Concept to Clinic Documentation

Release

DrivenData

Jan 11, 2018

Specification

CHAPTER 1

ALCF Concept to Clinic Design Document

Introduction

User story: how to think about this software

Bob is 55 years old. He goes to his family doctor for a routine checkup. The family doctor notes that Bob fits the right diagnostic profile for getting a lung cancer early screening scan. Bob's doctor schedules him for a low dose computed tomography (LDCT) scan at a teaching hospital down the street. Bob goes to this appointment a few days later where a technician helps him into the computed tomography (CT) machine and then captures imagery of his chest cavity.

Dr. Smith is a chief resident in diagnostic radiology at the hospital where Bob's scans were taken, and works in a lab that conducts experimental research using new technologies and treatments. She is an experienced and competent professional with years of experience making diagnoses using standard Picture Archiving and Communication System (PACS) software.

Still, like most radiologists she acknowledges that there is an art to detection and diagnosis and that if several other radiologists examined the same CT scans they might all reach slightly different conclusions.

She also knows that computer aided detection (CADe)/computer aided diagnosis (CADx) technology has been advancing rapidly and she's open to using an easy-to-use tool that will help her do her job more efficiently.

Dr. Smith has just pulled up Bob's imagery in order to review and interpret the imagery that the CT scanner generated. In front of Dr. Smith are two computers:

- To the left is her ordinary hospital workstation running a standard PACS software package. This is a system she uses every day, and she is extremely practiced at using this software; ever since starting residency, she has used some similar version of PACS software nearly every day. The PACS software is a highly specialized and fully featured software suite geared towards making it easy for clinicians to explore imagery. For instance, the user can pan around the image, zoom in and out of specific areas, scroll up and down the Z axis through image layers, change image brightness and contrast, and measure lengths between points.
- To the right is a standalone laptop. The software from this project is running locally on a fresh Linux install, and a browser window is opened on the homepage of the web-based interface service. This computer is trusted and only has intranet access to the DICOM imagery server.

The key question is this: armed with recent developments in artificial intelligence and machine learning, how might we help Dr. Smith catch cancer early and give her patients a better chance of having more time with their families?

Here are some issues in the current PACS diagnosis workflow that Dr. Smith feels could be improved upon:

- Radiologists don't want to miss any spots with potentially concerning tissue.The current software is great at presenting imagery but doesn't adequately help detect nodules.
- Radiologists worry a lot about false positives because exposing patients to additional CT scan radiation or unnecessary surgical procedures carry risks of their own and can end up being more dangerous than the original nodules. The current software doesn't adequately help minimize false positives.
- The more accurately a nodule is measured, the more accurate the diagnosis. Right now, most radiologists just eyeball which slice looks like it has the widest diameter. Then they use the measuring tool to take one horizontal measurement in the direction that looks widest. This method is imprecise and results in a loss of useful information when a complex 3D shape having volume and surface features is reduced to a single estimated width. Additionally, not knowing the precise nodule boundaries also means it's difficult to quantify whether and how a nodule has changed over time. The absence or presence and characteristics of such change over time is a major cancer indicator for radiologists. So the current software doesn't adequately help measure nodule boundaries.
- There is an existing set of best practices for what should be included in a lung cancer screening report (i.e. RSNA Standard Template For Lung Cancer Screening, http://www.radreport.org/template/0000268). Most radiologists haven't adopted this standard yet, partially because it's a lot of work and partially because the tools aren't helpful in filling this out. The current software doesn't help radiologists organize their thoughts or automatically annotate the records in ways that conform with best practices, so it doesn't adequately help streamline reporting.

Goals and scope

This application can be thought of as a prototype in the research and development (R&D) phase that is closely focused on building and presenting features that are **novel** and **build or improve upon existing tools** for lung cancer CADe/CADx. To borrow terms from "The Lean Startup," this application will serve as a minimum viable product (MVP) from which we can quickly gather feedback from radiologist users.

> A minimum viable product is the "version of a new product which allows a team to collect the maximum amount of validated learning about customers with the least effort" — "The Lean Startup" by Eric Ries, 2014

In this MVP, we are focusing on three core predictive **task areas**:

- `IDENTIFICATION`: Analyze CT scan images to detect and pinpoint the location of concerning nodules from background tissue.
- `CLASSIFICATION`: Use what we've seen from nodules in the past to predict whether the identified nodules are cancerous or benign.
- `SEGMENTATION`: Find the boundaries of nodules and create automatic measures to help radiologists refine and build out the computer-aided diagnosis.

We are also focusing on three ancillary but useful task areas:

- `IMPORTING`: Going from DICOM formatted images on disk (or, ideally, available through a server process that mimics the DICOM servers in hospitals) and (1) loading these DICOM images into a data format that is useful for analysis, e.g. Numpy array, and (2) persisting these imported data structures to a central location for access by various services. For the purposes of this project we will simply use the local filesystem to which all processes will have access, but in the future this file abstraction could easily be extended to cover cloud storage or network files.

- `REPORTING`: Present what has been found in a concise and clinically useful format that corresponds to identified best practices (such as the RSNA standard template) and evidence based risk diagnosis tools (such as the Lung-RADS™ assessment methodology).
- `EXPORTING`: Allow results to be exported in a variety of sensible formats which ideally fit existing standards, align with real world usage, or at least would be easy to repurpose in other applications.

Non Goals

For this proof of concept, we only want to spend time on value-added features that advance the core research and development efforts of the project. In order to focus work solely on things that are important, it helps to explicitly outline what's not in scope for this project.

Here's one important non-goal: we are **not** trying to recreate the features of a production PACS system. Millions of dollars and many engineer years have been spent creating commercial off-the-shelf (COTS) software that lets radiologists explore imagery in a fast, native application; PACS is a solved problem that we're not tackling here.

The goal is also certainly **not** to create a HIPAA-compliant, cloud-ready SaaS app completely with enterprise-grade security, load balancing, SSL certificates and so forth. In other words, this project's scope entails running on a local, trusted computer in a research clinic authorized to use non-standard applications. That means the app does not have to be ready to run in a public venue or with complicated security and privacy controls; the reasoning is not that this tool could never be put in production in such a system, it's just that productionizing an app for privacy, security, user workflow, and so forth are all (1) largely solved problems, (2) highly specific to the application setting, and (3) not relevant to this project's R&D focus.

So while a limited set of features common to PACS systems or SaaS applications may incidentally need to be built or incorporated if it improves our ability to demonstrate core capabilities, the following features are examples of things that aren't relevant or do not fit within the scope of this R&D effort unless user feedback indicates otherwise:

- User authentication, including password resets and transactional emails
- Role based access control or file access controls
- Service discovery and sophisticated container orchestration
- Most types of production-centric optimization like page caching
- Native or GPU accelerated exploration of CT images

Design goals and architecture

It's important to remember that the overarching goals is for this project is that the models are (1) useful, and (2) able to be adapted and adopted for use in hospitals with as little modification as possible. By adopting a hypothesis-driven development model and incorporating iterative product releases and validated learning in accordance with the Lean Startup principles, we can achieve our development goals more efficiently and with reduced risk of failure.

Let's imagine that somebody is creating a plugin that recreates our functionality on an existing COTS PACS system. Our goal is not to replace PACS, just to build on it, so this would be a great outcome. Such an implementation of this prototype would not use this software unmodified — indeed, it would use the existing PACS frontend which is unlikely to be web-based. So even though our interface will demonstrate what is possible, it needs to be **loosely coupled** to the trained models which are doing the core predictive tasks.

In order to preserve a bright line separation of concerns, we'll break the overall application down into several constituent pieces. Here's an extremely high level view of the component hierarchy:

- The job of the **prediction service** (via the **trained models** it wraps) is to implement these core predictive tasks and expose them for use, respectively. The models themselves shouldn't need to know about the prediction service which in turn should not need to know anything about the interface application.

- The job of the **interface backend (API)** is to ferry data back and forth between the client browser and model service, handle web requests, take care of computationally intensive transformations not appropriate for frontend Javascript, and persist user-entered data to a data store. It should not need to know much about the interface frontend, but it's main job is to relay data for frontend manipulation so it's acceptable for this part to less abstractly generalizable than the prediction service.
- The job of the **interface frontend (UI)** is to demonstrate as much value as possible by exposing functionality that the models make possible in an intuitive and attractive format. Each of these jobs to be done is described in greater detail below.

Here's a visual representation of this architecture:

Jobs to be done

Imagery selection

Task areas: `IMPORTING`

Imagery - Prediction service

RESERVED

Imagery - Interface API

1. List available DICOM images on local filesystem.
2. Display such details as may be available (e.g. metadata) for any particular DICOM image set on local filesystem.
3. Return a preview of the DICOM image on local filesystem as an HTML-displayable image.
4. Allow a specific image to be selected and create a new "Case" to work on for the remainder of the steps.
5. Items 1-3, but connecting to a DICOM server running in a separate container.

Imagery - Interface frontend

1. List available DICOM images on local filesystem.
2. Display such details as may be available (e.g. metadata) for any available DICOM image.
3. Display a preview of an available DICOM image.
4. Allow user to select a given image and start a new "Case." This image will be used in the identification step next.

Detect and select

Task areas: `IDENTIFICATION`, `CLASSIFICATION`

Detect - Prediction service

1. Given a pointer to a DICOM image on the local filesystem, generate a response with an array of predicted nodules ("candidates"). Each candidate in the response should indicate the centroid location tuple (X # voxels from left, Y # voxels from top, Z as slice number), and P(concerning) that each model outputs for each predicted location. Parameters that the request may include are a threshold for minimum P(concerning) to include a candidate in the response, and (2) a specific list of which models to use.
2. If multiple models are used to make predictions, combine and deconflict the predictions in a useful way. For example, if each of the models comes up with a slightly different set of candidates (which is to be expected), try to figure out if candidates that are close together are duplicates or bona fide separate tissue locations. If this approach has some tunable sensitivity (e.g. tweaking a distance threshold at which candidate sites are assumed to be the same nodule) then look for such a parameter in the model request.
3. Items 1 and 2, but accept an appropriate URI as the pointer to a DICOM image and implement fetching so that a DICOM server running in a separate container may be used in place of the local filesystem.

Detect - Interface API

1. Act as the data broker for the image being worked on (selected in our current session), requesting predictions from the Prediction service along with filtering or other passed parameters and relay them back to the frontend.
2. Provide an endpoint that can receive a payload of all nodule locations, (1) nodule centroid location tuple (X # voxels from left, Y # voxels from top, Z as slice number), (2) whether the nodule was predicted or manually added, and (3) whether the nodule was marked for further analysis or not (those manually identified by the user are presumably concerning). Persist this location list to the database attached to the current Case.

