CMP/DIG6200 - A3 Submission of Final Dissertation

DeTraC modification with initial dataset

Zan Zver 18133498

Computer and Data Science - BSC (HONS)

Faculty of Computing, Engineering and the Built Environment
Birmingham City University

BIRMINGHAM CITY
University

Project supervisor: Dr. Mohammed Abdelsamea
Submitted on the April of 2022

Abstract

DeTraC is a neural network model that stands for key technologies it uses (decomposition, transfer learning and composition) and is researched as part of BCU.
This project is research based and it focuses on modification of the original model to determine if initial dataset can be used instead of the composed one and if so, what are the results occurring from it.
In order to get results, project was recreated with Python and medical data was used for producing the results.
Desired outcome was initially a question if we can use initial dataset or not instead of composed data. As project was developed, it turned out that the answer is yes, using initial dataset instead of composed dataset in DeTraC model is possible. Now, the question that we had to answer is: should we use the initial data?
At first, tests needed to be reproduced from original paper with new code. This showed some baseline where we are at. After that, parameters were modified to fit initial dataset or composed dataset the most. At the end, it turns out that regardless of parameter turning, usage of initial dataset is not as good as using original composed dataset.

Acknowledgements

First of all, a big thank you to Birmingham City University. They have accepted me to the university, showed me the passion of Data Science and inspired me to continue with further education. BCU was also able to help with providing the server that helped with additional computing power. I have met and learned from many knowledgeable lecturers that I am really thankful for. In particular Dr. Mohammed Abdelsamea and Emmett Cooper who inspired me the most to grow in my final year.

Special thanks goes to the project supervisor Dr. Mohammed Abdelsamea. Throughout the project, Dr. Mohammed was supportive and understanding on the roadblocks that occurred. For me, Dr. Mohammed was a perfect supervisor for this project and a great personal tutor in my final year.

Throughout my studies, I have successfully completed industry placement in Blueberry consult. As part of the Sysadmin team (with the junior position), they have shown me industry best practices and interesting techniques that are applied on real world projects.

Big thanks goes to the my family and my partner as well. Across the project, I have had some ups and downs. In situations like this my partner helped me or celebrated small victories with me. As of my family, they have been supportive in every way, since me and my partner moved to the UK for our studies. Thank you!

Contents

List of Figures

List of Tables

Glossary

BCU - Birmingham City University
CI/CD - Continues Integration / Continues Deployment
Cuda - computing platform and application programming interface from Nvidia (Nvidia n.d.)
DNN - deep neural network
GPU - Graphic Processing Unit, also known as graphic card
MLOPS - Machine Learning Operations
NN - neural network
OS - operating system
SkLearn - Scikit learn
VM - virtual machine

1 Introduction

This paper is research based and it is focusing on an area of data science - neural networks. This report is going to walk you across what the project is (1), literature that correlated or inspired the project (2), how the project was created (3) and how the results were tested and evaluated (4).
Before we can get deeper into the paper, lets discuss what DeTraC actually is. It stands for:

- Decomposition
- Transfer learning
- Composition

It was developed by Dr. A. Abbas, Dr. M. Abdelsamea (who is also project supervisor) and Dr. M. Gaber. Project is being researched as part of BCU (Abbas, Abdelsamea, and Gaber 2020). What DeTraC does well is learning at subclass level. This allows it for faster convergence. Deeper discussion on DeTraC can be found on original paper as well as with the original code (Asmaa 2020) that is being the groundwork for this project.

1.1 Problem Definition

The task this paper covers is not aiming to see if a problem can be solved, rather it is opposing a research question being: can composed dataset be replaced with an initial dataset? If so, is there a performance gain/loss?

1.2 Scope

Scope of this project is to add functionality to the existing model of DeTraC. Since results needed to be evaluated, it needs to be tested against original model (using composed dataset) vs using new model (using initial dataset). That is giving us equal ground for comparison and an opportunity to compare it to the original paper.

1.3 Rationale

This project is a research based. It is to determine if we can improve the functionality of DeTraC based on the theory and to evaluate the work that has already been done.
After the project is complete, if the results are successful (as planned) it can be used in any kind of industry as an image recognition. Project can be tested on different datasets to see if there are any weak points of it, but that should not be the case.

1.4 Project Aim and Objectives

Goal of the project is to modify DeTraC existing code to be capable of training itself on the original dataset since at the moment training is done on the composed dataset. To achieve this goal, main objectives must be meet along the way. Those are:

- Rewriting the code of DeTraC
  - In order to achieve the main goal, old code needs to be rewritten and more modular. There are also some functional bugs in the code that are going to be removed with accomplishing this task.
  - When code is rewritten, implementation of the goal can be achieved.
- Reuse and improvement of technology
  - DeTraC has already been implemented with some of the technologies in the old code (such as transfer learning, clustering, etc...). Some of the technologies that are in use can be improved. For example, in the old code VGG16 is used as neural network, but in the new code it will be replaced with ResNet16.
  - Second item is comparison. It can be done between using composed dataset vs original dataset on the newly written code.

2 Literature Review

This section is focusing on review of articles that are relevant to the project theme. Section 2.1 is describing themes and technologies of the projects and how research that has already been done can help the project while section 2.2.4 is overviewing development of the project over time with the actors who contributed the most.

2.1 Themes

As name suggests, DeTraC is made of:

- Decomposition,
- Transfer learning,
- Composition.

Due to that, transfer learning is one of the first themes on the list. In the subject of technology, transfer learning has the same principle as in real world. We would train an algorithm on one dataset, and then transfer the knowledge of it on another one. Second topic is neural networks. To have working algorithm, neural network is going to be used to train and predict an output. Neural networks do create a mesh or simply network of neurons on which computer decides what to do. The only downside of them is that they sometimes cannot be as flexible.
Since the goal of DeTraC is to process images, we would be focusing on Convolutional neural networks (CNN). This opens up the third topic of discussion. As mentioned above, neural networks do have some downsides, but image comparison is one of the strong points. But CNN is improved point of it, therefore we would increase our accuracy / efficiency of the algorithm.
Forth theme is deep learning. If we are using neural networks (in general) we could focus on a bit more with deep learning. Deep learning is a branch of neural networks and what it does is teaches itself by example as humans do. Example from industry would be autonomous vehicles which are able to recognise traffic signs.
The final theme is clustering. The data (images in our case) that is going to be used by algorithm needs to be organised to some extent. With this, neural network can be guided to train itself to recognise correct patterns. Example from real world would be list of animals. If we would like to teach the difference between a cat and a dog to the kid, we would show them different images of each and tell them what is what. Classification uses the same principle, to achieve the same result.

2.2 Review of Literature and its theory

In this section, we are going to go across the themes and technologies that are the backbone of our project. Bellow, we have listed all of them and tags, so it is

easier to know what each article is discussing. Do note that first 3 technologies (VGG16, ResNet and ResNext) cover bigger scope, therefore titles covering them are organised together. Technologies and its tags:

- VGG16 [VGG16],
- ResNet [ResNet],
- ResNext [ResNext],
- Transfer learning [TL],
- Neural networks [NN],
- Convolutional neural networks [CNN],
- Deep learning [DL],
- Clustering [CL].

2.2.1 VGG16

Source: (Qassim, Verma, and Feinzimer 2018)
Article title: Compressed residual-VGG16 CNN model for big data places image recognition
Tags: [VGG16], [CNN], [CL], [NN]
Description:
Focus of this article is how CNN and VGG16 can increase speed and decrease the size of NN. This can help to form a better understanding for us on how CNN and VGG16 do help our algorithm underneath since we are going to be using CNN in the implementation and VGG16 was a good implementation candidate.

Source: (B. Liu et al. 2017)
Article title: Weld Defect Images Classification with VGG16-Based Neural Network
Tags: [VGG16], [NN], [CL]
Description:
In our example, we are using x-ray images for classification. This article uses similar structure in terms of technology, therefore in the implementation stage (or beforehand) we could take advantage of that and see if there are any improvements we can do based on the article.

Source: (Rezende et al. 2018)
Article title: Malicious Software Classification Using VGG16 Deep Neural Network's Bottleneck Features
Tags: [VGG16], [NN], [CL]
Description:

Classification is one of the technologies that is going to be implemented. This article is explaining how they are using classification with the help of VGG16 for classifying malicious software. In our case, this article is interesting in terms of classification explanation with connection to VGG16.

Source: (Zhao et al. 2018)
Article title: Synthetic Medical Images Using F AND BGAN for Improved Lung Nodules Classification by Multi-Scale VGG16
Tags: [VGG16], [NN], [CL]
Description:
Classification of cancer images with VGG16 – we are doing similar Focus of this article is image classification with VGG16. This project is taking the similar functionality approach to ours, therefore we can learn from it. Do note that the article is doing classification based on cancer images, but in our case we could change the dataset to the same one as in the article.

Source: (Arsa and Susila 2019)
Article title: VGG16 in Batik Classification based on Random Forest
Tags: [VGG16], [NN], [CL], [DL]
Description:
Corelation of this project with ours is small, but they are exposing an interesting way of handling unique patterns with deep learning, clustering and VGG16. Due to the combination of technologies used, it can be an interesting topic to go by.

Source: (Hridayami, Putra, and Wibawa 2019)
Article title: Fish Species Recognition Using VGG16 Deep Convolutional Neural Network
Tags: [VGG16], [NN], [CL], [CNN]
Description:
The article is tackling the best approach of image recognition on fish based on colour (RGB). They do use different colour sectors and that is important for us since there is some colour in images we are using and if this can be enhanced, this article can help us with it.

Source: (Z. Liu et al. 2019)
Article title: Improved Kiwifruit Detection Using Pre-Trained VGG16 With RGB and NIR Information Fusion
Tags: [VGG16], [NN], [CNN], [CL]
Description:
This article is a bit of an extinction on colour based on previous one. The main difference is that this article is using method RGB-D (Red Green Blue-depth). In the article they are using CNN and VGG16 with RGB-D to improve fruit detection. Might not be as useful for us since we are not dealing with much colour in

the dataset.

Source: (Krishnaswamy Rangarajan and Purushothaman 2020)
Article title: Disease Classification in Eggplant Using Pre-trained VGG16 and MSVM
Tags: [VGG16], [NN], [CL]
Description:
In this study, they are trying to use multi class support vector machines (MSVM) and VGG16 for disease detection. In our project, we are trying to detect disease as well, therefore this can be helpful.

Source: (Theckedath and Sedamkar 2020)
Article title: Detecting Affect States Using VGG16, ResNet50 and SE-ResNet50 Networks Tags: [VGG16], [ResNet] ,[NN], [CNN], [CL]
Description:
This article is comparing different networks, VGG16 and ResNet which are our top candidates are in the list. Due to that, this article could help to guide us to which network is the best for our use.

Source: (Yang et al. 2021)
Article title: A novel method for peanut variety identification and classification by Improved VGG16
Tags: [VGG16], [NN], [CL], [CNN], [ResNet]
Description:
This article is describing the new and improved ways VGG16 can detect images. It is compared to ResNet, therefore it can help to form a better decision for the final algorithm of chaise. If VGG16 is the better algorithm, this article can help us to improve the efficiency of it.

2.2.2 ResNet

Source: (Targ, Almeida, and Lyman 2016)
Article title: Resnet in Resnet: Generalizing Residual Architectures
Tags: [ResNet], [CNN]
Description:
This article is explaining how ResNet can be used within itself. This could help us improve the performance of ResNet since based on the article there is no computational overhead.

Source: (S. Li et al. 2016)
Article title: Demystifying ResNet
Tags: [ResNet], [NN] Description:
This article is deeply connected with our project since it goes into the depths of ResNet. Due to ResNet being a candidate algorithm, this article can give us more

information if the choice is correct or not.

Source:(Szegedy et al. 2017)
Article title: Inception-v4, Inception-ResNet and the Impact of Residual Connections on Learning
Tags: [CNN], [ResNet]
Description:
Inception networks are example of "very deep convolutional networks". Due to that, they have been in the centre of image recognition over the years. This article would explain to us how inception networks with combination of ResNet can accelerate the speed and efficiency.

Source: (Chen et al. 2017)
Article title: ResNet and Model Fusion for Automatic Spoofing Detection
Tags: [ResNet], [CNN]
Description:
In this article, they are using ResNet to detect if speakers have been spoofed or not. The article is not related to our topic, but it is an interesting view of ResNet in the sound industry and how it can be applied on different systems.

Source: (Akiba, Suzuki, and Fukuda 2017)
Article title: Extremely Large Minibatch SGD: Training ResNet-50 on ImageNet in 15 Minutes
Tags: [ResNet]
Description:
This is one of the key articles for ResNet. The algorithm (ResNet) is being praised for its efficiency, therefore with this study we can see why. They are testing and proving how efficient ResNet50 is and based on that we can get some information if this is appropriate for our use.

Source: (Lin and Jegelka 2018)
Article title: ResNet with one-neuron hidden layers is a Universal Approximator
Tags: [ResNet], [NN]
Description:
This article is explaining how increasing representational power for narrow deep networks is due to one-neuron hidden layers. The article itself does not corelate well with our theme, but in case ResNet is chosen as primary algorithm, this might be something worth looking into.

Source: (Allen-Zhu and Y. Li 2019)
Article title: What Can ResNet Learn Efficiently, Going Beyond Kernels?
Tags: [ResNet], [NN]
Description:

This article is talking about efficiency of ResNet. The algorithm itself is known to be efficient, but this article is proving that with questioning how it can get accuracy of more than 96percent in CIFAR-10. The article is a great opportunity for us to evaluate performance of ResNet.

Source: (Farooq and Hafeez 2020)
Article title: COVID-ResNet: A Deep Learning Framework for Screening of COVID19 from Radiographs
Tags: [ResNet], [CNN]
Description:
This article is in the similar zone in terms of topic compared to ours. The researchers are scanning Covid19 x-ray images and determining result with ResNet. Due to this, we could get a sense of ResNet, its limits and if it is good for our application.

