Speeding up the deep learning development life cycle for cancer diagnostics

Marc Päpper
30.07.2021
Our Mission

Increase cancer diagnostic accuracy and make it accessible to everyone who is in need.
Cancer diagnostics today

1. Biopsy
2. Examination
3. Diagnosis
4. Treatment
Future cancer diagnosis **not** for everyone?

Demand for pathological diagnostics is increasing by 6% globally.

But numbers of qualified pathologists cannot meet this demand.
Cancer diagnostics tomorrow

Scanner → AI pre-analysis → Check on screen → Improve model
About MindPeak

• Automation tools for visual diagnosis in pathology
• Support cancer experts for reliable and reproducible diagnoses.
Our Team and Advisors

Prof. Markus Tiemann
Pathologist and Managing Director at Institute of Haematopathology Hamburg

Prof. Axel Wellmann
Pathologist and Managing Director at Institute of Pathology Celle

Dr. Thomas Fenner
MD for Humane Medicine and Head of Laboratory Fenner

Geoff Baum
VP of Product at Adobe, co-founded Garage Ventures with Guy Kawasaki, U Stanford
Example: cancer cell detection
Robustness against many variations in the lab
Robustness against many variations in the lab
Simplicity

One AI for all labs

Lab 1
Lab 2
Lab 3
Lab 4

AI 1
AI 2
AI 3
AI 4

Other Vendors
Goal: Test new ideas quickly

- Idea
- Implementation
- Training & Evaluation
Stage: Idea
Overview: Idea stage

- Idea generation
  - Without data
  - Data-driven
- Efficient Annotations
- Metrics - define your target goals
Idea Generation - without data

- Use brainstorming techniques
  - But avoid groupthink
- Organize and group cards together
- Have a short, timeboxed discussion
- Then use limited voting sticky dots
- Evaluate e.g. with the Business Model Canvas
Data-driven idea generation

- Use a subset of your validation set for decisions:
  - Categorize the errors
  - Focus on high error categories

Example: let’s improve Stroma -> Immune misclassification!

<table>
<thead>
<tr>
<th>Image number</th>
<th>Stroma -&gt; Immune</th>
<th>Immune -&gt; Stroma</th>
<th>Scan artefacts</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>99</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>100</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Total Counts</td>
<td>4</td>
<td>7</td>
<td>35</td>
</tr>
</tbody>
</table>
Efficient Annotations

VS
Metrics - define your target goals

- Single metric for comparability
  - Aggregates like F1
  - Weighted average of aspects
  - Min operator to enforce equal importance
- Single metric validates your ideas
- Sub metrics
  - figure out where to improve
  - guide to generate next ideas
Metrics - Mindpeak example

**Cell Detection:** Precision + Recall $\rightarrow$ F1 Score

**Cell Classification:** Precision + Recall $\rightarrow$ F1 Score for each class
Weighted combination of single F1 scores $\rightarrow$ Overall F1 Score

**Target Metric:** Combination of Detection F1 Score + Overall Classification F1 Score
Idea stage – summary

• Think about efficient annotations
• Define single metric to guide experimentation
• Use your data to drive ideas
• Track errors your current model makes and categorize them
• Identify ideas matching high error categories
Stage: Implementation
Overview: Implementation stage

- Code quality (Linter, Typing, Refactorings)
- Automation: CI pipelines
- Profiling:
  - Interplay CPU vs GPU usage -> Nvidia Profiler Nsight
- Reproducibility

```python
def greeting(name: str) -> str:
    return 'Hello ' + name
```
Code quality - comments as code

```python
def loss(predictions, targets, background_index):
    # filter out background targets
    mask = targets != background_index
    predictions = predictions[mask]
    targets = targets[mask]

    loss = F.l1_loss(predictions, targets)

    return loss

def _filter_background_targets(predictions, targets, background_index):
    mask = targets != background_index

    return predictions[mask], targets[mask]

def loss(predictions, targets, background_index):
    predictions, targets = _filter_background_targets(predictions, targets, background_index)

    loss = F.l1_loss(predictions, targets)

    return loss
```

https://www.paepper.com/blog/tags/refactoring/
Code quality - use einops library

```python
def _flatten_by_channel(prediction):
    batch_size, channels, height, width = prediction.shape
    permuted = prediction.permute((0, 2, 3, 1))
    final = permuted.contiguous().view((batch_size * height * width, channels))
    return final
```

```python
def _flatten_by_channel(prediction):
    return rearrange(prediction, 'batch channel height width -> (batch height width) channel')
```

https://www.paepper.com/blog/tags/refactoring/
CI pipelines

• Take advantage of automation
  • Automatic checks for code format / PEP 8 etc
  • Automatic unit tests
    • Especially metrics / losses / data loading / data transformation
  • Automatic docker image build
  • Automatic deployment of demo model / visualization prototype
• Bugs caught early are the best -> save time
On reproducibility

