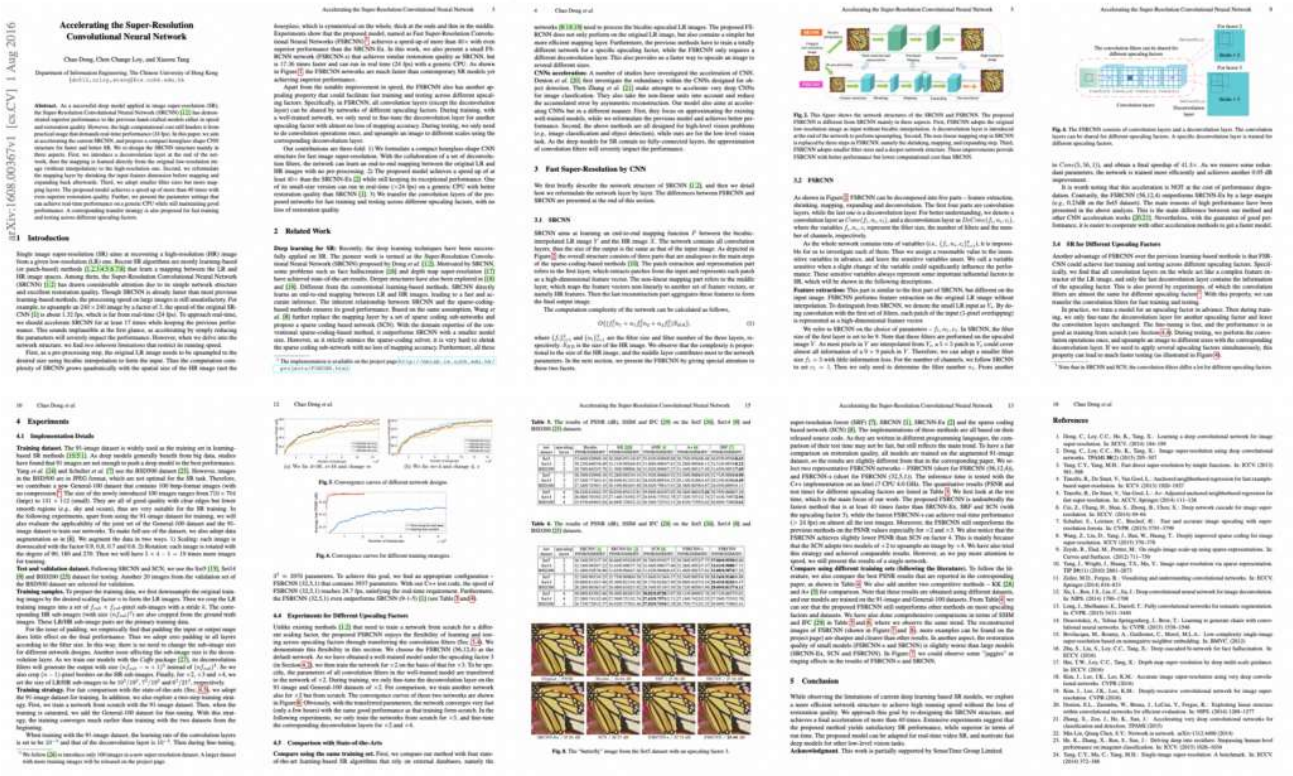
Published in Geek Culture

Olga Chernytska Follow

Dec 17, 2021 · 7 min read · Listen

Learn To Implement Papers: Beginner's Guide

Step-by-step instructions on how to understand Deep Learning papers and implement the described approaches.



Source: https://arxiv.org/abs/1608.00367

If you want to be a thinker, understand what is happening inside a black box, boost your creativity or become the first developer who brings recent scientific research into business — this post is for you.

Today we will discuss how to choose a "good" paper to start with, that will be relatively easy for a beginner; we will overview a typical paper structure and where important information is located; I'll give you step-by-step instructions on how to approach the paper implementation and share links that should help you when you get stuck.

Where to start?

