HELLENIC MEDITERRANEAN UNIVERSITY
DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING
SCHOOL OF ENGINEERING

Integration of gene expressions with polymorphism data in Systems Biology

Comparison with imaging techniques in Gliomas

A Bachelor of Science Thesis
Iosifidis Petros Konstantinos (4132)

Supervisor A: Prof. Dr. Tsiknakis Manolis
Supervisor B: Dr. Koumakis Lefteris

Heraklion, September 2022

Acknowledgements

I would like to start this thesis off by thanking several people. First and foremost I would like to thank my family for their continuous love and support throughout the journey of my life and subsequently my bachelors' degree. I would like to thank the educational staff of HMU for providing me the tools, knowledge and inspiration to attempt the field of computer science. A special thanks to Dr. Manolis Tsiknakis for supervising this thesis. Furthermore, a special thanks to Dr. Kostas Marias for introducing me to CV, taking me up on my wild goose chase and the pep talks given in every lecture. A warm thanks to Evangelia Maniadi for her quick walk-throughs through the bureaucracy of the university system. Lastly I would like to address a special thanks to Dr. Lefteris Koumakis for not losing hope in me and despite our limited time frame coming up with an exciting strategy to tackle the subject of this thesis; his contribution was invaluable.

Thanks, everyone.

-Petros Iosifidis

Abstract

The present work outlines core aspects of machine learning in the fields of radiomics, genomics, transcriptomics and radiogenomics. More specifically, it's attempting through the usage of multi-type data (including medical images, gene expressions, trascriptome expressions) to advance the diagnostic power of predictive models. In the same time, it's trying to advance the survival rate metrics using the same type of data in order to help with cancer correlations and treatment observation and evaluation.

Starting off the reader will understand core concepts of the biomedical field, the nature of the problem as well as the scope and target of this thesis. Continuing we will also give the reader the necessary computational knowledge needed to follow up with the experiments. Moving forward we perform a multi-type experiment attempting to merge radiogenomic classifiers with a better cancer survival rate. Lastly we present our results, give our outlook and discuss about the work done & problems we encountered and close off by pondering over future research.

The begin of the experiments starts with a lengthy preprocessing of approximately 4000 MRI blocks of multiple modalities (FLAIR, T1, T1CE, T2) and generation of custom input objects. Through the use of a DNN, namely a 3D CNN with modified inputs, we establish cancer classification and semantic segmentation into 4 major classes(background, necrotic core/non enhancing tumor, peritumoral edema, enhancing tumor) through the training and evaluation of multiple segmentation models.

Using the imaging data, we extract a plethora of imaging features that we later use in gradient boosting (XGBOOST) to approximate survival prediction from the imaging data analysis. Continuing with the genomic & trascriptomic data, we establish two major classes of "dead" or "alive for over 100 days" and generate classifiers based on the multi-omic profiling of our samples. Lastly we use the multi-omic data to generate powerful regressors for survival rate prediction.

Key words: bioinformatics, radiogenomics, MRI, multi-omics, cancer, gliomas, Data Preprocessing, 3D-CNN, Classification, Regression, Semantic segmentation, tumor classification, Survival prediction

Περίληψη

Η παρούσα εργασία σκιαγραφεί τις βασικές πτυχές της μηχανικής μάθησης στους τομείς της ραδιονομικής, της γονιδιωματικής, της μεταγραφτομικής και της ραδιογονιδιωματικής. Πιο συγκεκριμένα, επιχειρεί μέσω της χρήσης δεδομένων πολλαπλών τύπων (συμπεριλαμβανομένων ιατρικών εικόνων, εκφράσεων γονιδίων, εκφράσεων μεταγραφωμάτων) να προωθήσει τη διαγνωστική δύναμη των προγνωστικών μοντέλων. Ταυτόχρονα, προσπαθεί να προωθήσει τις μετρήσεις του ποσοστού επιβίωσης χρησιμοποιώντας τους ίδιους τύπους δεδομένων, προκειμένου να βοηθήσει με τις συσχετίσεις του καρκίνου και την παρατήρηση και αξιολόγηση της καρκινικής θεραπείας.

