DeepTumour: Deep Learning for the detection of lung tumours in Computed Tomography scans.

UNIVERSITY OF LINCOLN

SCHOOL OF COMPUTER SCIENCE

Joseph David Pitts
PIT16657094

16657094@students.lincoln.ac.uk

School of Computer Science

College of Science

University of Lincoln

Submitted in partial fulfilment of the requirements for the Degree of BSc(Hons) Computer Science

Supervisor Dr. Vassilis Cutsuridis

May 2021

Word Count: 16336

Acknowledgements

I want to thank my supervisor, Dr. Vassilis Cutsuridis, for his help and assistance throughout this Project. A special thanks to Matthew Hollingworth for his inspiration. All the love to Hannah Moore for being there with me through the highs and the lows. And finally, to my friends, family and housemates thank you for your constant support.

Abstract

Cancer is a horrible disease that currently has no cure, as such the detection of cancer as early as possible is vital for successful treatment. For years researchers have tried developing systems to find tumours in scans. Early attempts made use of image processing techniques to locate tumours, these systems had some success but often only worked if conditions were perfect. The first machine learning techniques such as Bayesian networks showed promise but lacked accuracy. Deep learning, the latest in machine learning research, promises to make up for the lack of accuracy of previous methods. DeepTumour, the system put forward in this work, is a deep learning network using the Mask-RCNN architecture. The model is trained on the LIDC-IDRI dataset and achieves an F1 score of 0.8034 on three of the four classes identified. This work shows that deep learning can be successfully used for medical imaging and has the potential to save lives through fast and accurate tumour detection.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

Lung cancer is the “leading cause of cancer death among men and the second leading cause of cancer death among women worldwide” (Torre et al. 2016). It not only accounts for 1.59 million deaths per year but also has one of the poorest survival outcomes of all cancers (del Ciello et al. 2017). However, if tumours can be caught in the early stages of the disease they can often be treated with surgery, whereas advanced cases are most often treated with chemotherapy (Islam et al. 2015). Furthermore, The International Early Lung Cancer Action Program Investigators found that the “10-year survival rate was 88%” (Investigators 2006) for patients with stage 1 lung cancer. This all indicates that the sooner the cancer is found the more effective the treatment is and the higher chance it’s survivable.

A radiologist is needed to review the computed tomography (CT) scan of each patient and locate any tumours. Unfortunately, in some instances tumours can be missed owing to “observer error (scanning, recognition, and decision-making error), specific characteristics of the undetected lesion (size, conspicuity, and location), or technical inaccuracies” (del Ciello et al. 2017). Missed tumours can lead to missed cancer ultimately resulting in a potentially preventable death. A Machine Learning model could limit the impact of observer error as it is not susceptible to the same errors a Human makes.

Deep learning is a subset of machine learning that is better at “solving the tasks that are easy for people to perform but hard for people to describe formally—problems that we solve intuitively, that feel automatic, like recognizing spoken words or faces in images” (Goodfellow et al. 2016). This makes it an ideal candidate for finding tumours on a lung CT scan, locating tumours is a skill that humans can do with a little training. Large tumours can even be obvious to those completely untrained. As such a deep learning model is well suited to this problem.

Deep learning works by having a deep, multi-layered neural network that is trained on huge datasets. The network makes small tweaks to itself over the training period in an attempt to increase performance.

This paper discusses the creation of DeepTumour, a deep-learning model to locate tumours on lung CT scans with the aim to reduce the number of errors in diagnosis by highlighting potential tumours. DeepTumour could also decrease the time taken by a radiologist to review a CT scan allowing them to view more in the same amount of time, resulting in quicker diagnosis and therefore quicker treatment.

1.1 Aim & Objectives

The overall aim of this project is to show that deep learning can be used to assist in the detection of lung tumours in CT scans.

1. **Objective One: Dataset Acquisition.**
    - Dataset should have a sufficient number of scans for training and testing.
    - The scans must be Computed Tomography (CT) scans not PET.
2. **Objective Two: Dataset Pre-Processing.**
    - The scans must be split into individual images.
    - Any unnecessary detail should be removed.
3. **Objective Three: Model Evaluation.**
    - The model should be trained to a high degree ($F_1 \geq 0.7$).
    - The model must have few false negatives.
    - The model should minimise false positives.
    - The model should be able to identify all types of tumour.

Chapter 2

Literature Review

2.1 Background of Deep Learning

Medical image analysis is a field that has captivated researchers for years. Early attempts used expert systems, cascades of if-else statements and rule-based image processing techniques, i.e. edge detection filters and mathematical modelling. These early systems, however, "were often brittle" (Litjens et al. 2017).

By the 1990s supervised techniques started being used (Litjens et al. 2017). Supervised machine learning is still in heavy use today, being employed in "SVMs [Support Vector Machines], neural nets, logistic regression, naive bayes, memory-based learning, random forests, decision trees, bagged trees, boosted trees, and boosted stumps" (Caruana & Niculescu-Mizil 2006). A review of supervised models found that some models such as support vector machines (SVMs) can "achieve excellent performance that would have been difficult to obtain just 15 years ago" (Caruana & Niculescu-Mizil 2006). The shift from systems that are completely designed by humans to systems that are trained by computers was the catalyst that enabled modern deep learning techniques to emerge.

The paper 'Deep Learning' by LeCun et al. (2015) takes a modern look at machine learning, reviewing the benefits of deep learning compared to the limited conventional machine-learning techniques. LeCun et al. (2015) goes on to write that "Deep learning is making major advances in solving problems that have resisted the best attempts of the artificial intelligence community for many years". One of the benefits of deep learning is that the task of feature engineering has been shifted from the person onto the machine. "Deep learning requires only a set of data with minor preprocessing, if necessary" (Shen et al. 2017), this has had the effect of significantly easing the cost of entry to machine learning allowing "nonexperts in machine learning to effectively use deep learning for their own research" (Shen et al. 2017).

The next big leap in Deep Learning is wildly thought to be unsupervised learning, as LeCun et al. (2015) puts it; "Human and animal learning is largely unsupervised: we discover the structure of the world by observing it, not by being told the name of every object". This method would allow for training on a dataset that hasn't been labelled. This would be the ideal solution when labelling a dataset is impractical, be it due to time-constraint, budgetary or impractical. (Metz et al. 2018). Unfortunately, unsupervised learning has yet to live up to the hopes many place upon it, as such the model used in this paper will be trained using a supervised approach.

2.2 Tumour detection by architecture

2.2.1 Faster R-CNN

Faster R-CNN is the third paper in the R-CNN family, it was published in 2017. The first paper from 2014, R-CNN, proposes a region-based network for object detection (Girshick et al. 2014). The network first extracts ≈ 2000 regions, these are then re-shaped to a square and fed into a CNN for feature extraction and finally an SVM for classification. R-CNN, however, is very slow needing ≈ 47 seconds to process an image. The paper Fast R-CNN released in 2015 aimed to fix this, instead of extracting regions the whole image is used as input for the CNN. The CNN then creates a convolutional feature map that is used to generate the regions. This change results in a network that is "213×faster at test-time [than R-CNN]" (Girshick 2015). Faster R-CNN further increases the speed of the network by feeding the feature map into another CNN to generate regions, rather than using a selective search algorithm. "When compared to efficient detection networks, Selective Search is an order of magnitude slower"(Ren et al. 2015). The addition of the Region Proposal Network (RPN) in faster R-CNN has allowed for near real-time object detection.

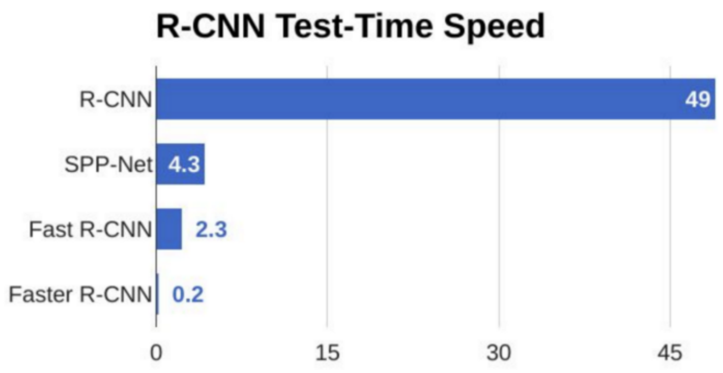


Figure 2.1: Speed Comparison of the R-CNN family.

Zhu et al. (2018) use Faster R-CNN as the base for their detection of lung nodules. They expand the network to work in 3D. They use the LUNA16 dataset, a subset of the LIDC-IDRI dataset. The LUNA16 set has removed all of the meta-info on the nodules such as size, location, texture and more, keeping only the annotations. The dataset consists of 888 CT scans. For pre-processing, the scans are normalised to a range of [0, 1] and the background is removed. 10 fold cross-validation is used to train and evaluate the model. Overall they achieved 90.44% accuracy with a sensitivity of 95.8%. This algorithm does perform excellently, however, it is in 3D whereas this project is looking at classification per slice, a 2D problem.

2.2.2 U-Net

The U-NET architecture proposed by Ronneberger et al. (2015) is a CNN based architecture designed specifically for biomedical imaging. By utilising extensive image augmentation techniques U-NET allows for accurate models to be trained on limited training data; "it works with very few training images and yields more precise segmentations" (Ronneberger et al. 2015). The network operates on a sliding window approach. Pixels inside the window are passed into successive convolution layers and upscalers. This allows for context to be provided and for more data to be extracted from the limited data sets U-NET is designed to worth with. The architecture gets its name from the two halves of it; the left encoder arm and the right decoder arm which form a U shape. A diagram of this can be seen in Figure 2.2.

The paper by Tong et al. (2018) uses an enhanced U-Net architecture. "We

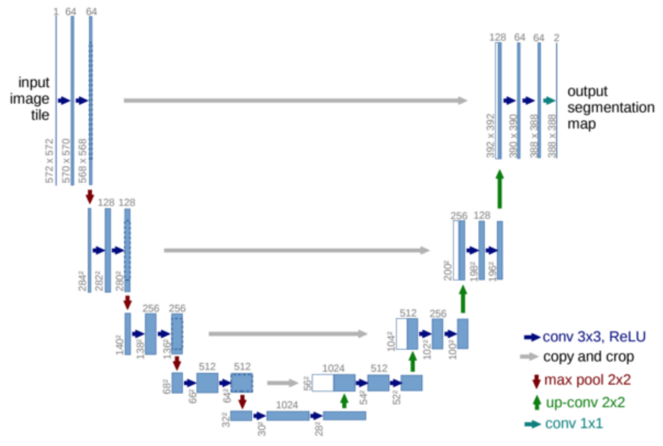


Figure 2.2: The U-NET architecture, encoder (left) and decoder (right)

introduced residual network, which has improved the network training" (Tong et al. 2018). They use the LUNA16 dataset, a subset of the LIDC dataset, to train and evaluate their improved U-Net architecture. They use an 8:1:1 ratio to split the dataset into Training, Validation and Testing respectively. This resulted in 975 scans in the training set. For pre-processing they normalise the pixels to the range [0, 1] and then segment the scan to remove surrounding data and extract the lungs. They use the Sørensen–Dice coefficient to compare the algorithm to the ground truth. The Sørensen–Dice coefficient gauges the difference between the two samples. Overall their modified U-Net results in a Sørensen–Dice coefficient of 0.736, compared to the un-modified U-Net's score of 0.713. In conclusion, the work by Tong et al. (2018) clearly shows that U-Net can be effectively used for the task of Nodule Segmentation. The work, however, results in a less accurate result compared to some of the previous techniques discussed.

2.2.3 ResNet

Deep-learning models use representation-learning methods in which specific layers are used to learn the low, mid and high-level features. In an image classification problem, "the learned features in the first layer of representation typically represent the presence or absence of edges . . . the second layer typically detects motifs . . . [and] the third layer may assemble motifs into larger combinations that correspond to parts of familiar objects" (LeCun et al. 2015). By this thinking to build a better neural network we can just stack more of these layers to identify more and more specific features. (He et al. 2016) show that this is not the case. In

Figure 2.3 we see that simply adding more layers to a network leads to a higher training error.

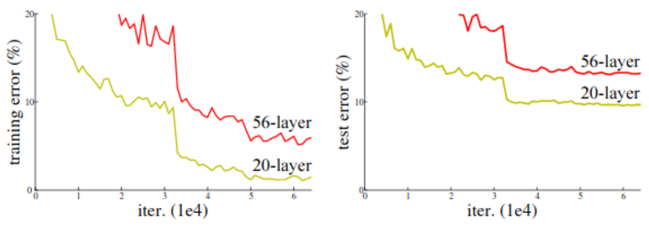
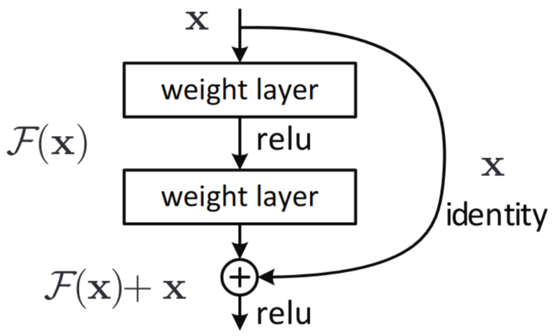


Figure 2.3: “Training error (left) and test error (right) on CIFAR-10 with 20-layer and 56-layer “plain” networks. The deeper network has higher training error, and thus test error.” (He et al. 2016)



x
weight layer
$\mathcal{F}(\mathbf{x})$
relu
weight layer
x
identity
$\mathcal{F}(\mathbf{x}) + \mathbf{x}$
relu

Figure 2.4: The Residual Block

He et al. (2016) present a solution to this issue in the form of a new layer for the network, a residual block. These residual blocks take input from the previous layer as normal, but also take the input from the previous layer. This can be seen in Figure 2.4. These ‘skip connections‘ allow for the network to learn identity mappings easier.

The consequence of these added residual blocks is that networks generally have a lower error when compared to traditional methods and as such can be built deeper. He et al. (2016) compared their new ResNet network with a traditional network without residual blocks and found that a 34 layer network can obtain sub 30% error, unlike the traditional network. They also found that a deeper ResNet with 34 layers performed better than a ResNet with 18 layers, reinforcing the idea that more layers can be used to increase network performance.

The paper by Jiang et al. (2018) uses a full resolution residual neural network (FRRN) (Pohlen et al. 2017) that extends ResNet by passing features at full resolution to each layer. They use three datasets, TCIA for training, MSKCC for validation and LIDC for testing. The Training was done using 160 x 160 region of interests (ROIs) around the tumour augmented with flips, shifts, noise and elastic deformation (Jiang et al. 2018). This lead to a total of 57,793 training image samples. They found that the FRRN was better at segmenting tumours than Random Forest, UNet and SegNet. On the MSKCC dataset, it performed very well with an accuracy of 86%, however, on the LIDC dataset the accuracy was only 64% although it had a higher sensitivity of 76% as opposed to the 69% sensitivity on the MSKCC dataset. While this piece of work does show that deep learning can be used, the poor accuracy on the testing set leads one to believe that ResNet may not be the best architecture for this project.