2.2.3 ResNext

Source: (Koné and Boulmane 2018)
Article title: Hierarchical ResNeXt Models for Breast Cancer Histology Image Classification
Tags: [ResNext], [CNN], [CL]
Description:
The article is using BACH system challenge for image classification. With that, they are trying to predict the stage of breast cancer. With the train/test split of 75/25, the accuracy they are getting with ResNext is 0.99.

Source: (Sharma and Muttoo 2018)
Article title: Spatial Image Steganalysis Based on ResNeXt
Tags: [ResNext], [CNN], [CL]
Description:
This article is an example of great use of ResNeXt. They are using CNN on ResNext to generate stronger image representations for steganalysis. With this they can easily detect hidden secret messages within images (steganalysis).

Source: (Orhan 2019)
Article title: Robustness properties of Facebook's ResNeXt WSL models
Tags: [ResNext] Description:
Facebook (now known as Meta) has published Facebook AI which is trained on 1B images from Instagram. Researchers in this article have compared the model with the ResNext and tried its robustness.

Source: (Pant, Yadav, and Gaur 2020)
Article title: ResNeXt convolution neural network topology-based deep learning

model for identification and classification of Pediastrum
Tags: [ResNext], [CNN]
Description:
This article is describing use of ResNext model on data set of 42k images. ResNext topology differentiates cells based on information in the data and gives the experimental result of 98.45percent and F1 score of 98. Since ResNext is a contender for algorithms, this is an amazing discovery.

2.3 Review

In previous section(s) we have gone across themes and technologies that are going to help us. This section is describing how technologies have evolved over time and who the most important actors are.

VGG16 originated at Oxford university under the name of Visual Geometry Group (VGG) in the year 1989 (Zisserman, Giblin, and Blake 1989). The authors did publish the research at first as a mathematical paper, but nowadays it is used in the computer vison (Simonyan and Zisserman 2014). In the 1991, VGG team used the mathematics for the first time for the 3D object recognition (Forsyth et al. 1991). Now days, team is working on different parts of image recognition and VGG network is still being developed, now moving into VGG19 (Xiao et al. 2020).

Arguably roots of ResNet originated in 1996 (Fitzpatrick 1996) when Fitzpatrick D. did research on Functional Organization of Local Circuits. Thomson A. (Thomson 2010) has published another research in 2010 that prompted the research of ResNet. But why did nobody do any work prior to that? The reason is that ResNet is neural network that has similar structure to residential nets in human body, cortical layer VI neurons (Proulx et al. 2014) is a bit more specific example of it. In 2015, paper was published with the title of "Highway Networks" (Srivastava, Greff, and Schmidhuber 2015). It describes what ResNet shall become. In 2016, ResNet is created in the paper "Deep Residual Learning for Image Recognition" (He et al. 2016). To this day, ResNet is developing itself as ResNet, ResNext, Wide ResNext... it all depends on the task that is needed on.

In the 2017, Facebook (with the help of University San Diego) has made a publicly available code of ResNext (Xie et al. 2020). A research paper has also been published on that theme (Xie et al. 2017). Paper is describing how they have changed ResNet to ResNext with adding some features (example: new dimension "cardinality"). In the paper, they have compared ResNet101 and ResNext101 and seen that ResNext is an improvement over ResNet, especially in classification accuracy.

Transfer learning started its development in 1976 by Stevo B (Bozinovski 2020). The paper gives the mathematical and geometrical model on how transfer learning could work with neural networks. In 1993, first paper was published with machine learning in the combination of transfer learning (L. Y. Pratt 1992). Since then, the field has advanced to include multitask learning (Thrun and L. Pratt 1998). Now, transfer learning is becoming more and more popular in machine learning tasks, Andrew Ng has predicted that this is going to happen in 2016 (Ng 2016).

Neural network development started in 1943 with algorithms and threshold logic (McCulloch and Pitts 1943). In 1940 mechanise of neural plasticity or Hebbian learning was discovered (Hebb and Penfield 1940). All the information combined, helped with first computers at the time as well as "primitive neural networks". Werbos's research in 1975 (Werbos 1975) reignited the interest in neural networks. The research paper showed the practical training in multi-layer networks. In 1986, neural networks with parallel distributed processing became popular due to the article that was published on that theme (Rumelhart, Hinton, McClelland, et al. 1986). The article explained how neurons can be simulated with the help of connectionism. In the year of 1992, multi-level networks were proposed with pretrained one layer and using unsupervised learning (Schmidhuber 1992). In 2006 it was Hilton G. that proposed used of RBM with the high-level representation and real valued variables (Hinton and Salakhutdinov 2006).

Convolutional neural networks (CNN) use mathematical operation called convolution for its work. They are mostly used in image processing now days. The work on CNNs started in 1959 by analysing cats visual cortices (Hubel and Wiesel 1959). This work was then upgrade in 1980 by Fukushima (Fukushima and Miyake 1982). The newly discovered work (neocognitron) introduced two new layers: convolutional layers, and downsampling layers. In 1989, they have used CNN to read handwritten ZIP code numbers (LeCun et al. 1989). Besides CNNs being discovered early on, they needed a lot of computational power to run. Due to that, CNNs started becoming more and more popular in 2000s since computing power is increasing every year (Oh and Jung 2004). Now days, CNNs are implemented in a lot of NN technologies due to their efficiency.

In the book questioning who started with deep learning (Tappert 2019), conclusions are drown towards Rosenblatt F in his book (Rosenblatt 1961). in 1961. First working example of deep learning was published in 1967 (Ivakhnenko 1967). The term "deep learning" was firstly introduced in 1986 (Dechter and Pearl 1988). Due to computational power, deep learning was not developing so fast compared to neural network development, but other fields in the AI have overcome some problems of it beforehand. After the computational power became cheaper (2000s), CNNs have evolved and that allowed deep learning to evolve as well.

Clustering (or k-means clustering) is a method of vector quantization. The terms "k-means" was firstly used in 1967 by James M (Huber, Le Cam, and Neyman 1967). The algorithm was first proposed in 1957, but article was published later in 1982 (Lloyd 1982). Now days, clustering is used in AI all the time.

2.4 Summary

This section describes the themes, technologies, and articles applicable based on the project. Section 2.2 if mainly focusing on the use of literature based on the article information while in the section 2.2.4 information regarding development over time can be found.
Based on the section 2.2, we have found out that different technologies have different approaches and capabilities. The algorithm of choice that seems the most convincing so far is ResNet due to its efficiency and ResNext being the second candidate that needs to be tested on our application if it makes sense.

3 Methodology and implementation

3.1 Methodology

Methodology type that makes the most sense for this project is agile. The reason for agile is its continuing circle of development and testing. In comparison to waterfall methodology, agile allows to do continues development in corelation of research.

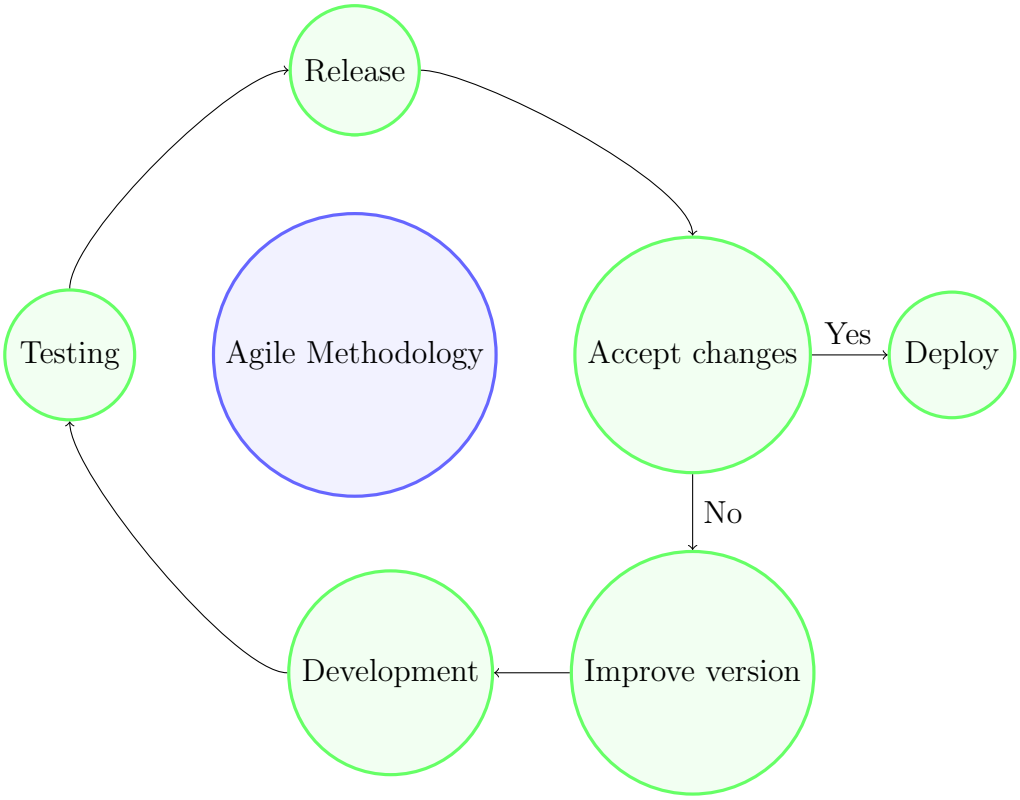

Figure 1: Representation of agile approach

To manage time, Gantt chart was used. If this would be a group project, an online solution that can be shared across the team and track the progress would be better fit (example: Trello, Monday.com).

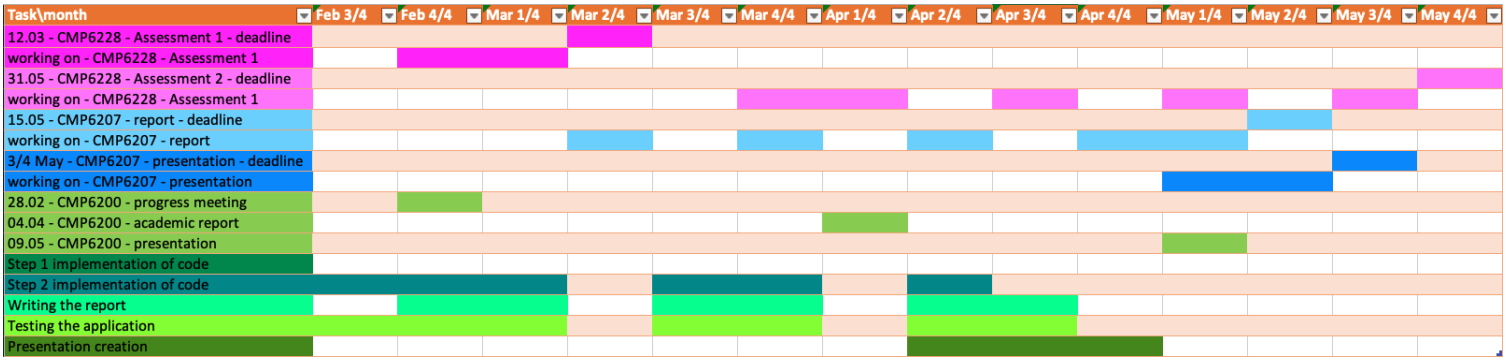

Figure 2: Gantt chart figure

3.2 Dataset description

DeTraC is not limited to work with only one dataset. It can work with any image dataset provided to it. The only thing that needs to be checked is that the dataset has is already classified. So for example, if we are using cats and dogs dataset, there needs to be one folder with cats and one with dogs inside the "Data/inital-dataset" folder.

3.2.1 Dataset in use

So far, different datasets have been tested while code was under the stage of the development. Now in this report, we are going to mainly use "Collection of textures in colorectal cancer histology" dataset (Kather et al. 2016).
The origin of the images is from authors archive - Institute of Pathology, University Medical Center Mannheim, Heidelberg University, Mannheim, Germany and all of the experiments were approved by the institutional ethics board (approval 2015-868R-MA).

3.2.2 About dataset licence

Dataset is licensed under "Creative Commons Attribution 4.0 International Public License". The only element bounding us to the licence is us attribution. With that in mind, author is asking to be cited upon use of their work (Commons n.d.).

3.2.3 About dataset

The dataset is representing a collection of textures of human colorectal cancer. It contains two files but we are only using "Kather_texture_2016_image_tiles_5000.zip" file. This file contains 5000 images of the size 150px * 150px.
Upon decompressing the zip file, we are represented with 8 classes that we can use. Each class has 625 images inside in the tif format. Classes that are available are:

- tumor,
- stroma,
- complex,
- lympho,
- debris,
- mucosa,
- adipose,
- empty.