Ξεκινώντας ο αναγνώστης θα κατανοήσει τις βασικές έννοιες του βιοϊατρικού τομέα, θα κατανοήσει τη φύση του προβλήματος καθώς και το εύρος και τον στόχο αυτής της διατριβής. Συνεχίζοντας αποτυπώνουμε τον αναγνώστη τις υπολογιστικές γνώσεις που απαιτούνται για την παρακολούθηση των πειραμάτων. Προχωρώντας, πραγματοποιούμε ένα πείραμα πολλαπλών τύπων επιχειρώντας να συγχωνεύσουμε ραδιογονιδιωματικούς ταξινομητές με καλύτερο ποσοστό επιβίωσης από καρκίνο. Τέλος, παρουσιάζουμε τα αποτελέσματά μας, δίνουμε τις προοπτικές μας και συζητάμε για τη δουλειά που έχει γίνει και τα προβλήματα που αντιμετωπίσαμε και κλείνουμε με το στοχασμούς για μελλοντική έρευνα.

Η αρχή των πειραμάτων ξεκινά με μια μακρά προεπεξεργασία περίπου 4000 μπλοκ MRI πολλαπλών τύπων (FLAIR, T1, T1CE, T2) και δημιουργία προσαρμοσμένων αντικειμένων εισαγωγής. Μέσω της χρήσης ενός DNN, συγκεκριμένα ενός τρισδιάστατου CNN με τροποποιημένες εισόδους, καθιερώνουμε την ταξινόμηση του καρκίνου και τη σημασιολογική κατάτμηση σε 4 κύριες κατηγορίες (υπόβαθρο, νεκρωτικός πυρήνας/μη ενισχυτικός όγκος, περιογκικό οίδημα, ενισχυτικός όγκος) μέσω της εκπαίδευσης και της αξιολόγησης έξι μοντέλων πολλαπλής τμηματοποίησης εικόνας.

Χρησιμοποιώντας τα δεδομένα απεικόνισης μας, εξάγουμε μια πληθώρα χαρακτηριστικών απεικόνισης που αργότερα χρησιμοποιούμε στους ταξινομητές ενίσχυσης κλίσης (XGBOOST) για να προσεγγίσουμε την πρόβλεψη επιβίωσης από την ανάλυση δεδομένων απεικόνισης. Συνεχίζοντας με τα γονιδιωματικά και μεταγραφικά δεδομένα, καθιερώνουμε δύο κύριες κατηγορίες «νεκρών» ή «ζωντανών για περισσότερες από 100 ημέρες» και δημιουργούμε ταξινομητές με βάση το πολυ-ομικό προφίλ των δειγμάτων μας. Τέλος, χρησιμοποιούμε τα πολλυ-ομικά δεδομένα για να δημιουργήσουμε έναν ισχυρό παλινδρομητή για την πρόβλεψη του ποσοστού επιβίωσης.

Λέξεις Κλειδιά: βιοπληροφορική, μαγνητική τομογραφία, χαρακτηριστικά ιατρικής εικόνας, 3D-CNN, multi-omics, καρκίνος, προ-επεξεργασία δεδομένων, γλοίωμα, κατηγοριοποίηση, παλινδρόμηση, σημασιολογική κατάτμηση, κατηγοριοποίηση όγκου, πρόβλεψη επιβίωσης

Table of Contents

List of Figures

List of Tables

List of Abbreviations and Symbols

GBM	Glioblastoma
CV	Computer Vision
csv	comma-separated values
FLAIR	Fluid Attenuated Inversion Recovery
LGG	Low Grade Glioma
HGG	High Grade Glioma
miRNA	micro RNA
GAN	Generative Adversarial Network
GD	Gadolinium
CE	Contrast Enhanced
DNA	Deoxyribonucleic acid
RNA	Ribonucleic acid
miRNA	micro RNA
LR	Learning Rate
ML	Machine Learning
AI	Artificial Intelligence
MSE	Mean Square Error
RMSE	Root Mean Square Error
I/O	Input / Output
ncDNA	non coding DNA
mRNA	messenger RNA
ncRNA	non Coding RNA
rRNA	ribosomal RNA
tRNA	transfer RNA
CNS	Central Nervous System
ROI	Region of Interest
NN	Nearest Neighbour
ADASYN	Adaptive Synthetic
SMOTE	Synthetic Minority Over-sampling Technique
IDH	Isocitrate dehydrogenase
CSF	Cerebrospinal Fluid
FC	Fully Connected
MLP	Multi Layer Perceptron

1 Biomedical Literature

Biology is the study of living things. In this section we give a brief overview of core concepts the reader needs to know to understand the problem this thesis is trying to tackle.