2.2.4 You Only Look Once (YOLO)

The YOLO architecture (Redmon et al. 2016) treats object detection as a regression problem on spatially separated bounding boxes. This approach enables for high accuracy while running in near real-time "45 frames per second with no batch processing on a Titan X GPU" (Redmon et al. 2016).

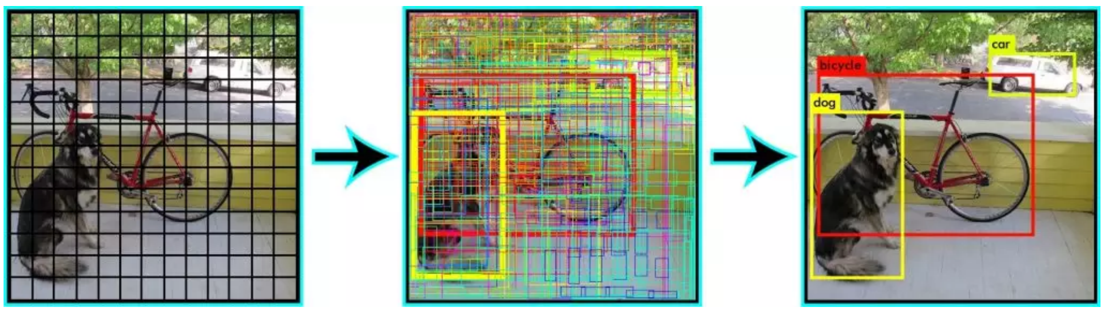


Figure 2.5: YOLO; Bounding Boxes to Detection

The architecture struggles to detect lots of closely packed small objects, like birds in a flock and does not deal well when detecting items in a new context. This shouldn't affect its usefulness's when detecting tumours, however since tumours will always be in the same context (lungs).

The work by Hammami et al. (2020) uses YOLO in conjunction with a Cycle GAN for the task of organ detection in CT images; although this use case does not directly translate to the use proposed in this project it is still a very relevant paper that shows that YOLO can be utilised within medical imaging and to great effect. Hammami et al. (2020) uses the third version of YOLO (YOLOv3) (Redmon & Farhadi 2018) to perform the organ detection; the architecture used consists of 53 convolutional layers. They train YOLO with "450 epochs and a decreasing learning rate" (Hammami et al. 2020). The dataset for the study was acquired from the Visceral Anatomy Benchmarks (Jimenez-del Toro et al. 2016). Two datasets were created from this data, the first created using manual segmentation and the second's "labels of which were obtained by merging the segmentations produced by the algorithms of benchmark participants" (Hammami et al. 2020). For pre-processing, the images were cropped to 320x320 pixels. The proposed method performs very well and achieves a mean average distance of 7.95$\pm$6.2mm. This is very promising and hopefully could be transferred to the task of detecting lung tumours.

2.2.5 Mask RCNN

Mask RCNN (He et al. 2017) is an extension to the Faster RCNN network that adds a segmentation branch to generate a segmentation mask. This extra step only adds a small computational overhead to the process (He et al. 2017); this is apparent as "models can run at about 200ms per frame on a GPU" (He et al. 2017).

The paper by Liu et al. (2018) uses Mask RCNN for the task of tumour segmentation. They use the LUNA16 dataset for testing and training, they do not give details on the split between test and train, however. Multiple pre-processing techniques are used, firstly slices that do not contain tumours are discarded; then the images are segmented to extract only the lungs and finally, they are normalised to a range of [0, 1]. Liu et al. (2018) used transfer learning to speed up the training process. The weights they used were trained from the COCO (Common Objects in Context) dataset (Lin et al. 2014). Overall they achieved a mean average precision of 0.796. This paper, although promising, is very brief and does not go into much detail on the methods they used, it does, however, show that Mask RCNN can be effectively used for tumour segmentation.

2.2.6 Summary and Key Findings

The purpose of this review was to explore the approaches taken before in tumour classification, aid in network architecture selection and note any techniques used that could be applied in this project. The papers reviewed have achieved accuracy scores between 60% and 90% indicating that the methods used can have a very large impact on model performance. The work by Zhu et al. (2018) using the faster RCNN architecture performed the best out of the papers discussed; as such faster RCNN will be the base of this project; mask RCNN may also be used as it is an extension upon the faster RCNN architecture with only minimal computational overhead. Segmenting the lungs out of the CT images is a common pre-processing step and the approach by Liu et al. (2018) of using K-Means clustering to segment the lungs may be used in this project. Furthermore, normalising the data between zero and one is also a pre-processing step taken in a few of the papers discussed. The transfer learning also done by (Liu et al. 2018) is also a technique that could greatly benefit this project. The decrease in training time is a huge benefit for two reasons. Firstly it will allow for training to be undertaken on the limited hardware available. Secondly, as training time will be substantially shortened training the model will not be a large endeavour taking multiple days but instead a task that can be done in a matter of hours. Because of this if the model has to be trained multiple times to increase accuracy the limit to the time impact on the project.

Chapter 3

Methodology

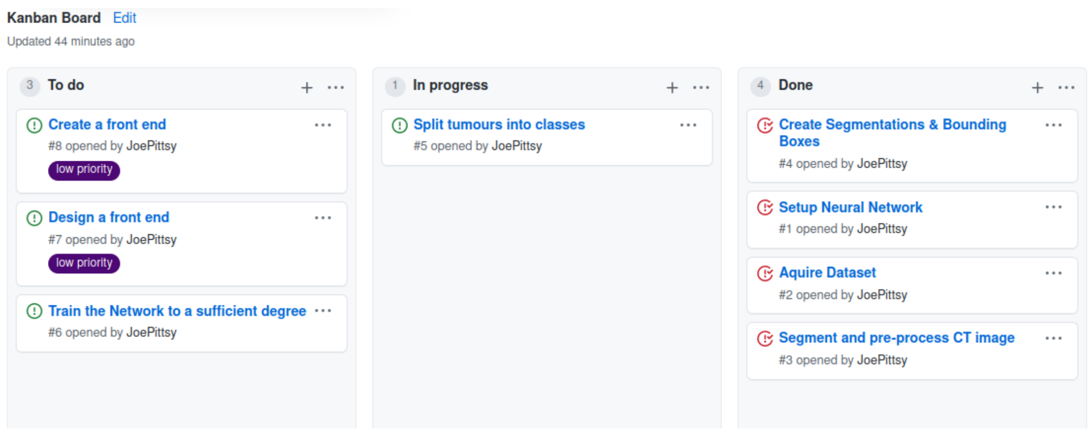
This section will discuss the management techniques and tools used throughout the Project as well as an overview of software development relating to the specifics of the Project. The section is split into sub-sections for better organisation; firstly, Project Management. This sub-section will cover the software management tools used. The second section, Software Development, discusses what software management techniques were used and others that were not deemed suitable as they did not facilitate the specific needs of the project. Finally, the Toolsets and Machine Environments section discusses the different tools used in the Projects life cycle, including IDE's and Version Control.

3.1 Project Management

A Guide to the Project Management Body of Knowledge describes Project Management as "the application of knowledge, skills, tools and techniques to project activities to meet the project requirements" (Institute 2017). This section will be a review of the tools that were used and others that were considered.

3.1.1 Kanban Board

Kanban was first introduced as a method for just in time (JIT) manufacturing at Toyota (Ohno 1988), since then, however, it has been adopted by the software development world as a tool for agile development as it can promote "improved software quality and lead time delivery" (Ahmad et al. 2013). For this Project a kanban board has been used throughout; originally as a physical whiteboard and post-it notes were used but this was eventually ported over to an online kanban board using GitHub. This was done to enable the board to be updated anywhere.


Kanban Board Edit
Updated 44 minutes ago

3 To do
- Create a front end
  #8 opened by JoePittsy
  low priority
- Design a front end
  #7 opened by JoePittsy
  low priority
- Train the Network to a sufficient degree
  #6 opened by JoePittsy

1 In progress
- Split tumours into classes
  #5 opened by JoePittsy

4 Done
- Create Segmentations & Bounding Boxes
  #4 opened by JoePittsy
- Setup Neural Network
  #1 opened by JoePittsy
- Aquire Dataset
  #2 opened by JoePittsy
- Segment and pre-process CT image
  #3 opened by JoePittsy

Figure 3.1: GitHub version of the Kanban board used

The Kanban board proved to be invaluable during the project, it allowed for the many tasks to be managed with ease and without overwhelming development.

3.1.2 Whiteboard

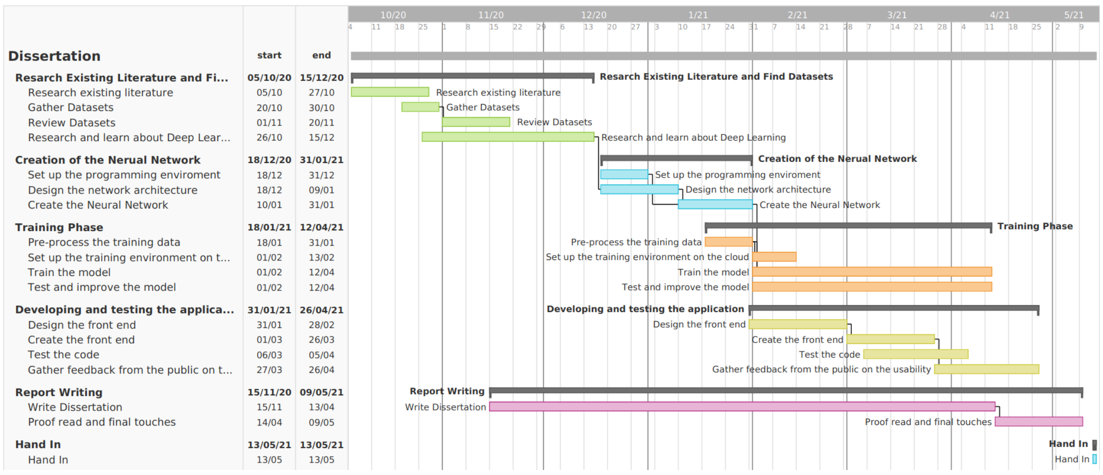
While not an established tool like a Gantt chart or a Kanban board a Whiteboard was an essential management tool, the sub-tasks identified during the creation of the Gantt chart were used as headings on the whiteboard and then even smaller tasks were broken out under those headings. A simple tick system was then used when a sub-task was completed, this not only provided a visual overview of the tasks both completed and pending but also provided a morale boost with each ticked off task.

3.1.3 Gannt Chart

“The most widely used method for displaying and tracking progress is the Gantt chart, which enumerates activities along the vertical axis and their start and stop dates on the horizontal axis” (Stoneburner 1999). A Gantt chart was created as part of the proposal phase of this project, it can be seen in Figure 3.2.

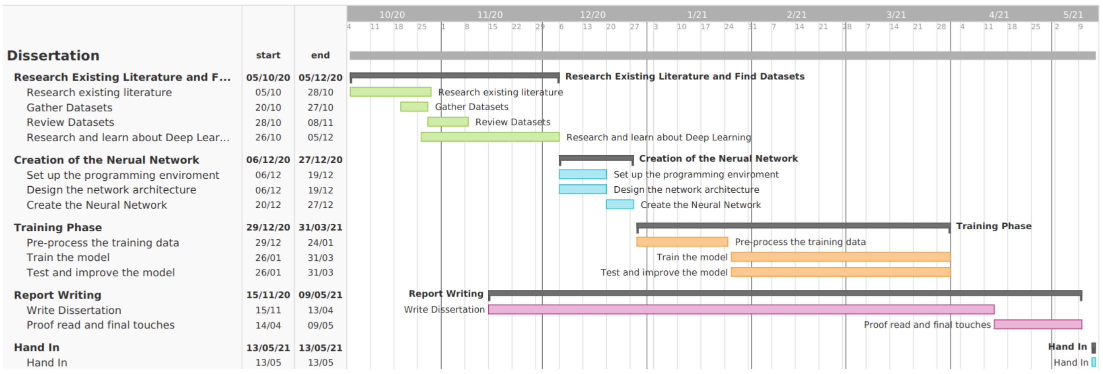
Unfortunately, this initial Gantt chart was not very accurate, some tasks were completed much faster than indicated on the chart and others much slower. The Gannt chart was still useful in the way that it decomposed the problem into smaller chunks, while the times given turned out to be inaccurate the breakdown of the Project proved to be very useful.

A Gantt chart that more actively reflects the actual timescale of the project can



Dissertation	start	end
Resarch Existing Literature and Fi...	**05/10/20**	**15/12/20**
Research existing literature	05/10	27/10
Gather Datasets	20/10	30/10
Review Datasets	01/11	20/11
Research and learn about Deep Lear...	26/10	15/12
Creation of the Nerual Network	**18/12/20**	**31/01/21**
Set up the programming enviroment	18/12	31/12
Design the network architecture	18/12	09/01
Create the Neural Network	10/01	31/01
Training Phase	**18/01/21**	**12/04/21**
Pre-process the training data	18/01	31/01
Set up the training environment on t...	01/02	13/02
Train the model	01/02	12/04
Test and improve the model	01/02	12/04
Developing and testing the applica...	**31/01/21**	**26/04/21**
Design the front end	31/01	28/02
Create the front end	01/03	26/03
Test the code	06/03	05/04
Gather feedback from the public on t...	27/03	26/04
Report Writing	**15/11/20**	**09/05/21**
Write Dissertation	15/11	13/04
Proof read and final touches	14/04	09/05
Hand In	**13/05/21**	**13/05/21**
Hand In	13/05	13/05

Figure 3.2: Initial Gannt Chart



Dissertation	start	end
Research Existing Literature and F...	**05/10/20**	**05/12/20**
Research existing literature	05/10	28/10
Gather Datasets	20/10	27/10
Review Datasets	28/10	08/11
Research and learn about Deep Lear...	26/10	05/12
Creation of the Nerual Network	**06/12/20**	**27/12/20**
Set up the programming enviroment	06/12	19/12
Design the network architecture	06/12	19/12
Create the Neural Network	20/12	27/12
Training Phase	**29/12/20**	**31/03/21**
Pre-process the training data	29/12	24/01
Train the model	26/01	31/03
Test and improve the model	26/01	31/03
Report Writing	**15/11/20**	**09/05/21**
Write Dissertation	15/11	13/04
Proof read and final touches	14/04	09/05
Hand In	**13/05/21**	**13/05/21**
Hand In	13/05	13/05

Figure 3.3: Updated Gannt Chart

be seen in Figure 3.3. As you can see the decision was made between the proposal and starting the project that a front end would not be created and instead efforts were to be focused on training the model. The environment setup and creation of the network took less time than initially stated. The training and improvement of the model, however, took much longer.

3.1.4 Risk Assessment

Before undertaking any large project it is prudent to understand the risks and to have devised a strategy on how to mitigate the issues that arise. Table 3.1 outlines a risk analysis undertaken. It measures each risk by the likelihood of occurrence and the potential impact if not mitigated, as such a mitigation plan is then provided. By following this analysis any of these risks should not cause a significant danger to the project.