Detect - Interface frontend

1. List all available predicted candidate sites. Allow each to be selected to view details.
2. When a candidate is selected, show a detail view in an image viewer control. By default, show the slice containing the predicted centroid and a marker clearly indicating where the predicted centroid occurs.
3. Allow the user to zoom, pan, and navigate through slices in the image viewer.
4. Allow the user to window grayscale levels.
5. Allow the user to toggle the candidate centroid marker's visibility.
6. Allow the user to mark any predicted candidate for further analysis.
7. Allow the user to navigate through the imagery freestyle and mark other locations which the models missed.
8. When the user has finished marking candidates, send all of this labeled data to the backend in the format specified above.

Annotate

Task areas: `CLASSIFICATION`, `REPORTING`

Annotate - Prediction service

RESERVED

Annotate - Interface API

1. List nodules identified as part of the current Case for detail view.
2. Receive and persist to the database a radiologist-supplied label for how concerning each nodule actually is.
3. Receive and persist to the database the other fields expected for each case in the RSNA Radiology Reporting Template for CT Lung Cancer Screening (http://www.radreport.org/template/0000268) — for example, solid/semi-solid/ground glass consistency, left lung/right lung, and any freehand notes they entered in the text box.
4. Receive and persist to the database the other fields expected for each nodule in the RSNA Radiology Reporting Template for CT Lung Cancer Screening. Namely: a choice between left lung/right lung; a choice between solid, semi-solid, and ground glass; a choice between unchanged/increased/decreased/new; and a field for text entry of notes. These should all be persisted to the database.

Annotate - Interface frontend

1. Provide UI controls for adding sending general RSNA template notes for the case.
2. Provide UI controls for selecting a nodule for detail view for a single nodule and display that nodule in a corresponding image viewer.
3. In the detail view for a single nodule, provide a UI control for specific a user-supplied ground truth label for P(concerning). Also show an indication of what the predicted P(concerning) was for each nodule.
4. In the detail view, provide UI controls for the nodule specific RSNA template data outlined above.
5. In the detail view, enrich the image viewer with pan, zoom, and slide traversal for taking a closer look at and around nodules.

Segment

Task areas: `SEGMENTATION`

Segment - Prediction service

1. Given a nodule centroid location tuple (X # voxels from left, Y # voxels from top, Z as slice number), return a 3D boolean mask with true values for voxels associated with that nodule and false values for other tissue or voids.

Segment - Interface API

1. For any given slice of any given nodule, transform the true values in the binary mask from the Prediction service into a series of vertices defining an irregular polygon which can be displayed on the image. In the transformation to vertices, take a precision parameter which causes the transformation to use more or fewer vertices to make the polygon fit the binary mask more precisely.
2. Given a stack of such irregular polygons, calculate a number of summary statistics of interest, to include: a new estimated centroid location tuple (X # voxels from left, Y # voxels from top, Z as slice number); a volume in cubic millimeters; a longest axial diameter.
3. Provide an endpoint that allows the current vertices to be persisted to the database attached to the current Case and Nodule.
4. Provide an endpoint that allows a 3D visualization of each nodule to be provided in a static image format (.png).

5. Provide an endpoint that allows a 3D visualization of each nodule to be viewed in an interactive format.

Segment - Interface frontend

1. Display an image view of a given slice of the selected nodule. Allow navigation between slices.
2. Overlay the current vertices on top of the nodule image to see the predicted boundary.
3. Allow the user to move existing vertices.
4. Allow the user to delete existing vertices.
5. Allow the user to add a new vertex splitting a line segment connecting two vertices.
6. Allow the user to extend the nodule boundary along the Z axis by extending the currently terminal polygon up or down to the adjacent slice which still has unhighlighted nodule.
7. Indicate to the user the Z axis extent of slices.

Results

Task areas: `REPORTING`, `EXPORTING`

Results - Prediction service

RESERVED

Results - Interface API

1. Summarize all of the data from the Case, including the general notes, and details for each nodule into a single JSON report.
2. Allow export of the whole report as an Extensible Markup Language (.xml) file.
3. Allow export of the whole report as a Portable Document Format (.pdf) file.
4. Allow export of the whole report as a Microsoft Word document (.docx) file.
5. Allow export of 3D nodule reconstructions in STL format.
6. Allow export of 3D nodule reconstructions in SHP format.

Results - Interface frontend

1. Display all of the results from the case matching the RSNA format in a clean, organized, and hyperlinked report.
2. Display the results
3. Provide links to all of the export options in sensible locations throughout the report.

CHAPTER 2

Getting Started

Contributing is as easy as submitting a PR to the GitHub repository! If you're already a GitHub pro, you can get started just with the repo in the way you would with any open source project. If you're not sure how to get started:

- Sign up for the challenge
- Find an open issue you're interested in working on
- Get your local environment running with the developer documentation
- Submit your pull request to the GitHub repo

If you need any help, jump into the forums and ask away!

Community Guidelines

This project is about building a community that is working together towards a shared goal. We've got a code of conduct that we expect participants to respect, but it really boils down to:

- **Keep the goal in mind.** We're focused on the audacious goal of providing advanced algorithms for clinicians. Points and prizes are great, but they're just a tool for keeping the project fun and interesting.
- **Leave your ego at the door.** We're all aiming to achieve something bigger by working together.
- **Practice empathy.** Not everyone has the same skills or background, assume participants have the best intentions.
- **Encourage other participants.** The more people who can contribute the better for everyone. Help each other and recognize good work when you see it.
- **Respect the process.** This project involves tricky technical tradeoffs. Sometimes the administrators may make a decision that you don't agree with. Contributors should assume good faith in all discussions and be willing to move on once the project moves past a certain decision.

And here are some concrete things that you can do:

- Answer questions on the forum

- Keep the developer documentation up to date as you change code
- See a good PR, add a +:heart: emoji! Challenge administrators will take notice.
- Let others know when you're working on an issue—invite them to join your fork and solve it together!

Making the Project Better

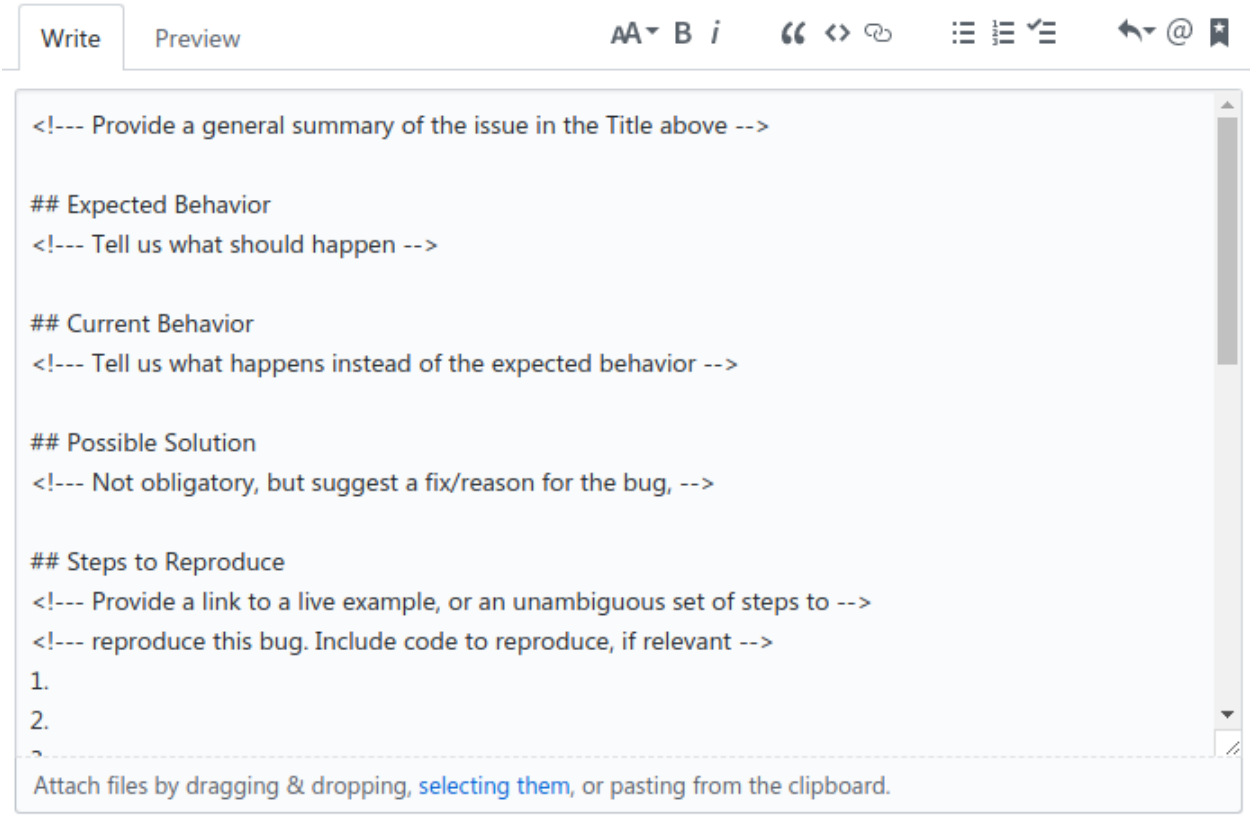
As with any open source project, the specification of how we reach our goals, the features that we implement, and how the software works will continue to evolve. We welcome your input in refining this vision.

We didn't think of everything beforehand, so contribute your ideas to the cause. We won't be able to implement every one, but filing good issues that become marked `official` after review will earn you points like a code contribution.

- Search the issues to make sure it doesn't already exist
- Use the *issue template* to ensure all the relevant details are included
- Wait for review and acceptance by the project maintainers

Filing an issue

Creating issues is encouraged! When you create a new issue, you'll see a pre-populated template — please fill this out as carefully and thoroughly as possible. Vague or insufficient issues will be closed quickly to help keep the project on track.



Upon review, project admins may tweak the issue or ask for more clarification, or may attach the `official` label to demonstrate both agreement and that supplying a PR that closes the issue will result in an award of points.

Cloning the repository and Git LFS

The problem with large files in Git (such as test images for this project) is that the entire history of these files is transferred to the client. This means that every time a big file is edited, its whole history needs to be downloaded by the contributors. Git Large File Storage (LFS) circumvents that problem by creating pointer files and lazily downloading only the needed version of big files while checkout rather than downloading the whole history of it while cloning or fetching (Githubs explanation can be found here).

If you don't want to use Git LFS, you can clone the repository in a normal way and contribute to it, but you won't be able to use e.g. test images shared by other contributors tracked by Git LFS.

Installing LFS

If you want to use Git LFS, please install Git LFS from the official website. To initialize LFS, you might need to run

```
$ git lfs install
```

Cloning Using LFS

Change to the project directory and run

```
$ git lfs pull
```

This should indicate that files are being downloaded.

Track files using LFS

To manage files using LFS, run

```
$ git lfs track myfile.psd
```

or to manage all files of a certain type, run

```
$ git lfs track "*.psd"
```

You can also adapt .gitattributes manually. Next, track said file

```
$ git add .gitattributes
```

Now you just have to commit and push the file(s) as you normally would

```
$ git add myfile.psd
$ git commit -m "Add big file"
$ git push origin master
```

Removing big files from Git history

If you already tracked big files using Git but moved them to LFS, you might want to remove them from the Git history. The preferred approach would be to use the BFG Repo-Cleaner since it is simpler and faster than Git's built-in command `filter-branch`. Please refer to this great and detailed Github article about removing data from the repository history.