3.3 Implementation

3.3.1 Flowchart

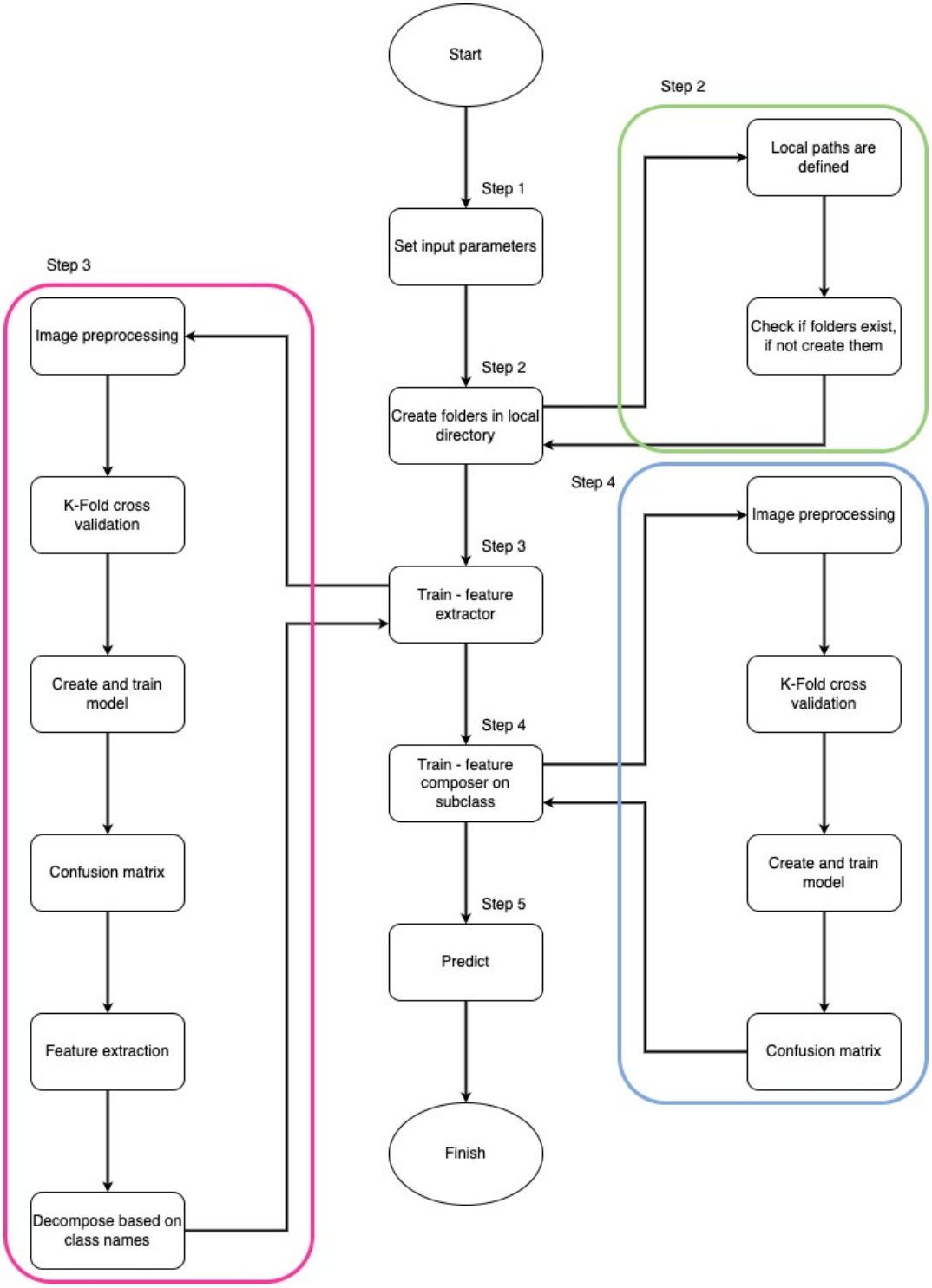

Figure 3: Figure representing flowchart of the code

The figure above is describing the steps that are taking place in order to have a successful program. If we have a look at the program from general perspective, we can deduct it into the 5 different steps. With this flowchart, we can go across each step and see where problem did occur and why.

Step name	Step description
Step 1	Configuration of parameters
Step 2	Folders are created if they do not exist
Step 3	Train the model on feature extractor
Step 4	Train the model on composed or initial data
Step 5	Model prediction

Table 1: Description of steps in flowchart

3.3.2 Tools

Listed bellow, we can see two tables describing what tools have been used in this project. This project was done on 100% open source tools to reduce the cost.
Do note that we are using the server from BCU that has Microsoft Windows operating system installed. This was not something we could change, so to work around that WSL (Appendix B: WSL) was added with Ubuntu OS on it. With that in mind, this project can be reproducible without Windows.
Third table has server description listed. Server is running as a VM in the host OS.

Tool name	Tool description
Server	Windows server, rented by BCU for computational power.
WSL - Ubuntu	For executing code in Ubuntu terminal.
Python	Programming language used for this project.
Pip/Conda	Pythons package manager tools.

Table 2: Main project tools

Python package name	Package description
cv2	image augmentation (eg: resize image)
datetime	get the current date
keras	for data augmentation
loguru	for better logging
matplotlib	allows us to draw graphs
numpy	better mathematical number handling (ex: numpy array)
os	allows us to access files/folders with Python
pytorch	AI and ML libraries
random	for random image shuffle
shutil	for creating new files/folders with Python
sklearn	for data science libraries (ex: KMeans)
tensorflow	AI and ML libraries
tqdm	displays progress bar in terminal

Table 3: Main Python packages

VM component	VM component specification
OS	Windows 10
CPU	16 cores
RAM	64GB
GPU	48GB
Storage	700GB
Privileges	Administrator

Table 4: Server specification

3.3.3 Versioning

To maintain different versions of the code, Git (Appendix A: Git) was used for versioning. In more specific terms, GitHub was the chosen platform for hosting code online.
In the projects life cycle, we have moved from DeTraC to DeTraC2. In the original development (that was forked from (asmaa4may n.d.)), there were some bugs and unnecessary features (for our use case) implemented. To combat this, new repository was created (DeTraC2) which is hosting the current up to date code. There was a different approach that was taken when new code base was started and that being "add what is needed". To be a bit more specific, one feature was added, tested and improved. So over time, features that were not needed were not added.
In the future, this process could be improved with CI/CD (Appendix C: CI/CD) or MLOPS (Appendix D: MLOPS).

3.3.4 Execution

Code execution is kept simple. In the WSL-Ubuntu, we navigate to the correct folder where code is located and then execute with command `python3 Code/main.py`. If there are not any folders in place, that are required, they will be automatically generated.
Do note that the folder Data/initial-dataset is the main data source. We need to put the data inside this folder and the data needs to be classed together (ex: folder for dogs, folder for cats).
If there is a need to change the parameters or algorithm (for example, switch from ResNet18 to ResNet50), user can do that in Code/main.py. There is a set of parameters that can simply be adjusted (such as number of epochs, batch size, learning rate, etc...)

3.3.5 Saving results

In the new version of DeTraC code, "Graphs" folder is created for storing graphs and model results inside. After a few executions, it became clear that storing graphs inside one folder can be messy. To overcome the problem, new sub-folder is generated every time model is executed. With this, we get "Model-Run-execution-time" (execution-time is when code was executed) structure. This systems allows us for better organisation and better ability to compare results over time.

3.3.6 Development execution

Main development folder is created under the name "DeTraC_V2". The folder contains 4 sub-folders that are needed in order for code to work.
The folders bellow are automatically generated if they could not be found. The exception is Code folder since it hosts actual code.

- Code
  - Has all the Python files.
- Data
  - Contains original data and rearranged data (based on original).
- Graphs
  - All of the results are saved here.
- Models
  - Past models are saved under this location.

Tree folder structure (Figure 2) is showcasing how each folder is structured. Do note that this is only representing roots folders in depth 1 and items in the folders. Each section bellow is going in more depth regarding items stored in them.

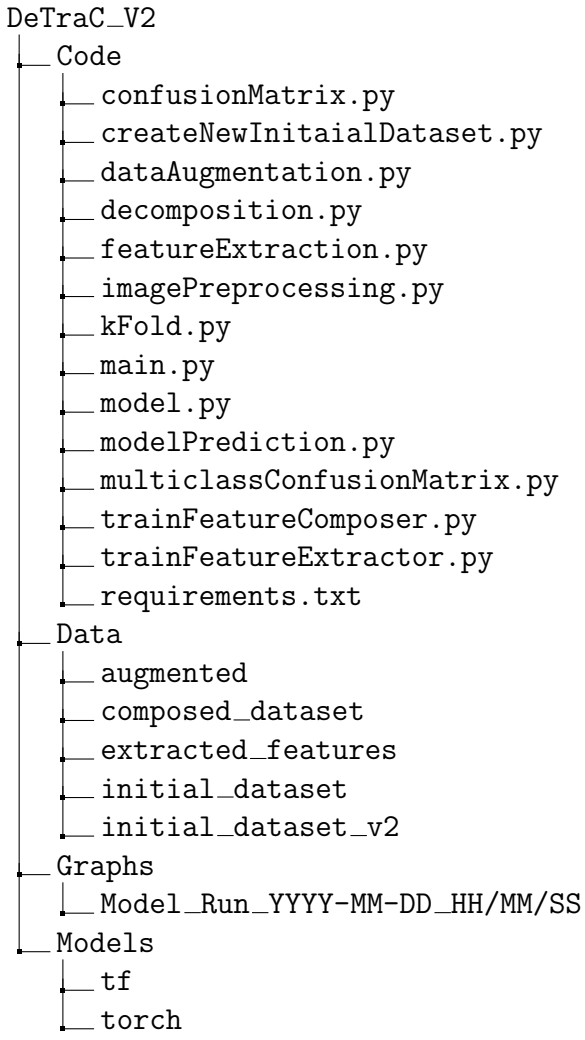

Figure 4: Example of tree folder structure of root folder

3.3.6.1 Code

This folder is home to all of the Python files. Main Python file (main.py) is responsible for executing the correct subprograms when needed. This modular approach is better for the long run if we want to change any functionality. Another bonus of this approach is readily. It is simple to read the code that is organised across multiple files in compare to having one big file.
In the table bellow, all of the programs that are used are listed and their functionality is explained. There is also a tree structure provided for visualisation of the "Code" folder.
Note: requirements.txt file contains versions of libraries used in the program.

Program name	Program description
confusionMatrix.py	Creates new confusion matrix
createNewInitaialDataset.py	Creates new folder initial_dataset_v2 and fills it with original data
dataAugmentation.py	If needed, data can be augmented by adding new "synthetic" images
decomposition.py	Decomposes the images into the correct folder
featureExtraction.py	Extracts features and save them locally
imagePreprocessing.py	Preprocesses images to fit the size and correct colour
kFold.py	Does k fold cross validation
main.py	Body of the code is here
model.py	Model is build in this section
modelPrediction.py	Predictions are executed here
multiclassConfusionMatrix.py	Helper class for confusion matrix
trainFeatureComposer.py	Main body for feature composer - everything is build here
trainFeatureExtractor.py	Main body for feature extractor - everything is build here

Table 5: Programs described

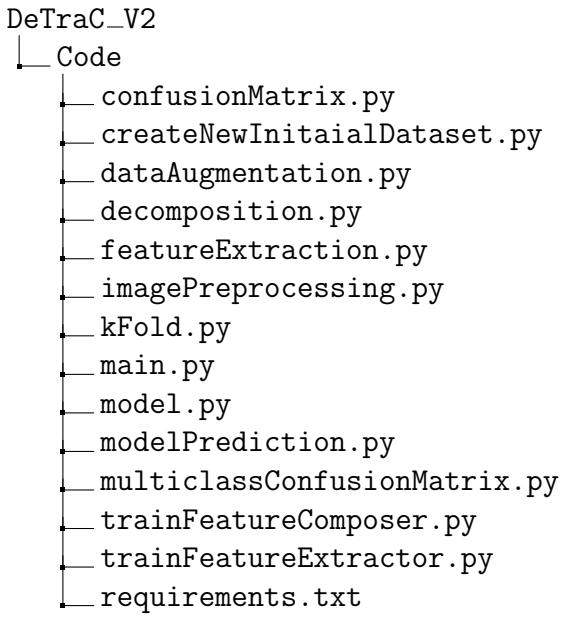

Figure 5: Example of tree folder structure of Code folder

3.3.6.2 Data

The Data folder has 5 subfolders inside as seen in the tree structure bellow. At the start of the run, Python does a check if folder exists or not and it crates missing folders if needed.
In the table bellow, we can see a short description of what each folder does.

```
Data
├── augmented
├── composed_dataset
├── extracted_features
├── initial_dataset
└── initial_dataset_v2
```

Figure 6: Data folder structure

Program name	Program description
Augmented	For augmented (synthetic images) created if more images are needed
Composed dataset	Contains images that are being marked as composed
Extracted features	Python saves model features in this subfolder
Initial dataset	Source of original data that
Initial dataset v2	Same dataset as initial dataset, but rearranged so it can be used for a prediction

Table 6: Data folders explained

Augmented folder contains synthetic images. In the main.py, we can specify if we want to use augmented data or not. In case we don't initial dataset is going to act like main dataset. Otherwise if we want to use augmented data, new images are going to be created by changing features of existing images. This is done in subprogram dataAugmentation.py. The subprogram creates a subfolder under the Data folder with name augmented. This folder then hosts folders with the same name as in initial dataset but with data to be different. In each of the folders, images are constructed as label (folder name) + some set of random characters. In the figure bellow we can see how folder structure is represented for augmented images.