• Track your data with your code
  • Which experiment ran with what data?
  • Comparability between experiments
• We use data version control (dvc)
  • Tracking the data
  • Generating pipelines for training and evaluation
• Track your metrics and tensorboard logs
• Have an easy automated pipeline for comparing with the previous model
Implementation stage - summary

- Push for high code quality
  - Touch code -> refactor
  - Linting + typing + tests
- Take advantage of automation: CI
- Use a profiler to avoid easy bottlenecks
- Achieve reproducibility and track your data
Stage: Training + Evaluation
Training & Evaluation
Overview: Training & Evaluation stage

- Multi GPU Training for speedup
- Dataset reduction techniques
- Single metric as guidance
PyTorch Data Parallelization

Training Data
1632 Images

GPU 0
Batch 0  Batch 1  Batch 2  ...  Batch 101

GPU 0
Batch 0  Batch 3  Batch 6  ...  Batch 99

GPU 1
Batch 1  Batch 4  Batch 7  ...  Batch 100

GPU 2
Batch 2  Batch 5  Batch 8  ...  Batch 101

Speedup 3x?!
PyTorch Data Parallelization

Training Data
1632 Images

GPU 0
Batch 0  Batch 1  Batch 2  ...  Batch 101

GPU 0
Batch 0  Batch 3  Batch 6  ...  Batch 99

GPU 1
Batch 1  Batch 4  Batch 7  ...  Batch 100

GPU 2
Batch 2  Batch 5  Batch 8  ...  Batch 101

Training Time $t$
PyTorch Data Parallelization

Batch 0

GPU 0
Master

Batch 0

Send data
and model weights

GPU 0
Master

Batch 0

Process batch on
each GPU
independently

GPU 1
Worker

GPU 2
Worker

Batch 1

Batch 2

GPU 0
Master

Gather outputs
and compute loss on master

GPU 1
Worker

GPU 2
Worker

Batch 1

Batch 2

GPU 0
Master

Scatter loss and compute
gradients on each GPU

GPU 1
Worker

GPU 2
Worker

Batch 1

Batch 2

GPU 0
Master

Gather gradients and update
model on master

GPU 1
Worker

GPU 2
Worker
Pytorch Distributed Data Parallelization

1. Load and process batch on each GPU separately
   - GPU 0
   - Batch 0
   - GPU 1
   - Batch 1
   - GPU 2
   - Batch 2

2. Compute loss and gradient separately on each GPU
   - GPU 0
   - Batch 0
   - GPU 1
   - Batch 1
   - GPU 2
   - Batch 2

3. Gradient all-reduce between all processes
   - GPU 0
   - GPU 1
   - GPU 2
   - Batch 0
   - Batch 1
   - Batch 2

4. Same model update on each GPU!
You need to use a distributed sampler which takes care of loading different data on each GPU.

Reference:
torch.utils.data.distributed.DistributedSampler

Goyal et al. 2018:
Dataset reduction techniques

- Shrink your dataset to a representative smaller set
  - Not easy to define representative
  - Risk of losing out on information
  - Easy to do
- Create a toy dataset
  - Much more work
  - Highly customizable via parameters
  - Not sure how it translates to real data
  - Very good for idea prototyping
    - e.g. label noise experiments
Training + evaluation stage – summary

- Take advantage of multi GPU training
- Dataset reduction techniques for faster training
  - Small representative subset
  - Toy dataset
- Single metric to select if your idea is a winner
Disappointment

WHEN YOU HAVE A FANTASTIC IDEA

BUT IT DOESN'T IMPROVE OVER THE BASELINE
Disappointment

- ML: lots of iteration
- Ideas
- Learn from it:
  - Why?
  - Your data / problem?
  - New ideas?
- Write a diary / Confluence page
- Quality + Speed -> Next idea
Summary

• Use data-driven idea generation
• Have a single evaluation metric to target
• Automate all the things (especially tests / CI pipelines)
• Track your data with your code by using dvc
• Strive for high code quality
  • Comments as code
  • Use einops for dimensional readability
• Use Distributed Data Parallel to train on multiple GPUs
• Take advantage of learning opportunities from experiments
• Iterate quickly + have fun
Looking forward to your questions!

Get in touch

@ marc@mindpeak.ai
@mpaepper
Marc Päpper