Risk	**Likelihood**	**Impact**	**Mitigation Plan**
Trained Model is not sufficiently accurate.	High	Low	Retrain with tweaked hyperparameters
The dataset is imbalanced	Medium	Medium	Use data augmentation techniques to correct for the imbalance.
Lack of processing power.	High	High	Use cloud solutions or university supplied machines.
Time Constraints	Low	Medium	Work on the project continuously and start the report early in the process.
Underfitting	Medium	High	An underfit model cannot detect a tumor. To solve this train for more epochs.
Overfitting	Medium	Medium	An overfit model cannot generalise and only preforms well on the training dataset. Training for fewer epochs should solve this.

Table 3.1: Risk Analysis table

3.2 Software Development

This section will be a review and selection of a software development methodology. The specific needs and requirements of the project will be considered in this selection process

3.2.1 Methodology Selection

Methodologies can be broadly split into two categories, heavyweight and lightweight. Heavyweight, traditional, methodologies "require defining and documenting a stable set of requirements at the beginning of a project" (Awad 2005). Lightweight, agile, methods instead focus on responsiveness to change where requirements are not set in stone. The most significant heavyweight methodologies are the Waterfall and Spiral models.

To select the correct methodology for the project, first, we must determine what is required for this project. Most methodologies expect a team of developers to work on the same project, this project, however, has only one developer. This will limit the number of methodologies that can be employed. This project also has a fixed deadline, therefore methodologies that prioritise fast development times or fixed development timescales will be preferred.

The waterfall method is one of the oldest software development methodologies (Huo et al. 2004), it describes a linear progression through a series of five steps commonly labelled Requirement Analysis, Design, Development, Testing & Maintenance. These steps are often allocated a period of time for completion. The previous steps are frozen when moving on to the next step (Balaji & Murugaiyan 2012) since water cannot flow uphill in a waterfall. The main benefits of waterfall are the clear requirements set from the start of the project and its ease to implement. The strict progression, however, can lead to inflexibility if the requirements of the project change. This methodology does guarantee a fixed timescale from the start which is desirable for this project, however, its inflexibility does pose an issue as with a one-person development team problems are likely to arise.

The spiral methodology takes a linear step approach like waterfall but these stops are repeated until project completion. This helps to mitigate the main downfall of waterfall, its inflexibility. Spiral is a risk-driven model that requires risk assessments to be carried out throughout (Boehm 1988). A development team with little experience in this area may find the spiral model unpractical to implement. As discussed spiral does fix the inflexibility downside of waterfall making it attractive for use in this project, however, it is much more suitable for larger projects with substantial development teams.

The Agile Manifesto (Fowler et al. 2001) is a response to the traditional methodologies and laid out the key principles of agile methods. It promotes the need for flexibility in the face of changing requirements, frequent working releases and simplicity. These principles line up very well with the requirements of this project. Frequent working releases, even if not fully featured, would ensure that by the end of the fixed time frame of this project an artefact will have been produced. Kanban and Scrum are two popular agile methodologies.

The Kanban methodology leverages just in time (JIT) principles to prioritise tasks (Lei et al. 2017). Kanban focuses on limiting the amount of work and visualising the development process through cards and Kanban boards (Discussed in sub-section 3.1.1). Work under Kanban is deconstructed into small tasks, this can help visualise the work needed and render a previously daunting task manageable. A key benefit of Kanban is that "A kanban workflow can change at any time" (Rehkopf n.d.), this is great for this project where requirements may shift throughout.

Scrum is an agile methodology that uses 'sprints' which are fixed time periods dedicated to completing specific work tasks. These sprints are the foundation of the Scrum methodology. Another key aspect of Scrum is the daily stand-ups, these meetings are used to structure the day of work. The obvious downside of Scrum, in the context of this project, is the daily stand-ups. As a one-person development team, this key aspect of Scrum can not be applied.

Considering the above methodologies and the specific requirements of this project Kanban was selected. Kanbans focus on continuous release and limiting work in progress will allow for the project to be completed piece by piece without overwhelming the developer. Furthermore, the flexibility offered should accommodate any changes to the project at any stage of development.

3.3 Toolsets and Machine Environments

3.3.1 Integrated Development Environments (IDEs)

Muşlu et al. (2012) states that "Integrated development environments make recommendations and automate common tasks, such as refactoring, auto-completions, and error corrections", these tools can greatly increase the efficiency of software developers. Furthermore, many IDEs contain debugging tools that help save time. In this subsection, we will be comparing a range of IDEs and will conclude which is the most suitable for this project.

PyCharm

PyCharm is an IDE designed specifically for Python, it runs on all three major operating systems; Windows, MacOS & Linux. The IDE comes in two 'flavours', Community Edition and Professional. The Community edition is free open source software (FOSS) and is licensed under the Apache 2.0 License (Haagsman 2017).

The other version of PyCharm, Professional Edition, is based upon the free version but has closed source features. The Professional edition is £6.90 monthly or £69.00 for the first year, £55.00 for the second year and £41.00 every year after that. If a user has had a license for a year and then chooses not to renew they have a fallback license for the version of the IDE that the subscription was started on (JetBrains n.d.). The professional option does have an educational license that would allow the software to be used for free in this project.

There are two main advantages of PyCharm professional over PyCharm community, firstly a suite of web development tools. Secondly, and more relevant to this project, built-in data science tools including Pandas, Numpy, Matplotlib and Jupyter.

Visual Studio

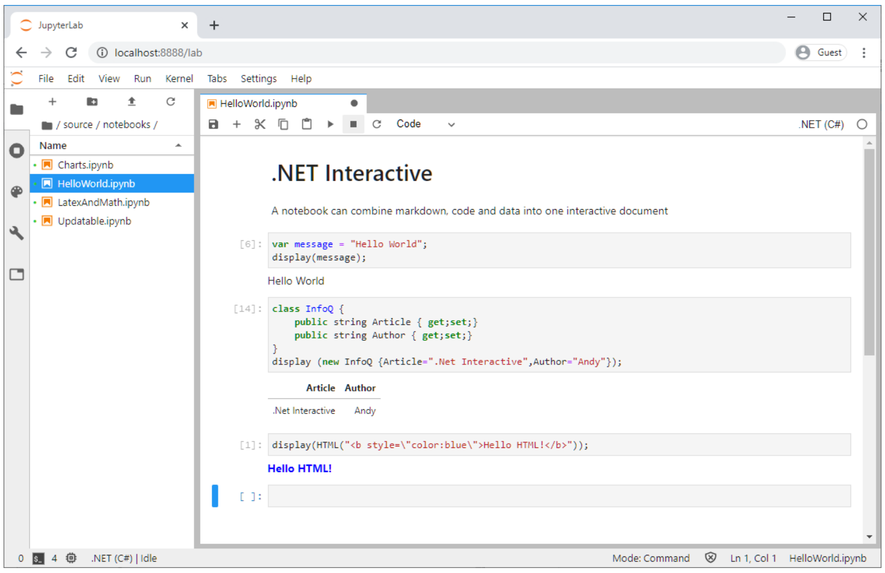
Visual Studio is Microsoft's multi-language IDE. Visual Studio only runs on Windows, a MacOS version is available, however, it is a stripped-back version and cannot develop Python (Microsoft n.d.). It provides no support for Linux at all. Visual Studio comes in three variations; Community, Professional and Enterprise.

Visual Studio Enterprise is designed for enterprise use and as such will not be discussed here. All version of Visual Studio are closed source.

Visual Studio Community is a free IDE for individuals, education, academia and open-source projects. It provides the development tools to work with Python and can be expanded upon with extensions if features are missing. Visual Studio Professional is only needed for a commercial team as Community disallows use for profit. In this Project, Visual Studio Community would be fine as it is both open source and non-commercial.

Jupyter and Google Colab

Jupyter is a web-based notebook development environment. Jupyter notebooks are a mix of markdown and code cells, this allows for inline documentation that is more powerful than comments. An example of this can be seen in Figure 3.4.



.NET Interactive

A notebook can combine markdown, code and data into one interactive document

```
[6]: var message = "Hello World";
     display(message);
```

Hello World

```
[14]: class InfoQ {
          public string Article { get;set;}
          public string Author { get;set;}
      }
      display (new InfoQ {Article=".Net Interactive",Author="Andy"});
```

Article	Author
.Net Interactive	Andy

```
[1]: display(HTML("<b style=\"color:blue\">Hello HTML!</b>"));
```

Hello HTML!

```
[ ]:
```

Figure 3.4: An example Jupyter notebook showing the combination of markdown and code.

"The Jupyter system supports over 100 programming languages . . . including Python, Java, R, Julia, Matlab, Octave, Scheme, Processing, Scala, and many more" (Barba et al. 2019). Unfortunately, Jupyter is lacking many features that a full IDE has such as debugging, refactoring and control. It is, however, fully free and open-source operating under The 3-Clause BSD License.

Google Colab is Googles free Jupyter environment that runs in the cloud, it has one feature that sets it apart however, they provide free access to GPU's and

TPU's for code execution. This is very convenient for developing on the go where you might not have access to powerful hardware to train models.

Conclusion

This Project will be mainly developed on the Ubuntu 20.04 operating system, this rules out the use of Visual Studio Community and Visual studio pro as they are Windows-only applications. The rest of the IDEs discussed can be installed on Ubuntu or run in the browser in the case of Google Colab / Colab Pro. A Comparison matrix can be seen in Table 3.2; from this, we can see that Jupyter lab and Google Colab / Colab pro do not have any inbuilt debugging tools. Perscheid et al. (2017) states that "developers are in need of tools that support them in analyzing the interactions of different parts of the program", I agree and because of this, I do not believe that Jupyter lab, Google Colab or Colab pro are the right tools to use for this project. That leaves us with PyCharm and PyCharm Pro, looking again at the comparison matrix in Table 3.2 we can see that the two are very similar and offer many of the same functionality, however, PyCharm Pro has inbuilt support for NumPy, MatPlotlib and more. I believe that these inclusions result in PyCharm Pro being the IDE of choice for this project, especially considering it is free for Students. Google Colab will be utilised to train the model using the free GPU access, however, the code will be programmed and debugged in PyCharm.

	PyCharm Community	**PyCharm Pro**	**Visual Studio Community**	**Visual Studio Pro**	**Jupyter Lab**	**Google Colab**	**Google Colab Pro**
Price	Free	£6.90/Month Free for Students	Free	$45/Month	Free	Free	$9.99/Month
Open Source	Yes	No	No	No	Yes	Yes	Yes
License	Apache 2	N/A	N/A	N/A	BSD 3-Clause	Apache 2	Apache 2
Version Control	Yes	Yes	Yes	Yes	No	Limited	Limited
Debugging Tools	Yes	Yes	Yes	Yes	No	No	No
Integrated Terminal	Yes	Yes	Yes	Yes	No	Limited	Limited
Notes	Made for Python	Integrated support for data science tools	Windows Only	Windows Only		Free GPUs'	Faster GPU's

Table 3.2: Comparison Matrix for different IDE's

3.3.2 Version Control

Somasundaram (2013) describes version control as "A system capable of recording the changes made to a file or a set of files over a time period in such a way that it allows us to get back in time from the future to recall a specific version on that file". This sub-section will review the different types of version control, the one selected for this project and how it was used to manage the project.

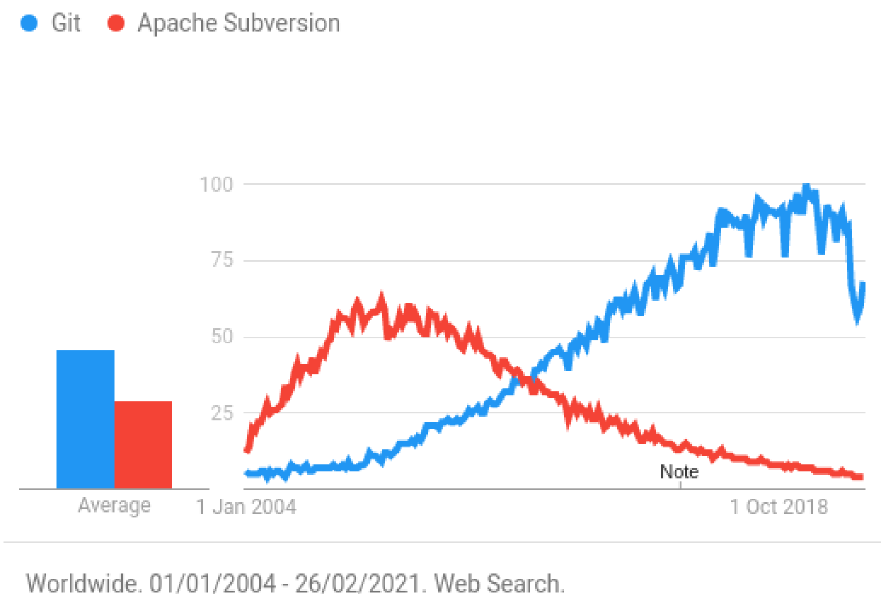


Figure 3.5: Google Trends data for Git vs SVN

There are two distinct groups of version control; Centralised Version Control Systems (CVCS) and Distributed Version Control Systems (DVCS). CVCS's have developers pulling and committing to the same directory hosted locally by the team; "These VCSs [CVCS] are centralized as they have a single canonical source repository. All developers work against this repository through a checkout taken from the repository, essentially a snapshot at some moment in time." (De Alwis & Sillito 2009). Distributed Version Control Systems, however, allow for developers to check out the entire history, not just a snapshot, this has led the way to 'Feature Branches' which De Alwis & Sillito (2009) describes as "prospective change is done within a branch and then merged into the mainline, rather than being directly developed against the mainline. CVS has taken the lead recently and has become the most used variant of the two; In figure 3.5 we can see that recently Git, the most common CVCS software, overtook Subversion (SVS a very popular DVCS software) in December 2011 and since then has exploded in popularity.

A Distributed Version Control System and by extension Git, the most popular implementation of DCVS is to be the chosen VCS for this project, still, the decision on where to store the repository has to be made. We can either store it locally or online using one of the available git-based platforms; below follows a discussion on the different solutions, pros and cons for each and a conclusion on the chosen platform.

Locally Hosted

The easiest way to get started with git is to run `git init` in any directory and a new git repository will be created, from there you can commit changes and use git to start tracking changes. The downside of a local git repository is that in the case of a drive failure then all of the code will be lost, there is no backup. Furthermore, when working with a team a local repository may be less suitable.

GitLab

GitLab is a consideration if you wish to use git with a team while still having complete control. GitLab is an application that allows for changes to be pushed and pulled from a main point. Developers can pull the repository to their local machine and then push the changes back without ever interacting with the files directly. GitLab sill has the same downside of a single point of failure, however. Regular backups could alleviate this issue.

GitHub

GitHub allows for repositories to be stored on their servers, it is software as a service (SASS) meaning that GitHub manages everything apart from the data. This means that programmers can focus their efforts on development and leave the rest to GitHub. Furthermore, GitHub has extra features such as kanban boards, Continuous Integration (CI) tools and more. GitHub also offers its GitHub Pro service for free for students.

3.3.3 Conclusion

Carefully considering the options, GitHub is the solution that meets the most criteria for this project. It is free and very hands-off allowing more time to be devoted to developing DeepTumour.