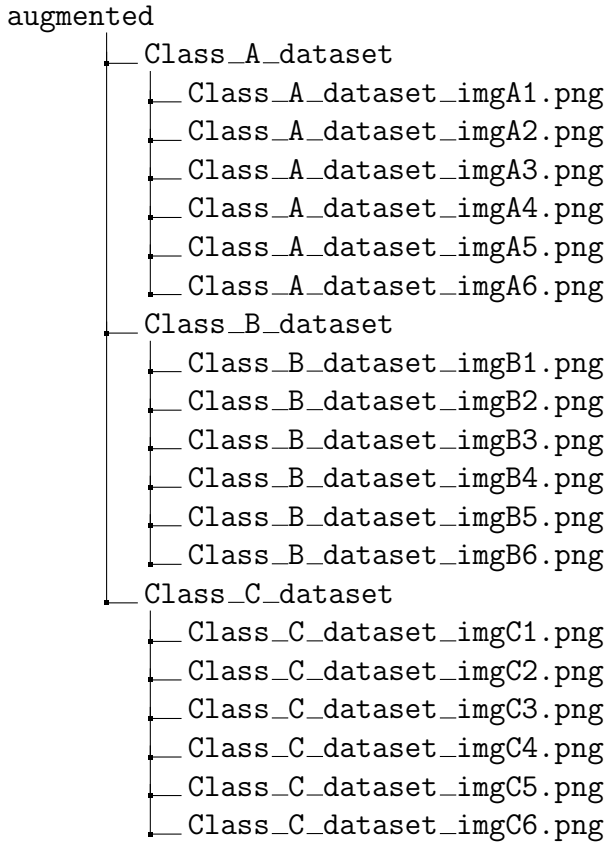

Figure 7: Example of tree folder structure for augmented folder

Composed dataset is created at the start. When we execute the code, trainFeatureExtractor.py is called, which calls function execute_composition (in decomposition.py). This function has 4 inputs (initial dataset path, composed dataset path, extracted features path and K). Based on those features, it generates data in composed dataset folder.
When Python is creating Composed dataset, it checks first if folder exists, if not it creates it. In case that there is already a folder with data in place located, old data is removed. In the process of creation, 2 clusters are created for each class name. Once folders (clusters) are created, data is filled into the correct cluster. Tree diagram bellow is representing an example how clusters are structured. If we have classes A, B and C, class gets its own cluster 1 and 2 with data (from initial dataset) in both of them.

```
composed_dataset
    ├── Class_A_dataset_1
    │   ├── imgA1.png
    │   ├── imgA2.png
    │   └── imgA3.png
    ├── Class_A_dataset_2
    │   ├── imgA4.png
    │   ├── imgA5.png
    │   └── imgA6.png
    ├── Class_B_dataset_1
    │   ├── imgB1.png
    │   ├── imgB2.png
    │   └── imgB3.png
    ├── Class_B_dataset_2
    │   ├── imgB4.png
    │   ├── imgB5.png
    │   └── imgB6.png
    ├── Class_C_dataset_1
    │   ├── imgC1.png
    │   ├── imgC2.png
    │   └── imgC3.png
    └── Class_C_dataset_2
        ├── imgC4.png
        ├── imgC5.png
        └── imgC6.png
```

Figure 8: Example of tree folder structure for composed_dataset folder

Features are extracted at the beginning as well, when trainFeatureExtractor.py is called. In there we call fuction extract_features (from featureExtraction.py) which takes in 6 arguments (initial dataset path, class names, width, height, model and framework). Based on this, it reads grayscale image, converts img to RGB, resizes the image if necessary and saves image to the aray. This allows it to be saved locally. Process is repeated for each class. Example of how folder is structured can be seen in the tree structure bellow.

```
extracted_features
    ├── Class_A.npy
    ├── Class_B.npy
    └── Class_C.npy
```

Figure 9: Example of tree folder structure for extracted_features folder

Initial (or source) dataset accepts different image formats (for example: JPG, PNG, TIF). DeTraC works with different kind of datasets, it was tested with multiple of them. In this report, we are using medical dataset (3.2 Dataset description). In order for code to work properly, initial folder needs to have at least two classes. Tree structure bellow is an example of how source folder is expected to be structured.

```
initial_dataset
    ├── Class_A_dataset
    │   ├── imgA1.png
    │   ├── imgA2.png
    │   ├── imgA3.png
    │   ├── imgA4.png
    │   ├── imgA5.png
    │   └── imgA6.png
    ├── Class_B_dataset
    │   ├── imgB1.png
    │   ├── imgB2.png
    │   ├── imgB3.png
    │   ├── imgB4.png
    │   ├── imgB5.png
    │   └── imgB6.png
    └── Class_C_dataset
        ├── imgC1.png
        ├── imgC2.png
        ├── imgC3.png
        ├── imgC4.png
        ├── imgC5.png
        └── imgC6.png
```

Figure 10: Example of tree folder structure for initial_dataset folder

Initial dataset v2 is a copy of source data. Images in this folder are the same as in initial dataset but it has the same structure as composed dataset. This is for code consistency reasons. With this structure, we can use original code and compare it to new one (using initial dataset) and be sure that there aren't any bias changes in place. The folder is created automatically at the start when function execute_newInitaial_dataset (from createNewInitaialDataset.py) is called. It creates clusters same as in composed dataset and it splits the data from initial data set to cluster 1 or 2 at random. Do note that it is even distribution between clusters. Example of how initial dataset v2 structure looks like is shown in the tree structure bellow.

```
initial_dataset_v2
    ├── Class_A_dataset_1
    │   ├── imgA1.png
    │   ├── imgA2.png
    │   └── imgA3.png
    ├── Class_A_dataset_2
    │   ├── imgA4.png
    │   ├── imgA5.png
    │   └── imgA6.png
    ├── Class_B_dataset_1
    │   ├── imgB1.png
    │   ├── imgB2.png
    │   └── imgB3.png
    ├── Class_B_dataset_2
    │   ├── imgB4.png
    │   ├── imgB5.png
    │   └── imgB6.png
    ├── Class_C_dataset_1
    │   ├── imgC1.png
    │   ├── imgC2.png
    │   └── imgC3.png
    └── Class_C_dataset_2
        ├── imgC4.png
        ├── imgC5.png
        └── imgC6.png
```

Figure 11: Example of tree folder structure for initial_dataset_v2 folder

3.3.6.3 Graphs

In order to visualise results of model runs, graphs are saved locally. After each successful run, under the Graphs folder, subfolder with model run and time is created. Each model run has three subfolers generated as well: accuracy graphs, loss graphs and results confusion. Based on what is needed, we can navigate to accuracy graphs, loss graphs or results from confusion matrix. There is also a file with the name of "Parameters.txt", which is generated to provide information on parameters (or weights) that were used for this model run.
The subfolder (accuracy and loss) has three different subfolders inside as well for composed path, initial path and feature extractor. After successful run is complete all tree graphs do have graphs saved inside for comparison.
Results from confusion matrix are saved under the results confusion subfolder. This folder creates three different subfolders inside: Composed, FeatureExtractor and Initial. The goal is to save results (accuracy, F1, etc...) from confusion matrix into the correct run. So if we are looking at the graph that used composed dataset we can see results from confusion matrix that are saved in composed folder as Results.txt.
With this structure, we can see past model runs and compare results. This has came in handy in the past since it is organised in the simple way. As mentioned above, MLOPS concepts would improve workflow like this. Structure of the Graphs folder can be seen in the tree diagram bellow.

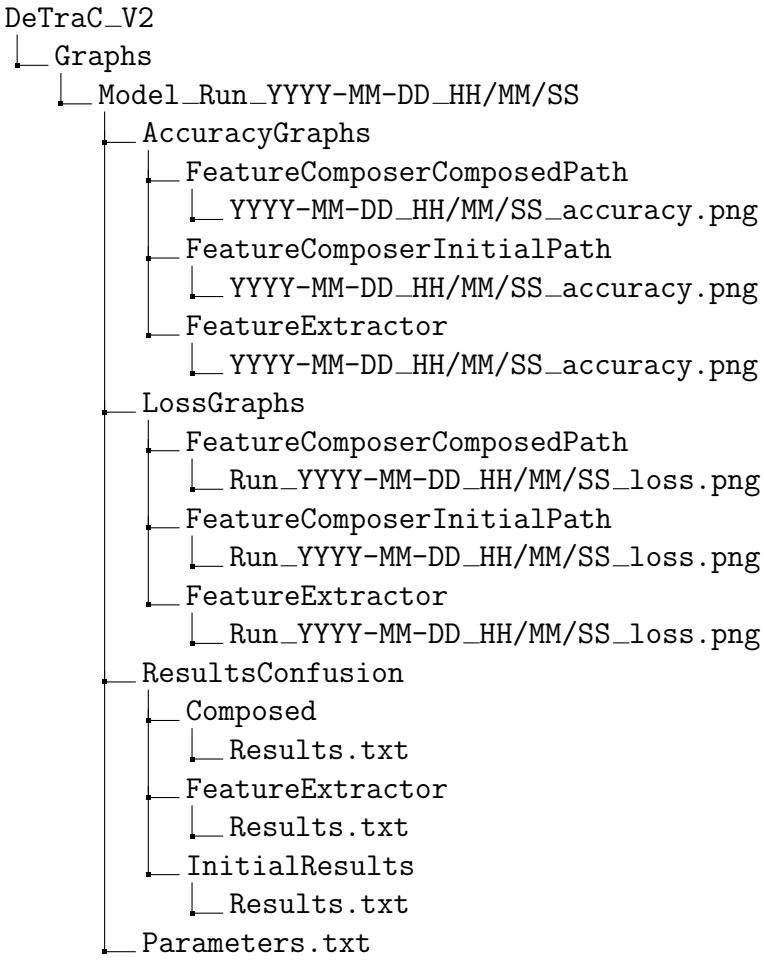

Figure 12: Example of tree folder structure of Graphs folder

3.3.6.4 Models

Once we have created a model that we are happy with, we can reuse it. Each model is saved locally under the folder Models. In there we have tensorflow and pythorch models from our previous runs. With this, we can replicate the results and reuse it in other scenarios.
Models are saved in the correct folder with the prefix of DeTraC, what model is (for example: feature_composer) and when it was generated. Example can be seen under the tree structure bellow.

```
DeTraC_V2
└── Models
    ├── tf
    │   └── DeTraC_feature_composer_2021-12-22_12-13-51.pth
    └── torch
        └── DeTraC_feature_composer_2021-12-22_12-13-51.pth
```

Figure 13: Example of tree folder structure of Models folder

4 Evaluation

4.1 Testing

For splitting the data, K-fold cross validation is used instead of regular train/test split. K-fold cross validation allows us to split data into the small sections (buckets) (Brownlee 2018). Each section is then spit again into train/test and desired algorithm is tested per each bucket. After all the sections have been covered, an average of each section is given back to us. With this, data is the most optimised in terms of split between train/test (Refaeilzadeh, Tang, and H. Liu 2009).
To implement K-fold cross validation, SkLearn K-Fold library was used (SkLearn n.d.[c]). Firstly we are passing features (as X), labels (as Y) and number of splits (as K) into our custom function. The function returns 4 parameters back (X_train, X_test, Y_train, Y_test).
In our custom function, data is being split into K sections (buckets) and shuffled as well. After that, we go into the for loop that does the split for train/test and saves the result into the parameters that are then passed back from the function.

4.2 Evaluation Methodology

The mostly used testing strategies in the industry are accuracy and precision. Testing with this is valid, but inclusion of F1 score or Recall is also a solid approach since all of the techniques are used in the industry (Brownlee 2019). Based on that, five different types of tests are being implemented:

- confusion matrix,
- accuracy,
- precision,
- recall,
- f1 score.

4.2.1 Confusion matrix

Confusion matrix is a technique for feature selection. It allows for mapping of different values from the dataset to the table for visualisation (AskPython n.d.). Confusion matrix is used (most often) in classification, attribute selection and k-nearest neighbors (Visa et al. 2011)
There are four elements confusion matrix represents:

- True Positives (TP),
- True Negatives (TN),

- False Positives (FP),
- False Negatives (FN).

Confusion matrix can represent these three (main) calculations:

- precision,
- recall,
- f1 score.

In this case, we are using module confusion matrix from SkLearn (SkLearn n.d.[a]). The data that is passed in the function of confusion matrix is y_true (confirmed true cases) and y_pred (predicted true cases). Function is then returning four elements (TP, TN, FP and FN).

4.2.2 Accuracy

Accuracy is a calculation that provides results of model overall. It is calculated by meaning the fraction of the total items that were marked correctly by classifier (Mohajon 2020).

$$\frac{TruePositives+TrueNegatives}{TruePositives+TrueNegatives+FalsePositives+FalseNegatives} = Accuracy$$

4.2.3 Precision

Precision is a ratio between items that are classified as True Positives divided from True Positives added with False Positives. The equation bellow (Davis and Goadrich 2006) explains in mathematical terms how calculation can is done.

$$\frac{TruePositives}{TruePositives+FalsePositives} = Precision$$

For calculating precision SkLearn module (precision_score) was used (SkLearn n.d.[e]) at first. Based on the y_true and y_pred, this module does calculation and returns precision. There are also different parameters for average can be passed into the function, based on the need:

- binary - for binary predictions
- micro - metrics of True Positives (TP), False Negatives (FN) and False Positives (FP) are calculated
- macro - metrics for each label are calculated and their unweighted mean is found. Do note that if there is imbalance (like in the real world) in labels, this is not taken in the consideration.

- weighted - metrics for each label are calculated and their average weight is found. This is done by looking the number of true instances for each label. Using this approach can cause F1 score not to be between precision and recall.

- samples - metrics is calculated for each instance and their average is found (this is used in multilabel classification).

Later on, another module was discovered from SkLearn with the name of precision_recall_fscore_support (SkLearn n.d.[d]). As the name suggests, this module calculates all three items we need in one go, with the same parameters above. Due to its simplicity, precision, recall and f1 are being calculated with this new module. Regardless, the sections about precision and recall do explain how we could calculate it individually.

4.2.4 Recall

This is a ratio between True Positive divided from sum of True Positives and False Negatives. In the equation bellow (Davis and Goadrich 2006) we can see mathematical formula for it.

$$\frac{TruePositives}{TruePositives+FalseNegatives} = Recall$$

To calculate recall, SkLearn provides another module with the name of recall_score (SkLearn n.d.[f]). We pass y_true and y_pred are passed into the function and recall is returned. Average can be selected same as above in precision.

4.2.5 F1 score

F1 score is a combination of precision and recall. It is calculated by multiplying by 2 ratio between precision multiplied by recall and sum of precision and recall. Equation for the math can be seen bellow (Lipton, Elkan, and Naryanaswamy 2014). Where F1 score is better is in imbalanced data. Due to the nature of raw real life data, F1 score attempts to "balanced out" differences in precision and recall to get a better representation.

$$2 * \frac{precision*recall}{precision+recall} = F1$$

Calculating F1 can be done with another SkLearn module with the name of f1_score (SkLearn n.d.[b]). In this case, we pass in the parameters of y_true and y_pred to the module. Parameters of average can be set the same way as in precision. The result that is returned from the module is F1 score.