Opening a Pull Request

It's beyond the scope of this document to explain pull requests in detail, but Github has some great resources to help people new to git and Github.

Once you're familiar with the concept, the steps are pretty simple:

- Fork the `concept-to-clinic/concept-to-clinic` project
- Clone your own fork to your local development computer
- Check out a branch in your local repo, named something descriptive like `issue-8-sphinx-autodoc`
- Make your changes
- If necessary, rebase the changes onto the project's `master` branch
- Open the PR, and fill out the prepopulated PR template to the best of your ability

CHAPTER 3

Project Structure

Here is an annotated version of the project structure.

CHAPTER 4

Local development with Docker

Docker Compose is the only officially supported development configuration. We're doing this to stay focused on the project goals rather than triage lots of "not working on my machine" issues. If you wish to use virtualenvs instead, please ensure that you run all of the tests with Docker Compose before opening any Pull Requests.

The steps below will get you up and running with a local development environment.

Note: All of these commands assume you are in the root of your generated project.

Prerequisites

Install Docker

You'll need a recent version of Docker. If you don't already have it installed, follow the instructions for your OS:

- On Mac OS X, you'll need `Docker for Mac`
- On Windows, you'll need `Docker for Windows` for 64bit Windows 10 Pro, Enterprise and Education or `Docker Toolbox` for other versions
- On Linux, you'll need `docker-engine`

Docker for Mac and Windows include Docker Compose, so most Mac and Windows users do not need to install it separately. For Linux users, follow the instructions for Linux installation in the Docker documentation

Build the Stack

This can take a while, especially the first time you run this particular command on your development system:

```
$ docker-compose -f local.yml build
```

Boot the System

This brings up the `interface app` (running Django), `prediction app` (running Flask), and PostgreSQL.

The first time it is run it might take a while to get started, but subsequent runs will occur more quickly.

Open a terminal at the project root and run the following for local development:

```
$ docker-compose -f local.yml up
```

You can also set the environment variable `COMPOSE_FILE` pointing to `local.yml` like this:

```
$ export COMPOSE_FILE=local.yml
```

And then run:

```
$ docker-compose up
```

You should now be able to open a browser and interact with the services:

- `interface` frontend at http://localhost:8080
- `interface` API at http://localhost:8000/api/
- `prediction` service at http://localhost:8001
- `documentation` service at http://localhost:8002

Run tests

For the vue frontend interface, you can run the unit test with:

```
$ docker-compose -f local.yml run vue_unit_test
```

And the end to end test with:

```
$ docker-compose -f local.yml run vue_e2e_test
```

Code changes

Because `local.yml` has the volume mappings `./interface/:/app` and `./prediction/:/app` (the project dirs containing all of the code into the container's source code directory), changes made to files during development should be reflected in the running `interface` or `prediction` container as soon as the dev server process restarts.

Create JSON data

First use the testing fixture factories to create some data in your local database:

```
docker-compose -f local.yml run interface python manage.py shell_plus

Python 3.6.2 (default, Sep  8 2017, 06:15:13)
[GCC 4.9.2] on linux
Type "help", "copyright", "credits" or "license" for more information.
```

```
(InteractiveConsole)
>>> from backend.cases.factories import *
>>> case = CaseFactory()
>>> candidates = CandidateFactory.create_batch(5, case=case)
>>> nodule0 = NoduleFactory(candidate=candidates[0])
>>> nodule1 = NoduleFactory(candidate=candidates[1])
>>> nodule4 = NoduleFactory(candidate=candidates[4])
[exit]
```

Then run the API: `docker-compose -f local.yml up interface`

And if you go to `http://[IP of your docker, could be localhost]:8000/api/` you can play with the data you just created:

For instance, here's the list of nodules we just created:


```
[
    {
        "url": "http://localhost:8000/api/nodules/1/",
        "centroid": {
            "id": 6,
            "x": 324,
            "y": 285,
            "z": 59,
            "series": 6
        },
        "created": "2017-10-10T22:16:35.620318Z",
        "lung_orientation": 0,
        "case": "http://localhost:8000/api/cases/2/",
        "candidate": "http://localhost:8000/api/candidates/1/"
    },
    {
        "url": "http://localhost:8000/api/nodules/2/",
        "centroid": {
            "id": 7,
            "x": 49,
            "y": 227,
            "z": 31,
            "series": 7
        },
        "created": "2017-10-10T22:16:44.436150Z",
        "lung_orientation": 0,
        "case": "http://localhost:8000/api/cases/3/",
        "candidate": "http://localhost:8000/api/candidates/2/"
    },
    {
        "url": "http://localhost:8000/api/nodules/3/",
        "centroid": {
            "id": 8,
            "x": 242,
            "y": 339,
            "z": 22,
            "series": 8
        },
        "created": "2017-10-10T22:16:50.403528Z",
        "lung_orientation": 0,
        "case": "http://localhost:8000/api/cases/4/",
        "candidate": "http://localhost:8000/api/candidates/5/"
    }
```

```
]
```

Running the tests

The `tests/test_docker.sh` shows you how to run the tests for both the prediciton and interface applications.

For example, if you are actively developing the prediction app, you could run the following to execute the tests:

```
$ docker-compose -f local.yml run prediction pytest
```

Other notes

Pre-commit hooks

Git pre-commit hooks are useful tools that run commands before you commit. To enable easier maintainance of style guide and running tests, we implement pre-commit git hook commands that will run every time before you create a commit.

To set it up on your local development repository:

```
$ cd concept-to-clinic
```

Copy the `.githook-pre-commit` file into the `.git/hooks/` folder as `pre-commit` with `-f` flag so that you overwrite any existing `pre-commit` file:

```
$ cp -f .githook-pre-commit .git/hooks/pre-commit
```

Install `flake8` locally for the styling scripts:

```
$ pip install flake8 pep8
```

You are set! The next time you want to commit, styling and testing pre-commit commands will be executed first and if they fail, your commit will not be committed.

If you want to by-pass the pre-commit check, use `--no-verify` flag in your commit:

```
$ git commit --no-verify -m "Ignore pre-commit rules"
```

To check if the created pre-commit hook is working without having to do a commit, you can run from the root of the project:

```
$ .git/hooks/pre-commit
```

Detached Mode

If you want to run the stack in detached mode (in the background without console output), use the `-d` argument:

```
$ docker-compose -f local.yml up -d
```

CHAPTER 5

Challenge Rules

View the Challenge Rules

CHAPTER 6

Code of Conduct

View the Code of Conduct

CHAPTER 7

Contributor License Agreement

View the Contributor License Agreement

CHAPTER 8

Algorithms

Aidence

Summary

Author: Tim Salimans, Mark-Jan Harte, Gerben van Veenendaal **Repository:** https://bitbucket.org/aidence/kaggle-data-science-bowl-2017/src/38c4f2f67294?at=master The 3rd place at the Data Science Bowl 2017 on the private leaderboard.

License

Prerequisites

Dependency	Name	Version
Language	Python	3.4
ML engine		
ML backend	Tensorflow	1.1
OS		
Processor	CPU	yes
	GPU	Nvidia K80
GPU driver	CUDA	8.0
	cuDNN	6.0

Dependency packages:

```
tensorflow==1.1
opencv>=3.1
scipy==0.17.0
numpy==1.13
```

```
scikit-learn==0.19.0
pydicom==0.9.9
SimpleITK==1.0.1
pandas==0.20.3
pycuda==2017.1.1
```

Algorithm design

Preprocessing

Resampling to the isotropic resolutions of and for the final model.

Nodule detection

Fully convolutional Resnet has been employed in order to detect for each pixel whether it is contained in the center of a nodule. It was trained it over the LIDC/IDRI dataset. Two of those models has been trained: one for normal sized nodules and one for masses. The masses on the train data of Kaggle have been annotated and the mass network has been trained on both masses from LIDC/IDRI as well as masses from Kaggle. Takes the logit output of that network for the whole volume and thresholds it to determine candidates. It also masks out nodules outside the lung.

Prediction of cancer probability

Takes the candidates and trains some attributes of the LIDC dataset (malignancy, etc.) and trains the cancer label for the Kaggle scans in a multi-task model.

Trained model

Source: From the issue description followed that the trained model is already requested. There are two `.pkl` models in localization and localization-large

Usage instructions: README

Model Performance

Training- / prediction time

Test system:

Component	Spec	Count
CPU		
GPU	Nvidia K80	4 for everything but the final model 8 for the final model
RAM		

Training time:

> It takes about 3-5 days to run everything (infer+train) on a decent machine with 8 GPUs. **Prediction time:**unknown, but must be less than 14 min per CT, since it processes the 506 CTs for the 5 days

Model Evaluation

Dataset: Data Science Bowl evaluation dataset

Metric	Score
Log Loss	0.40127

Use cases

When to use this algorithm

- The annotation for the mass and nodules over the Kaggle dataset, provided by the aidence team, can be used in futher fine-tunings / retrainings.

When to avoid this algorithm

- even with GPU support the approach of per voxel examination may consume a huge amount of time. The authors have used 8 GPUs Nvidia K80 which is

Adaptation into Concept To Clinic

Porting to Python 3.5+

The solution is already compatible with Python 3.5+

Porting to run on CPU and GPU

The approach consists of two deep 3D residual networks for classification (which runs through each `voxel` from a CT scan). It'll require a huge amount of time to even predict with this pipeline using CPU only.

Improvements on the code base

The code itself looks good to me.

Adapting the model

Comments

The major benefits for the concept-to-clinic from the aidence approach will be to include provided mass- ans nodule-annotations over the Kaggle dataset into the overall dataset for further retraining other models on it.

References

Aidence algorithmREADME Mass-annotations over Kaggle data. Nodule-annotations over Kaggle data.

Alex |Andre |Gilberto |Shize algorithm

Summary

The approach consists of 3D CNN data model which slide through the z coordinate of a CT volume, followed xgboost and extraTree models trained on different subsets of extracted features. by was custom built to reflect how radiologists review lung CT scans to diagnose cancer risk.

> A sliding 3D data model was custom built to reflect how radiologists review lung CT scans to diagnose cancer risk. As part of this data model - which allows for any nodule to be analyzed multiple times - a neural network nodule identifier has been implemented and trained using the Luna CT dataset. Non-traditional, unsegmented (i.e. full CT scans) were used for training, in order to ensure no nodules, in particular, those on the lung perimeter are missed.

Source

Author: Alexander Ryzhkov, Gilberto Titericz Junior, Andre, Shize Su **Repository:** https://github.com/astoc/kaggle_dsb2017 The approach scored the 8th place at the Data Science Bowl 2017.

License

Prerequisites

Some of the cells' values were restored from the AWSs' setups and CUDA compatibility.