3.3.4 Packages and Libraries

Throughout this project many open source packages and libraries were used, as Li et al. (2009) states “Using off-the-shelf (OTS) components such as COTS or open-source software (OSS) promises to reduce development time and cost while increasing software quality”. This sub-section serves to document the key libraries used, where they were used and why.

pylidc

The python package pylidc (Hancock & Magnan 2016) “is an Object-relational mapping (using SQLAlchemy) for the data provided in the LIDC dataset” (Hancock 2018). It was used extensively in the pre-processing phase when working with the LIDC dataset, it allowed for very fast access to the radiologists' augmentations, tumour bounding box's and malignancy information. The TCIA team recommends users use the pylidc package before time is wasted writing custom tools to analyse the XMLs themselves (TCIA Team 2020).

Pylidc is available under the MIT License (MIT).

NumPy

NumPy (Harris et al. 2020) “is an open-source project aiming to enable numerical computing with Python”. It is an essential package to have when doing anything mathematical in Python. It is used throughout the Project in many capacities.

NumPy is available under the BSD 3-Clause license.

Detectron2

Detectron2 (Wu et al. 2019) is Facebook's framework for creating object detection models. It acts as an abstraction layer on top of PyTorch (Paszke et al. 2019), an open-source machine learning library. Detectron2 has a ‘model zoo’ of common machine learning architectures such as Faster R-CNN, RPN & Fast R-CNN, Mask R-CNN and more. Detectron2 is used for the creation and training of the model.

Detectron2 is released under the Apache 2.0 license.

scikit-learn

Scikit-learn (Pedregosa et al. 2011) is a machine learning library for Python with functions for regression, clustering and more. Scikit-Learn was used extensively in the pre-processing stage to segment the lungs from the CT scans (This is discussed in section 4.3.1).

Scikit-learn is available under the BSD 3-Clause license.

Others

The packages shown above are the primary libraries used. Other packages such as OpenCV, MatPlotLib, MedPy and more were used throughout this project. See the requirements.txt file in the source code for a more comprehensive list.

Chapter 4

Design, Development and Evaluation

4.1 Requirements elicitation, collection and analysis

4.1.1 Requirements Elicitation

Before developing one must first determine the requirements for the project. Furthermore, an understating on which requirements are essential and which are less of a priority can streamline the development process. Below is a list of requirements created for this project, those listed as must are essential requirements.

1. The dataset must be freely available for use.
2. The data must be CT scans.
3. The dataset should be sufficiently large ($\geq$ 500 scans).
4. The CT scans should be segmented.
5. The model must have few false negatives.

The most important requirement here is that the model must have few false negatives. Due to the nature of the project, an increase in false positives is preferred if it decreases false negatives. A false positive can be reviewed by a radiologist and found to be harmless, but a false negative may be missed by the radiologist resulting in the cancer being undetected.

4.1.2 Dataset Analysis

A crucial step in any machine learning solution is to collect a dataset to train the model upon. Deep Learning requires a very large dataset in order to effectively train (LeCun et al. 2015). This section will discuss the different datasets publicly available, their strengths and weakness for use in this project, a comparison matrix and finally, a decision on the correct dataset to use for this project.

Lung Image Database Consortium (LIDC-IDRI)

The LIDC-IDRI dataset is a collection of thoracic computed tomography (CT) scans of 1010 patients with lesions annotated by four radiologists. The scans are provided in the DICOM (Digital Imaging and Communications in Medicine) format, it provides additional metadata such as patient and hospital details and more, since these scans have been anonymised the extra information provided by this format is mute and as such we can convert the images to another format such as PNG. "The Database contains 7371 lesions marked "nodule" by at least one radiologist. 2669 of these lesions were marked $nodule \geq 3mm$ by at least one radiologist, of which 928 (34.7%) received such marks from all four radiologists. These 2669 lesions include nodule outlines and subjective nodule characteristic ratings." (Armato III et al. 2011). The LIDC-IDRI dataset is available under the Creative Commons Attribution 3.0 Unported License.

LUNA16

LUNA16 is a sub-set of LIDC-IDRI where slices of a thickness greater than 2.5mm are excluded, this leaves the dataset with 888 CT scans. They also remove Nodules that have been marked $nodule \geq 3mm$ by less than 3 radiologists keeping those only marked by 3 or 4 radiologists. The images are provided in ten different zip files to allow for easy use with 10-fold cross-validation techniques. The files are provided as MHD (MetaImage MetaHeader) files which just contain the raw pixel data, unlike the DICOM format. The LUNA16 Dataset is made available under the Creative Commons Attribution 4.0 International License.

National Institutes of Health (NIH) DeepLesion Dataset

The NIH DeepLesion dataset is unlike the other datasets discussed here in that it contains scans and annotations of note all over the body it contains "lung nodules, liver tumors, enlarged lymph nodes, and so on." (NIH Clinical Center 2018). It contains a total of 32,120 CT slices from 10,594 CT scans on 4,427 patients. Approximately 25% of the lesions are in the lungs resulting in ≈ 8183 lung lesions (Yan et al. 2018). The dataset was harvested for use in a deep learning study for the creation of a general/universal lesion detector. Most of the scans are $512x512$ pixels with 48.3% being 1mm slices and 48.9% are 5mm (Yan et al. 2018). This dataset would require significant pre-processing to extract just the Lung CT-Scans are remove any unwanted annotations.

4.1.3 Dataset Conclusion

Overall the researcher believes that the LIDC dataset is the best fit for this project. The NIH DeepLesion Dataset is good but it is not specific to lungs and as such would require a significant time sink, furthermore since the LIDC and LUNA16 sets are lungs only the images are likely to be more uniform in slice width and image size. The decision to use LIDC over LUNA16 is that using the original dataset that LUNA was derived from allows for more control for the researchers and exposes for scans to work with.

Criteria	**Value**
Number of Scans	1010
Number of Tumours	13,299
Number of Slices	244,527
Number of Slices with Tumours	12,165
Slice Width	512
Slice Height	512

Table 4.1: Statistics for the LIDC dataset

4.2 Design

4.2.1 Network

The architecture in use for this project is the Mask-RCNN architecture as discussed in Section 2.2.5. Table 4.2 shows a breakdown of the layers that make up the network as well as the number of trainable parameters per layer. In total the network has 62,635,542 trainable parameters.

Transfer Learning

This project is making use of transfer learning, A method of re-training an existing model to recognise a new class, or classes. Transfer learning is normally only used when the two domains are similar, i.e. a network trained on horses could be transfer learned to recognise zebras instead. This project attempts to transfer learn across domains. The model was originally trained on the COCO dataset (Common items in context). None of the classes it is trained on are related to tumours in any way. Despite the perceived lack of relationship between the two problems, transfer learning can produce very accurate models in just a fraction of the time it would take to train a model from scratch on the LIDC-IDRI dataset.

Hyperparameters

"Hyperparameters are critical in machine learning, as different hyperparameters often result in models with significantly different performance" (Wang & Gong 2018). To train the best model tweaking and finding the optimal hyperparameters is key. The hyperparameters with the most impact in the context of this project were the base learning rate, learning rate decay, maximum iterations and warm-up iterations. A value of 0.001 was used for the base learning rate, this decayed by 0.5 every 250 iterations. The models were all had 100 warm-up iterations. The max number of iterations differed from class to class to optimise training, categories one and three for 1000 epochs and category two for 1500.

Layers	Parameters	Layers	Parameters	Layers	Parameters
backbone.fpn_lateral2.weight	65536	backbone.bottom_up.res4.5.conv1.weight	262144	backbone.bottom_up.res4.20.conv1.weight	262144
backbone.fpn_lateral2.bias	256	backbone.bottom_up.res4.5.conv2.weight	589824	backbone.bottom_up.res4.20.conv2.weight	589824
backbone.fpn_output2.weight	589824	backbone.bottom_up.res4.5.conv3.weight	262144	backbone.bottom_up.res4.20.conv3.weight	262144
backbone.fpn_output2.bias	256	backbone.bottom_up.res4.6.conv1.weight	262144	backbone.bottom_up.res4.21.conv1.weight	262144
backbone.fpn_lateral3.weight	131072	backbone.bottom_up.res4.6.conv2.weight	589824	backbone.bottom_up.res4.21.conv2.weight	589824
backbone.fpn_lateral3.bias	256	backbone.bottom_up.res4.6.conv3.weight	262144	backbone.bottom_up.res4.21.conv3.weight	262144
backbone.fpn_output3.weight	589824	backbone.bottom_up.res4.7.conv1.weight	262144	backbone.bottom_up.res4.22.conv1.weight	262144
backbone.fpn_output3.bias	256	backbone.bottom_up.res4.7.conv2.weight	589824	backbone.bottom_up.res4.22.conv2.weight	589824
backbone.fpn_lateral4.weight	262144	backbone.bottom_up.res4.7.conv3.weight	262144	backbone.bottom_up.res4.22.conv3.weight	262144
backbone.fpn_lateral4.bias	256	backbone.bottom_up.res4.8.conv1.weight	262144	backbone.bottom_up.res5.0.shortcut.weight	2097152
backbone.fpn_output4.weight	589824	backbone.bottom_up.res4.8.conv2.weight	589824	backbone.bottom_up.res5.0.conv1.weight	524288
backbone.fpn_output4.bias	256	backbone.bottom_up.res4.8.conv3.weight	262144	backbone.bottom_up.res5.0.conv2.weight	2359296
backbone.fpn_lateral5.weight	524288	backbone.bottom_up.res4.9.conv1.weight	262144	backbone.bottom_up.res5.0.conv3.weight	1048576
backbone.fpn_lateral5.bias	256	backbone.bottom_up.res4.9.conv2.weight	589824	backbone.bottom_up.res5.1.conv1.weight	1048576
backbone.fpn_output5.weight	589824	backbone.bottom_up.res4.9.conv3.weight	262144	backbone.bottom_up.res5.1.conv2.weight	2359296
backbone.fpn_output5.bias	256	backbone.bottom_up.res4.10.conv1.weight	262144	backbone.bottom_up.res5.1.conv3.weight	1048576
backbone.bottom_up.res3.0.shortcut.weight	131072	backbone.bottom_up.res4.10.conv2.weight	589824	backbone.bottom_up.res5.2.conv1.weight	1048576
backbone.bottom_up.res3.0.conv1.weight	32768	backbone.bottom_up.res4.10.conv3.weight	262144	backbone.bottom_up.res5.2.conv2.weight	2359296
backbone.bottom_up.res3.0.conv2.weight	147456	backbone.bottom_up.res4.11.conv1.weight	262144	backbone.bottom_up.res5.2.conv3.weight	1048576
backbone.bottom_up.res3.0.conv3.weight	65536	backbone.bottom_up.res4.11.conv2.weight	589824	proposal_generator.rpn_head.conv.weight	589824
backbone.bottom_up.res3.1.conv1.weight	65536	backbone.bottom_up.res4.11.conv3.weight	262144	proposal_generator.rpn_head.conv.bias	256
backbone.bottom_up.res3.1.conv2.weight	147456	backbone.bottom_up.res4.12.conv1.weight	262144	proposal_generator.rpn_head.objectness_logits.weight	768
backbone.bottom_up.res3.1.conv3.weight	65536	backbone.bottom_up.res4.12.conv2.weight	589824	proposal_generator.rpn_head.objectness_logits.bias	3
backbone.bottom_up.res3.2.conv1.weight	65536	backbone.bottom_up.res4.12.conv3.weight	262144	proposal_generator.rpn_head.anchor_deltas.weight	3072
backbone.bottom_up.res3.2.conv2.weight	147456	backbone.bottom_up.res4.13.conv1.weight	262144	proposal_generator.rpn_head.anchor_deltas.bias	12
backbone.bottom_up.res3.2.conv3.weight	65536	backbone.bottom_up.res4.13.conv2.weight	589824	roi_heads.box_head.fc1.weight	12845056
backbone.bottom_up.res3.3.conv1.weight	65536	backbone.bottom_up.res4.13.conv3.weight	262144	roi_heads.box_head.fc1.bias	1024
backbone.bottom_up.res3.3.conv2.weight	147456	backbone.bottom_up.res4.14.conv1.weight	262144	roi_heads.box_head.fc2.weight	1048576
backbone.bottom_up.res3.3.conv3.weight	65536	backbone.bottom_up.res4.14.conv2.weight	589824	roi_heads.box_head.fc2.bias	1024
backbone.bottom_up.res4.0.shortcut.weight	524288	backbone.bottom_up.res4.14.conv3.weight	262144	roi_heads.box_predictor.cls_score.weight	2048
backbone.bottom_up.res4.0.conv1.weight	131072	backbone.bottom_up.res4.15.conv1.weight	262144	roi_heads.box_predictor.cls_score.bias	2
backbone.bottom_up.res4.0.conv2.weight	589824	backbone.bottom_up.res4.15.conv2.weight	589824	roi_heads.box_predictor.bbox_pred.weight	4096
backbone.bottom_up.res4.0.conv3.weight	262144	backbone.bottom_up.res4.15.conv3.weight	262144	roi_heads.box_predictor.bbox_pred.bias	4
backbone.bottom_up.res4.1.conv1.weight	262144	backbone.bottom_up.res4.16.conv1.weight	262144	roi_heads.mask_head.mask_fcn1.weight	589824
backbone.bottom_up.res4.1.conv2.weight	589824	backbone.bottom_up.res4.16.conv2.weight	589824	roi_heads.mask_head.mask_fcn1.bias	256
backbone.bottom_up.res4.1.conv3.weight	262144	backbone.bottom_up.res4.16.conv3.weight	262144	roi_heads.mask_head.mask_fcn2.weight	589824
backbone.bottom_up.res4.2.conv1.weight	262144	backbone.bottom_up.res4.17.conv1.weight	262144	roi_heads.mask_head.mask_fcn2.bias	256
backbone.bottom_up.res4.2.conv2.weight	589824	backbone.bottom_up.res4.17.conv2.weight	589824	roi_heads.mask_head.mask_fcn3.weight	589824
backbone.bottom_up.res4.2.conv3.weight	262144	backbone.bottom_up.res4.17.conv3.weight	262144	roi_heads.mask_head.mask_fcn3.bias	256
backbone.bottom_up.res4.3.conv1.weight	262144	backbone.bottom_up.res4.18.conv1.weight	262144	roi_heads.mask_head.mask_fcn4.weight	589824
backbone.bottom_up.res4.3.conv2.weight	589824	backbone.bottom_up.res4.18.conv2.weight	589824	roi_heads.mask_head.mask_fcn4.bias	256
backbone.bottom_up.res4.3.conv3.weight	262144	backbone.bottom_up.res4.18.conv3.weight	262144	roi_heads.mask_head.deconv.weight	262144
backbone.bottom_up.res4.4.conv1.weight	262144	backbone.bottom_up.res4.19.conv1.weight	262144	roi_heads.mask_head.deconv.bias	256
backbone.bottom_up.res4.4.conv2.weight	589824	backbone.bottom_up.res4.19.conv2.weight	589824	roi_heads.mask_head.predictor.weight	256
backbone.bottom_up.res4.4.conv3.weight	262144	backbone.bottom_up.res4.19.conv3.weight	262144	roi_heads.mask_head.predictor.bias	1

Table 4.2: Layers and trainable parameters per layer. Total Trainable Parameters: 62,635,542

4.3 Development

This section has been split into subsections. In each sub-section key ideas, techniques and concepts of note will be discussed. Furthermore, an overview of any problems encountered, and how the problems were overcome or mitigated. Then a decomposition of the code is provided.

