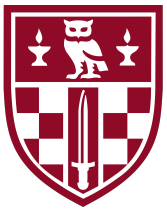
Birkbeck, University of London

Pneumonia Detection from Chest X-Ray Images

Author:
Baran Buluttekin

Supervisor:
Dr. George Magoulas



A project report submitted in fulfillment of the requirements for the degree of MSc Data Science

in the

Department of Computer Science

July 9, 2020

Declaration

I have read and understood the sections of plagiarism in the College Policy on assessment offences and confirm that the work is my own, with the work of others clearly acknowledged. I give my permission to submit my report to the plagiarism testing database that the College is using and test it using plagiarism detection software, search engines or meta- searching software.

The report may be freely copied and distributed provided the source is explicitly acknowledged.

Abstract

Placeholder, to be completed later.

Contents

1 Introduction

Medical diagnosis and specifically computer-aided diagnosis (CAD) is a hot topic in the field of technology. One of the main reasons for becoming a hot topic is the recent innovation and breakthroughs achieved by computer vision research. Combined with poor healthcare coverage around the globe, CAD systems offer a promising solution to mitigate the devastating impact of fatal diseases such as pneumonia. Achieving human-level accuracy in computer vision task in a wide array of classification task such as ImageNet large scale visual recognition challenge (ILSVRC) [1] sparked the debates about whether these CAD systems can reduce or altogether replace the jobs such as radiologist in the future. Controversial topics such as whether or not artificial intelligence will replace the radiologist in the future aside, these automated systems can offer answers for patient's questions in absence of medical help or to very least offer much needed second opinion in the face of unsatisfied diagnoses. Given all the mentioned possible benefits of the CAD systems, this project is focused on building classification CAD systems for diagnosing pneumonia from the chest X-ray images.

1.1 Aims and Objectives

The aim of this project is to build a fully functional chest X-ray image classification pipeline that implements CI/CD principals to experimentation and deployment.

1.1.1 Objectives

Project will be implemented with execution of fallowing objectives:

1. **Carrying out general data exploration:** This part involves general check on dataset.

2. **Data pre-processing and augmentation:** Preparing the data for model ready state.

3. **Building baseline model with well known neural network architectures:** This step involves setting additional benchmarks with out of the box models from section 4.

4. **Using pre-trained network to increase model performance:** Using pre-trained networks to help training and accuracy of the model.

5. **Visualizing neural network to ensure learning quality:** For making sure model learning as intended and focusing on correct parts of the image.

6. **Model ensembling:** Using ensemble method with different neural network architectures.

7. **Applying different deployment options:** Implementation of different deployment options. Based on their trade offs.

It's worth emphasizing that the objective of this project is not to achieve the state of the art result in pneumonia detection but to offers a preferred method for improving and enhancing the existing models. The intuition behind choosing the above objectives instead of attempting to build novel architecture from scratch is the process of choosing a novel architecture has a very large search space and requires a lot of iteration and experimentation. Due to the limited time frame of this project attempting to find new architecture would not be feasible. Additionally, objectives designed to serve the project goal with consistent aims. For example, item 1 and 2 will focus on reducing the model over-fitting while item 5 would serve as a tool to detect over-fitting. Objective 2 serves as a selection for a suitable model and setting benchmark while 6 is aimed at improving the model.

1.2 CI/CD Pipeline

In this section, I will give a brief introduction to the CI/CD pipeline to explain what CI/CD is and why it is chosen as a preferred way to build this project.