Dependency packages: Neither the repository nor the authors specified exact versions of the Python packages:

Andre	Shize
Keras 1.2.2	numpy
Theano	pandas
spyder	xgboost
opencv	scikit-learn
pydicom	
scipy	
scikit-image	
SimpleITK	
numpy	
pandas	

Algorithm design

Preprocessing

1. Resampling all patient CT scans to a relatively rough resolution of 2x2x2mm.
2. CT voxels' values standardisation to Hounsfield scale.
3. Lungs segmentation.

Nodule detection

> Train a nodule identifier on a slicing architecture using Luna dataset and intermediate files created (3 options provided).

The slicing architecture itself is made of UNets. One of the aforementioned options is also a good data augmentation method:

> [..] Special mosaic-type aggregation of training of the nodule identifier has been deployed, as illustrated below.

Prediction of cancer probability

> The most important feature is the existence of nodule(s), followed by their size, location and their other characteristics. For instance, a very significant number of patients for which no nodule has been found, proved to be no cancer cases. [..] Key features include existence/size of the largest nodule, and its vertical location, existence of emphysema, volume of all nodules, and their diversity.

The authors also have mentioned that the code location of nodules versus the segmented lungs centre of gravity as a feature provide higher significance in comparison with convenient upper/lower parts of lungs feature.

> As outlined, our combined approach uses the neural network as a feature generator and then applying xgboost and extraTree models on the extracted features to generate predictions and submissions. To make the model performance more stable, we also run some of the models with multiple random seeds (e.g., for xgb, use 50 random runs; for extraTree, use 10 random runs) and take the average. Our final winning submission (private LB0.430) is a linear combination of a couple of xgb models and extraTree models.

Trained model

Source: nodule_identifiers

Usage instructions: Shize algorithm, Andre algorithm

Model Performance

Training- / prediction time

Test system:

Component	Spec	Count
CPU	C3.8 Intel Xeon	
GPU	P2 NVIDIA K80 12GB	>1
RAM		

Training time: days on AWS

> Training some of the nodule models took days using high end 12GB GPUs.

Prediction time: unknown, but must be less than 14 min per CT, since it processes the 506 CTs for the 5 days

Model Evaluation

Dataset: Data Science Bowl evaluation dataset

Metric	Score
Log Loss	0.43019

Use cases

When to use this algorithm

- The nodules detection system seems to be a good contribute to a concept-to-clinic's ensemble, by the reason listed in comments.

When to avoid this algorithm

- The nodules detection method provided by the authors requires inconvenient rough CT's spacing (`2x2x2mm`) which may conflict with other pipelines, if the high order interpolation polynomials will be employed then the additional spacing transaction may considerably affect on a computation time.
- The training from scratch, as it was mentioned by the authors, for only one of the sliding architectures may take days even over AWS P2 equipped by NVIDIA K80 12GB GPUs.

Adaptation into Concept To Clinic

Porting to Python 3.5+

The Andre part had been already written in python 3.5. However Shize used the python 2. The main difficulties seems to be the lack of specified versions for the packages employed by Shize. Nonetheless, Shize's part consists merely of ensembling already extracted features via xgboost and extraTree models, and GPUs are not required.

Porting to run on CPU and GPU

The noodles detector written on `Keras` with `Theano` as the backend, thus it shall run on CPU out of the box.

Improvements on the code base

Adapting the model

Worth noting that simpler model consisted only of a single xgb has performed similarly (0.434 on private LB). Thus it will be better to drop away the cumbersome combination of different xgb and extraTree models , and some of them were using averaged prediction from 50 or 10 random runs (i.e., using 50 (or 10) different random seeds)

Comments

The whole pipeline relies on the nodules detector, and at the same time the approach has reached 8th place on the DSB17 private LB, it's worth to admire that method then and consider it into account. Moreover, the authors stated that they didn't use the information relative nodule malignancy as they've incorrectly assumed it's unavailable, therefore training the model from scratch or fine tune it over the data within malignancy status seems to be beneficial.

References

Repository: https://github.com/astoc/kaggle_dsb2017Technical Report: https://github.com/astoc/kaggle_dsb2017/blob/master/Solution_
dataset: https://luna16.grand-challenge.org/home/

Julian de Wit Algorithm

Summary

The algorithm stands out with the idea to pre-detect strange tissue, estimate the malignancy of the nodules using a C3D network and predict the cancer probability using XGBoost and some other features. It is combined with the algorithm by Daniel Hammack at the prediction level.

Source

Author: Julian de Wit**Repository:** https://github.com/juliandewit/kaggle_ndsb20172nd place at the Data Science Bowl 2017 together with the algorithm by Hammack

License

MIT

Prerequisites

Dependency	Name	Version
Language	Python	3.5
ML engine	Keras	2.0.8
ML backend	Tensorflow	1.3.0
OS	Windows/Ubuntu	64 Bit
Processor	CPU	yes
	GPU	yes
GPU driver	CUDA	8.0
	cuDNN	6.0

Dependency packages:

```
beautifulsoup4==4.6.0
lxml==3.8.0
numpy==1.13.1
pandas==0.20.3
scipy==0.19.1
scikit-learn==0.19.0
scikit-image==0.13.0
tensorflow-gpu==1.3.0
Keras==2.0.8
xgboost==0.6a2
opencv-python==3.3.0.10
pydicom==0.9.9
SimpleITK==1.0.1
```

Algorithm design

Preprocessing

Every scan was rescaled so that every voxel represented a volume of 1x1x1 mm. Next, the pixel intensities were clipped to the minimum and maximum of the interesting Hounsfield Units. Then, they were scaled between 0 and 1. Lastly, the author ensured that all the scans have the same orientation.

Strange tissue detection

The author used a C3D network with an input of 32x32x32 mm which is a receptive field that is 8 times smaller than the one of Hammack. This way it is much lighter and more diverse with respect to the used architecture of Hammack.

Prediction of cancer probability

The author states:

```
In the end I only used 7 features for the gradient booster to train upon.
These were the maximum malignancy nodule and its Z location for all 3 scales and the␣
↪amount of strange tissue.
```

Trained model

Source: https://retinopaty.blob.core.windows.net/ndsb3/trained_models.rar

Usage instructions:

```
[...] The models are placed in the ./models/ directory.
From there the nodule detector step3_predict_nodules.py can be run to detect nodules␣
↪in a 3d grid per patient.
The detected nodules and predicted malignancy are stored per patient in a separate␣
↪directory.
The masses detector is already run through the step2_train_mass_segmenter.py and will␣
↪stored a csv with estimated masses per patient.
```

Model Performance

Training- / prediction time

Unfortunately, neither the blog entry nor the readme mention the system that was used for training and testing.**Test system:**

Component	Spec	Count
CPU		
GPU		
RAM		

Training time: 10 hours per 3D ConvNet**Prediction time:** unknown

Model Evaluation

Dataset: Data Science Bowl 2017

Metric	Score
Log-Loss	0.40117

Use cases

When to use this algorithm

- when we don't want to port the code (as it is already written for Python 3)
- when we don't want to train the models (as they are downloadable)

When to avoid this algorithm

- It's unclear how well the algorithm performs without being ensembled with the solution of Hammack - especially since its respective field is 8 times smaller than the one of Hammack. Also, the author already mentions himself that the network architecture still needs finer tuning.

Adaptation into Concept To Clinic

Porting to Python 3.5+

It's already written to run with Python 3.

Porting to run on CPU and GPU

It is possible to make Tensorflow use the CPU instead of the GPU.

Improvements on the code base

The author states that he did not clean up the complete repository to keep its reproducibility. It might make sense to contact the author to task for further suggestions for the clean up.

Adapting the model

The author suggests to play around with the architecture of the CNNs since he put very few time in that, although the architecture of a neural network is a critical factor of its performance.

Comments

The differences between the two approaches of Hammack and de Wit can be seen in the following table:

References

ReportCode

Deep Breath Algorithm

Summary

This algorithm trains a network for determining if a voxel in a scan is part of a nodule to generate a list of nodule candidates and their centroids, and additional networks to reduce the number of candidates and predict the malignancy of the remaining nodules. Finally, another network is trained to predict the probability of lung cancer.

Source

Authors: Elias Vansteenkiste, Matthias Freiberger, Andreas Verleysen, Iryna Korshunova, Lionel Pigou, Frederic Godin, Jonas Degrave**Repository:** https://github.com/EliasVansteenkiste/dsb3The approach scored 9th place at the Data Science Bowl 2017.

License

MIT license

Prerequisites

Dependency	Name	Version
Language	Python	2.7
ML engine	Lasagne	0.2.dev1
ML backend	Theano	0.9.0b1
OS	Ubuntu	16.04
Processor	CPU	yes
	GPU	yes
GPU driver	CUDA	8
	cuDNN	5.1

Dependency packages:

```
theano==0.9.0b1
lasagne==0.2.dev1
scikit-image==0.12.3
scipy==0.18.1
numpy==1.12.0
scikit-learn==0.18.1
```

Algorithm design

The authors provide a detailed overview of the algorithm in [1].

Preprocessing

As described by the authors:

```
The chest scans are produced by a variety of CT scanners, this causes a difference in␣
↪spacing between voxels of the original scan.
We rescaled and interpolated all CT scans so that each voxel represents a 1x1x1 mm␣
↪cube.
```

Nodule detection

First, a network for segmenting nodules in the input scan was built using the LUNA dataset as training data. The dataset contains the location and diameter of nodules.

```
To train the segmentation network, 64x64x64 patches are cut out of the CT scan and␣
↪fed to the input of the segmentation network.
For each patch, the ground truth is a 32x32x32 mm binary mask. Each voxel in the␣
↪binary mask indicates if the voxel is inside the nodule.
The masks are constructed by using the diameters in the nodule annotations.
```

The Dice coefficient was used as an objective function, as it is well suited to counter the imbalance of bigger and smaller nodules.

```
The downside of using the Dice coefficient is that it defaults to zero if there is no␣
↪nodule inside the ground truth mask.
There must be a nodule in each patch that we feed to the network. To introduce extra␣
↪variation, we apply translation and rotation augmentation.
The translation and rotation parameters are chosen so that a part of the nodule stays␣
↪inside the 32x32x32 cube around the center of the 64x64x64 input patch.
```

The network architecture is based on the U-net architecture, which was adapted to 3D input data. It mainly consists of convolutional layers with 3x3x3 filter kernels without padding, and one max pooling layer. (See [1] for a detailed schematic.)

```
The trained network is used to segment all the CT scans of the patients in the LUNA␣
↪and DSB dataset. 64x64x64 patches are taken out the volume with a stride of␣
↪32x32x32 and the prediction maps are stitched together.
In the resulting tensor, each value represents the predicted probability that the␣
↪voxel is located inside a nodule.
```

Blob detection

After nodule segmentation, there is a prediction for each voxel in the scan if it is inside a nodule. To find the center of nodules, blobs of high probability voxels are searched. To find such blobs, the Difference of Gaussian method is used as implemented in the scikit-image package. After blob detection, there is a list of nodule candidates with their respective centroids. Some problems the authors discovered with this method:

```
Unfortunately the list contains a large amount of nodule candidates. For the CT scans␣
↪in the DSB train dataset, the average number of candidates is 153. The number of␣
↪candidates is reduced by two filter methods:

    Applying lung segmentation before blob detection
    Training a false positive reduction expert network
```

Lung segmentation

After discovering that 2D segmentation didn't work well because the segmentation network didn't have a global context, the authors developed a 3D approach which focused on cutting out the non-lung cavities from the convex hull built around the lungs.