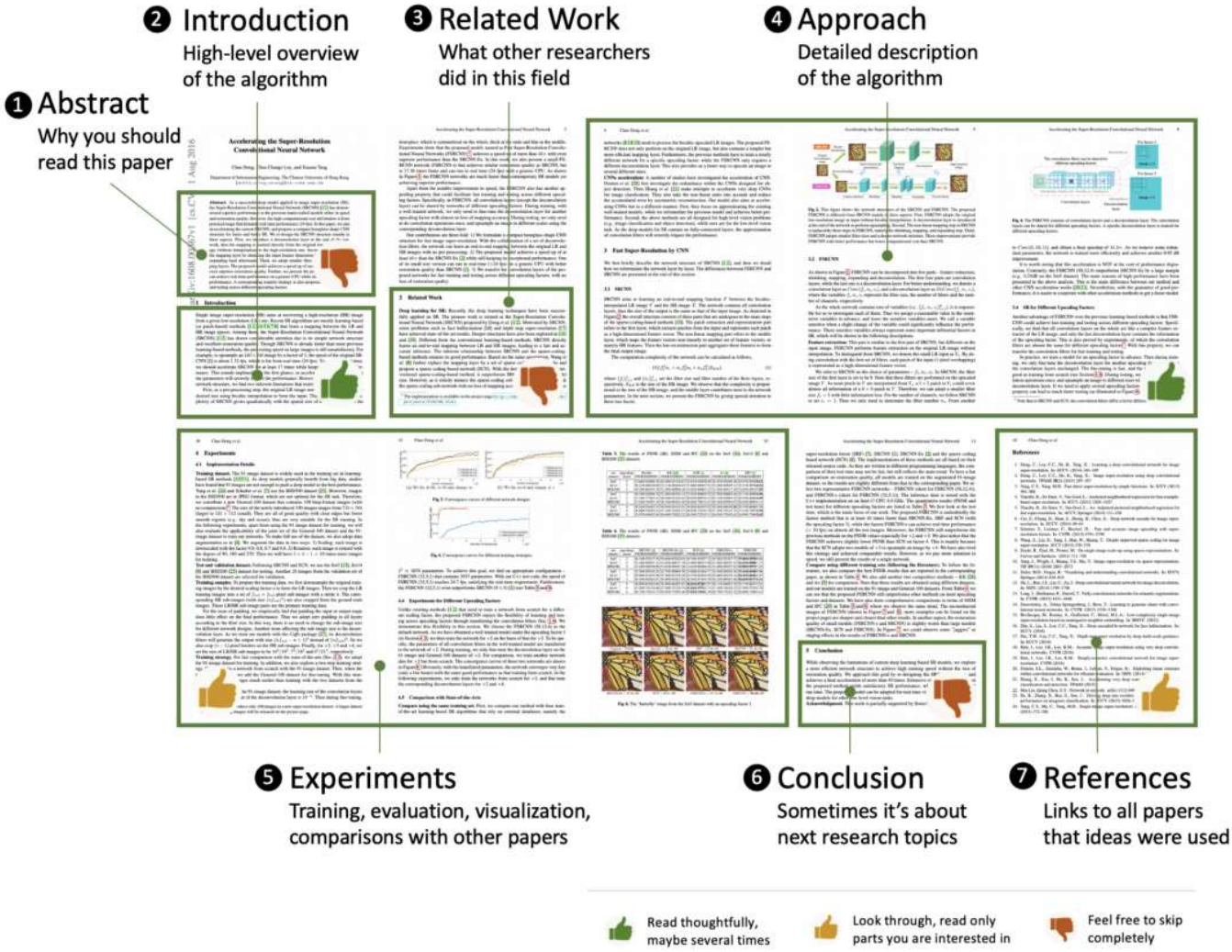
If you want your learning to be smooth and stressless, you should find a "good" paper. As a starting point, I really recommend you to choose an ***old and highly cited paper*** that describes the concept you're familiar with.

- Old highly cited papers usually explain very fundamental concepts that became a basis for more recent research. You know the fundamentals — you'll better understand recent papers as well. For Deep Learning, papers before 2016 are considered to be already old.
- Highly cited papers are reproducible. This means that many other scientists were able to understand and implement the approach. To find out the number of citations for a particular paper, google it in Google Scholar. A paper with more than 1000 citations is considered to be highly cited.
- Usually, older papers describe simpler concepts, which is a big plus for you as a beginner.

Paper structure: what to skip, what to read

Typical Deep Learning paper has the following structure:

1. Abstract



Structure of a typical Deep Learning paper. Image by Author

1. Abstract is a "marketing" summary. It is very short and focuses on why this approach is better than previous ones and what is novel about it. Abstracts are published in the conference schedules and online archives (such as Arxiv), and their only goal is to convince you to read this particular paper. You've already chosen a

Introduction section first, warm up your brain before diving deeper into algorithm details.

