Assignment 1 - Initiate

Due 22th October 2024

OVERVIEW AND GOALS

In this exercise you will research the topic that you want to work on during this course and plan your work. You should read scientific research, understand what's currently possible and produce a detailed plan of your next steps, including a time estimation for each step.

DELIVERABLES

1. References to at least two scientific papers that are related to your topic
2. A decision of a topic of your choice (see below for inspiration)
3. A decision of which type of project you want to do (see below)
4. A written summary that should contain:
    a. Short description of your project idea and the approach you intend to use
    b. Description of the dataset you are about to use (or collect)
    c. A work-breakdown structure for the individual tasks with time estimates (hours or days) for dataset collection; designing and building an appropriate network; training and fine-tuning that network; building an application to present the results; writing the final report; preparing the presentation of your work.

The result can be a single README file (Markdown can improve the visual appearance significantly) or a PDF document. Please put everything into a Git repository that can either be public or private. In case you prefer your work to remain private, create a private repository and grant us read-access to it (both the user handle **`apacha`** and **`padraig20`** on Github or **`apacha`** and **`e12122372`** on TU Wien's Gitlab). Please use a spell-checker, e.g., Grammarly.

Once you're done, please notify us by writing an e-mail to applied-deep-learning@outlook.com with the subject **[Applied Deep Learning] Exercise 1 - Your Matr.Number** with the link.

TOPICS

The list of topics for your project includes but is not limited to:

- Computer Vision
  - Classification of Images
  - Detection of Object
  - Segmentation of Objects
  - Style Transfer
  - Text Recognition
  - Image Colorization
  - Image Generation and Adversarial Image Generation
- Audio Processing
  - Music and Genre Classification
  - Tracking of musical concepts, such as beats, key, onset, and tempo
  - Audio source separation
  - Music audio generation
  - Harmonizing music
- Natural Language Processing
  - Fake News Detection
- Reinforcement Learning
  - Control a cart
  - Play chess

The list above should be understood as inspiration for you to come up with a suitable topic. You can also find interesting project ideas on Kaggle or by looking at research papers with their respective source code at http://paperswithcode.com/ and https://huggingface.co/.

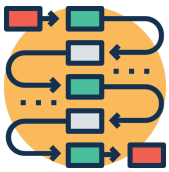
PROJECT TYPES

Bring your own data

The focus of this project type is to collect a comprehensive dataset that is suitable for training deep neural networks. Typically this project would include collecting a large number of samples (hundreds or thousands, depending on the topic) in a semi-automatic way and then starting to annotate these samples for supervised learning. Although the focus of this project type lies in collecting a dataset, you should at least train and run a simple neural network that showcases how to work with your data and establish a baseline for future researchers to refer to.

You will get bonus points if you make your dataset publicly available at the end of the course.

Bring your own method



The focus of this project type is to build or re-implement a neural network architecture that operates on an existing dataset that is already publicly available. Your method should reflect the state of the art, so it is fine to use an existing implementation but you should alter it and attempt to find ways how you can improve the results.

You will get bonus points if you can improve the state of the art.

Beat the classics

Deep Learning hasn't been around for a very long time, so many tasks can still be efficiently solved with traditional approaches. In this type of project, you will pick a problem, for which there is an established traditional algorithm (e.g., edge detection) and try to come up with an alternative approach that uses deep learning to do the same thing. Keep in mind that you must use the same metric for both approaches to have a fair comparison.

You will get bonus points if your approach works better than the traditional approach.

Beat the stars

Deep Learning can significantly improve the state of the art, often by making small incremental changes. The number of neurons, a clever loss function, a novel regularization method, or a specific kind of layer can have a large impact on the results. Many of these ideas are being published on a daily basis on Arxiv.org and are awaiting their adoption. In this type of project you would pick one of these novel ideas (e.g., Deformable Convolutional Layers) and try to beat the current state of the art on a specific problem. Given the fast pace in the field, it is acceptable to beat state-of-the-art results that are up to a year old.

You will get bonus points if you actually manage to beat the state of the art. You may even consider publishing your work in a scientific conference or journal.

CRITERIA FOR GRADING

As discussed in the preliminary lecture, every assignment will be graded according to these five criteria (as applicable) with a maximum score of 10 points:

- Results
- Creativity
- Complexity
- Code Quality
- Presentation

Late Policy

Submissions will still be accepted after the deadline, but keep in mind that there is a penalty of one point per day. So if you submit your assignment two days late, you will only be able to achieve a maximum of 8 points.