False Positive Reduction

To reduce the large size of nodule candidates that are returned by the blob detection algorithm, an expert network was trained for predicting if the candidate really is a nodule.

```
We used lists of false and positive nodule candidates to train our expert network.␣
↪The LUNA grand challenge has a false positive reduction track which offers a list␣
↪of false and true nodule candidates for each patient.
For training our false positive reduction expert we used 48x48x48 patches and applied␣
↪full rotation augmentation and a little translation augmentation (±3 mm).
If we want the network to detect both small nodules (diameter <= 3mm) and large␣
↪nodules (diameter > 30 mm), the architecture should enable the network to train␣
↪both features with a very narrow and a wide receptive field.
The inception-resnet v2 architecture is very well suited for training features with␣
↪different receptive fields. Our architecture is largely based on this architecture.␣
↪We simplified the inception resnet v2 and applied its principles to tensors with 3␣
↪spatial dimensions.
We distilled reusable flexible modules. These basic blocks were used to experiment␣
↪with the number of layers, parameters and the size of the spatial dimensions in our␣
↪network.
```

(See [1] for a detailed schematic.)

Malignancy prediction

Another network was trained on the LUNA dataset to predict the malignancy of nodules:

```
The network we used was very similar to the FPR network architecture. In short it has␣
↪more spatial reduction blocks, more dense units in the penultimate layer and no␣
↪feature reduction blocks.
We rescaled the malignancy labels so that they are represented between 0 and 1 to␣
↪create a probability label. We constructed a training set by sampling an equal␣
↪amount of candidate nodules that did not have a malignancy label in the LUNA␣
↪dataset.
As objective function, we used the Mean Squared Error (MSE) loss which showed to work␣
↪better than a binary cross-entropy objective function.
```

Prediction of cancer probability

```
After we ranked the candidate nodules with the false positive reduction network and␣
↪trained a malignancy prediction network, we are finally able to train a network for␣
↪lung cancer prediction on the Kaggle dataset.
Our strategy consisted of sending a set of n top ranked candidate nodules through the␣
↪same subnetwork and combining the individual scores/predictions/activations in a␣
↪final aggregation layer.
```

Transfer learning was used to improve the results:

```
At first, we used the the fpr network which already gave some improvements.␣
↪Subsequently, we trained a network to predict the size of the nodule because that␣
↪was also part of the annotations in the LUNA dataset. In both cases, our main␣
↪strategy was to reuse the convolutional layers but to randomly initialize the dense␣
↪layers.
```

To aggregate nodule predictions, the two most successful strategies were described as follows:

P_patient_cancer = 1 - P_nodule_benign The idea behind this aggregation is that the probability of having cancer is equal to 1 if all the nodules are benign. If one nodule is classified as malignant, P_patient_cancer will be one. The problem with this approach is that it doesn't behave well when the malignancy prediction network is convinced one of the nodules is malignant. Once the network is correctly predicting that the network one of the nodules is malignant, the learning stops.

Log Mean Exponent The idea behind this aggregation strategy is that the cancer probability is determined by the most malignant/the least benign nodule. The LME aggregation works as the soft version of a max operator. As the name suggest, it exponential blows up the predictions of the individual nodule predictions, hence focussing on the largest(s) probability(s). Compared to a simple max function, this function also allows backpropagating through the networks of the other predictions.

For the final predictions, the results of the last 30 stage models were ensembled.

Trained model

Not available

Model Performance

Training- / prediction time

As taken from the official documentation:

Test system: 7 machines with following specs: GPU: GTX 1080, GTX 980, Titan X, Tesla K40 Mem: 32GB to 64GB per machine CPU: mostly 6 cores, 3GHz to 4GHz

Training time: 4 days **Prediction time:** 3 days

Model Evaluation

Dataset:

Metric	Score
Accuracy	

Use cases

When to use this algorithm

-
-
-

When to avoid this algorithm

-
-
-

Adaptation into Concept To Clinic

Porting to Python 3.5+

Porting to run on CPU and GPU

With training times and specs as specified in the section above, it does not seem to be worth to try running this on CPU only.

Improvements on the code base

Code is spread across lots of (ambiguously named) files with what seem to be independent Python scripts, lots of cleanup/refactoring needs to be done here.

Adapting the model

The models used for nodule detection are a good approach to find candidates, but the process has to be turned into a reusable pipeline.

Comments

References

[1] https://eliasvansteenkiste.github.io/machine%20learning/lung-cancer-pred/

DL Munich

Summary

Author: Niklas Köhler, Alex Wolf, Mo, Julian Jungwirth **Repository:** https://github.com/NDKoehler/DataScienceBowl2017_7th_place The 7th place at the Data Science Bowl 2017 on the private leaderboard.

License

Prerequisites

Dependency	Name	Version
Language	Python	3.4.3
ML engine		
ML backend	Tensorflow	1.0.1
OS	Ubuntu	14.04
Processor	CPU	Intel(R) Core(TM) i7-4930K CPU
	GPU	Nvidia GTX 1080
GPU driver	CUDA	8.0
	cuDNN	5.1

Dependency packages:

```
opencv-python==3.2.0.6
Python 3.4.3
dicom==0.9.9-1
joblib==0.10.3
tensorflow-gpu==1.0.1
SimpleITK==0.10.0.0
numpy==1.12.0
pandas==0.19.2
scipy==0.18.1
scikit-image==0.12.3
scikit-learn==0.18.1
```

Algorithm design

Preprocessing

1. Resampling to the isotropic resolution of .
2. Segment the lungs on the resampled CT scan via a lung segmentation neural network, trained with around 150 manually annotated lung wings.
3. Crop CT scan by the minimal cube which will fits the lungs.

Nodule detection

Training

In objective to segment lung nodules 5-channel UNet architecture with receptive field has been employed. The model has been trained over the LUNA|LIDC dataset. A random crop of lung slices sampled randomly from all 3 room axis (not only axial) along with four adjacent slices (+2/-2) were feded into the network as an input, the corresponding elliptical nodule annotation mask for the central slice was used as a target output. Training data was made of exclusively croped slices which contain a nodule annotation and 15 random negative slices for each room axis from each patient scan, which do not contain any nodule annotation. Finally, a training batch consists of 30 random nodule sectioning slices and 2 random negative slices.

Prediction

The trained segmentation network has been applied to all slices of a CT scann from all 3 room axis. This results in three 3D probability maps which has been be averaged. This final 3D probability map tensor was the basis for the candidate proposal clustering.

Candidate proposal clustering

Regions in the probability map tensor that have a high probability for being nodules have been identified using DBSCAN. The sklearn.cluster.DBSCAN implementation has been used.

> We chose a relatively low threshold on the probability map to generate an input for DBSCAN and deal with the high number of proposed regions that results from this in two ways.
>
> 1. Regions that are too large to be processed in the later stage of cancer classification are again clustered with a threshold just sufficiently high for producing small enough candidates.

2. Proposed regions are sorted according to a score compute as the sum over the m highest pixel values in the region, where m is approximately the number of pixels in the smallest annotated nodule.

The resulting cluster centers are used to crop candidates from the original data in and resolutions.

Prediction of cancer probability

The cancer classification network uses multiple candidates from the candidate proposal steps as input and predicts a single cancer probability for each patient. From 5 to 20 candidates per patient has been used. Additionally to the pixel information from the lung CT-scan we include the nodule probability map information from the segmentation network in a second input channel. Therefore, the classifier maps a tensor of shape (C,64,64,64,2), where C = 10 is the number of candidates, to a single output probability. The classifier is either a 3D residual convolution network of an 2D residual convolution network. In the case of 3D classifier each candidate is represented by the full (64,64,64,2) tensor. In the case of the 2D classifier each candidate is presented as the 3 center slices from all 3 room axis. The package of these three layers then is treated by the network as a multichannel input (no weight-sharing for different axis).

When applied to a single candidate the 3D residual core model reduces the candidate tensor from (64,64,64,2) pixels to an (8,8,8,128) feature map, by applying multiple stacked convolution and pooling layers (details in code). We reduce this final feature tensor with shape (8,8,8,128) with an 1x1 convolution layer with a single kernel to (8,8,8,1). A final global average pooling followed by a sigmoidal activation function reduces this map to the single final output. We apply this residual core model to all candidates of a patient simultaneously and share the weight in the candidate dimension. When applying the core-module to a multi-candidate input with shape (C,64,64,64,2) we arrive at an (C, 1) output tensor. We then apply a reduction function (max) to this tensor in the second dimension to obtain a single value. The entire graph is schematically shown for the 2D case in figure:

Final Ensemble

For the final solution four different Patient-Classifier-Networks have been employed, the outputs of which were averaged to a single prediction.

Network	isotropic scan resolution	architecture
05res 3DNet	0.5	3D residual network
07res 3DNet	0.7	3D residual network
05res 2DNet	0.5	2D residual network
07res 2DNet	0.7	2D residual network

Trained model

Source: seems that it's no access to the trained models. @reubano has opend an issue at the 3rd of July but there's no responce.

Usage instructions: README

Model Performance

Training- / prediction time

Test system:

Component	Spec	Count
CPU	Intel(R) Core(TM) i7-4930K CPU	
GPU	Nvidia GTX 1080	
RAM	32GB	
	200GB	

Training time: from the training log it seems that one 3D resnet model requires 13 epochs with average 750s per epoch. For the 2D resnet from the coresponding train log folloed that the authors train the model over 17 epochs with the average 150s per epoch. Taking into account that there're two classification models of each type the resulting training time of the classification part will consume about 7 hours. **Prediction time:** unknown, but must be less than 14 min per CT, since it processes the 506 CTs for the 5 days

Model Evaluation

Dataset: Data Science Bowl evaluation dataset

Metric	Score
Log Loss	0.42751

Use cases

When to use this algorithm

- For the nodules candidates detection and segmentation, also in combination with the Alex | Andre | Gilberto | Shize segmentation algorithm.
- As a part of classification batch, may be used only one architecture out of four.

When to avoid this algorithm

- If there's is no GPU support it may consume a huge amount of time. Also the isotropic scan spacings which was used at classification stage are not common, this will lead to additional time consumption caused by resizing.

Adaptation into Concept To Clinic

Porting to Python 3.5+

The solution is already compatible with Python 3.5+

Porting to run on CPU and GPU

The approach consists of four deep 3D / 2D residual networks for classification (which runs through each candidate from a CT scan) and two 3D2D Unet models for nodules and lungs segmentation. It'll require a huge amount of time to even predict with this pipeline using CPU only.