4.3.1 Pre-Processing

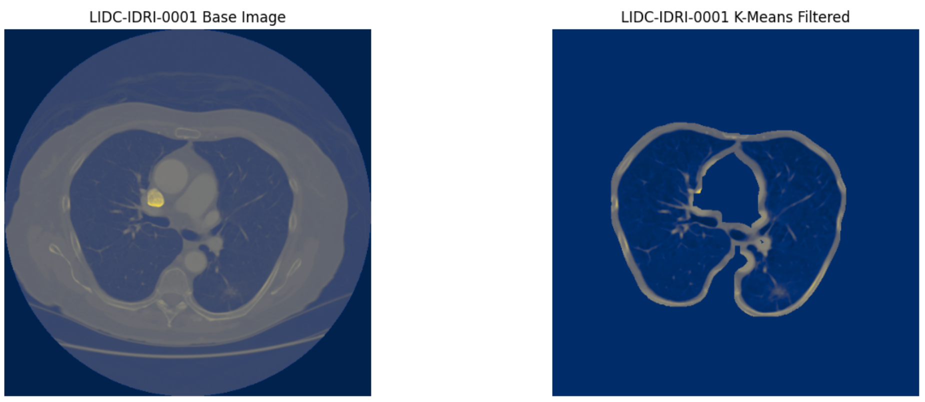
Kuhn & Johnson (2013) describes data pre-processing and data pre-processing techniques as the “addition, deletion, or transformation of the training set data”. Kuhn & Johnson (2013) then follows this up by stating that “data preparation can make or break a model’s predictive ability”. Common problems with datasets include missing data, impossible data, and more specifically when dealing with image datasets; improperly scaled images. Since the dataset that we are using in this project was not collected by us but by the NIC Clinical Center the scans have had pre-cursory processing to ensure they are all the same size. Below we will discuss a variety of pre-processing methods used.

Annotation

Annotation of the dataset was mainly done by the radiologists, however since multiple radiologists mark the same tumour these values needed to be averaged to provide just one annotation per tumour. For values such as malignancy, calcification and subtlety the high median was taken. For the segmentation’s, however, the task was more difficult. To generate a consensus for segmentation’s we first use the `cascaded_union` function from Shapely (Gillies et al. 2007–) to join all of the shapes together. Next, the shape is rasterised into an image using rasterio (Gillies et al. 2013–); this is done so we can apply a Gaussian blur, which is used to help blend the polygons together. Finally, we convert the blurred raster image back into a polygon and return it as the tumour mask.

K-Means for Lung Segmentation

The CT images contain a lot of the surrounding area and not just the two lungs. K-Means clustering with a cluster size of 2 was used to find the centre points of the two lungs. A series of erosions and dilations are then performed in order to segment the two lungs out and remove the surrounding area. An example of this can be seen in Figure 4.1.



(a) Base Image (b) Pre-Processed

Figure 4.1: Before and after pre-processing (with false colour)

Pearson Correlation Matrix

For each annotation, we have 11 features, as listed below (Hancock 2018).

1. Height of the bounding box
2. Width of the bounding box
3. Subtlety - how obvious the tumour is
4. Internal structure - internal composition of the nodule
5. Calcification - the pattern of calcification, if present
6. Sphericity - the three-dimensional shape of the nodule in terms of its roundness from linear to round
7. Margin - how well-defined the nodule margin is from poorly defined to sharp
8. Lobulation - degree of lobulation ranging from none to marked
9. Spiculation - the extent of spiculation present
10. Texture
11. Malignancy - subjective assessment of the likelihood of malignancy, assuming the scan originated from a 60-year-old male smoker

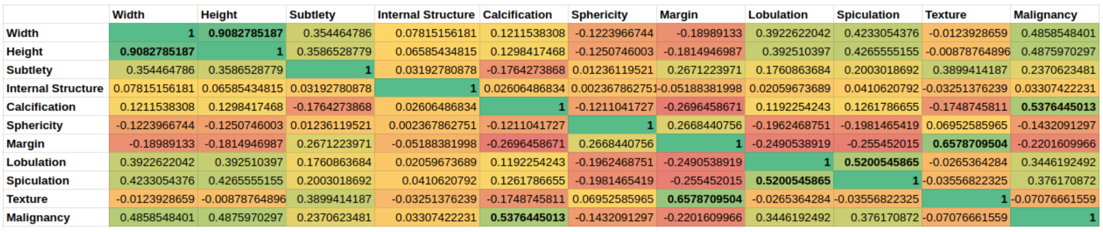
We can use the Pearson Correlation Coefficient (PCC) to determine which of these variables are linked and how strongly.

The PCC is used to determine the relationship between two variables. Adler & Parmryd (2010) writes a "range of +1 (perfect correlation) to -1 (perfect but negative correlation) with 0 denoting the absence of a relationship". The formula for computing the coefficient (r) can be seen in Equation 4.1.

$$r = \frac{\sum(x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum(x_i - \bar{x})^2 \sum(y_i - \bar{y})^2}} \tag{4.1}$$

I had already written an implementation for the PCC in MATLAB so I decided to use that code rather than re-implementing it in Python. The code can be seen in Appendix B.

Running the code on the dataset we get the matrix seen in Table 4.3. Correlations greater than 0.5 are highlighted in bold. From looking at this matrix we can see that Width and Height are very strongly correlated, as expected. The next strongest correlation is between Texture and Margin, then Calcification and Malignancy and the last correlation > |0.5| is Lobulation and Spiculation.

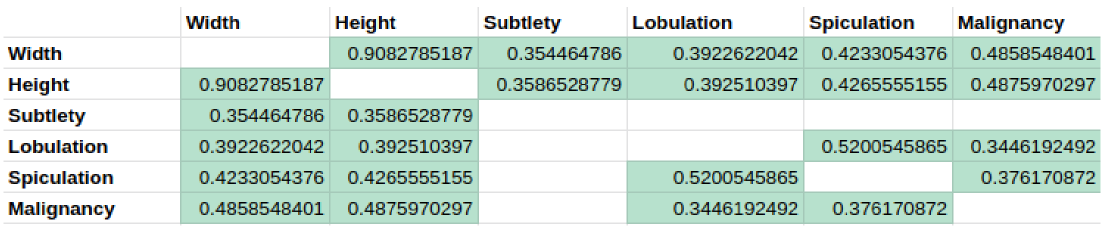


	Width	Height	Subtlety	Internal Structure	Calcification	Sphericity	Margin	Lobulation	Spiculation	Texture	Malignancy
Width	**1**	**0.9082785187**	0.354464786	0.07815156181	0.1211538308	-0.1223966744	-0.18989133	0.3922622042	0.4233054376	-0.0123928659	0.4858548401
Height	**0.9082785187**	**1**	0.3586528779	0.06585434815	0.1298417468	-0.1250746003	-0.1814946987	0.392510397	0.4265555155	-0.00878764896	0.4875970297
Subtlety	0.354464786	0.3586528779	**1**	0.03192780878	-0.1764273868	0.01236119521	0.2671223971	0.1760863684	0.2003018692	0.3899414187	0.2370623481
Internal Structure	0.07815156181	0.06585434815	0.03192780878	**1**	0.02606486834	0.002367862751	-0.05188381998	0.02059673689	0.0410620792	-0.03251376239	0.03307422231
Calcification	0.1211538308	0.1298417468	-0.1764273868	0.02606486834	**1**	-0.1211041727	-0.2696458671	0.1192254243	0.1261786655	-0.1748745811	**0.5376445013**
Sphericity	-0.1223966744	-0.1250746003	0.01236119521	0.002367862751	-0.1211041727	**1**	0.2668440756	-0.1962468751	-0.1981465419	0.06952585965	-0.1432091297
Margin	-0.18989133	-0.1814946987	0.2671223971	-0.05188381998	-0.2696458671	0.2668440756	**1**	-0.2490538919	-0.255452015	**0.6578709504**	-0.2201609966
Lobulation	0.3922622042	0.392510397	0.1760863684	0.02059673689	0.1192254243	-0.1962468751	-0.2490538919	**1**	**0.5200545865**	-0.0265364284	0.3446192492
Spiculation	0.4233054376	0.4265555155	0.2003018692	0.0410620792	0.1261786655	-0.1981465419	-0.255452015	**0.5200545865**	**1**	-0.03556822325	0.376170872
Texture	-0.0123928659	-0.00878764896	0.3899414187	-0.03251376239	-0.1748745811	0.06952585965	**0.6578709504**	-0.0265364284	-0.03556822325	**1**	-0.07076661559
Malignancy	0.4858548401	0.4875970297	0.2370623481	0.03307422231	**0.5376445013**	-0.1432091297	-0.2201609966	0.3446192492	0.376170872	-0.07076661559	**1**

Table 4.3: Pearson Correlation Matrix of all features

A Pearson Correlation Matrix shows us not only strong correlations but also weak correlations. We can see in Table 4.3 that Internal Structure and Sphericity have no correlation to any of the other features at all. Furthermore, Margin and Calcification have very weak correlations with most features. By removing all features that are only correlated with one other feature (Using a threshold of 0.3) we are left with the Matrix seen in Table 4.4

The outcome of using the PCC is that now we have a list of 6 features that are correlated, rather than 11 that are only somewhat. By removing the un-correlated features we can better use a clustering algorithm to split the tumours into classes.



	Width	Height	Subtlety	Lobulation	Spiculation	Malignancy
Width		0.9082785187	0.354464786	0.3922622042	0.4233054376	0.4858548401
Height	0.9082785187		0.3586528779	0.392510397	0.4265555155	0.4875970297
Subtlety	0.354464786	0.3586528779				
Lobulation	0.3922622042	0.392510397			0.5200545865	0.3446192492
Spiculation	0.4233054376	0.4265555155		0.5200545865		0.376170872
Malignancy	0.4858548401	0.4875970297		0.3446192492	0.376170872	

Table 4.4: Thresholded Pearson Correlation Matrix

Clustering for Class selection

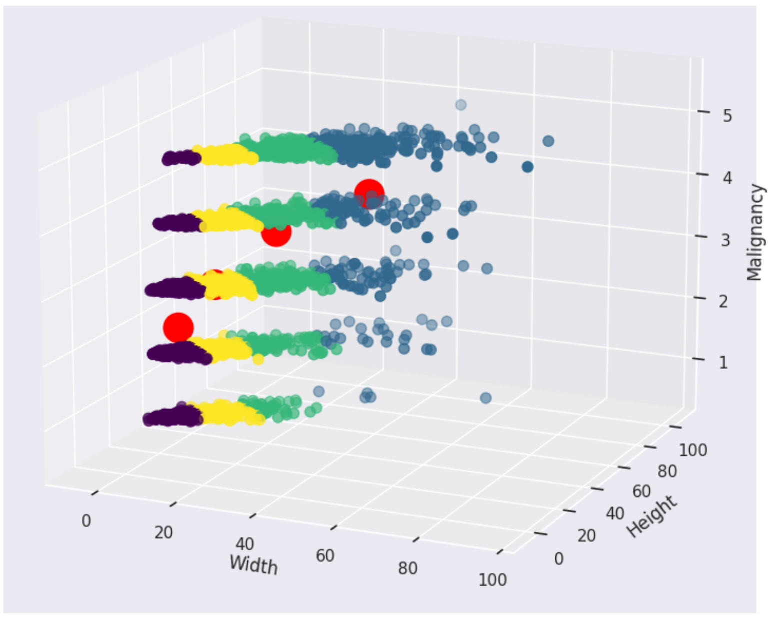


Figure 4.2: Width vs Height vs Malignancy with cluster centres in red.

The tumours in the dataset despite having a plethora of metadata are not categorised. The significant change in visual appearance between the tumours requires that they are split into discrete categories in order to aid training. To accomplish this K-Means clustering is used to cluster the tumours into four categories. The clustering is performed in 6 dimensions using the features previously identified from the person correlation matrix (Width, Height, Subtlety, Lobulation, Spiculation and Malignancy). In Figure 4.2 we can see the data plotted in 3D with each colour identifying a category and each red dot denoting the cluster centre.

Problems Encountered

Originally each tumour was counted as exported as a training sample. This was not seen to be a problem until it came to training. This image has only one annotated tumour in each image, this led to inefficient training as the network was only being informed of the one tumour, even if multiple existed on the image.

This was rectified by having each image annotated with all of the tumours on it, this increased the accuracy of the model considerably. In Figure 4.3 we can see an example of a scan with multiple tumours, before this issue was resolved this image would have appeared twice, each time only showing one of the tumours.

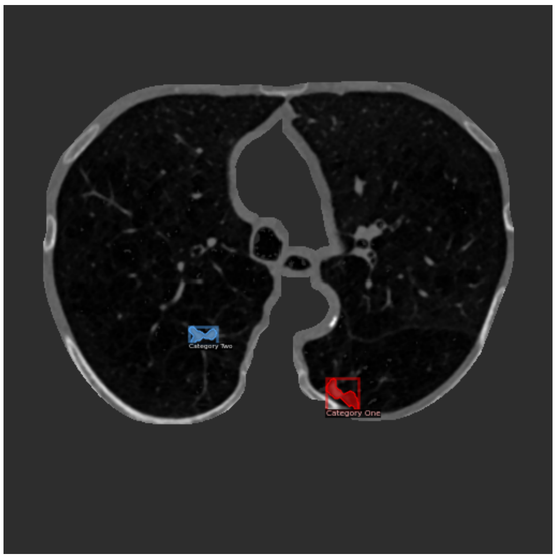


Figure 4.3: LIDC-IDRI Patient 0606 Slice 424 with Two Tumours.

4.3.2 Pre-Processing Files

prepare_dataset.py

This script performs all the image pre-processing required on the dataset and saves each CT slice with a tumour to its own .png file. The use of multiprocessing using a number of sup-processes is used to speed up the process. Even with the use of multiprocessing it still takes a few hours to process all of the scans. The code for this can be seen in Appendix A.

utils.py

The utils.py file contains the functions used to perform the pre-processing used in prepare_dataset.py. It contains multiple functions to provide filtering, segmentation and the combination of masks. The code for this can be seen in Appendix A.

cluster.py

Cluster.py performs K-Means clustering of the tumours to determine the labels used during classification. The model created is then saved to be used during the annotation process. The code for this can be seen in Appendix A.

annotate.py

The annotate script is used to create the train and test text files. Originally the functionality was included in the prepare_dataset.py file but it was extracted into its own file so tweaks could be made to the classes without having to process all of the images again. Using the clustering model created by cluster.py each tumour is assigned a label. The code for this can be seen in Appendix A.

4.3.3 Training Code

Dependencies

The Dependencies section is the smallest of the sections. It installs the `pyyaml` and `detectron2` packages that are either not installed by default or the wrong version is installed. After the necessary packages are installed the run time needs to restart so the new packages can be loaded. After the restart, the rest of the needed dependencies such as `numpy, detectron2, os & random` are loaded. The code for this can be seen in Appendix A.

Load Dataset

Detectron2 expects the datasets to be in a specific format. This section reads in the .txt files made by annotate.py and converts them into the required format.

The dataset has a very large class imbalance. The class containing the largest tumours, class 3, only occurs 120 times in the training dataset and 54 times in the test dataset. To remedy this we artificially increase the amount of training data for class 3, each slice is submitted five times instead of just one. The code for this can be seen in Appendix A.