1.1 Cell

Dubbed as the smallest unit of life that can live on it's own by the dogma of Biology, the cell is the principal building block of all organisms (even if it's a one cell organism!). It consists of three main parts(fig.1[1]):

1. the cell membrane, surrounds the cell and controls it's I/O stream
2. the cytoplasm, the fluid within the cell that contains multiple smaller cell parts that perform certain functions (energy production, protein forming, etc.)
3. the nucleus, which contains the cell's DNA

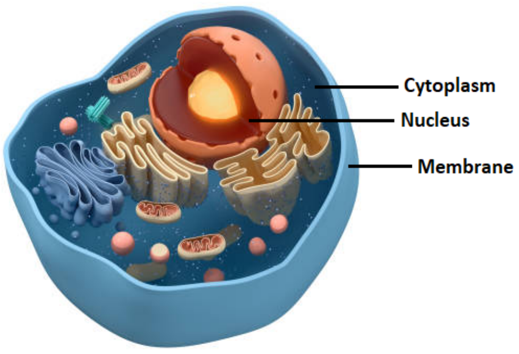


Figure 1: The Cell

Fun fact: the average human consists of more than 30 trillion cells!

1.2 DNA

DNA(fig.2[2]) is a polymer composed of two polynucleotide chains that coil around each other to form a double helix as proposed by Watson & Crick [16]. It carries genetic instructions for the development, growth and reproduction of all known organisms. Each DNA strand is made of four chemical units, called nucleotide bases, which comprise the genetic "alphabet." The bases are adenine (A), thymine (T), guanine (G), and cytosine (C).

The vast majority of our DNA (named ncDNA that composes 98% of our DNA) doesn't code proteins but serves functional roles (like the regulation of gene expression). It is believed that it has functions that are yet to be discovered[3].

[1] https://media.istockphoto.com/photos/internal-structure-of-an-animal-cell-3d-rendering-section-view-picture-id1306045773?k=20&m=1306045773&s=612x612&w=0&h=81ecNdkPSXfw8gAYvZW-Aj_rocDDfjlfBqTrmPg5--M=

[2] https://en.wikipedia.org/wiki/DNA#/media/File:Eukaryote_DNA-en.svg

[3] https://www.lsi.umich.edu/news/2018-04/scientists-discover-role-%E2%80%98junk%E2%80%99-dna

A gene is a hereditary unit that we inherit from our parents that define our characteristics(color of eyes, height, etc). A human has about 23,000 such instruction snipets. They are composed of DNA. The complete set of an organisms genes is called the genome. In humans the genome is approximately ~2% of our total DNA. Our genome is distributed in 46 chromosomes (23 pairs), half taken from our mother and the other half from our father.

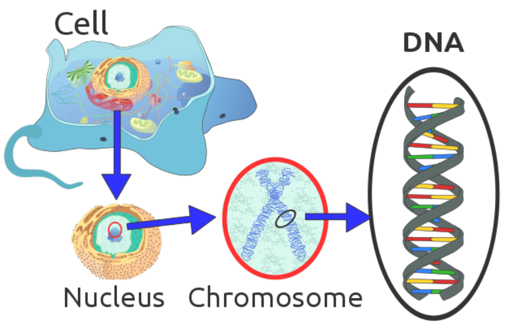


Figure 2: Location of our DNA

1.2.1 Genomics

The study of the genome and it’s environment is called Genomics. It is an interdisciplinary field of Biology. It aims at the collective characterization and quantification of all of an organism's genes, their interrelations and influence on the organism[19].

A major milestone of the field is the completion of “The Human Genome Project”. It is dubbed as one of the greatest scientific feats in history. It started in 1997 and it’s aim was to decipher the chemical makeup of the human genome[17] and it finished in 2003[18] having completed about 92% of the total human genome sequencing.