Improvements on the code base

Since the objective of the DSB17 was to guess whether the patient has canser or not, the approach of DL Munich doesn't provide any information for each nodule, but only for a candidates set. It'll be beneficial for the concept-to-clinic to recieve the information for each candidate respectively. It can be done merely by extracting values from before the `max` operator.

Adapting the model

Comments

The algorithm of nodules segmentation is well generalised and may be integrated with the Alex | Andre | Gilberto | Shize segmentation algorithm. Since the main motivation for four architectures was the generalization then if will be used in combination with other approaches it seems to be beneficial to use only one (or two: 3D and 2D) out of four.

References

DL Munich descriptionREADME

grt123 Algorithm

Summary

The algorithm consists of nodule detection and cancer classification based on 3D convolution neural networks. The model receives preprocessed 3D lung scans as input and outputs both the bounding boxes of suspicious nodules and the probability of getting cancer.

Source

Author: Team grt123 **Repository:** https://github.com/lfz/DSB20171st place at the Data Science Bowl 2017

License

Neither mentioned in the repository nor in the technical report, but since the authors had to accept the Data Science Bowl rules, the code must be published under MIT license.

Prerequisites

Dependency	Name	Version
Language	Python	2.7
ML engine	pyTorch	0.1.10+ac9245a
ML backend		
OS	Ubuntu	14.04
Processor	CPU	yes
	GPU	yes
GPU driver	CUDA	8.0
	cuDNN	5.1

Dependency packages:

```
h5py==2.6.0
SimpleITK==0.10.0
numpy==1.11.3
nvidia-ml-py==7.352.0
matplotlib==2.0.0
scikit-image==0.12.3
```

```
scipy==0.18.1
pyparsing==2.1.4
pytorch==0.1.10+ac9245a
```

Algorithm design

First, a binary mask of a lung is applied to the data to filter out as much noise as possible. Next, a CNN is trained to detect nodules that might be malicious. Lastly, another CNN takes the top 5 nodules that appear to be most malicious and tries to predict whether the patient has cancer. For both networks data augmentation is used to artificially increase the amount of data on which they can be trained.

Preprocessing

All raw data is first converted into Hounsfield units. It is then clipped with the window [-1200, 600] and then transformed linearly to [0, 255]. Next, a binary mask of the space inside a lung is calculated.

The mask is multiplied with the raw data in order to quickly remove noise that does not belong to the lung. Everything outside of the mask is padded with 170. All values greater than 210 (typically the bones) are replaced with 170 as well. The padding value 170 is chosen to resemble the intensity of tissue.

Nodule detection

Due to the memory constraints of the GPU, patches of sizes 128 x 128 x 128 are cropped from the lung scans and then fed to the network. 70% of the patches contain at least one nodule while the remaining 30% may contain no modules. If a patch goes beyond the range of the lung scan, it is padded with the value 170. For data augmentation, patches are randomly left-right flipped and resized with a ratio between 0.8 and 1.15. Oversampling was used for big nodules in the training set to achieve a more balanced distribution of nodule sizes. Also, hard negative mining was used to further balance the training set. The network structure looks as follows:

Prediction of cancer probability

After the nodule detection net (N-Net) was trained, the authors applied it onto the whole 3D scan of each patient. The top 5 proposed nodules of the N-Net are then fed to the cancer classification net. Next, they proceeded as follows:

-

We extracted the last convolution layer of N-Net for each proposal, which is a $32 \times 32 \times 32$ cube of 128 features shown in Fig. 3 as a green volume. We max pool over the central $2 \times 2 \times 2$ voxels of each proposal, and pass the resulting 128-D features to two succeeding fully connected layers to generate a prediction of cancer probability. We assume the final diagnosis is made from these 5 proposals and a hypothetical dummy nodule (in case N-Net missed some nodules) independently.*The whole approach can be seen in Figure 4.

The prediction loss is defined at the right hand side of Figure 4 where P_{d} is a free parameter representing the cancer probability from the dummy nodule.

Trained model

Source:https://github.com/lfz/DSB2017/tree/master/model

Usage instructions: Running main.py should use the final models to create the output files for the Data Science Bowl.

Model Performance

Training- / prediction time

Test system:

Component	Spec	Count
CPU		
GPU	TITAN X	8
GPU	Memory	>= 12GB
RAM		

Training time: 4 days **Prediction time:**

Model Evaluation

Dataset: Private Kaggle Test Data

Metric	Score
Log-Loss	0.39975

Use cases

When to use this algorithm

- when you want to achieve the least log-loss on the Data Science Bowl test data

When to avoid this algorithm

- when you want to train the model by yourself from scratch (it took the authors 4 days on 8 TITAN X)

Adaptation into Concept To Clinic

Porting to Python 3.5+

I did not manage to run the complete algorithm so far as I'm lacking CUDA. @reubano attempted to port it to Python 3.5 here.

Porting to run on CPU and GPU

Since modules such as DataParallel are used, it appears to be non-trivial to make the algorithm run only using a CPU.

Improvements on the code base

As expected from a code base created at a competition, most of it is completely undocumented.

Adapting the model

Multiplying the data with a binary mask of a lung is a nice way to remove noise that does not belong to the lung itself. Also, using CNNs definitely makes sense as the weight sharing decreases the amount of memory needed by the model.

Comments

The accuracy and ranking of the algorithm speaks for itself. Though, I'm not sure how easy it is to re-use as it is completely undocumented, written in Python 2.7 and it took 8 TITAN X to train it. Hopefully, to segment a nodule it also works on a regular GPU of a desktop PC.

References

RepositoryTechnical Report

Daniel Hammack Algorithm

Summary

The algorithm stands out with the idea to pre-detect possible abnormal regions which are then fed to two ensembles of 17 3D convolutional neural networks (CNNs). It is combined with the algorithm by Julian de Wit at the prediction level.

Source

Author: Daniel Hammack**Repository:** https://github.com/dhammack/DSB2017The approach scored 2nd place at the Data Science Bowl 2017.

License

Neither mentioned in the repository nor in the technical report, but since the authors had to accept the Data Science Bowl rules, the code must be published under MIT license.

Prerequisites

Dependency	Name	Version
Language	Python	2.7
ML engine	Keras	
ML backend	Theano, scikit-learn	
OS		
Processor	CPU	yes
	GPU	yes
GPU driver	CUDA	
	cuDNN	

Dependency packages: Neither the repository nor the authors specified exact versions of the Python packages:

```
cPickle
joblib
matplotlib
numpy
pandas
pdb
PIL
```

```
pydicom
pylab
scipy
SimpleITK
skimage
```

Algorithm design

Preprocessing

The author states:

```
Each CT scan is resized so that each voxel represents a 1 mm$^3$ volume. This is␣
↪necessary so that the
same model can be applied to scans with different 'slice thickness'. Slice thickness␣
↪refers to the distance
between consecutive slices (when viewing a 3D image as a collection of 2D slices) and␣
↪can vary by up to
4x between scans. The scans are also clipped between -1000 and 400 Hounsfield units.␣
↪Air has a value of
-1000 HU and bone has 400, so values beyond these two endpoints are not informative␣
↪for the diagnosis.
After this, each scan is rescaled to lie between 0 and 1 with -1000 mapping to 0 and␣
↪+400 mapping to
1. A crude lung segmentation is also used to crop the CT scan, eliminating regions␣
↪that don't intersect
the lung.
```

Nodule detection

To detect candidate nodules, the author built a model for identifying regions of the scan that appear abnormal:

```
[...] This model has an architecture similar to our nodule models, but it was trained␣
↪on data with 90% normal
samples and 10% abnormal samples. It used a regression objective with a single output␣
↪representing a
weighted combination of nodule attributes (with more weight for the more important␣
↪attributes). Thus
a larger value is assigned the more 'abnormal' the region.
Each scan is limited to have between 1 and 50 'abnormal regions'. If more than 50 are␣
↪found, the
50 most abnormal are kept (and if none are found we pick the most abnormal␣
↪regardless). An abnormal
region is defined to be any 64 mm$^3$ region of the scan which received a prediction␣
↪higher than a chosen
threshold (our choice was 1 but it's relatively arbitrary as the units for this␣
↪metric aren't meaningful).
```

The author found out that it is very helpful to predict further attributes of the nodules such as diameter, lobulation, spiculation, and malignancy. He suspects that

```
[...] training simultaneously on multiple objectives smooths out the gradients (as we␣
↪were
able to increase the learning rate substantially when training on multiple␣
↪objectives).
```

To create more training data, the author performed random 3D transformations of the input.

Prediction of cancer probability

```
The last step in our pipeline is to forecast a cancer diagnosis given the nodule␣
↪attribute predictions. The
number of nodules per scan is variable and there are 4 predicted attributes for each␣
↪nodule. We manually
constructed features thought to be informative for diagnosis from the nodule␣
↪attribute predictions. Our
features are:
• max malignancy, spiculation, lobulation, and diameter across nodules (4 features)
• stdev of malignancy, spiculation, lobulation, and diameter predictions across␣
↪nodules (4 features)
• location of most malignant nodule (3 features, one for each dimension in the scan)
• stdev of nodule locations in each dimension (3 features)
• nodule clustering features (4 features)
```

When plotting the feature importances, one can clearly recognize that `malignancy` is the feature that correlates the most with the cancer probability. Another interesting fact is that the Z-dimension, so whether the nodule is closer to the head or closer to the feet, is also a very useful feature.

Trained model

Source: https://github.com/dhammack/DSB2017/commit/896282c0f41a4fd752aabe7fa9de6a65943ba745

Usage instructions: The code used to score and save the predictions is in scoring_code.

Model Performance

Training- / prediction time

Test system:

Component	Spec	Count
CPU		
GPU	Tesla K80 / amazon p2.xlarge	1
RAM		

Training time: 8 hours on K80 / 1.5 hours on AWS**Prediction time:** unknown

Model Evaluation

Dataset: Data Science Bowl evaluation dataset

Metric	Score
Log Loss	0.401172

Use cases

When to use this algorithm

- the output of the first network, which finds the 50 most abnormal regions and predicts the attributes `diameter, lobulation, spiculation` and `malignancy`, could be used to suggest abnormal regions to the user including the predicted attributes to offer further support in detecting candidate nodules
- using multiple small networks and ensembling them creates in theory a solution that generalizes very well

When to avoid this algorithm

- running multiple CNNs and combining them adds a lot of computational overhead which results in a longer prediction time. Though, it's still unclear how long the prediction given one CT scan really takes (so this is just an assumption)

Adaptation into Concept To Clinic

Porting to Python 3.5+

A very tough problem might be that the author was not able to specify exact dependency versions he used. This could become a problem when we try to "estimate" the used versions. In the README the author already mentions: `Also - I have noticed that a newer version of OpenCV can break some of my code`. What might be even more devastating is what the author means by stating `Also I sometimes make modifications to my local Keras install to try out new things`. This makes it really complicated to estimate whether porting it to Python 3 is feasible.

Porting to run on CPU and GPU

I haven't found the code that specifies which device to use for training so far. I'd estimate that porting the code to work on a CPU should definitely be possible.