Training

The training section firstly setups up the config for training, this includes loading a pre-trained model (to transfer learn from). Setting the training and test weights. And finally configuring the hyperparameters to the values discussed in the Design sub-section.

Once the config has been set we create a new folder for the model's files to be saved to then create a Trainer object and start the training process. The code for this can be seen in Appendix A.

4.3.4 Evaluating the Model

Intersection Over Union (IoU)

Intersection Over Union is a metric that measures to what extent two bounding boxes overlap. An IoU of 1 is a perfect overlap where 0 dictates that the boxes do not overlap at all. This is used to 'score' the predictions made by the network. If a prediction has an IoU that is greater than a set threshold then that is counted as a true positive. If the IoU is less than the threshold then it is a false positive. In this use case, an IoU of > 0 is considered positive as it will draw the radiologist to the area for further consideration.

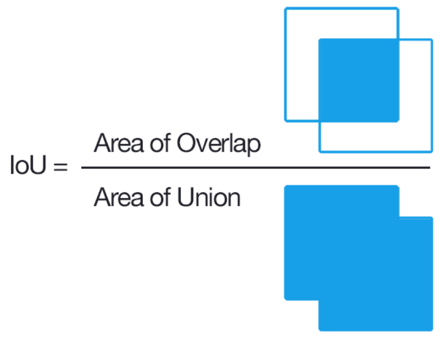


Figure 4.4: Intersection over Union (Rosebrock 2016)

Recall, Precision & F1 Score

Recall, also known as sensitivity, is the proportion of successfully classified 'things' (In this case tumours). A model that has no false negatives would have a recall score of 1.

$$Recall = \frac{True\ Positives}{True\ Positives\ +\ False\ Negatives} \tag{4.2}$$

Precision tells us of all the predictions the model made how many were true. A model with no false positives would have a precision of 1.

$$Precision = \frac{True\ Positives}{True\ Positives\ +\ False\ Positives} \tag{4.3}$$

The F1 score, also known as the Sørensen–Dice coefficient, is the harmonic mean of the recall and precision figures. It more accurately reflects the performance of the model than accuracy can. A model that always predicts everyone has cancer would be very accurate, as it would successfully categorise all cancer cases. However, as the majority of the population do not have cancer, a simple accuracy measurement would score this model very highly when in reality it is completely useless. The F1 score of this model would be zero.

$$F_1 = 2 * \frac{Recall * Precision}{Recall + Precision} \tag{4.4}$$

Problems Encountered

The code for evaluating the entire dataset had a bug that remained undetected for some time. An 'if' statement that should have been an 'elif' resulted in false-positive results being counted twice. The bug was caught when it was noticed the sum of False Positives + True Positives + False Negatives was greater than the number of predictions made by the network. Once this bug was fixed accurate F1 scores were computed.

4.3.5 Evaluation Code

Evaluation Setup

Before we can use the trained model we must first load it into a predictor and set the threshold. We set the threshold low to extract most predictions from the network. After that, we create an IoU (subsection 4.3.4) function which is used to compare the predictions against the ground truth. The code for this can be seen in Appendix A.

Single Image Inference

This function performs inference on a single image (Or a random sample from the test dataset) and reports the IoU score(s). An example outcome from this function can be seen in Figure 4.5. We can see that this prediction is a very accurate one with an IoU of 0.8647 (With 1 being perfect), the model is also very confident that this is a tumour at 89.27% certainty. The code for this can be seen in Appendix A.

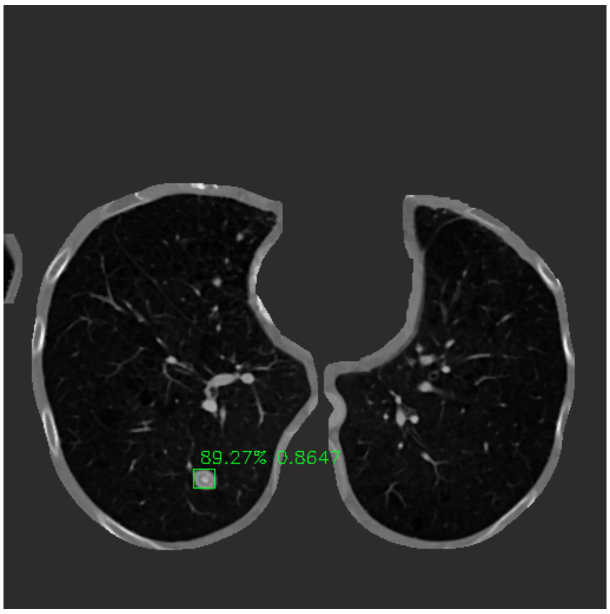


Figure 4.5: Single inference on LIDC-IDRI 0119 Slice 164.

Full Dataset Evaluation

This section runs inference on every single image in the test dataset. It records the number of True Positives, False Positives and False Negatives for each class and overall. At the end, the Precision, Recall and F1 scores are calculated and displayed. The code for this can be seen in Appendix A.

4.4 Training and Results

An iterative approach was taken for training the model, due to the speed benefits of transfer learning we can afford to train the model many times in a short space of time. This section will discuss each iteration of the training process, what went well, what didn't and how the model performed.

4.4.1 First Train

For the first training session, the model was trained for 1000 epochs on all four categories of tumours. The Third category was artificially inflated in the training dataset to counteract the dataset imbalance. Overall there were 9883 training images with a ratio of 6730:831:1755:567 between the four categories respectively.

Since Mask RCNN is an extension to Faster RCNN we have graphs for both the Faster RCNN section and Mask RCNN section.

FasterRCNN Graphs

Looking at the graphs in Figure 4.9 we can see that the training had limited success, accuracy isn't increasing with time like we would expect and false negative isn't decreasing.

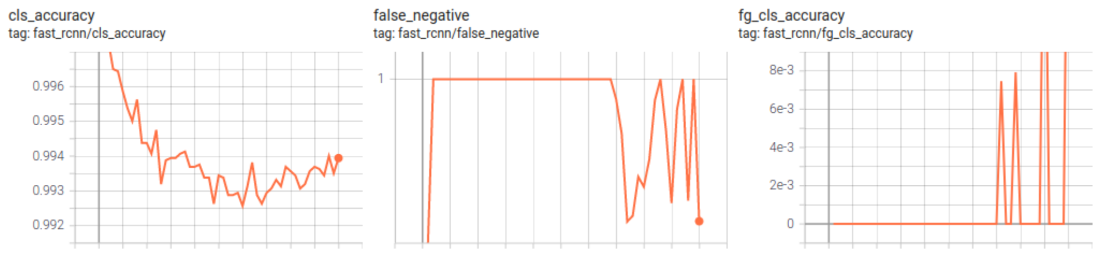


Figure 4.6: Faster RCNN training graphs.

MaskRCNN Graphs

The graphs in Figure 4.7 also show us that something is definitely wrong, the accuracy is somewhat increasing but not ideal, the false negative rate is jumping up and down instead of decreasing.

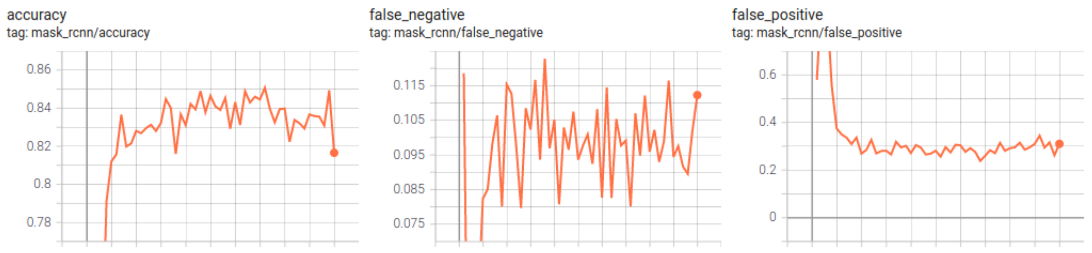


Figure 4.7: Mask RCNN training graphs.

Overall Loss

The overall loss graph seen in figure 4.8 shows us that training stagnated and stayed at ≈ 0.5 thought the thousand epochs. At the end of training the total loss was 0.547.

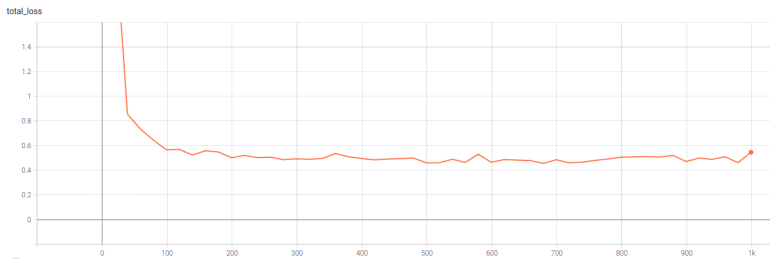


Figure 4.8: Overall Loss Graph.

Confusion Matrix and Evaluation

Overall	**Positive**	**Negative**
Positive	1781 (TP)	2112 (FN)
Negative	934 (FP)	N/A (TN)
F1 Score:	**Recall:**	0.4575
0.539	**Precision:**	0.656

Table 4.5: Overall Confusion Matrix

Table 4.5 shows us that this model is not very accurate with an overall F1 score of 0.539. In addition, the model fails to identify many tumours resulting in a high false-negative rate, our requirements elicitation identified that false negatives are to be minimised. Looking at the Confusion Matrix for category zero (Table 4.6a) we see that the F1 Score is only 0.4781 compared to 0.7114 for category one (Table 4.6b) or even 0.6585 for category three (Table 4.6d). This indicates that category one may be responsible for the large training error.

Next Steps

It is obvious that the network has not trained to its fullest extent here, the poor statistics of category zero suggest it is responsible for reducing the accuracy of the other three categories. For the next train, just one category will be trained to see how accurate training can get in isolation.

Category 0	Positive	Negative
Positive	1176 (TP)	1648 (FN)
Negative	919 (FP)	N/A (TN)
F1 Score:	**Recall:**	0.4164
0.4781	**Precision:**	0.5613

(a) Confusion Matrix Category 0

Category 1	Positive	Negative
Positive	196 (TP)	150 (FN)
Negative	9 (FP)	N/A (TN)
F1 Score:	**Recall:**	0.5665
0.7114	**Precision:**	0.9561

(b) Confusion Matrix Category 1

Category 2	Positive	Negative
Positive	382 (TP)	287 (FN)
Negative	5 (FP)	N/A (TN)
F1 Score:	**Recall:**	0.571
0.7235	**Precision:**	0.9871

(c) Confusion Matrix Category 2

Category 3	Positive	Negative
Positive	27 (TP)	27 (FN)
Negative	1 (FP)	N/A (TN)
F1 Score:	**Recall:**	0.5
0.6585	**Precision:**	0.9643

(d) Confusion Matrix Category 3

Table 4.6: Confusion Matrices for all categories

4.4.2 Second Train

After the disappointing results of the first training session, I decided to train a model on just one category of tumour, in this case, category 3. Since category three had the least amount of training data it would rule out the issue of insufficient training data if a successful model could be trained. As such the data was not augmented and the model was trained on only 119 images.

FasterRCNN Graphs

Looking at the graphs in Figure 4.9 we can see that the accuracy is approaching one and false negatives are approaching zero. This indicates that the model has been very successful at learning the training dataset.

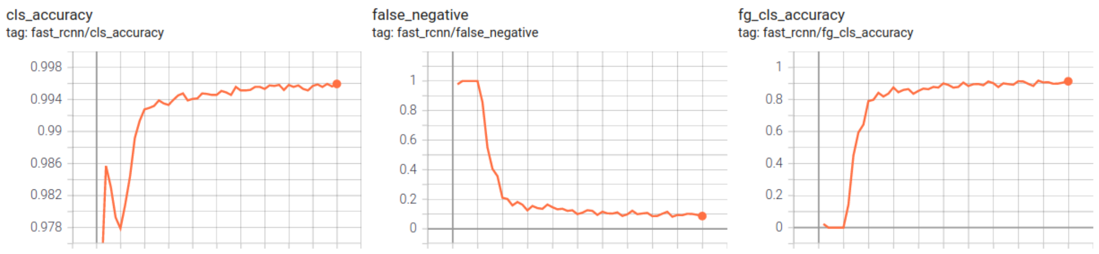


Figure 4.9: Faster RCNN training graphs.

MaskRCNN Graphs

The graphs in Figure 4.10 show a similar story, the accuracy for the masks is approaching one however it is only 0.9531 at the end of the training, the false-negative amount is declining and is 0.04205 at the end of training and given more epochs it is likely this would decrease even more. The false positives reached a value of ≈ 0.05 very quickly in the training process and stayed at that level throughout.

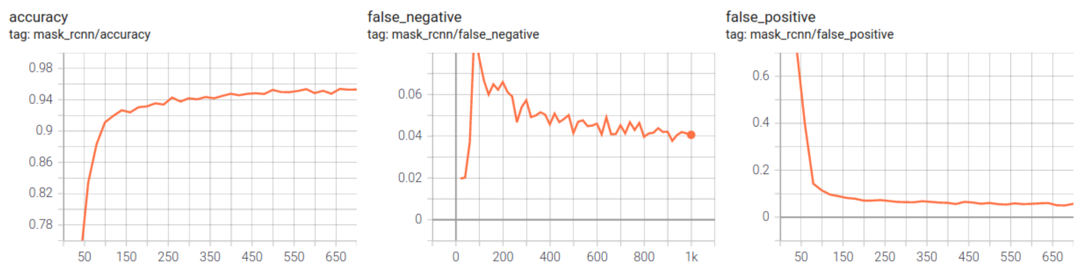


Figure 4.10: Mask RCNN training graphs.

Overall Loss

The overall loss graph seen in figure 4.11 seems to indicate that training after epoch ≈ 300 is unnecessary as the graph is flat, however, we know from the FasterCNN and MaskRCNN graphs that this is not so as improvements were being made. This is why just looking at an overall loss is not the best metric for gauging training progress. Nevertheless, the overall loss value at the end of the training was 0.1355, a lot better than the previous training's value of 0.547.

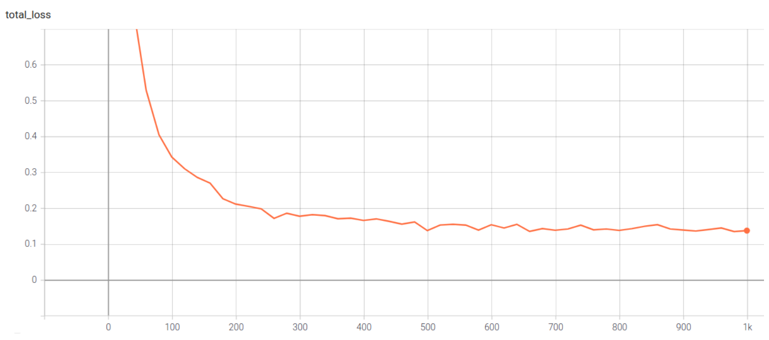


Figure 4.11: Overall Loss Graph.