OTHER QUESTIONS

In case you have other questions regarding the assignment, please send an e-mail to applied-deep-learning@outlook.com or post your question in the discussion forum in ILEA (preferred).

Assignment 2 - Hacking

Due 17th December 2024

OVERVIEW AND GOALS

In this exercise, you will deep dive into your project, try to build a baseline model that operates on your data as one would expect (although not necessarily in an ideal way). Then try to optimize your neural network as much as you can (given your available time budget), i.e., perform hyperparameter optimization and/or gather more data.

N.B. If you picked the project type “Bring your own data,” you still have to build and train a suitable pipeline, even if it is a very simple one. The results will then serve as a baseline for your newly created dataset.

RECOMMENDED APPROACH

1. Specify an error metric (how you want to evaluate your algorithm) and a reasonable target value for that error metric (how good it should be)
2. Establish a working end-to-end pipeline as a baseline for further experimentation
3. Instrument the system to detect defects or bottlenecks that cause underfitting/overfitting
4. Repeatedly make incremental changes, i.e., gather new data, adjust hyperparameters, change algorithm depending on new insights

Be aware that hyperparameter optimization can be costly and time-consuming, so start early and stop if you’ve reached your own target. Gathering new data can be more useful than you think and there are many ways to artificially expand your dataset at low costs (= data augmentation).

Whenever you work on your project, please keep track of the time to help you better understand which parts of the project potentially require more time than you initially estimated.

DELIVERABLES

For this exercise, you are expected to implement the project you specified in exercise 1. Changes should be documented in your Git repository. Please also provide a brief summary (e.g., in your README) of

- the error metric you specified
- the target of that error metric that you want to achieve
- the actually achieved value of that metric
- the amount of time you spent on each task, according to your own work breakdown structure.

Keep in mind that the actual amount of time you spend on your exercise does not affect grading!

As this is a software project, students are expected to adhere to the principles of good software design, which include:

- Use of good names for your variables, methods, functions, and modules. These names should be self-explanatory, easy to understand, and intention-revealing. They may be short if they are confined in a small scope (e.g., 'i' for a counter in a loop) but should be long enough to grasp their intended use (e.g., 'input_image', 'number_of_epochs', 'training_batch_size'). Avoid abbreviations because they are hard to understand. The only exception to this rule is if the abbreviation is better known than its spelled out counterpart (e.g., 'HTML', 'XML', 'HTTP').
- Providing as much documentation as necessary (useful documentation includes architectural diagrams or comments in the code that explain *why* a certain operation was implemented in a specific way). Avoid useless comments that simply repeat what is already in the code (e.g., 'Computes the scaling factor' for a method called 'compute_scaling_factor'). A great way to create documentation directly from the source code is by using Read the docs or similar tools.
- Test your code. Create at least a few tests that make sure your pre-processing and post-processing work correctly. Testing the training procedure itself is difficult and therefore not mandatory (but you will receive bonus points if they are done, e.g., train on the CPU of the CI with a tiny fraction for a single epoch). Ideally, you write the tests before the actual implementation (Test-Driven-Development).
- Make sure your application can be run easily, document how to do so, and ensure that the tests are green. A great way to do this is by using a Continuous Integration server that automatically builds your application and runs the tests on each commit. There are plenty of free services available, including Travis, Gitlab, AppVeyor, CircleCI, Bitbucket Pipelines, Azure DevOps, or Bitrise (depending on your platform and programming language). Setting up a CI in the beginning can seem like unnecessary overhead, but it will help you significantly in the long run.
- If you use Python, please specify which version and create at least a requirements.txt file that specifies the used libraries, such that they can be installed easily in a fresh virtual

environment. Ideally you use a complete dependency management tool like Poetry or Pipenv.
- Please auto-format your code before you submit it. All major IDEs offer this functionality.
- If you have large files (training data, model weights, ...), use tools that were designed to store them, e.g., Github releases. Do NOT commit them into your Git repository. Git is not designed to store and keep track of large binary files. While Git LFS *can* be used, be aware that Github has quotas that limit the free use of Git LFS.

There are several platforms that can help you execute and log your experiments, including Google Colab, Weights and Biases or Gradient.

Once you're done, please notify us by writing an e-mail to applied-deep-learning@outlook.com with the subject **[Applied Deep Learning] Exercise 2 - Your Matr.Number**. Since you've already given us access to your repository, no more action is needed.