3. Related Work. All scientific papers (and Deep learning as well) are related: every discovery is built upon the work of dozens of researchers before. Related Work overview is a required section for every paper. Authors have to make sure that their work is relevant, solves important problems, and doesn't repeat work done before by other researchers. It is an important section for the scientific community — but not for us (practitioners), so skip it!
(Okay. Sometimes you may need it — but only in case you are looking for other fundamental papers/concepts in the field to read.)

4. Approach in Details. Here is where the fun begins. It is the most complicated and challenging section in the paper, and the most important (read it for sure!). Don't put your expectations too high and don't expect to understand everything from a single read. It is the section you are going to come back to again and again while coding.

Don't be afraid of complicated formulas, in most cases, they explain basic concepts. I believe that's how the researchers make jokes. In a little while, you'll get used to it.

While reading the paper catch all the information you may need — data pre-processing techniques, detailed neural network architecture, loss function, training tricks, and post-processing. Try your best to get it. It's okay if you cannot understand something even after several attempts, later I'll tell you what to do with that.

5. Experiments. This section is full of charts, tables, and images. Usually, it contains detail on datasets, training, and evaluation, as well as a review of how the model performs with various hyperparameters and compared to state-of-the-art approaches from other papers. If the paper is on Computer Vision, there will be visualizations of model predictions as well. I would say that this section is to read partly, the only what you are interested in.

6. Conclusion is the summary of the paper, sometimes it contains the authors' ideas

reference to the original work (citation). Most of such references you'll probably skip when the concept from these references is already explained in the paper, or just unimportant. However, sometimes, authors may say: "We used model architecture described in paper [2], only modified the activation in the final layer". In this case, you'll need to find and read the paper [2] to fully understand the approach.

And now — it's time to read the paper. Turn off the music, switch your phone to airplane mode, and take a cup of tea. For the next 30 minutes, you should be highly focused, as you are diving into a new world — exciting, but quite challenging.

Where to find help?

For many people "implement the paper" means "quickly read the paper and then look for ready implementations over the internet". It is the easiest way, but not the rewarding one. I really recommend you start from scratch and not look for the ready solutions right away. Do at least something by yourself — that's the time when you are learning.

Even if you are a complete beginner in implementing papers, there is always something that you can do:

Download the dataset, explore it, write a data loader.

Simple and easy task, but when done it gives you confidence and helps to move on:

- Start writing model architecture, simplify or skip the parts you don't understand. There is a strange weights initializer — skip it for now and use the default one. You've never worked with PReLU activation before — use ReLU instead. Your goal now is to create a trainable model, not the model from the paper or the model with good performance, just TRAINABLE. It has input, it has output, so you can run the training.
- There is a custom loss in the paper — replace it with a similar one implemented in the deep learning library.


You'll end up with the draft. You may even train the draft and see how it goes — maybe the results will be not so bad 🙂

Then fill the gaps and fix the poorly working parts. First, experiment on your own — test ideas that came to your mind while writing the draft, read the paper once again, and hopefully catch the concepts you previously missed. Don't be upset if you get stuck very quickly. You've created a draft, and it is great progress, you've already learned a lot. So next time and with the next paper you'll draft will be much better. It is a learning process.

Feel completely stuck? Perfect time to google.

Remember, I recommended you choose a highly cited paper. And now you'll feel the benefits. Popular papers have dozens of implementations over the internet and blog posts describing the complex parts. Enjoy!

The first place to check is Papers With Code, a large library with code implementations of probably all the popular papers. These implementations are official or from researchers like you and me. For instance, word2vec has 67 implementations on Papers With Code available in PyTorch and Tensorflow.

You may copy-paste, but invest time in understanding that code. That's the last piece of advice.

Good Luck!

Once again:

1. Choose an old and highly cited paper.
2. Thoroughly read it and try to catch as much as you can — data preparation, model architecture, loss function, and training details.
3. Don't be upset if you don't understand everything.


5. Try to improve the draft on your own.
6. When you get stuck — look over the internet for posts and code with the paper implementation. Copy-paste, but read and understand.
7. Wrap up your work as a Github project (why not?). Look, how I did it.
8. Repeat with a new paper. And feel how smoother it goes the second time 🙂

True learning is happening on steps 2–5, so the more time you spend here — the faster you learn. Good luck!

What's next?

If you'd like to read more about paper implementation, check my other posts:

- Learn To Reproduce Papers: Beginner's Guide (it's a longer version of the current post accompanied by a paper implementation example)
- Word2vec with PyTorch: Implementing the Original Paper