4.2.6 Evaluation Metrics

- Number of epochs (num_epochs)
  - Specify number of epochs (iterations).
  - Expected type: positive integer (example: 20)
- Batch size (batch_size)
  - How many items are taken into the group (batch) at once.
  - Expected type: positive integer (example: 32)
- Number of classes for feature extractor (feature_extractor_num_classes)
  - Number of classes we have in our folder.
  - Expected type: positive integer (example: 3)
- Number of classes for feature composer (feature_composer_num_classes)
  - This is the same number as in feature extractor but doubled. Do note that if we are using initial dataset as input, parameter with this name will be used since it is only the name.
  - Expected type: positive integer (example: 6)
- How many folds we want (folds)
  - For K-fold cross validation we need to specify the number of folds (K) that we would like.
  - Expected type: positive integer (example: 10)
- Learning rate for feature extractor (feature_extractor_lr)
  - What is the learning rate for feature extractor.
  - Expected type: positive decimal (example: 0.001)
- Learning rate for feature composer(feature_composer_lr)
  - What is the learning rate for feature composer.
  - Expected type: positive decimal (example: 0.001)
- If we want to use Cuda (use_cuda)
  - If we are using Nvidia Cuda capable GPU for faster processing, enabling that would allow for faster processing
  - Expected type: boolean (example: True)
- Number for K (k)

  - Number for clusters in k-means clustering
  - Expected type: positive integer (example: 2)

- What type deep neural network model we want to use (modelType)
  - We can chose from different models of DNN that are being implemented (for example: VGG16, ResNet18, etc...)
  - Expected type: string that contains model name listed (example: ResNet18)

- What the momentum should be for the model (momentumValue)
  - The momentum model should have while it is operating
  - Expected type: positive decimal in the range between 0 and 0.9^{∞} (example: 0.60)

- What is the dropout value for the model (dropoutValue)
  - For improved performance, we can specify for DNN model to kill some connections with dropout. This can be specified here.
  - Expected type: decimal in the range between 0 and 0.9^{∞} (example: 0.60)

- Is data augmentation enabled (dataAugmentationEnabled)
  - If there is lack of data, we can generate new synthetic data by turning data augmentation on. This should improve model performance if there is any overfitting.
  - Expected type: boolean (example: True)

4.2.7 Baseline systems

Optimal (stock) settings for comparing with composed dataset:

- Number of epochs (num_epochs): 80
- Batch size (batch_size): 50
- Number of classes for feature extractor (feature_extractor_num_classes): 3
- Number of classes for feature composer (feature_composer_num_classes): 6
- Number of folds for k-fold cross validation (folds): 20
- Learning rate for feature extractor (feature_extractor_lr): 0.0001
- Learning rate for feature composer(feature_composer_lr): 0.0001
- If we want to use Cuda (use_cuda): True

- Number for K (k): 2
- Type of deep neural network model (modelType): resnet18
- Momentum for the model (momentumValue): 0.99
- Dropout value for the model (dropoutValue): 0.75
- Data augmentation enabled (dataAugmentationEnabled): False

4.2.8 Dataset

We are using 3 classes from that dataset:

- tumor,
- stroma,
- adipose.

As mentioned before, this project is not restricted by this dataset. In case of project reproduction, this can be used or tested with any other dataset (but parameters might need to be tuned slightly).
"Data" section is explaining how folders are structured and what is done to some extend. This section is going to add to this. Folder with the name of initial_dataset is root of where the data is. In this folder, we place our initial data separated with classes.
For the purpose of testing, we have used only 3 classes out of the 8 available in total. Each class has 625 images that we are dealing with. In case that the number of images is not enough for us, we have a function that deals with data augmentation. This function creates 801 new images that we can use. In the section of data augmentation, images are rotated, zoomed, flipped and brightness is changed. By creating new images model overfitting problems can be reduced (Bloice, Stocker, and Holzinger 2017).
After we execute the program, it is going to go into feature extractor to extract features from images. It will adjust images if needed, split the data (with k-fold cross validation) and normalise it. Next step is to build NN and fit the data in. After that, the confusion matrix is executed, the model is build and feature extraction is done which results in a new folder with the name of extracted_features. This folder contains features from each class as the numpy array. Based on the data in this folder, new folder is build with the name of composed_dataset. New folders are created (as 1_class1, 2_class1, etc...) as program does k-means clustering. With that, folder is full of clustered images.
Next step is to decide if we want to use feature composer (so use DeTraC original code) or use initial dataset (the modified DeTraC version). Method that is build, does accept both versions, the only difference is parameters (example: composed data path or initial path). If we pass in the composed data path, it is going to use

the clustered composed class that was created in the section above. After that, similar process is done on image processing (example: resize the image), k-fold cross validation, normalisation, building and testing the model.
Otherwise if we want to use initial dataset, we want to pass in the custom path. This folder has the same structure as composed dataset would, the only difference is that the images are split (based on the class) in sub-folders on random (as 1_class1, 2_class1, etc...). With that, we did not modify images, just split them presenting same structure as in composed folder while having same data as in initial folder. After the data is passed, same concept is used (for reproduction purposes) as if we would use feature composer path above (processing, k-fold, etc..).

4.3 Results

In this section, we are going to compare results that are used in composed version to the initial version. Number of test cases were executed in order to compare one version against the other. Do note that all of the test cases were executed in "deep tuning" mode for better accuracy. When test was stared, it was an in automatic mode, that meaning both composed dataset and initial dataset scenarios were tested at once.
Main test cases represented:

- Test case 1
  - Using same parameters as original paper to see how close the program is.
- Test case 2
  - Parameters are slightly tuned for representing baseline systems.
- Test case 3
  - Experimenting with smaller dataset.
- Test case 4
  - Using same dataset as Test case 3 but adding data augmentation to it.

4.3.1 Test case 1

This test case has the same parameters as original paper for comparison. The goal of the test is to establish a close reproduction point of the papers results and see what our results would be in comparison between composed dataset (original to the paper) and initial dataset. In this case, deep mode was used to train the model.
For the data, we have used 650 images for each class (mentioned on top), without

any synthetic data. As seen in the first graph representing validation accuracy for composed dataset, the plot seems a lot more analog and close to the median. The second graph is representing validation accuracy but for initial dataset. Seen from the graph, it is plotted in the same style as composed dataset but with results lower than composed dataset.

Variable names	Variable values
num_epochs	80
batch_size	50
feature_extractor_num_classes	3
feature_composer_num_classes	6
folds	10
feature_extractor_lr	0.0001
feature_composer_lr	0.001
use_cuda	True
k	2
modelType	ResNet18
momentumValue	0.9
dropoutValue	0.0

Table 7: Test case 1 variables

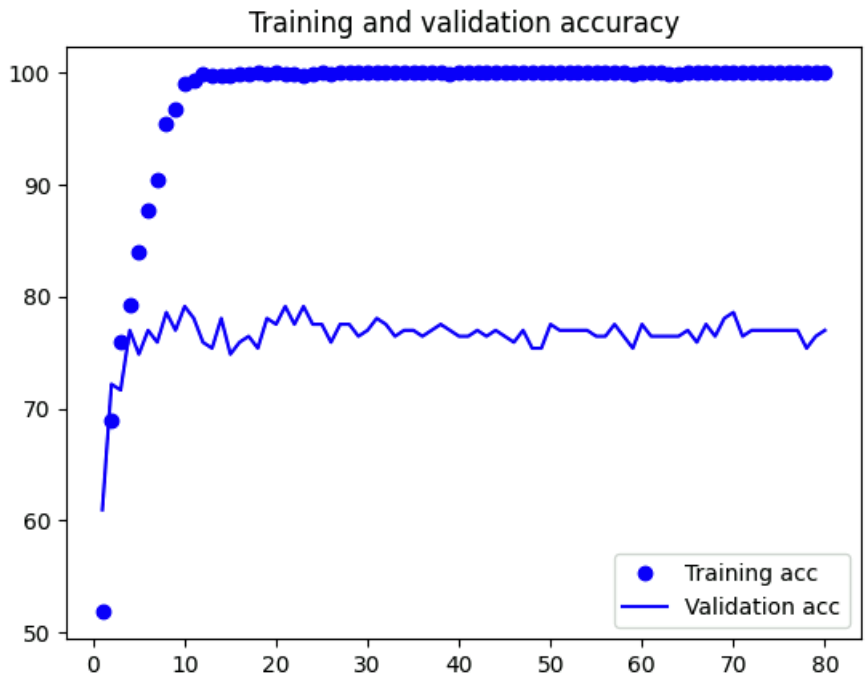

Figure 14: Test case 1 training and validation accuracy - composed dataset

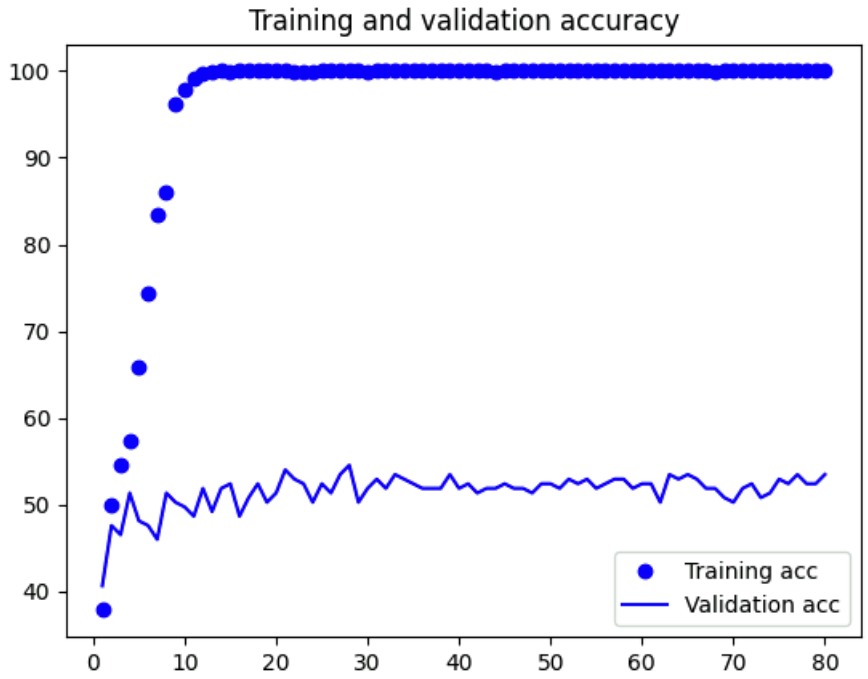

Figure 15: Test case 1 training and validation accuracy - initial dataset

After executing first test case, we can see results:

Testing names	Composed results	Initial results
Accuracy	0.16	0.24
Micro Precision	0.16	0.24
Micro Recall	0.16	0.24
Micro F1 score	0.16	0.24
Macro Precision	0.03	0.07
Macro Recall	0.17	0.19
Macro F1 score	0.04	0.10
Weighted Precision	0.02	0.09
Weighted Recall	0.16	0.24
Weighted F1 score	0.04	0.13

Table 8: Test case 1 results

```
Confusion Matrix
[[0.         0.         0.35828877]
 [0.         0.         0.31016043]
 [0.         0.         0.3315508 ]]
Accuracy: 0.16
Micro Precision: 0.16
Micro Recall: 0.16
Micro F1-score: 0.16
Macro Precision: 0.03
Macro Recall: 0.17
Macro F1-score: 0.04
Weighted Precision: 0.02
Weighted Recall: 0.16
Weighted F1-score: 0.04

Classification Report
              precision    recall  f1-score   support

  01_TUMOR_1       0.00      0.00      0.00        19
  01_TUMOR_2       0.00      0.00      0.00        48
 02_STROMA_1       0.00      0.00      0.00        31
 02_STROMA_2       0.00      0.00      0.00        27
07_ADIPOSE_1       0.00      0.00      0.00        33
07_ADIPOSE_2       0.16      1.00      0.27        29

    accuracy                           0.16       187
   macro avg       0.03      0.17      0.04       187
weighted avg       0.02      0.16      0.04       187
```

Figure 16: Test case 1 confusion matrix results - composed dataset

```
Confusion Matrix
[[0.         0.         0.29411765]
 [0.         0.         0.28877005]
 [0.         0.         0.4171123 ]]
Accuracy: 0.24
Micro Precision: 0.24
Micro Recall: 0.24
Micro F1-score: 0.24
Macro Precision: 0.07
Macro Recall: 0.19
Macro F1-score: 0.10
Weighted Precision: 0.09
Weighted Recall: 0.24
Weighted F1-score: 0.13

Classification Report
              precision    recall  f1-score   support

  01_TUMOR_1       0.00      0.00      0.00        27
  01_TUMOR_2       0.00      0.00      0.00        28
 02_STROMA_1       0.00      0.00      0.00        35
 02_STROMA_2       0.00      0.00      0.00        19
07_ADIPOSE_1       0.13      0.21      0.16        38
07_ADIPOSE_2       0.29      0.90      0.43        40

    accuracy                           0.24       187
   macro avg       0.07      0.19      0.10       187
weighted avg       0.09      0.24      0.13       187
```

Figure 17: Test case 1 confusion matrix results - initial dataset

4.3.2 Test case 2

Second test is showing results of much more intensive testing. The goal of this test is to get deeper information by testing the model a bit longer with heavier weights.
This test produced interesting results for comparison. First of all, if we have a look at training and validation accuracy graph of composed dataset, we can see that validation accuracy seems to be lower than expected but median of the plotted curve is a lot more centralised. If we compare that to initial dataset graph, we can see that validation accuracy is lower but plotted curve seem to have a bit more offset from its median in comparison to composed dataset.