Confusion Matrix and Evaluation

Category 3	**Positive**	**Negative**
Positive	50	4
Negative	3	N/A
F1 Score:	**Recall:**	0.9259
0.9346	**Precision:**	0.9434

Table 4.7: Confusion Matrix and F1 Score for Category 3

From the confusion matrix seen in Table 4.7 we can see that the network managed to become very accurate at detecting Category three tumours when only being trained on the single category, much more accurate than during the first training session. Furthermore, an F1 Score of 0.9346 is impressive, however, due to there only being 54 instances of category 3 in the testing set we do have to acknowledge the accuracy may be lower in a real-world setting.

Next Steps

Now that we know the model can train to a high degree for one class the next step is to train it on multiple classes and try to replicate the success there. We saw that category zero was the worst-performing in the first train, as such the next model will be trained on categories 1, 2 and 3. The decision to drop category zero was made due to time constraints on the project, the very low precision, recall and F1 scores were seen for category zero in the first train singled it out as the bottleneck that was holding back effective training of the other categories.

4.4.3 Third Train

As previously discussed this model will be trained on categories 1, 2 and 3. Category 3 will be augmented again to increase the amount of training data and mitigate the dataset imbalance.

FasterRCNN Graphs

The graphs in Figure 4.12 clearly show trends in the right direction, accuracy is increasing and almost at 1. The false-negative rate is falling, although it still has a significant way to go. The foreground class accuracy is also increasing, but like the false-positive rate, it could be higher. These graphs indicate that training is going well although more epochs are needed.

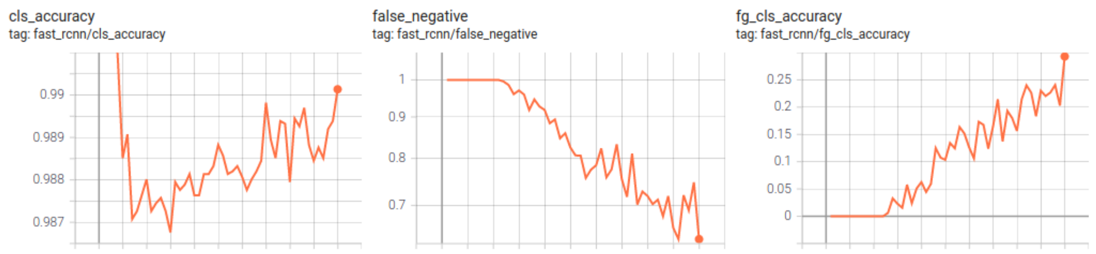


Figure 4.12: Faster RCNN training graphs.

MaskRCNN Graphs

The graphs in Figure 4.13 show very promising signs, accuracy is near one, the flase negative rate is very low at 0.075 at the end of training and the false positive rate is steady at 0.2.

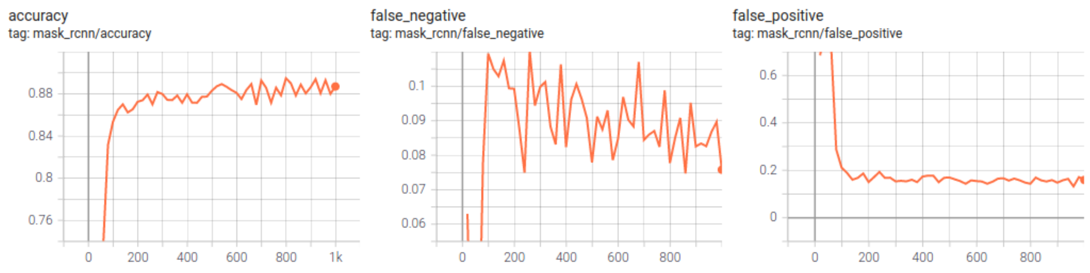


Figure 4.13: Mask RCNN training graphs.

Overall Loss

The overall loss graph seen in Figure 4.14 shows that the loss value, while a little unstable, is steadily decreasing. At the end of training the total loss value was 0.3386, higher than the loss of the previous train but as discussed when looking at the Faster RCNN graphs, more epochs may be needed.

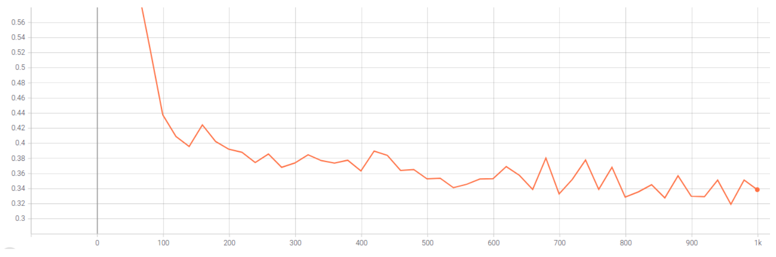


Figure 4.14: Overall Loss Graph.

Confusion Matrix and Evaluation

The confusion matrices for this training session are somewhat disappointing. The F1 score for category 2 is not great at all at 0.6568, the F1 scores for the other two category's are high at ≈ 0.8 but we have already seen that we can train a model on category 3 to an F1 score of 0.9346. While not a bad model overall with an F1 of 0.7183 there is still room for improvement.

Category 1	**Positive**	**Negative**
Positive	285 (TP)	61 (FN)
Negative	41 (FP)	N/A
F1 Score:	**Recall:**	0.8237
0.8482	**Precision:**	0.8742

(a) Confusion Matrix for Category 1

Category 2	**Positive**	**Negative**
Positive	530 (TP)	139 (FN)
Negative	415 (FP)	N/A
F1 Score:	**Recall:**	0.7922
0.6568	**Precision:**	0.5608

(b) Confusion Matrix for Category 2

Category 3	**Positive**	**Negative**
Positive	48 (TP)	6 (FN)
Negative	15 (FP)	N/A
F1 Score:	**Recall:**	0.8889
0.8205	**Precision:**	0.7619

(c) Confusion Matrix for Category 3

Overall	**Positive**	**Negative**
Positive	863 (TP)	206 (FN)
Negative	471 (FP)	N/A
F1 Score:	**Recall:**	0.8073
0.7183	**Precision:**	0.6469

(d) Overall Confusion Matrix

Table 4.8: Confusion Matrices for all categories and overall

Next Steps

The current issue is that a model trained on all three of the categories results in a less accurate prediction for category three tumours than a model just trained on category three. A potential solution to this would be to use model stacking, combining the outputs of multiple models into one unified output. This technique would allow us to train each category for more or fewer epochs depending on the needs of that category, furthermore, we can independently adjust the threshold for each of the models during evaluation to achieve the best results possible.

4.4.4 Final Model

The trained model for category three was kept and two new models were trained for categories one and two. Category one was trained for 1000 epochs and category two for 1500. In this section, we will be discussing the model stacking and final evaluation for the model. The downside of the model stacking approach is that the segmentation masks get discarded. Given more time it is possible that they could be re-implemented.

Combining the models

Now we have three models trained we need to unify the three outputs into one. Independent thresholds were used for each model, these figures were settled upon after a process of trial and error. The category one model had a testing threshold of 0.4, category 2 was 0.6 and category 3 at 0.9. Each model then performs inference on the image, this provides three separate outcomes. It is very possible that the category 2 model has detected a category 1 tumour and vice-versa. Because of this, a system to determine which predictions are predicting the same potential tumour is needed. To accomplish this the IoU algorithm (Section 4.3.4) is used. IoU is run against the predictions and those that have overlapping bounding boxes are grouped. The group of predictions is then converted into a single prediction by combining the multiple bounding boxes into one. The code for this can be seen in Appendix A.

Confusion Matrix and Evaluation

Category 1	**Positive**	**Negative**
Positive	304 (TP)	42 (FN)
Negative	112 (FP)	N/A
F1 Score:	**Recall:**	0.8786
0.7979	**Precision:**	0.7308

(a) Confusion Matrix for Category 1

Category 2	**Positive**	**Negative**
Positive	497 (TP)	172 (FN)
Negative	75 (FP)	N/A
F1 Score:	**Recall:**	0.7429
0.8010	**Precision:**	0.8689

(b) Confusion Matrix for Category 2

Category 3	**Positive**	**Negative**
Positive	51 (TP)	3 (FN)
Negative	13 (FP)	N/A
F1 Score:	**Recall:**	0.9444
0.8644	**Precision:**	0.7969

(c) Confusion Matrix for Category 3

Overall	**Positive**	**Negative**
Positive	852 (TP)	217 (FN)
Negative	200 (FP)	N/A
F1 Score:	**Recall:**	0.7970
0.8034	**Precision:**	0.8099

(d) Overall Confusion Matrix

Table 4.9: Confusion Matrices for all categories and overall

The confusion matrices (Table 4.9) for the final model show that the model stacking approach was a resounding success. An overall F1 score of 0.8034 (Table 4.9d) is a significant improvement on the previously trained model's F1 of 0.7183 (Table 4.8d). Furthermore, the precision has improved greatly from 0.6469 to 0.8099. Unfortunately, the recall has decreased ever so slightly (by 0.0103), however, this is worth the huge increase in precision. Overall model stacking has resulted in a very accurate model (on three of the four categories), has an excellent recall rate, a metric identified as important, whilst also having a very high precision rate ensuring that the false positive rate isn't too high.

Final Results

Figure 4.15 shows six random slices ran through the DeepTumour network. The percentage is an accuracy value denoting how confident DeepTumour is with its prediction. The second value is the IoU value that represents the overlap between the predictions and the ground truth. The second scan, LIDC-IDRI-0834 Slice 61, shows two predictions. A successful one is shown in green and an incorrect one in red. The testing dataset had the category zero tumours removed from it once the decision to not train for them was made.

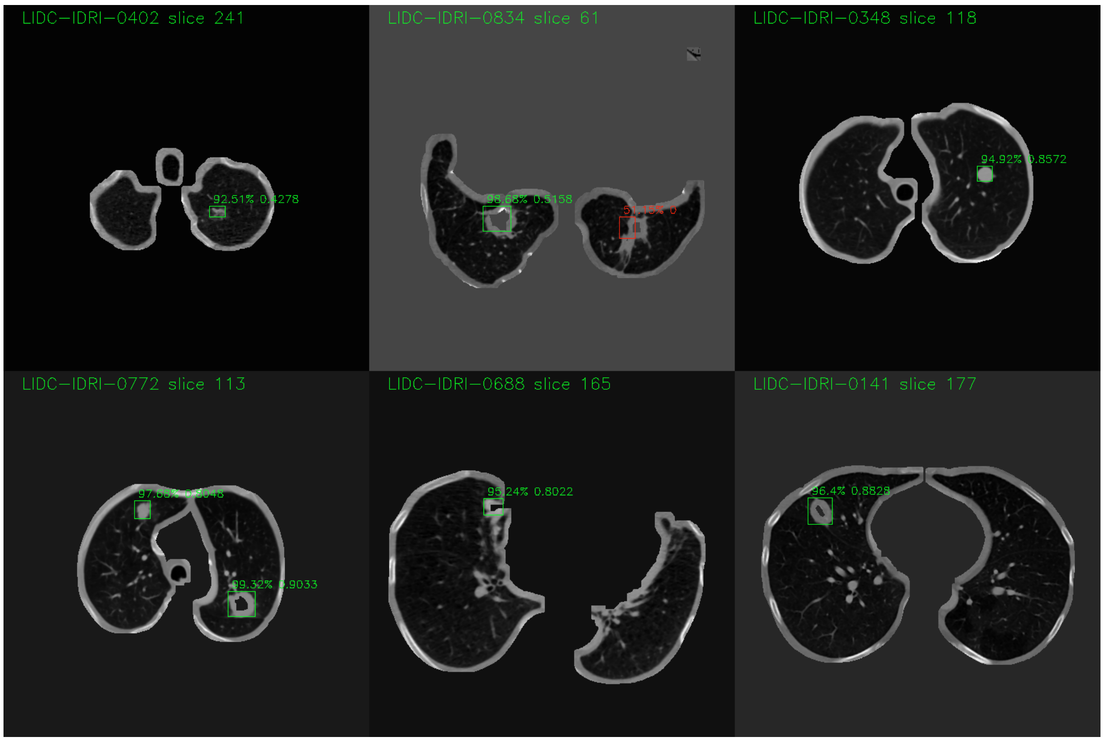


Figure 4.15: 6 random slices and the DeepTumour outputs.

Chapter 5

Project Conclusion

In this project, a model was trained to identify lung tumours with an F_1 score of 0.8304. The model takes an average of 354.76 milliseconds to make predictions on an image (Intel(R) Xeon(R) CPU @ 2.20GHz 13GB RAM NVIDIA Tesla T4). The model has a recall score of 0.7970, this indicates that the false-negative rate of the model is low, a necessity when a false negative on a lung tumour could have fatal consequences. Furthermore, the precision score of 0.8099 achieved by the model informs of a low false-positive rate. This is key as too many false positives could lead to 'The model that cried tumour' causing radiologists to lose faith in DeepTumour.

The model, however, cannot identify all tumours in the LIDC-IDRI dataset. Category 0 was proving very difficult to train, as such the decision was made to focus on the other categories of tumours to ensure the highest performance on those. Due to the use of model stacking, it would be relatively simple to add in the ability to detect this category of tumour at a later date. The modularity of DeepTumour also allows for the sub-models to be replaced if a model is trained to a higher degree for that category, be that due to advances in deep learning or an increase of training data. This feature will allow for a hospital to continuously improve DeepTumour thereby resulting in a further decrease in missed tumours.

Overall this project has been a success, even though category 0 tumours are not covered by DeepTumour currently. This project has still shown that deep learning can be used for the detection of lung tumours, and to an effective degree.

Chapter 6

Reflective Analysis

On the whole, I believe that the project went very well, that is not to say, however, that there were no issues. This chapter has been split into sections, each one discussing a certain aspect of the project, what went well, what didn't and any changes I would have made with hindsight.

6.1 Dataset

In general, I am happy with the selection of the LIDC-IDRI dataset for this project. The dataset was sufficiently large allowing for thousands of images for both training (2698 images) and testing (1069). The dataset was quite unbalanced, however, with significantly more category 0 tumours and significantly fewer of category 3, although this is most likely due to the commonality of these tumours. If I were to undertake this project again I may be tempted by the LUNA16 dataset, a subset of LIDC-IDRI. The LUNA16 dataset aims to standardise LIDC by removing slices of a certain thickness and more.

6.2 Pre-Processing

The pre-processing section had its ups and downs. While the segmentation of the lungs and removal of unnecessary detail works on the whole there are a few completely blank images, this was only caught later in the evaluation stage. This may have negatively impacted the training. With hindsight, I would have added checks to detect any images that are just one colour. A new technique of segmenting them may also yield better results. Transformations such as flips could have also been applied in this section to double the amount of training data available.

6.3 Training

As previously stated category zero tumours were disregarded, this was due to time constraints on the project, I believed it would be better to have a highly accurate model on three of the categories than a mediocre model for all four. Given more time I am confident a model could be trained to a high degree on the category 0 tumours and then integrated into the DeepTumour model stack.

The use of model stacking is a decision I am happy with and would not necessarily change if I were to re-do the project, however, I do believe it could be improved. Currently, the outputs of the sub-model are combined using handwritten code with arbitrary weightings. Given more time a machine learning network could be trained to consolidate the three inputs into one unified output. This network would theoretically be able to create a more accurate final prediction than what was managed by hand, this would allow DeepTumours F_1 score to increase past 0.8 resulting in an even better model.