Continuous integration (CI) is a workflow strategy that helps ensure everyone's changes will integrate with the current version of the project in the typical software engineering team. This lets members of the team catch bugs, reduce merge conflicts, and increase overall confidence that your software is working. While the details may vary depending on the development environment, most CI systems feature the same basic tools and processes. In most scenarios, a team will practice CI in conjunction with automated testing using a dedicated server or CI service. Whenever a developer adds new work to a branch, the server will automatically build and test the code to determine whether it works and can be integrated with the code on the main development branch. The CI server will produce output containing the results of the build and an indication of whether or not the branch passes all the requirements for integration into the main development branch. By exposing build and test information for every commit on every branch, CI paves the way for what's known as continuous delivery, or CD, as well as a related process, called continuous deployment. The difference between continuous delivery and continuous deployment is that CD is the practice of developing software in such a way that you could release it at any time. When coupled with CI, continuous delivery lets you develop features with modular code in more manageable increments. Continuous development is an extension of continuous delivery. It's a process that allows you to actually deploy newly developed features into production with confidence, and experience little if any, downtime. Even though

the benefits of using CI/CD pipelines are more prominent in the software teams, integration automated testing will help even individual projects such as this by reducing time for debugging.

In more granular detail, this system works with central version control services and in this project central version control service used is Github. GitHub uses a communication tool called *webhooks* to send messages to external systems about activities and events that occurred in the project. For each event type, subscribers will receive messages related to the event. Generally, events refer to action involving the software such as new commit push, pull (merge) request, or other software related action. In this case, whenever a new commit is pushed to any branch of the project, a message from Github will be sent out to a third party system called *travis.* [1] Travis is a hosted CI service that allows build and test software hosted in version control services. When travis receives the webhook call it will fetch the most recent version of the project and run the tests associated with it. When the test runs completed with the latest version of the software, test results will be sent back to relevant commits as status information using GitHub API. This information can either be used by developers for making decisions such as whether to accept the pull request versus reject it or if applicable can be used by service to initiate the deployment process for the software. In all cases, CI/CD will work as automation for software quality assurance process to speed up the development and improve the overall reliability of the software.

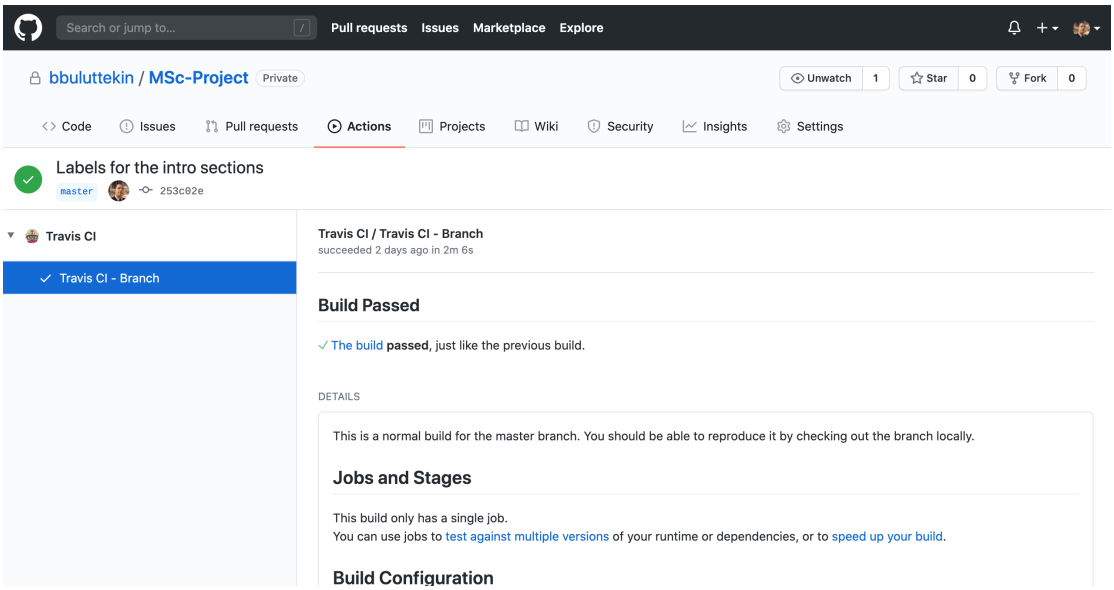


Figure 1.1: CI feedback received from Travis.