Variable names	Variable values
num_epochs	100
batch_size	30
feature_extractor_num_classes	3
feature_composer_num_classes	6
folds	20
feature_extractor_lr	0.0001
feature_composer_lr	0.0001
use_cuda	True
k	2
modelType	ResNet18
momentumValue	0.99
dropoutValue	0.2

Table 9: Test case 2 variables

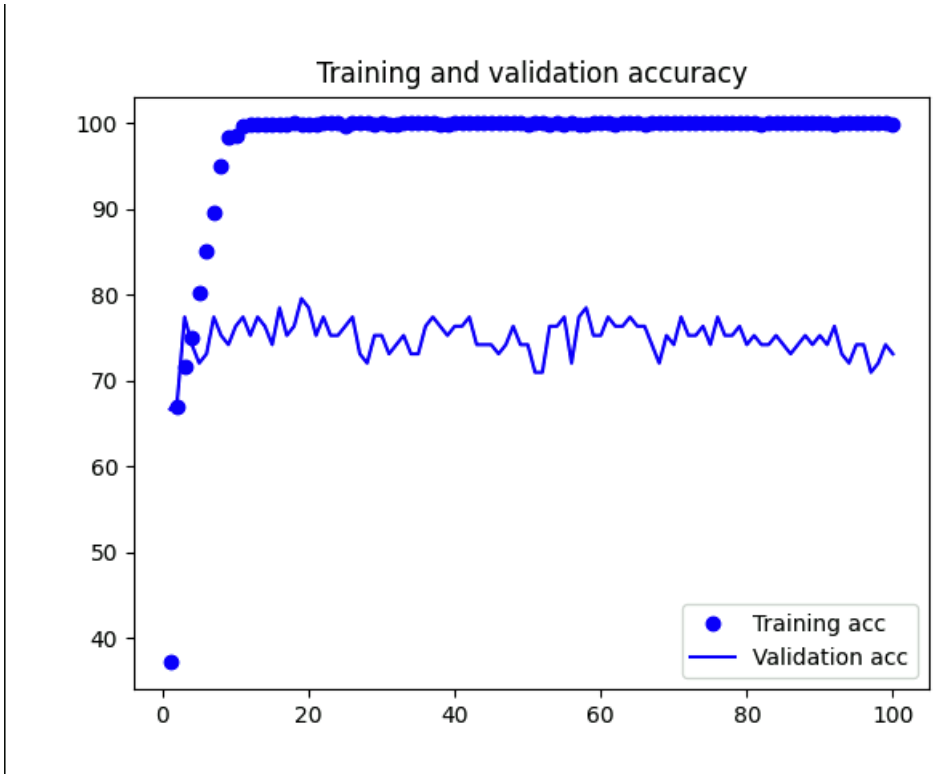

Figure 18: Test case 2 training and validation accuracy - composed dataset

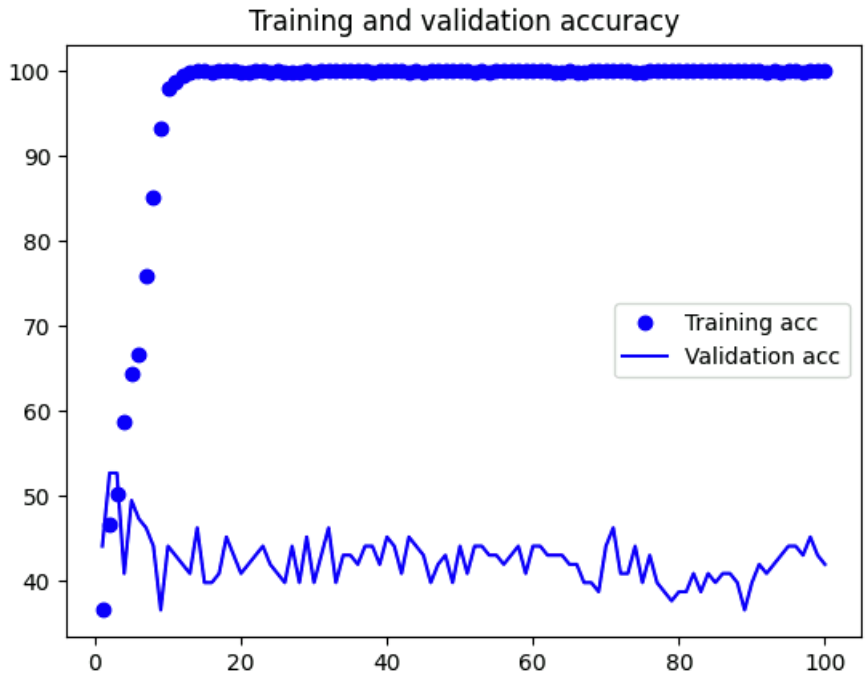

Figure 19: Test case 2 training and validation accuracy - initial dataset

After executing second test case, we can see results:

Testing names	Composed results	Initial results
Accuracy	0.08	0.16
Micro Precision	0.08	0.16
Micro Recall	0.08	0.16
Micro F1 score	0.08	0.16
Macro Precision	0.01	0.03
Macro Recall	0.17	0.17
Macro F1 score	0.02	0.05
Weighted Precision	0.01	0.03
Weighted Recall	0.08	0.16
Weighted F1 score	0.01	0.04

Table 10: Test case 2 results

```
Confusion Matrix
[[0.         0.         0.35483871]
 [0.         0.         0.31182796]
 [0.         0.         0.33333333]]
Accuracy: 0.08
Micro Precision: 0.08
Micro Recall: 0.08
Micro F1-score: 0.08
Macro Precision: 0.01
Macro Recall: 0.17
Macro F1-score: 0.02
Weighted Precision: 0.01
Weighted Recall: 0.08
Weighted F1-score: 0.01

Classification Report
              precision    recall  f1-score   support

  01_TUMOR_1       0.00      0.00      0.00        19
  01_TUMOR_2       0.00      0.00      0.00        14
 02_STROMA_1       0.00      0.00      0.00        20
 02_STROMA_2       0.00      0.00      0.00         9
07_ADIPOSE_1       0.08      1.00      0.14         7
07_ADIPOSE_2       0.00      0.00      0.00        24

    accuracy                           0.08        93
   macro avg       0.01      0.17      0.02        93
weighted avg       0.01      0.08      0.01        93
```

Figure 20: Test case 2 confusion matrix results - composed dataset

```
Confusion Matrix
[[0.         0.         0.35483871]
 [0.         0.         0.32258065]
 [0.         0.         0.32258065]]
Accuracy: 0.16
Micro Precision: 0.16
Micro Recall: 0.16
Micro F1-score: 0.16
Macro Precision: 0.03
Macro Recall: 0.17
Macro F1-score: 0.05
Weighted Precision: 0.03
Weighted Recall: 0.16
Weighted F1-score: 0.04

Classification Report
              precision    recall  f1-score   support

  01_TUMOR_1       0.00      0.00      0.00        18
  01_TUMOR_2       0.00      0.00      0.00        15
 02_STROMA_1       0.00      0.00      0.00        10
 02_STROMA_2       0.00      0.00      0.00        20
07_ADIPOSE_1       0.16      1.00      0.28        15
07_ADIPOSE_2       0.00      0.00      0.00        15

    accuracy                           0.16        93
   macro avg       0.03      0.17      0.05        93
weighted avg       0.03      0.16      0.04        93
```

Figure 21: Test case 2 confusion matrix results - initial dataset

4.3.3 Test case 3

This test is comparing composed dataset to initial dataset but on a smaller sample. Instead of 625 images for each run, we have only used 100 images per class. Test like this can give us information about overfitting and underfitting, so if quantity of our data makes any difference or not.
In the graph representing training and validation accuracy we can see that validation accuracy seems to be represented as digital signal. This is true for both cases (composed and initial). If we compare that to Test case 2 (which has more data), we can see the representation there being much more similar to analog signal. Overall, validation accuracy for composed dataset is in the area of expectancy. Initial dataset result is about 40% lower in comparison to composed datasets validation accuracy.

Variable names	Variable values
num_epochs	80
batch_size	50
feature_extractor_num_classes	3
feature_composer_num_classes	6
folds	20
feature_extractor_lr	0.0001
feature_composer_lr	0.001
use_cuda	True
k	2
modelType	ResNet18
momentumValue	0.99
dropoutValue	0.0

Table 11: Test case 3 variables

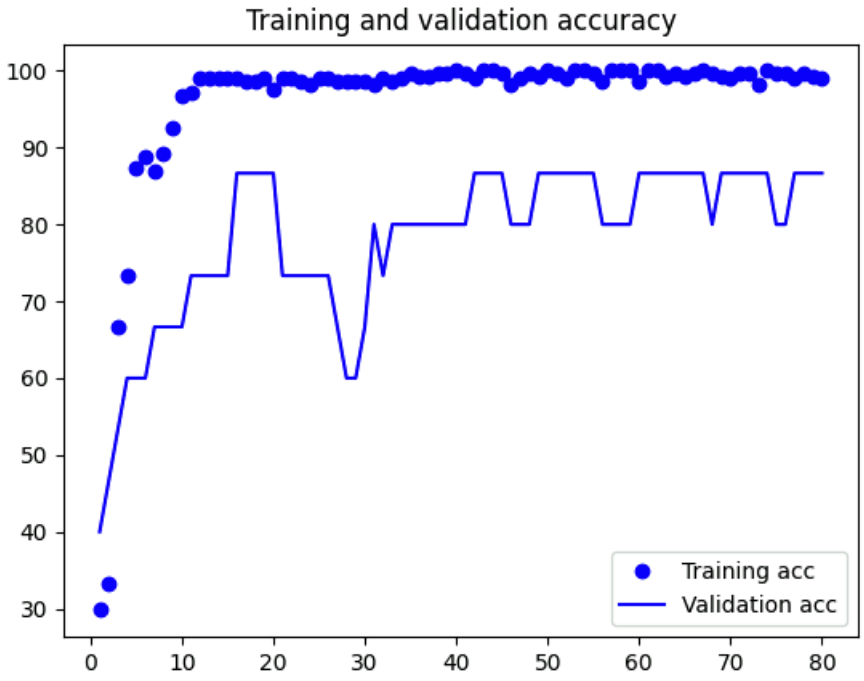

Figure 22: Test case 3 training and validation accuracy - composed dataset

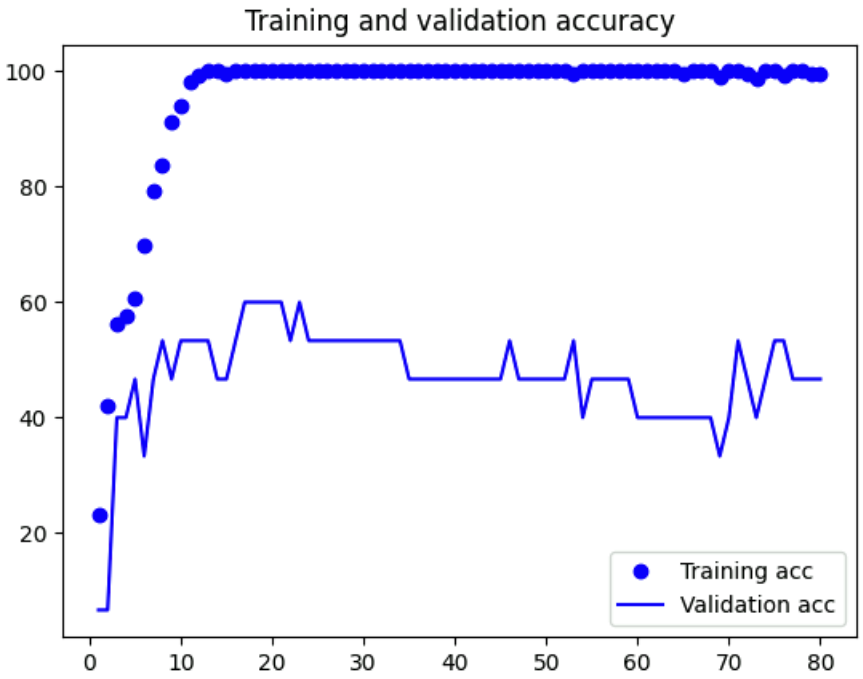

Figure 23: Test case 3 training and validation accuracy - initial dataset

After executing first test case, we can see results:

Testing names	Composed results	Initial results
Accuracy	0.40	0.20
Micro Precision	0.40	0.20
Micro Recall	0.40	0.20
Micro F1 score	0.40	0.20
Macro Precision	0.08	0.03
Macro Recall	0.20	0.17
Macro F1 score	0.11	0.06
Weighted Precision	0.16	0.04
Weighted Recall	0.40	0.20
Weighted F1 score	0.23	0.07

Table 12: Test case 3 results

```
Confusion Matrix
[[0.         0.         0.         0.         0.06666667]
 [0.         0.         0.         0.         0.26666667]
 [0.         0.         0.         0.         0.2       ]
 [0.         0.         0.         0.         0.06666667]
 [0.         0.         0.         0.         0.4       ]]
Accuracy: 0.40

Micro Precision: 0.40
Micro Recall: 0.40
Micro F1-score: 0.40

Macro Precision: 0.08
Macro Recall: 0.20
Macro F1-score: 0.11

Weighted Precision: 0.16
Weighted Recall: 0.40
Weighted F1-score: 0.23

Classification Report
              precision    recall  f1-score   support

           0       0.00      0.00      0.00         1
           1       0.00      0.00      0.00         4
           2       0.00      0.00      0.00         3
           3       0.00      0.00      0.00         1
           4       0.40      1.00      0.57         6

    accuracy                           0.40        15
   macro avg       0.08      0.20      0.11        15
weighted avg       0.16      0.40      0.23        15
```

Figure 24: Test case 3 confusion matrix results - composed dataset

```
Confusion Matrix
[[0.         0.         0.4       ]
 [0.         0.         0.33333333]
 [0.         0.         0.26666667]]
Accuracy: 0.20

Micro Precision: 0.20
Micro Recall: 0.20
Micro F1-score: 0.20

Macro Precision: 0.03
Macro Recall: 0.17
Macro F1-score: 0.06

Weighted Precision: 0.04
Weighted Recall: 0.20
Weighted F1-score: 0.07

Classification Report
              precision    recall  f1-score   support

           0       0.00      0.00      0.00         3
           1       0.00      0.00      0.00         3
           2       0.00      0.00      0.00         3
           3       0.00      0.00      0.00         2
           4       0.00      0.00      0.00         1
           5       0.20      1.00      0.33         3

    accuracy                           0.20        15
   macro avg       0.03      0.17      0.06        15
weighted avg       0.04      0.20      0.07        15
```

Figure 25: Test case 3 confusion matrix results - initial dataset

4.3.4 Test case 4

This test is about investing an effect of augmented data. In the Test case 3, we have had only 100 images which is not a lot. To combat the small dataset size, we can use data augmentation. So, in this dataset, we have used 100 images per class and augmented them. This gave us 730 images per class which in theory should result in the better accuracy.
Results that are represented in the graph for composed dataset bellow do show that more data indicates for validation accuracy to be displayed in analog style. The result is similar to Test case 2 while it differentiates from Test case 3. The only downside that this test case represents is lower accuracy. Compared to other test cases, this one if the lowest but still in the 70% range of validation accuracy. Due to increased sample size, we can see that in initial dataset, the plotted curve does look more analog. If we compare that to Test case 3, we can see that due to smaller data size the curve is much more digital. This gives us conformation that data augmentation has worked for this case.