Improvements on the code base

I reached out to the author about how to actually use the proposed algorithm (he suggested to get in contact with him in such a case). Unfortunately, he didn't respond so far.

Adapting the model

One could try to reduce the number of trained networks and predicted abnormal regions in order to speed up the prediction time.

Comments

From my point of view, this is a very promising approach. First using a network to suggest abnormal regions with helpful attributes that should further be investigated is a neat way to reduce the problem size. Unfortunately, there is very little to no documentation of the code. Furthermore, we don't know which exact dependency versions have been used which might result in further problems.

References

Repository: https://github.com/dhammack/DSB2017Technical Report: https://github.com/dhammack/DSB2017/blob/master/dsb_2017_

MDai Algorithm

Summary

Source

License

Apache 2.0

Prerequisites

Dependency	Name	Version
Language	Python	3.5
ML engine	Keras	1.2.2
ML backend	Tensorflow	1.0.0
OS	Ubuntu	16.04
Processor	CPU	(yes/no)
	GPU	yes
GPU driver	CUDA	8
	cuDNN	5.1

Dependency packages:

```
numpy==1.12.1
scipy==0.19.0
pandas==0.19.2
scikit-image==0.13.0
scikit-learn==0.18.1
joblib==0.9.4
pillow==4.0.0
xgboost==0.6a2
keras==1.2.2
tensorflow-gpu==1.0.0
hyperopt==0.1
h5py==2.7.0
redis-py==2.10.5
```

Additional installation instructions:

Install pydicom 1.0.0a1 from source.

```
git clone https://github.com/pydicom/pydicom.git pydicom-src
cd pydicom-src
git reset --hard bbaa74e9d02596afc03b924fe8ffbe7b95b6ff55
python setup.py install
```

Algorithm design

Preprocessing

The DICOM images were preprocessed applying the following steps:

```
1) DICOM images are loaded into a 3D-numpy ndarray.
2) The pixel values are converted into Hounsfield Units (HU).
3) The volume is resampled to isotropic spacing.
```

Sex detection

As a further feature for model training the sex is determined using a deep residual net (ResNet) with residual units based on the architecture proposed by He et al. that was trained on hand-labeled DICOM images. The network architecture is as follows:

Nodule detection

Two variants of ResNets are used to create two sets of probability maps out of the isotropic volume created from the DICOM image, that provide the further algorithm with regions of interest. This is done by extracting patches out of the full isotropic volume and using the model to predict the presence of a nodule in that patch. The model architectures are designed as follows:

750 random hyperparameter sets are used in the further prediction process. These parameters include:

- Probability map of model m05a or m09a
- ResNet to fit boundary box
- ResNet to predict cancer probability
- Cancer Probability threshold
- Size threshold
- Number of nodules used in cancer prediction
- Inner or outer boundary box
- Aggregation function for p(cancer)

A threshold is applied to the density map. Then for each configuration bounding boxes are fit around connected objects inside the probability map. A second ResNet is then used to refine the bounding boxes. These ResNets are designed as follows:

Too small bounding boxes are discarded. The remaining bounding boxes are used to create 3D image subsets out of the isometric volume. These ROIs are again fed to the yet another ResNet to predict whether a nodule is present in it. This ResNet has the following architecture:

ROIs with a too low probability to contain a nodule are discarded.

Prediction of cancer probability

The ROIs remaining at the end of the nodule detection are fed into a third ResNet to predict whether they are cancerous or not. These ResNets is designed as follows:

The resulting probabilities are then aggregated. Further some additional features are created out of the geometry of the fitted bounding boxes, the number of ROIs and the number of predicted nodules.

All features (sex and nodule features) as well as the single probabilities that the patient has cancer resulting of all hyperparameter configurations are used in a XGBoost ensemble model to get a final prediction of how probable the patient has cancer.

Trained model

A trained model is not publicly available, but was requested.

Source: -

Usage instructions: -

Model Performance

Training- / prediction time

Test system: AWS p2.16xlarge instance

Component	Spec	Count
vCPU	Broadwell 2.7 GHz	64
GPU	NVIDIA GK210	16
RAM	732 GB	

Training time: Not specified. **Prediction time:** Several days for the whole DSB dataset.

Model Evaluation

Dataset:

Metric	Score
LogLoss	0.41629

Use cases

When to use this algorithm

- If you want to take into account the patient's sex (only for final prediction, whether the patient has cancer.)
- If you want bounding boxes for probable nodules.

When to avoid this algorithm

- A lot of models are run in parallel for one prediction. Thus this model will probably be computational very expensive.

Adaptation into Concept To Clinic

Porting to Python 3.5+

Porting to run on CPU and GPU

Improvements on the code base

- There is basically no documentation (doc-strings, comments) to explain what the purpose of each function/module is.
- Some functions are extremely long and do a lot of stuff. Breaking them up in multiple functions will improve readability and maintainability.
- There are no unit tests so far.

Adapting the model

Since the goal of this project is to detect nodules and classify each nodule and not to predict whether the patient has cancer overall, the feature generation used for the ensemble model will not be needed, including the prediction of the patient's sex. The patient's sex would also probably be known by the doctor and does not have to be predicted in production. The other parts of the prediction process up until the ensemble is happening can be used mostly unchanged. The final aggregation of the probabilities of cancer for each nodule has to be removed. Also the ensemble model will have to be adjusted to aggregate the predictions for each nodule of each hyperparameter combination instead of an overall probability that the patient has cancer. If possible the number of models in the ensemble should be reduced. The authors also suggest that some parts of the algorithm can be optimized.

Comments

References

Owkin Algorithm

Summary

The algorithm ensembles an approach that uses 3 U-Nets and 45 engineered features (1) and a 3D VGG derivative (2). The cancer probability is predicted by both approaches using XGBoost and in the end ensembled using the average.

Source

Author: owkin (Pierre) and Simon Jegou**Repository:** https://github.com/owkin/DSB2017The approach scored 10th place at the Data Science Bowl 2017.

License

The repository doesn't mention a license but since the authors had to accept the Data Science Bowl rules, the code must be published under MIT license.

Prerequisites

Dependency	Name	Version
Language	Python	2.7
ML engine	Keras	2
ML backend	Tensorflow	1
OS		
Processor	CPU	(yes)
	GPU	yes
GPU driver	CUDA	
	cuDNN	

Dependency packages:

```
tensorflow-gpu
keras
SimpleITK
pandas
pyidcom
h5py
opencv-python
radiomics
xgboost
```

Algorithm design

Approach 1 uses 3 U-Nets which are structured as follows:

Approach 2 in turn uses a 3D VGG derivative that predicts 11 output variables:

Preprocessing

Approach 1 normalizes the Hounsfield values and then uses k-means clustering with k=2.Approach 2 segments the lung using Hounsfield units and fills the lung structures that is according to the author superior to morphological closing.

Nodule detection

Approach 1 applies `a U-Net to each slice in direction X, Y or Z to output a mask with the same shape where the 1s represent the nodules detected by the Unet.` A custom loss functions has been used to train the U-Nets (the Dice coefficient). It `appl[ies] these U-Net on all the slices to obtain 6 different 3D binary segmentation masks of the detected nodules (directions X, Y, Z + unions I1, I2 and I3)`Approach 2 passes 64x64x64 cubes of the patient's scan to the 3D VGG model, which predicts 11 features in turn.

Prediction of cancer probability

Approach 1 `extract[s] the 10 biggest nodules and compute[s] 45 features for each nodule (including location, area, volume, HU statistics etc.)`. These 450 features are then passed to a XGBoost classifier whereupon the 6 predictions are then averaged.Approach 2 passes the predicted features to a XGBoost classifier as well.

Trained model

Source:

- Approach 1: https://github.com/owkin/DSB2017/blob/master/sje_scripts/Unet_{X,Y,Z}.hdf5
- Approach 2: https://github.com/owkin/DSB2017/blob/master/pic_scripts/model64x64x64_v5_rotate_v2.h5

Usage instructions:

```
#Create train features with patch model (Pierre's model)
cd ./pic_scripts/
python kaggle_script_features_patch.py -i /XXX/train/ -o /tmp/patch_model_features_
↪train.csv

#Create train features with unet+radiomics (Simon's model)
cd ./../sje_scripts/
python kaggle_script_features_unet.py -i /XXX/train/ -o /tmp/unet_features_train.npz

#Create test features with patch model (Pierre's model)
cd ./../pic_scripts/
python kaggle_script_features_patch.py -i /XXX/test/ -o /tmp/patch_model_features_
↪test.csv

#Create test features with unet+radiomics (Simon's model)
cd ./../sje_scripts/
python kaggle_script_features_unet.py -i /XXX/test/ -o /tmp/unet_features_test.npz

#Build xgbmodel from train features
cd ./../pic_scripts/
python kaggle_train.py -p /tmp/patch_model_features_train.csv -u /tmp/unet_features_
↪train.npz -l /fusionio/KaggleBowl/stage1_labels.csv -s /tmp/model.bst
```

```
#Prediction from test features
python kaggle_predict.py -p /tmp/patch_model_features_test.csv -u /tmp/unet_features_
↪test.npz -l /fusionio/KaggleBowl/sample_submission.csv -s /tmp/model.bst -f /tmp/
↪sub_final.csv
```

Model Performance

Training- / prediction time

Test system: Unknown

Component	Spec	Count
CPU		
GPU		
RAM		

Training time: Unknown**Prediction time:** 3 min per Patient for approach 1

Model Evaluation

Dataset: Data Science Bowl evaluation dataset

Metric	Score
Log Loss	0.44068

Use cases

When to use this algorithm

- if you want to use other lung segmentation methods than Hounsfield Unit clipping since the presented approaches also use morphologic features, filling and clustering

When to avoid this algorithm

- if a GPU us not available or the memory is very limited (8 million parameters per U-Net)

Adaptation into Concept To Clinic

Since I was lacking the training data and CUDA, I was not able to completely run the scripts.

Porting to Python 3.5+

Porting it to Python 3.5 should be as easy as running `2to3`. After adapting `print` statements to Python 3, I was able to start all scripts without import errors, deprecation warnings or the like.

Porting to run on CPU and GPU

Loading the models was easily possible using Tensorflow CPU. However, I can't make a statement about how long it takes to make predictions using the CPU only.

Improvements on the code base

I like the code base so far.

Adapting the model

We should examine the approaches to segment lungs carefully that were used by the presented approaches. Also, we definitely should have a look at pyradiomics as this allowed one author to easily compute 45 features by just passing the image mask of a nodule and letting pyradiomics do the work.

Comments

I really like the code base. It is relatively clean and well documented for a competition submission. My only concerns are regarding how long it would take to predict nodules of a patient using a CPU only. One U-Net alone has 8 million parameters and the VGG derivative 400k. Approach 1 alone apparently took 2-3 minutes for one patient to predict his cancer probability, which certainly involved at least one GPU.