CRITERIA FOR GRADING

As discussed in the preliminary lecture, every assignment will be graded according to these five criteria (as applicable) with a maximum score of 10 points:

- Results
- Creativity
- Complexity
- Code Quality
- Presentation

Late Policy

Submissions will still be accepted after the deadline, but keep in mind that there is a penalty of one point per day. So if you submit your assignment two days late, you will only be able to achieve a maximum of 8 points.

OTHER QUESTIONS

In case you have other questions regarding the assignment, please send an e-mail to applied-deep-learning@outlook.com or post your question in the discussion forum in ILEA (preferred).

Assignment 3 - Deliver

Due 21st January 2025

OVERVIEW AND GOALS

In this third and final exercise your task is to demonstrate what you have achieved in the previous few months. You will do this in three different forms:

- Build a small demo application that runs your trained model and performs inference
- Write the final report of your project
- Prepare a presentation that we'll watch together in class

DELIVERABLES

Demo Application

Even the best scientific research is worthless if you can't share your knowledge and communicate your results. Therefore, you are asked to build a small demo application that a potential user could run to understand what you have done. There are many ways how you can realize this. In its simplest form, it is a command-line tool that takes the user input and produces a file as a result. A more sophisticated way would be to have a docker container that wraps such a command-line tool and could be deployed to a server for others to use. But depending on your project, a completely different form might be appropriate.

A great way to build a sample application is to develop a simple JavaScript web application that runs your trained model in the browser and provides a minimal interface to trigger inference. Some tools for doing this are ONNX.js, Tensorflow.js, Streamlit, Gradio, Anvil, Dash, and ConvNet.js. You can upload your trained models to Github (as an attachment to a release - don't commit it into the repository!) and dynamically fetch them when the user opens the webpage.

As a reference, you can play with the Deep Optical Measure Detector, which was developed for recognizing the structure (measures) in music scores. It is a Vue.js application and available on Github. For ONNX.js, there is also a range of publicly available demos that you can adapt.

Final Report

Your final report should contain a concise and informative written summary of your project. A large section of that report can be taken from your project proposal with appropriate edits. The entire report should be no longer than 5 pages.

It should answer the four fundamental questions:

- What is the problem that you tried to solve?
- Why is it a problem?
- What is your solution?
- Why is it a solution? (And in particular, why is or isn't deep learning a solution?)

Additionally, it should cover:

- The main take-aways and insights you gained from your project (e.g., batch-normalization improved the results significantly, Adadelta worked much better than SGD on my data, annotating data works really good with tool X, setting up the pre-processing takes much more time than I expected, ...)
- If you would do the same project again, what - if anything - would you do differently?
- How much time did you spend on your project? How does the number compare to your initial estimate? If you underestimated any part, what were the reasons for this?

The final report should be submitted as a single PDF file.

Presentation

Prepare and record a presentation that we will watch together in the last two lectures during class. The video must not be longer than four minutes. The first two minutes of that video should contain a quick introduction into your topic, the selected approach and the final results. The remaining two minutes should focus on insights that you want to share with your colleagues and ideally a small demo of the project.

Please record yourself giving the presentation in the best quality available, upload your presentation to YouTube, and send me the link. You can use free software like Davinci Resolve to edit your videos. The videos will not be made public, unless you want to. I will collect all the videos and create a playlist that we will watch together with one or two questions in between presentations. **Please don't flag the video as "specially made for children" or I won't be able to put it into a playlist.**

Please reserve the full 2 hours from 16:00 - 18:00 for the last two lectures. For fairness, all videos must be handed in before the first lecture and the actual presenters will be selected randomly.

Submission

Once you're done, please notify us by writing an e-mail to applied-deep-learning@outlook.com with the subject **[Applied Deep Learning] Exercise 3 - Your Matr.Number** and the final report as an attachment, as well as the link to your YouTube video.

CRITERIA FOR GRADING

As discussed in the preliminary lecture, every assignment will be graded according to these five criteria (as applicable):

- Results
- Creativity
- Complexity
- Code Quality
- Presentation

The demo application, the report, and the presentation are each worth up to 10 points.

Late Policy

Submissions will still be accepted after the deadline, but keep in mind that there is a penalty of one point per day. So if you submit your assignment two days late, you will only be able to achieve a maximum of 8 points.

Please note that the presentation (video) is exempt from this policy, as we watch the videos in class together. If your presentation is absent, it will receive 0 points.

OTHER QUESTIONS

In case you have other questions regarding the assignment, please send an e-mail to applied-deep-learning@outlook.com or post your question in the discussion forum in ILEA (preferred).