Variable names	Variable values
num_epochs	80
batch_size	50
feature_extractor_num_classes	3
feature_composer_num_classes	6
folds	20
feature_extractor_lr	0.0001
feature_composer_lr	0.001
use_cuda	True
k	2
modelType	ResNet18
momentumValue	0.99
dropoutValue	0.0

Table 13: Test case 4 variables

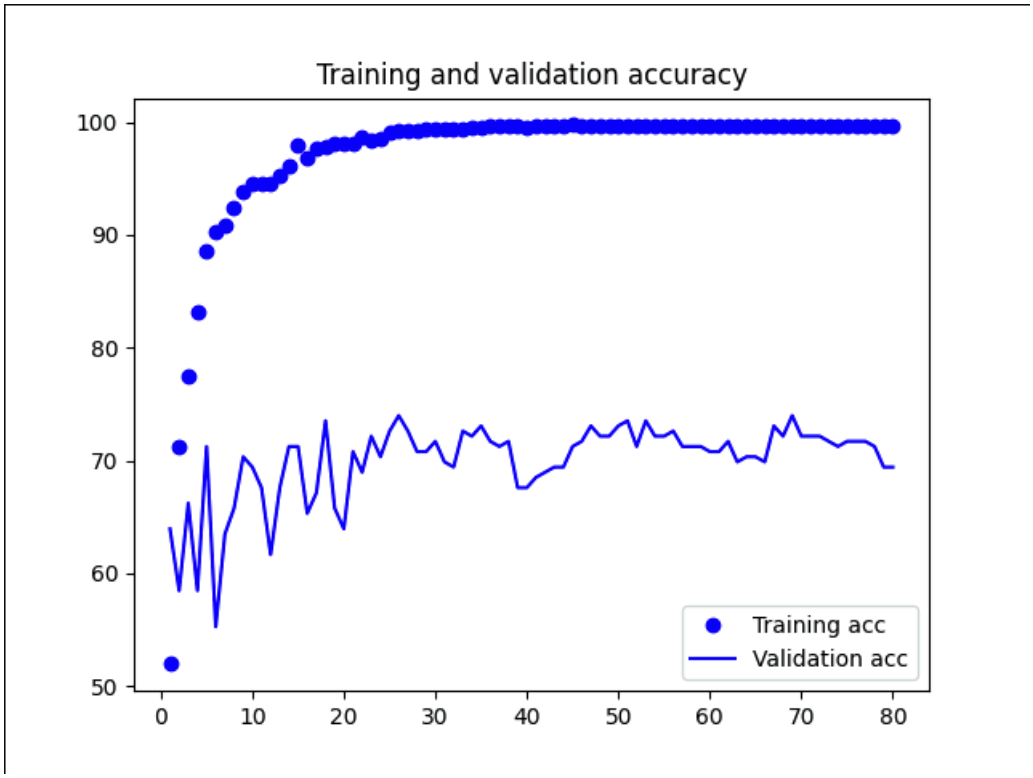

Figure 26: Test case 4 training and validation accuracy - composed dataset

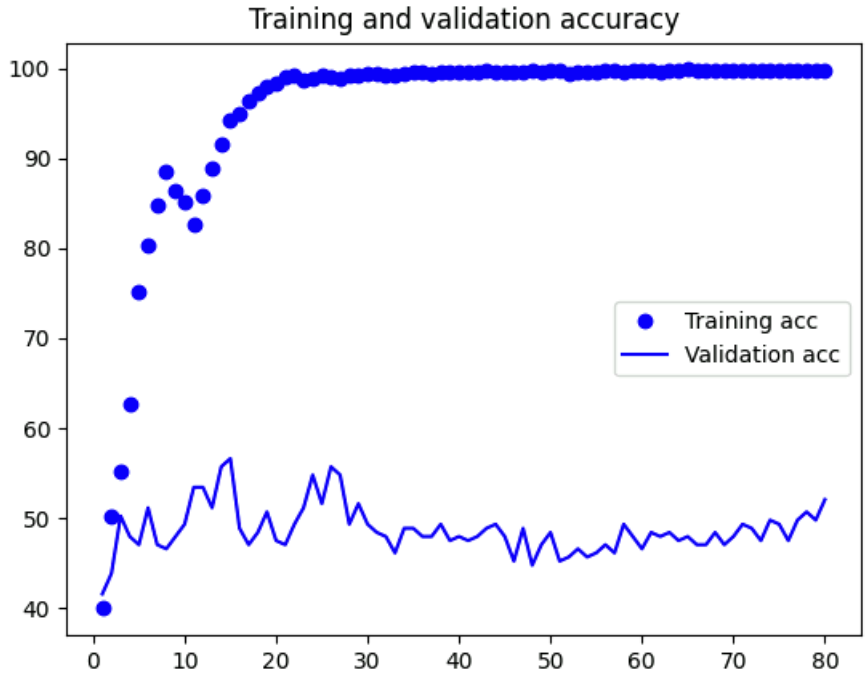

Figure 27: Test case 4 training and validation accuracy - initial dataset

After executing fourth test case, we can see results:

Testing names	Composed results	Initial results
Accuracy	0.23	0.18
Micro Precision	0.23	0.18
Micro Recall	0.23	0.18
Micro F1 score	0.23	0.18
Macro Precision	0.04	0.03
Macro Recall	0.17	0.17
Macro F1 score	0.06	0.05
Weighted Precision	0.05	0.03
Weighted Recall	0.23	0.18
Weighted F1 score	0.09	0.05

Table 14: Test case 4 results

```
Confusion Matrix
[[0.         0.         0.37442922]
 [0.         0.         0.26484018]
 [0.         0.         0.36073059]]
Accuracy: 0.23
Micro Precision: 0.23
Micro Recall: 0.23
Micro F1-score: 0.23
Macro Precision: 0.04
Macro Recall: 0.17
Macro F1-score: 0.06
Weighted Precision: 0.05
Weighted Recall: 0.23
Weighted F1-score: 0.09

Classification Report
              precision    recall  f1-score   support

  01_TUMOR_1       0.00      0.00      0.00        45
  01_TUMOR_2       0.00      0.00      0.00        37
 02_STROMA_1       0.00      0.00      0.00        21
 02_STROMA_2       0.00      0.00      0.00        37
07_ADIPOSE_1       0.00      0.00      0.00        28
07_ADIPOSE_2       0.23      1.00      0.38        51

    accuracy                           0.23       219
   macro avg       0.04      0.17      0.06       219
weighted avg       0.05      0.23      0.09       219
```

Figure 28: Test case 4 confusion matrix results - composed dataset

```
Confusion Matrix
[[0.         0.         0.34246575]
 [0.         0.         0.33789954]
 [0.         0.         0.3196347 ]]
Accuracy: 0.18
Micro Precision: 0.18
Micro Recall: 0.18
Micro F1-score: 0.18
Macro Precision: 0.03
Macro Recall: 0.17
Macro F1-score: 0.05
Weighted Precision: 0.03
Weighted Recall: 0.18
Weighted F1-score: 0.05

Classification Report
              precision    recall  f1-score   support

  01_TUMOR_1       0.00      0.00      0.00        44
  01_TUMOR_2       0.00      0.00      0.00        31
 02_STROMA_1       0.00      0.00      0.00        37
 02_STROMA_2       0.00      0.00      0.00        37
07_ADIPOSE_1       0.18      1.00      0.30        39
07_ADIPOSE_2       0.00      0.00      0.00        31

    accuracy                           0.18       219
   macro avg       0.03      0.17      0.05       219
weighted avg       0.03      0.18      0.05       219
```

Figure 29: Test case 4 confusion matrix results - initial dataset

4.4 Discussion

Comparing Test case 1 to Test case 2, we can see that there is a slight difference in the results. Overall, comparing composed dataset Test case 1 graph to composed dataset Test case 2 graph, they are almost identical. One big difference we can easily spot is that graph plotted for validation accuracy seems to be noisier in Test case 2.
The biggest difference is in the initial dataset graphs. In the Test case 1, we can see line plotting in the 50% range and up for validation accuracy. Comparing that to Test case 2, validation accuracy is noisier and seems to be decreasing. Hyper parameters for Test case 1 and Test case 2 are similar, but there are three key differences that seem to drive the behaviour in this way. In Test case 1, we have bigger batch size of 50 (compared to batch size of 30) and dropout is disabled (while it is enabled in Test case 2). Learning rate for feature composer (and initial dataset) is also increased in Test case 1. Combination of these parameters seems to be inefficient for our model.

If we directly compare results from Test case 2 and Test case 3, something interesting can be seen. Data in Test case 2 is 650 items per class while Test case 3 is only 100 items per class. Despite the data difference, the difference in validation

accuracy between initial and composed class is similar. We are using different hyper parameters (weights) for each test, that is resulting for difference in validation accuracy over all. But regardless in comparison of the two tests, offset between composed and initial dataset is still similar.

Initial dataset results seem to be similar. If we compare validation accuracy graph from Test case 4 to Test case 3, we can see that because of bigger data volume (730 compared to 100), the plotted results are not as digital. Plotted results of Test case 4 seem to be in the in the same way as in Test case 2 which is reasonable due to number of images being a lot closer. If we have a look at what validation accuracy presents compared across the three test cases, we can see that in Test case 4, it is in the range of 50%. Test case 3 is in the similar range and Test case 2 is in the 40% range. This can confirm our idea that the composed dataset is similar across the board.

Concluding the test cases, we can see that researchers over the original DeTraC model have made progress in the right sections. If this model would be performing better with the initial dataset compared to the composed, that would result in different questions.
Overall, the results seem to behave in the similar way in general:
If validation accuracy is improved for composed dataset, it is going going to be improved for initial dataset as well. If validation accuracy decreases for composed dataset, initial dataset will have decrease as well. The difference of results from validation accuracy between composed dataset and initial dataset seems to be in similar range across all the test cases.

5 Conclusions

At the start of the report we asked if modifying original DeTraC code is possible to the extend to use initial dataset instead of the composed dataset. In case if it is, what are performance results?
Originally, DeTraC was using decomposition → transfer learning → composed dataset to achieve high accuracy. To test, if we can replace last step with initial dataset, some modifications were crucial. At the end, it was a success, you can use initial dataset in the last step instead of the composed one, but there is a cost associated with it. It turns out that last step (composed dataset) allows accuracy to be much higher in comparison with initial dataset.
In regards to the original paper, this proves that researchers developing DeTraC have achieved the goal of high accuracy without using initial dataset.

6 Recommendations for future work

One of the things that could be added in in the future is testing of ResNext. As mentioned in literature review, ResNext is deeper and "newer" version of ResNet. For this project, testing was tried but it has failed and results of ResNext could not be added. Based on the technology ResNet is build on, assumption can be made that there should be no significant difference (as with VGG16 compared to ResNet18). But those are only assumptions and if there is difference, it would be interesting if it can be proven right or wrong.
Another recommendation that can be made is to use MLOPS concepts. If the project would start with MLOPS concepts in mind, we could track the progress easier and see where we have gotten and how we gotten there.

7 References

References

Abbas, Asmaa, Mohammed M. Abdelsamea, and Mohamed Medhat Gaber (2020). "DeTrac: Transfer Learning of Class Decomposed Medical Images in Convolutional Neural Networks". In: *IEEE Access* 8, pp. 74901–74913. DOI: 10.1109/ACCESS.2020.2989273.

Akiba, Takuya, Shuji Suzuki, and Keisuke Fukuda (2017). "Extremely large minibatch sgd: Training resnet-50 on imagenet in 15 minutes". In: *arXiv preprint arXiv:1711.04325*.

Alla, Sridhar and Suman Kalyan Adari (2021). "What is mlops?" In: *Beginning MLOps with MLFlow*. Springer, pp. 79–124.

Allen-Zhu, Zeyuan and Yuanzhi Li (2019). "What can resnet learn efficiently, going beyond kernels?" In: *Advances in Neural Information Processing Systems* 32.

Arsa, Dewa Made Sri and Anak Agung Ngurah Hary Susila (2019). "VGG16 in batik classification based on random forest". In: *2019 International Conference on Information Management and Technology (ICIMTech)*. Vol. 1. IEEE, pp. 295–299.

AskPython (n.d.). *Precision and Recall in Python*. Last accessed 16 April 2022. URL: https://www.askpython.com/python/examples/precision-and-recall-in-python.

Asmaa, Abbas (2020). *DeTraC COVId19*. Last accessed 20 March 2022. URL: https://github.com/asmaa4may/DeTraC_COVId19.

asmaa4may (n.d.). *DeTraC GitHub source code*. Last accessed 23 April 2022. URL: https://github.com/asmaa4may/DeTraC_COVId19.

Bloice, Marcus D, Christof Stocker, and Andreas Holzinger (2017). "Augmentor: an image augmentation library for machine learning". In: *arXiv preprint arXiv:1708.04680*.

Bozinovski, Stevo (2020). "Reminder of the first paper on transfer learning in neural networks, 1976". In: *Informatica* 44.3.

Brownlee, Jason (2018). *A Gentle Introduction to k-fold Cross-Validation*. Last accessed 15 April 2022. URL: https://machinelearningmastery.com/k-fold-cross-validation/.