References

Repository

qfpxfd algorithm

Summary

Kaggle solution Authors: Jia Ding, JunGao, WangDong**Luna solution Authors:** Jia Ding, Aoxue Li, Zhiqiang Hu and Liwei Wang The 4th place at the Data Science Bowl 2017 on the private leaderboard.

License

MIT License

Prerequisites

`Unknown`

Dependency	Name	Version
Language	Python	
ML engine	Keras	
ML backend	Tensorflow	

Dependency packages: `Unknown`

Algorithm design

The algorithm itself consists of 2 steps:

- Candidate detection
- False Positive Reduction

Preprocessing

The axial slices are used as inputs. For each axial slice in CT images, its two neighbors slices are concatenated in axial direction, and then rescaled into 600×600×3 voxels.

Candidate Detection

the architecture of the proposed candidate detection network is composed of two modules: a region proposal network (RPN) that aims to propose potential regions of nodules (also called Region-of-Interest (ROI)); a ROI classifier then recognizes whether ROIs are nodules or not. These two DCNNs share the same feature extraction layers.

The region proposal network takes an image as input and outputs a set of rectangular object proposals (i.e. ROIs), each with an objectness score. It is based on an original Faster R-CNN with a deconvolutional layer added.

To generate ROIs, a small network is slided over the feature map of the deconvolutional layer. At each sliding-window location, multiple ROIs are simultaneously predicted. The multiple ROIs are parameterized relative to the corresponding reference boxes, called anchors.

With the ROIs extracted by RPN, a DCNN is developed to decide whether each ROI is nodule or not. A ROI Pooling layer is firstly exploited to map each ROI to a small feature map.

The ROI pooling works by dividing the ROI into a grid of sub-windows and then max-pooling the values in each sub-window into the corresponding output grid cell. Pooling is applied independently to each feature map channel as in standard max pooling. After ROI pooling layer, a fully-connected network, which is composed of two 4096-way fully-connected layers, then map the fixed-size feature map into a feature vector. A regressor and a classifier based on the feature vector then respectively regress boundingboxes of candidates and predict candidate confidence scores.

False Positive Reduction Using 3D DCNN

With the extracted nodule candidates, a 3D DCNN is utilized for false positive reduction. This network contains six 3D convolutional layers which are followed by Rectified Linear Unit (ReLU) activation layers, three 3D max-pooling lay- ers, three fully connected layers, and a final 2-way softmax activation layer to classify the candidates from nodules to none-nodules.

As for inputs of the proposed 3D DCNN, each CT scan is firstly normalized with a mean of -600 and a standard deviation of -300. After that, for each candidate, the center of candidate is used as the centroid and then crop a 40 × 40 × 24 patch.

Trained model

Source: seems that it's no access to the trained models.

Usage instructions:`unknown`

Model Performance

```
Unknown
```

| Component | Spec | Count | |————|——-|——-| | CPU | | | | GPU | | | | RAM | | | | | | | –>

Training time:**

```
Unknown
```

Prediction time:** unknown, but must be less than 14 min per CT, since it processes the 506 CTs for the 5 days

Model Evaluation

The performance was evaluated on the LUNA16 Challenge with dataset which contains a total of 888 CT scans. In the LUNA16 challenge, performance is evaluated using the Free-Response Receiver Operating Characteristic (FROC) analysis.

Candidate Detection Results showed sensetivity of 0.946 with 15.0 canditates/scan. The competition performance metric (CPM, averaged FROC key-points' magnitude) score is 0.891.

Dataset: Data Science Bowl evaluation dataset

Problem	Score	Metric	Dataset
Lung Cancer Detection	Log Loss	0.40183	Kaggle
False Positive Reduction	CPM	0.891	Luna16
Candidate Detection	Sensitivity </br>	0.946 </br>	Luna16
	Candidates per scan	15.0	

Use cases

When to use this algorithm

- For the nodules candidates detection and segmentation.

When to avoid this algorithm

- If there's is no GPU support it may consume a huge amount of time.

Adaptation into Concept To Clinic

Since the actual repo reference as well as a trained model can not be found, there is no way to adapt the algorithm into Concept To Clinic except for implementing it from scratch. The authors of the solution were emailed in order to get an actual information.

References

Luna16 resultsData Science Bowl results

Therapixel

Summary

The algorithm screens and characterizes nodules as well as masses and scans for emphysema as well as aorta calcifications using pre-trained models. In the end, the results are aggregated and the cancer probability is calculated using XGBoost.

Source

Author: Pierre Fillard**Repository:** https://github.com/pfillard/tpx-kaggle-dsb2017

License

GNU General Public License v3.0

Prerequisites

This is the recommended / tested environment

Dependency	Name	Version
Language	Python	3.5
ML engine		
ML backend		Tensorflow 1.1
OS	Ubuntu	16.04 x64
Processor	CPU	2x Intel i7
	GPU	4xNVIDIA Titan
Alternative GPU	GPU	NVIDIA NVIDIA P6000
GPU driver	CUDA	8
	cuDNN	5.1

Dependency packages: All dependencies have been installed using `pip3`.

```
https://github.com/pfillard/tensorflow/tree/r1.0_relu1
xgboost
numpy
scipy
skimage
SimpleITK
h5py
optparse
```

Algorithm design

Unfortunately, there is no technical report or blog post about the approach so far. Thus, the following notes have been derived from the source code only.

Data format

The input data is expected to have been converted in ITK MetaImage (.mhd, .raw) format. Some suggested ressources for that are here and here.

Preprocessing

The algorithm first segments the lung using a VGG-like 3DConvNet.

Nodule detection

Using 5 pre-trained models, the algorithm searches for nodules and saves their location together with the probability of the detection really being a nodule. Next, two models estimate the malignancy and the size of the nodules using regression techniques.

Further features

Mass detection

Using the same models as for nodule detection, the algorithm tries to detect masses that might also exist beyond the lungs. The same models as for the nodule characterization are used to estimate size and malignancy of them.

Emphysema scan

For each image series, one model scans for emphysema (a lung disease that's responsible for short breath and is most often caused by smoking). The model outputs ten values labeled 'eh0' to 'eh9' whose meaning is still enigmatic to me.

Aorta calcification

Lastly, one model is applied to find calcifications of the aorta. The model outputs ten values labeled 'ah01' to 'ah10'.

Prediction of cancer probability

The features created in the previous steps, which have been saved to separate CSV files, are firstly aggregated into one CSV file. That file only uses the features: eh0, eh1, eh2, max_nod_size, max_nod_size, prob_max_nod, max_malignancy, prob_max_mal, ah0, ah1 and ah2. Lastly, XGBoost predicts the cancer probability based on these features.

Trained model

Source: https://github.com/pfillard/tpx-kaggle-dsb2017/tree/master/models

Usage instructions:

- Use `train.py` for feature extraction and model training
- Use `predict.py` to make predictions

Model Performance

Training- / prediction time

Unknown**Test system:**

Component	Spec	Count
CPU		
GPU		
RAM		

Training time: Prediction time:

Model Evaluation

Dataset: Data Science Bowl 2017

Metric	Score
Log Loss	0.40409

Use cases

When to use this algorithm

- if we also want to include information about masses beyond the lung, emphysema or aorta calcification
- if we're fine with working with black-box algorithms

When to avoid this algorithm

- if converting images in ITK MetaImage is not feasible
- if we want to understand what the design decisions of the author have been regarding network architecture etc. (no documentation given)

Adaptation into Concept To Clinic

Porting to Python 3.5+

The algorithm was written to run with Python 3.5.

Porting to run on CPU and GPU

When trying to run the code without a GPU, I'm getting errors when the models are restored. However, I think that it should be possible to port the models to run on CPUs. I'm just not that experienced with Tensorflow so far.

Improvements on the code base

The `kaggle_utils` and the `lidc` files need to be refactored as they already consist of over 600, respectively, 1100 lines of code.

Adapting the model

It's questionable how much impact the additional features have and how well the nodule detection works itself. Also, the models still are big black-boxes that first need to be interpreted in order to understand them and to make changes that can improve them.

Comments

I'm positively surprised by how well the author documented the technical requirements and how easy it is to overlook the codebase. Unfortunately, I haven't found a blog article or technical report about the approach. Thus, one is supposed to derive from the code what the author's solution concept is and what's the models architectures are. Furthermore, I'm not sure how one is supposed to install the Tensorflow 1.0 fork of the author which adds a patch for relu. It seems that one is required to compile it oneself using `./configure` and `bazel`. I contacted the author and asked for any advices on how to understand the pre-trained models and how to improve the approach to actually use it in an application. However, since the author seems to sell this approach or similar solutions to customers, I doubt that he will be able to help us a lot.

References

RepositorySales Homepage

CHAPTER 9

How to Edit This Document

This documentation is a sphinx project. From this section, you will get set on how to build, install, edit and view documentation locally on your development environment.

The documentation files are located in the `docs` folder.

For this documentation, source filenames that are allowed are Markdown (ending with `.md`) and reStructuredText (ending with `.rst`) files.

How to Install & Build Locally

Since Docker is the officially supported development configuration, when you start the local development with docker, the `documentation service` is also started.

Navigate to http://localhost:8002/ to view the docs, once your local server is up.

Code changes

Because `local.yml` has the volume mappings `./:/app` and `./docs/_build/html:/app/docs/_build/html`, changes made to files during development should be reflected in the running `documentation` container as soon as the dev server process restarts. The `compile_docs` container has a `restart: always` configuration to allow the docs application to automatically compile the markdown and reStructuredText files in the project into `html` and then, sync these files.

So when you do a change in the docs, instead of restarting all services, or if you're running them in daemon mode, when you reload the documentation page, your changes will reflect.

Running the tests

The `tests/test_docker.sh` shows you how to run the tests for the documentation application.

```
$ docker-compose -f local.yml run documentation make -C /app/docs doctest
```

How to Update Table of Contents

The table of contents can be updated using `markdown` or `reStructuredText` files.

To add a new table of contents section to the current one, the file(s) with the section's content have to be linked using sphinx from the entry-point file of the project i.e. `index.rst`.

Example:

In the current `index.rst`, it includes a root `toctree` directive, under which we can add content files or create new `toctree` sections for the table of contents.

```
...
.. toctree::
   :maxdepth: 4 #indicate maximum depth of your tree
   :caption: Descriptive caption

   top-level-file-1
   top-level-file-2
   ...
...
```

Here `top-level-file-1` and `top-level-file-2` can be markdown or reST files corresponding to `top-level-file-1.md` or `top-level-file-2.rst` in the local path.

This will render like this on the table of contents:

...

- *Descriptive caption*
  - *Top level file 1*
  - *Top level file 2*

...

You can read more about toctree here

Useful Links

When writing the documentation, **markdown** is the easiest to use and adapt to. You can have a look at this cheatsheet to help you in writing the markdown files.

If you prefer using **reST** markup, you can have a look at their documentation and also the official sphinx documentation which details the various directives, their purpose and how to use them.

This example PyPi project also details some of the commands available via sphinx.

CHAPTER 10

Indices and tables

- genindex
- modindex
- search