1.3 Project specification and design

This project I aimed to keep code and reporting together to provide easy reproduction. Codebase design to be extendable and modular. Therefore, I assign a

[1] https://travis-ci.org/

sub-folder for all the project-specific code under the name *src*. Having a module in the same directory level with the other components allows the ability to use the code in notebook experiment as well as with the tests in the CI integration. Both project proposal and report developed using the LaTeX typesetting system and documents kept in the version controlling to allow easy changes and rolling back to the desired version. Finally, root-level files such as *.travis.yml* and *requirements.txt* is instrumental in defining which steps to take in CI runs and constructing a near-identical environment for software dependencies. Below, I added a directory tree to serve as a guide for navigating and finding project files.

```
├── Proposal
│   └── Proposal files
├── Report
│   ├── chapters
│   │   └── Chapter files
│   ├── img
│   │   └── Images
│   └── Main project latex files
├── scripts
│   └── Utility scripts
├── notebooks
│   └── Experiment notebooks
├── src
│   └── Python library files
├── tests
│   └── Test files
├── .gitignore
├── .travis.yml
├── README.md
└── requirements.txt
```

1.4 Reproducibility Guidance

As a scientific project, it is very important that anyone can reproduce the experiments and findings in this project to verify the conclusions reached are accurate. Main components of reproducible research are open code, open data and repeatable software runtime specification. Open code component is the most straightforward among the other components as the source-code produced part of this project will be shared with the project reviewers and will be made public in GitHub [2] once the assessment of the project is completed. Dataset [2] used in this project is also available through the website URL cited and hosted in online data science community called Kaggle [3]. I have used Kaggle as the main source of accessing this data for two reasons, firstly for its functionality of allowing API calls to retrieve data

[2]https://github.com/bbuluttekin/MSc-Project

[3]https://www.kaggle.com/paultimothymooney/chest-xray-pneumonia

and secondly for managing the data versioning for the user. Data versioning is an integral component of the reproduction of machine learning projects because the model produced by the training will heavily depend on the data it trained. The current version of the dataset as of the writing of this project is version 2. To enable easier runtime replication and to leverage computational power I have chosen to use an online service called Google Colaboratory. [4] Colaboratory or *"Colab"* for short is a free service provided by Google Research. It will allow running Python code through the browser that connected to remote compute resources. Considering that Colab is a remote compute resource, I have created starter utility scripts to automate data acquisition. These files can be found inside the *scripts* folder. Please note that using these script will require obtaining API key from Kaggle platform and this API key file should be in the path specified in the scripts. However, reproducing in the Colab is optional and software dependencies required to produce local development environment is provided with the "requirements.txt" file. Lastly, custom software components for this project resides in the "src" folder and this folder must be placed in a location available to the scope of the python runtime.

[4] https://colab.research.google.com/

2 Background Material

General history of artificial neural networks.

2.1 Building Blocks of Neural Networks

2.2 Exploding and Vanishing Gradients

2.3 Optimization

2.4 Regularization and Over-fitting

2.5 Software Challenges Specific to Machine Learning Systems

Role of hardware accelerators. Debugging related challenges of Neural Networks Talk about core modules and how it will fit into general experimentations.

3 Data

Reminder of the dataset. [3] Data set comprises of jpeg files stored in classification based folders.

3.1 Data Augmentation

Data augmentations applied and example results of the augmentation.

3.2 Limitations of the Dataset

Issues related to validation set size. Variance of the image resolution.

3.3 Data Procession

Information about tf.data processing pipeline and advantages compare to other data feeds.

Information about data processing for scikit-learn models. Talk about the decisions for image size and how it relates to information loss and preservation. Briefly mention the high variance of the image sizes and image resizing options and choice. Discussing different representation of the dataset will be covered. Talk about data module design and functionality.

References

[1] J. Deng et al. “ImageNet: A Large-Scale Hierarchical Image Database”. In: *CVPR09*. 2009.

[2] Daniel Kermany and Kang Zhang. *“Labeled Optical Coherence Tomography (OCT) and Chest X-Ray Images for Classification”*. 2018. DOI: http://dx.doi.org/10.17632/rscbjbr9sj.2.

[3] *Open Access Biomedical Image Search Engine*. https://openi.nlm.nih.gov/. Accessed: 2019-03-12.