Another consequence of model stacking is that the segmentation masks predicted by the models are not used. This is a shame as segmentation masks could help radiologists more than a simple bounding box could do. Given more time the segmentation masks could be combined using the same rasterisation process used during pre-processing.

Bibliography

Adler, J. & Parmryd, I. (2010), ‘Quantifying colocalization by correlation: the pearson correlation coefficient is superior to the mander’s overlap coefficient’, *Cytometry Part A* **77**(8), 733–742.

Ahmad, M. O., Markkula, J. & Oivo, M. (2013), Kanban in software development: A systematic literature review, *in* ‘2013 39th Euromicro conference on software engineering and advanced applications’, IEEE, pp. 9–16.

Armato III, S. G., McLennan, G., Bidaut, L., McNitt-Gray, M. F., Meyer, C. R., Reeves, A. P., Zhao, B., Aberle, D. R., Henschke, C. I., Hoffman, E. A. et al. (2011), ‘The lung image database consortium (lidc) and image database resource initiative (idri): a completed reference database of lung nodules on ct scans’, *Medical physics* **38**(2), 915–931.

Awad, M. (2005), ‘A comparison between agile and traditional software development methodologies’, *University of Western Australia* **30**.

Balaji, S. & Murugaiyan, M. S. (2012), ‘Waterfall vs. v-model vs. agile: A comparative study on sdlc’, *International Journal of Information Technology and Business Management* **2**(1), 26–30.

Barba, L. A., Barker, L. J., Blank, D. S., Brown, J., Downey, A. B., George, T., Heagy, L. J., Mandli, K. T., Moore, J. K., Lippert, D., Niemeyer, K. E., Watkins, R. R., West, R. H., Wickes, E., Willing, C. & Zingale, M. (2019), ‘Teaching and learning with jupyter - chapter 5 jupyter notebook ecosystem’. Available from: `https://jupyter4edu.github.io/jupyter-edu-book/jupyter.html` [Accessed 7 February 2021].

Boehm, B. W. (1988), ‘A spiral model of software development and enhancement’, *Computer* **21**(5), 61–72.

Caruana, R. & Niculescu-Mizil, A. (2006), An empirical comparison of supervised learning algorithms, *in* ‘Proceedings of the 23rd international conference on Machine learning’, pp. 161–168.

De Alwis, B. & Sillito, J. (2009), Why are software projects moving from centralized to decentralized version control systems?, *in* ‘2009 ICSE Workshop on Cooperative and Human Aspects on Software Engineering’, IEEE, pp. 36–39.

del Ciello, A., Franchi, P., Contegiacomo, A., Cicchetti, G., Bonomo, L. & Larici, A. R. (2017), ‘Missed lung cancer: when, where, and why?’, *Diagnostic and interventional radiology* **23**(2), 118–125.

Fowler, M., Highsmith, J. et al. (2001), ‘The agile manifesto’, *Software Development* **9**(8), 28–35.

Gillies, S. et al. (2007–), ‘Shapely: manipulation and analysis of geometric objects’.

Gillies, S. et al. (2013–), ‘Rasterio: geospatial raster i/o for Python programmers’.

Girshick, R. (2015), Fast r-cnn, *in* ‘Proceedings of the IEEE international conference on computer vision’, pp. 1440–1448.

Girshick, R., Donahue, J., Darrell, T. & Malik, J. (2014), Rich feature hierarchies for accurate object detection and semantic segmentation, *in* ‘Proceedings of the IEEE conference on computer vision and pattern recognition’, pp. 580–587.

Goodfellow, I., Bengio, Y. & Courville, A. (2016), *Deep Learning*, MIT Press. `http://www.deeplearningbook.org`.

Haagsman, E. (2017), ‘Pycharm community edition and professional edition explained: Licenses and more’. Prague, Czechia: JetBrains. Available from: `https://lncn.ac/jet-brains` [Accessed 7 February 2021].

Hammami, M., Friboulet, D. & Kechichian, R. (2020), Cycle gan-based data augmentation for multi-organ detection in ct images via yolo, *in* '2020 IEEE International Conference on Image Processing (ICIP)', IEEE, pp. 390–393.

Hancock, M. (2018), 'pylidc — pylidc documentation'. Available from: `https://pylidc.github.io/` [Accessed 26 February 2021].

Hancock, M. C. & Magnan, J. F. (2016), 'Lung nodule malignancy classification using only radiologist-quantified image features as inputs to statistical learning algorithms: probing the lung image database consortium dataset with two statistical learning methods', *Journal of Medical Imaging* **3**(4), 044504.

Harris, C. R., Millman, K. J., van der Walt, S. J., Gommers, R., Virtanen, P., Cournapeau, D., Wieser, E., Taylor, J., Berg, S., Smith, N. J., Kern, R., Picus, M., Hoyer, S., van Kerkwijk, M. H., Brett, M., Haldane, A., del R'ıo, J. F., Wiebe, M., Peterson, P., G'erard-Marchant, P., Sheppard, K., Reddy, T., Weckesser, W., Abbasi, H., Gohlke, C. & Oliphant, T. E. (2020), 'Array programming with NumPy', *Nature* **585**(7825), 357–362.

He, K., Gkioxari, G., Dollár, P. & Girshick, R. (2017), Mask r-cnn, *in* 'Proceedings of the IEEE international conference on computer vision', pp. 2961–2969.

He, K., Zhang, X., Ren, S. & Sun, J. (2016), Deep residual learning for image recognition, *in* 'Proceedings of the IEEE conference on computer vision and pattern recognition', pp. 770–778.

Huo, M., Verner, J., Zhu, L. & Babar, M. A. (2004), Software quality and agile methods, *in* 'Proceedings of the 28th Annual International Computer Software and Applications Conference, 2004. COMPSAC 2004.', IEEE, pp. 520–525.

Institute, P. (2017), *A Guide to the Project Management Body of Knowledge (PMBOK(R) Guide–Sixth Edition / Agile Practice Guide Bundle*, PMBOK® Guide, Project Management Institute.

Investigators, I. E. L. C. A. P. (2006), 'Survival of patients with stage i lung cancer detected on ct screening', *New England Journal of Medicine* **355**(17), 1763–1771.

Islam, K. M., Jiang, X., Anggondowati, T., Lin, G. & Ganti, A. K. (2015), ‘Comorbidity and survival in lung cancer patients’, *Cancer Epidemiology and Prevention Biomarkers* **24**(7), 1083.

JetBrains (n.d.), ‘What is a perpetual fallback license?’. Prague, Czechia: JetBrains. Available from: `https://sales.jetbrains.com/hc/en-gb/articles/207240845-What-is-perpetual-fallback-license` [Accessed 7 February 2021].

Jiang, J., Hu, Y.-C., Liu, C.-J., Halpenny, D., Hellmann, M. D., Deasy, J. O., Mageras, G. & Veeraraghavan, H. (2018), ‘Multiple resolution residually connected feature streams for automatic lung tumor segmentation from ct images’, *IEEE transactions on medical imaging* **38**(1), 134–144.

Jimenez-del Toro, O., Müller, H., Krenn, M., Gruenberg, K., Taha, A. A., Winterstein, M., Eggel, I., Foncubierta-Rodríguez, A., Goksel, O., Jakab, A. et al. (2016), ‘Cloud-based evaluation of anatomical structure segmentation and landmark detection algorithms: Visceral anatomy benchmarks’, *IEEE transactions on medical imaging* **35**(11), 2459–2475.

Kuhn, M. & Johnson, K. (2013), Data pre-processing, *in* ‘Applied predictive modeling’, Springer, pp. 27–59.

LeCun, Y., Bengio, Y. & Hinton, G. (2015), ‘Deep learning’, *nature* **521**(7553), 436–444.

Lei, H., Ganjeizadeh, F., Jayachandran, P. K. & Ozcan, P. (2017), ‘A statistical analysis of the effects of scrum and kanban on software development projects’, *Robotics and Computer-Integrated Manufacturing* **43**, 59–67.

Li, J., Conradi, R., Bunse, C., Torchiano, M., Slyngstad, O. P. N. & Morisio, M. (2009), ‘Development with off-the-shelf components: 10 facts’, *IEEE software* **26**(2), 80–87.

Lin, T.-Y., Maire, M., Belongie, S., Hays, J., Perona, P., Ramanan, D., Dollár, P. & Zitnick, C. L. (2014), Microsoft coco: Common objects in context, *in* ‘European conference on computer vision’, Springer, pp. 740–755.

Litjens, G., Kooi, T., Bejnordi, B. E., Setio, A. A. A., Ciompi, F., Ghafoorian, M., Van Der Laak, J. A., Van Ginneken, B. & Sánchez, C. I. (2017), ‘A survey on deep learning in medical image analysis’, *Medical image analysis* **42**, 60–88.

Liu, M., Dong, J., Dong, X., Yu, H. & Qi, L. (2018), Segmentation of lung nodule in ct images based on mask r-cnn, *in* '2018 9th International Conference on Awareness Science and Technology (iCAST)', IEEE, pp. 1–6.

Metz, L., Maheswaranathan, N., Cheung, B. & Sohl-Dickstein, J. (2018), 'Meta-learning update rules for unsupervised representation learning', *arXiv preprint arXiv:1804.00222* .

Microsoft (n.d.), 'Compare visual studio for mac and pc'. Washington, USA: Microsoft. Available from: `https://visualstudio.microsoft.com/vs/mac/#vs_mac_table` [Accessed 7 February 2021].

Muşlu, K., Brun, Y., Holmes, R., Ernst, M. D. & Notkin, D. (2012), 'Speculative analysis of integrated development environment recommendations', *ACM SIGPLAN Notices* **47**(10), 669–682.

NIH Clinical Center (2018), 'Nih clinical center releases dataset of 32,000 ct images'. Maryland, USA: National Institutes of Health. Available from: `https://www.nih.gov/news-events/news-releases/nih-clinical-center-releases-dataset-32000-ct-images` [Accessed 17 February 2021].

Ohno, T. (1988), *Toyota production system: beyond large-scale production*, crc Press.

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Kopf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J. & Chintala, S. (2019), Pytorch: An imperative style, high-performance deep learning library, *in* H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox & R. Garnett, eds, 'Advances in Neural Information Processing Systems 32', Curran Associates, Inc., pp. 8024–8035.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M. & Duchesnay, E. (2011), ‘Scikit-learn: Machine learning in Python’, *Journal of Machine Learning Research* **12**, 2825–2830.

Perscheid, M., Siegmund, B., Taeumel, M. & Hirschfeld, R. (2017), ‘Studying the advancement in debugging practice of professional software developers’, *Software Quality Journal* **25**(1), 83–110.

Pohlen, T., Hermans, A., Mathias, M. & Leibe, B. (2017), Full-resolution residual networks for semantic segmentation in street scenes, *in* ‘Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition’, pp. 4151–4160.

Redmon, J., Divvala, S., Girshick, R. & Farhadi, A. (2016), You only look once: Unified, real-time object detection, *in* ‘Proceedings of the IEEE conference on computer vision and pattern recognition’, pp. 779–788.

Redmon, J. & Farhadi, A. (2018), ‘Yolov3: An incremental improvement’, *arXiv preprint arXiv:1804.02767* .

Rehkopf, M. (n.d.), ‘ Kanban vs. scrum: which agile are you? ’. Sydney, Australia: Atlassian Available from: `https://www.atlassian.com/agile/kanban/kanban-vs-scrum` [Accessed 23 March 2021].

Ren, S., He, K., Girshick, R. & Sun, J. (2015), ‘Faster r-cnn: Towards real-time object detection with region proposal networks’, *arXiv preprint arXiv:1506.01497* .

Ronneberger, O., Fischer, P. & Brox, T. (2015), U-net: Convolutional networks for biomedical image segmentation, *in* ‘International Conference on Medical image computing and computer-assisted intervention’, Springer, pp. 234–241.

Rosebrock, A. (2016), ‘Intersection over Union (IoU) for object detection ’. Available from: `https://www.pyimagesearch.com/2016/11/07/intersection-over-union-iou-for-object-detection/` [Accessed 25 March 2021].

Shen, D., Wu, G. & Suk, H.-I. (2017), ‘Deep learning in medical image analysis’, *Annual review of biomedical engineering* **19**, 221–248.

Somasundaram, R. (2013), *Git: Version control for everyone*, Packt Publishing Ltd.

Stoneburner, J. D. (1999), Project management methods for accelerated product development, PhD thesis, Citeseer.

TCIA Team (2020), 'LIDC-IDRI'. Maryland, USA: National Cancer Institute. Available from: `https://wiki.cancerimagingarchive.net/display/Public/LIDC-IDRI` [Accessed 26 February 2021].

Tong, G., Li, Y., Chen, H., Zhang, Q. & Jiang, H. (2018), 'Improved u-net network for pulmonary nodules segmentation', *Optik* **174**, 460–469.

Torre, L. A., Siegel, R. L. & Jemal, A. (2016), Lung cancer statistics, *in* 'Lung cancer and personalized medicine', Springer, pp. 1–19.

Wang, B. & Gong, N. Z. (2018), Stealing hyperparameters in machine learning, *in* '2018 IEEE Symposium on Security and Privacy (SP)', IEEE, pp. 36–52.

Wu, Y., Kirillov, A., Massa, F., Lo, W.-Y. & Girshick, R. (2019), 'Detectron2', `https://github.com/facebookresearch/detectron2`.

Yan, K., Wang, X., Lu, L. & Summers, R. M. (2018), 'Deeplesion: automated mining of large-scale lesion annotations and universal lesion detection with deep learning', *Journal of Medical Imaging* **5**(3), 036501.

Zhu, W., Liu, C., Fan, W. & Xie, X. (2018), Deeplung: Deep 3d dual path nets for automated pulmonary nodule detection and classification, *in* '2018 IEEE Winter Conference on Applications of Computer Vision (WACV)', IEEE, pp. 673–681.

Appendix A

Source Code

Github Repository

https://github.com/JoePittsy/DeepTumour/

prepare_dataset.py

https://github.com/JoePittsy/DeepTumour/blob/main/pre-processing/prepare_dataset.py

utils.py

https://github.com/JoePittsy/DeepTumour/blob/main/pre-processing/utils.py

clutser.py

https://github.com/JoePittsy/DeepTumour/blob/main/pre-processing/cluster.py

annotate.py

https://github.com/JoePittsy/DeepTumour/blob/main/pre-processing/annotate.py

Training Notebook

https://github.com/JoePittsy/DeepTumour/blob/main/pre-processing/DeepTumor_Training.py

Testing Notebook

https://github.com/JoePittsy/DeepTumour/blob/main/training/DeepTumor_
Training.ipynb

Appendix B

Pearson Correlation

```
function r = pearson_correlation(x, y)
    x_hat = mean(x);
    sxx = (1/ (length(x) - 1)) * sum(power(x - x_hat, 2));
    y_hat = mean(y);
    syy = (1/ (length(y) - 1)) * sum(power(y - y_hat, 2));
    sxy = (1/ (length(y) - 1)) * sum((x - x_hat) .* (y -y_hat));

    r = sxy / power(sxx*syy,0.5);
end
```