— (2019). *How to Calculate Precision, Recall, F1, and More for Deep Learning Models*. Last accessed 19 March 2022. URL: https://machinelearningmastery.com/how-to-calculate-precision-recall-f1-and-more-for-deep-learning-models/.

Chen, Zhuxin et al. (2017). "ResNet and Model Fusion for Automatic Spoofing Detection." In: *Interspeech*, pp. 102–106.

Commons, Creative (n.d.). *CC licence information*. Last accessed 19 March 2022. URL: https://creativecommons.org/licenses/by/4.0/legalcode.

Davis, Jesse and Mark Goadrich (2006). "The relationship between Precision-Recall and ROC curves". In: *Proceedings of the 23rd international conference on Machine learning*, pp. 233–240.

Dechter, Rina and Judea Pearl (1988). "Network-based heuristics for constraint-satisfaction problems". In: *Search in artificial intelligence*. Springer, pp. 370–425.

Farooq, Muhammad and Abdul Hafeez (2020). "Covid-resnet: A deep learning framework for screening of covid19 from radiographs". In: *arXiv preprint arXiv:2003.14395*.

Fitzpatrick, David (1996). "The functional organization of local circuits in visual cortex: insights from the study of tree shrew striate cortex". In: *Cerebral cortex* 6.3, pp. 329–341.

Forsyth, David et al. (1991). "Invariant descriptors for 3 d object recognition and pose". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 13.10, pp. 971–991.

Fukushima, Kunihiko and Sei Miyake (1982). "Neocognitron: A self-organizing neural network model for a mechanism of visual pattern recognition". In: *Competition and cooperation in neural nets*. Springer, pp. 267–285.

He, Kaiming et al. (2016). "Deep residual learning for image recognition". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778.

Hebb, Donald Olding and Wilder Penfield (1940). "Human behavior after extensive bilateral removal from the frontal lobes". In: *Archives of Neurology & Psychiatry* 44.2, pp. 421–438.

Hinton, Geoffrey E and Ruslan R Salakhutdinov (2006). "Reducing the dimensionality of data with neural networks". In: *science* 313.5786, pp. 504–507.

Hridayami, Praba, I Ketut Gede Darma Putra, and Kadek Suar Wibawa (2019). "Fish species recognition using VGG16 deep convolutional neural network". In: *Journal of Computing Science and Engineering* 13.3, pp. 124–130.

Hubel, David H and Torsten N Wiesel (1959). "Receptive fields of single neurones in the cat's striate cortex". In: *The Journal of physiology* 148.3, p. 574.

Huber, Peter J, Lucien Marie Le Cam, and Jerzy Neyman (1967). *Proceedings of the fifth Berkeley symposium on mathematical statistics and probability*.

Ivakhnenko, A. G. (1967). *Cybernetics and forecasting techniques*. Vol. 8. American Elsevier Publishing Company.

Kather, Jakob Nikolas et al. (May 2016). *Collection of textures in colorectal cancer histology*. Zenodo. DOI: 10.5281/zenodo.53169. URL: https://doi.org/10.5281/zenodo.53169.

Koné, Ismaël and Lahsen Boulmane (2018). "Hierarchical ResNeXt models for breast cancer histology image classification". In: *International Conference Image Analysis and Recognition*. Springer, pp. 796–803.

Krishnaswamy Rangarajan, Aravind and Raja Purushothaman (2020). "Disease classification in eggplant using pre-trained VGG16 and MSVM". In: *Scientific reports* 10.1, pp. 1–11.

LeCun, Yann et al. (1989). "Backpropagation applied to handwritten zip code recognition". In: *Neural computation* 1.4, pp. 541–551.

Li, Sihan et al. (2016). "Demystifying resnet". In: *arXiv preprint arXiv:1611.01186*.

Lin, Hongzhou and Stefanie Jegelka (2018). "Resnet with one-neuron hidden layers is a universal approximator". In: *Advances in neural information processing systems* 31.

Lipton, Zachary C, Charles Elkan, and Balakrishnan Naryanaswamy (2014). "Optimal thresholding of classifiers to maximize F1 measure". In: *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*. Springer, pp. 225–239.

Liu, Bin et al. (2017). "Weld defect images classification with vgg16-based neural network". In: *International Forum on Digital TV and Wireless Multimedia Communications*. Springer, pp. 215–223.

Liu, Zhihao et al. (2019). "Improved kiwifruit detection using pre-trained VGG16 with RGB and NIR information fusion". In: *IEEE Access* 8, pp. 2327–2336.

Lloyd, Stuart (1982). "Least squares quantization in PCM". In: *IEEE transactions on information theory* 28.2, pp. 129–137.

McCulloch, Warren S and Walter Pitts (1943). "A logical calculus of the ideas immanent in nervous activity". In: *The bulletin of mathematical biophysics* 5.4, pp. 115–133.

Microsoft (2021). *About WSL*. Last accessed 23 April 2022. URL: https://docs.microsoft.com/en-us/windows/wsl/about.

Mohajon, Joydwip (2020). *Confusion Matrix for Your Multi-Class Machine Learning Model*. Last accessed 23 April 2022. URL: https://towardsdatascience.com/confusion-matrix-for-your-multi-class-machine-learning-model-ff9aa3bf7826.

Ng, Andrew (2016). "Nuts and bolts of building AI applications using Deep Learning". In: *NIPS Keynote Talk*.

Nvidia (n.d.). *Nvidia Cuda technology*. Last accessed 16 April 2022. URL: https://www.nvidia.com/en-gb/geforce/technologies/cuda/.

Oh, Kyoung-Su and Keechul Jung (2004). "GPU implementation of neural networks". In: *Pattern Recognition* 37.6, pp. 1311–1314.

organisation, Git (n.d.). *About git*. Last accessed 23 April 2022. URL: https://git-scm.com/about/free-and-open-source.

Orhan, A Emin (2019). "Robustness properties of Facebook's ResNeXt WSL models". In: *arXiv preprint arXiv:1907.07640*.

Pant, Gaurav, DP Yadav, and Ashish Gaur (2020). "ResNeXt convolution neural network topology-based deep learning model for identification and classification of Pediastrum". In: *Algal Research* 48, p. 101932.

Poth, Alexander, Mark Werner, and Xinyan Lei (2018). "How to deliver faster with CI/CD integrated testing services?" In: *European Conference on Software Process Improvement*. Springer, pp. 401–409.

Pratt, Lorien Y (1992). "Discriminability-based transfer between neural networks". In: *Advances in neural information processing systems* 5.

Proulx, Eliane et al. (2014). "Nicotinic acetylcholine receptors in attention circuitry: the role of layer VI neurons of prefrontal cortex". In: *Cellular and Molecular Life Sciences* 71.7, pp. 1225–1244.

Qassim, Hussam, Abhishek Verma, and David Feinzimer (2018). "Compressed residual-VGG16 CNN model for big data places image recognition". In: *2018 IEEE 8th Annual Computing and Communication Workshop and Conference (CCWC)*. IEEE, pp. 169–175.

Refaeilzadeh, Payam, Lei Tang, and Huan Liu (2009). "Cross-validation." In: *Encyclopedia of database systems* 5, pp. 532–538.

Rezende, Edmar et al. (2018). "Malicious software classification using VGG16 deep neural network's bottleneck features". In: *Information Technology-New Generations*. Springer, pp. 51–59.

Rosenblatt, Frank (1961). *Principles of neurodynamics. perceptrons and the theory of brain mechanisms*. Tech. rep. Cornell Aeronautical Lab Inc Buffalo NY.

Rumelhart, David E, Geoffrey E Hinton, James L McClelland, et al. (1986). "A general framework for parallel distributed processing". In: *Parallel distributed processing: Explorations in the microstructure of cognition* 1.45-76, p. 26.

Schmidhuber, Jürgen (1992). "Learning complex, extended sequences using the principle of history compression". In: *Neural Computation* 4.2, pp. 234–242.

Sharma, Akash and Sunil Kumar Muttoo (2018). "Spatial image steganalysis based on resnext". In: *2018 IEEE 18th International Conference on Communication Technology (ICCT)*. IEEE, pp. 1213–1216.

Simonyan, Karen and Andrew Zisserman (2014). "Very deep convolutional networks for large-scale image recognition". In: *arXiv preprint arXiv:1409.1556*.

SkLearn (n.d.[a]). *Confusion matrix*. Last accessed 16 April 2022. URL: https://scikit-learn.org/stable/modules/generated/sklearn.metrics.confusion_matrix.html.

— (n.d.[b]). *F1 score*. Last accessed 16 April 2022. URL: https://scikit-learn.org/stable/modules/generated/sklearn.metrics.f1_score.html.

— (n.d.[c]). *Model selection KFold*. Last accessed 15 April 2022. URL: https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.KFold.html.

— (n.d.[d]). *Precision recall fscore support*. Last accessed 16 April 2022. URL: https://scikit-learn.org/stable/modules/generated/sklearn.metrics.precision_recall_fscore_support.html.

— (n.d.[e]). *Precision score*. Last accessed 16 April 2022. URL: https://scikit-learn.org/stable/modules/generated/sklearn.metrics.precision_score.html.

— (n.d.[f]). *Recall score*. Last accessed 16 April 2022. URL: https://scikit-learn.org/stable/modules/generated/sklearn.metrics.recall_score.html.

Spinellis, Diomidis (2012). "Git". In: *IEEE software* 29.3, pp. 100–101.

Srivastava, Rupesh Kumar, Klaus Greff, and Jürgen Schmidhuber (2015). "Highway networks". In: *arXiv preprint arXiv:1505.00387*.

Szegedy, Christian et al. (2017). "Inception-v4, inception-resnet and the impact of residual connections on learning". In: *Thirty-first AAAI conference on artificial intelligence*.

Tappert, Charles C (2019). "Who is the father of deep learning?" In: *2019 International Conference on Computational Science and Computational Intelligence (CSCI)*. IEEE, pp. 343–348.

Targ, Sasha, Diogo Almeida, and Kevin Lyman (2016). "Resnet in resnet: Generalizing residual architectures". In: *arXiv preprint arXiv:1603.08029*.

Theckedath, Dhananjay and RR Sedamkar (2020). "Detecting affect states using VGG16, ResNet50 and SE-ResNet50 networks". In: *SN Computer Science* 1.2, pp. 1–7.

Thomson, Alex M (2010). "Neocortical layer 6, a review". In: *Frontiers in neuroanatomy* 4, p. 13.

Thrun, Sebastian and Lorien Pratt (1998). "Learning to learn: Introduction and overview". In: *Learning to learn*. Springer, pp. 3–17.

Visa, Sofia et al. (2011). "Confusion matrix-based feature selection." In: *MAICS* 710, pp. 120–127.

Werbos, PJ (1975). "Experimental implications of the reinterpretation of quantum mechanics". In: *Il Nuovo Cimento B (1971-1996)* 29.1, pp. 169–177.

Xiao, Jian et al. (2020). "Application of a novel and improved VGG-19 network in the detection of workers wearing masks". In: *Journal of Physics: Conference Series*. Vol. 1518. 1. IOP Publishing, p. 012041.

Xie, Saining et al. (2017). "Aggregated residual transformations for deep neural networks". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 1492–1500.

— (2020). *ResNeXt code*. Last accessed 14 March 2022. URL: https://github.com/facebookresearch/ResNeXt.

Yang, Haoyan et al. (2021). "A novel method for peanut variety identification and classification by Improved VGG16". In: *Scientific Reports* 11.1, pp. 1–17.

Zhao, Defang et al. (2018). "Synthetic medical images using F&BGAN for improved lung nodules classification by multi-scale VGG16". In: *Symmetry* 10.10, p. 519.

Zisserman, Andrew, Peter Giblin, and Andrew Blake (1989). "The information available to a moving observer from specularities". In: *Image and vision computing* 7.1, pp. 38–42.

8 Appendices

8.1 Appendix A: Git

Git is a versioning platform used for developers to keep the track of code growth. It is used in industry for collaborative team work and in open source community to share their code with the rest of the world as well (Spinellis 2012).
It was originally developed by Linus Torvalds for Linux (open source operating system), therefore Git itself is open source and free to use by anyone under GNU General Public License version 2.0 (organisation n.d.).
Due to open source of Git, and its uses, a lot of companies started to host it. Some of the top companies are:

- GitHub - most well known, specially in the open source world, now owned by Microsoft
- GitLab - popular in the industry
- BitBucket - similar to GitLab

8.2 Appendix B: WSL

WSL stands for Windows Subsystem for Linux. The role WSL takes is, it uses native Windows functions and it passes it to Linux directly. This is different than virtual machine (VM), since VMs are isolated from host (or base) operating system, therefore requiring a lot more resources. With WSL, developers can develop applications/programs for Linux in Windows. One great example is multi compatibility of the programs. If program is required to run on Windows and Linux, developer can develop for both operating systems at the same time and test it simultaneously (Microsoft 2021).

8.3 Appendix C: CI/CD

CI/CD stands for continues integration / continues development and is part of devops (development operations) tools. The goal of CI/CD is to create a workflow where code of the developer is tested automatically on the push to the Git branch. Example of CI/CD would be that we have two developers working on a project. Developer 1 is responsible for frontend development and developer 2 is responsible for backend development. To see if the code is compatible, they (or devops team) can spend a day setting up CI/CD so upon git submission automatic test is done letting them know how compatible frontend is with the backend. This results in financial and time savings since at the end of the project, they know if the code is compatible or not rather to be surprised to see code not working and spend more developer time redoing code (Poth, Werner, and Lei 2018).

8.4 Appendix D: MLOPS

MLOPS stands for machine learning operations and is part of typical ML pipeline. It originates from devops and the goals it tries to solve are:

- faster experimentation and model development,
- faster deployment of the models,
- to see if quality of the model is up to the standard.

With the help of MLOPS, ML engineers and data scientists work together for faster development. MLOPS also provides versioning to see how model performed with different parameters (Alla and Adari 2021).