1.3 RNA

RNA is also a nucleic acid that exists in all living cells. It has structural similarities to DNA, but unlike DNA it is single stranded (with some exceptions to double stranded RNA viruses and special RNA types). It is using the same bases as DNA [1.2] with the only difference that it’s using uracil (U) instead of thymine (T). There are multiple types of RNA but the three major types are:

- mRNA: DNA is stored inside the nucleus and under normal circumstances it never leaves it. The mRNA comes into play to carry out information from the nucleus to the cytoplasm
- rRNA: becomes part of the ribosome, which is the factory for protein synthesis
- tRNA: is the carrier of amino acids to the ribosome in order to complete the protein synthesis

1.3.1 Transcription

Transcription(fig.3[4]) is the first step in gene expression, in which information from a gene is used to construct a functional product such as a protein. The goal of transcription is to make an RNA copy of a gene's DNA sequence[5]. For a protein-coding gene, the RNA copy(transcript), carries the information needed to build a polypeptide (protein or protein subunit).

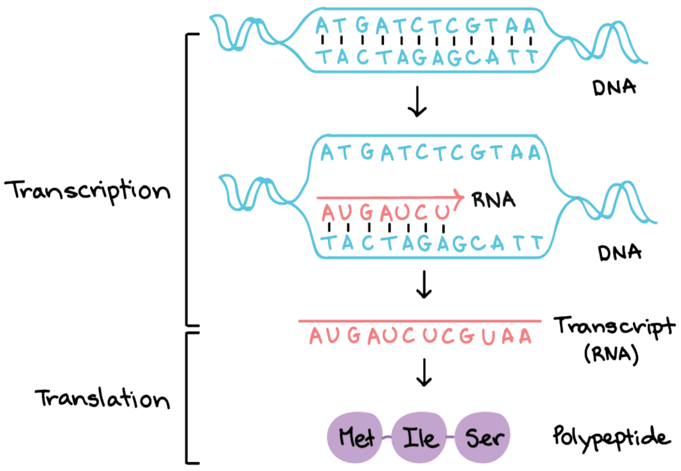


Figure 3: The process of transcription and translation

1.3.2 miRNA

The miRNA[18] is a small RNA segment that is produced by ncRNA. The job of miRNA is to act as a gene regulator by intercepting the mRNA and silencing genes. This happens because miRNA is partially complementary to the mRNA it's trying to oppress. As soon as it attaches itself to the mRNA, it will cause either it's degradation or prevent ribosomes from translating it.

The interesting thing about miRNA is that it can be associated with a plethora of diseases, cancer being one of them[20],[21]. The very odd thing about it is that it has both been associated with oncogenic events and as tumor suppressing agent!

We call this type of miRNA, an "oncomiR". A list of miRNAs directly associated with cancer can be found here[6]. Lastly, research has shown that miRNAs can directly be associated with survival prediction in cancer patients [22], [23].

4 https://cdn.kastatic.org/ka-perseus-images/20ce29384b2e7ff0cdea72acaa5b1dbd7287ab00.png
5 https://www.khanacademy.org/science/ap-biology/gene-expression-and-regulation/transcription-and-rna-processing/a/overview-of-transcription
6 https://en.wikipedia.org/wiki/Oncomir#Characteristics_and_mechanisms_of_some_well_defined_oncomirs

1.3.3 Transcriptomics

Same as with Genomics in [1.2.1], transcriptomics study the transcriptome (the complete set of RNA transcripts that are produced by the genome). The main focuses of transcriptomics is how transcripts of a cell, tissue or living organism are influenced by disease or other environmental factors[19], but scientists are also looking into other functions for ncRNA.

1.4 Brain

The brain is the most complex organ inside the human body. It controls our thoughts, it stores memories, expresses emotion through chemical reactions, understands and processes complex signals from our sensors (vision from our eyes, audio from our ears) and generally is the main operator behind most processes that are carried out inside our body.

1.4.1 Brain cell (Neuron)

To further our understanding of the brain we begin with the smallest biological computational unit. The neuron[24](fig.4[7]).

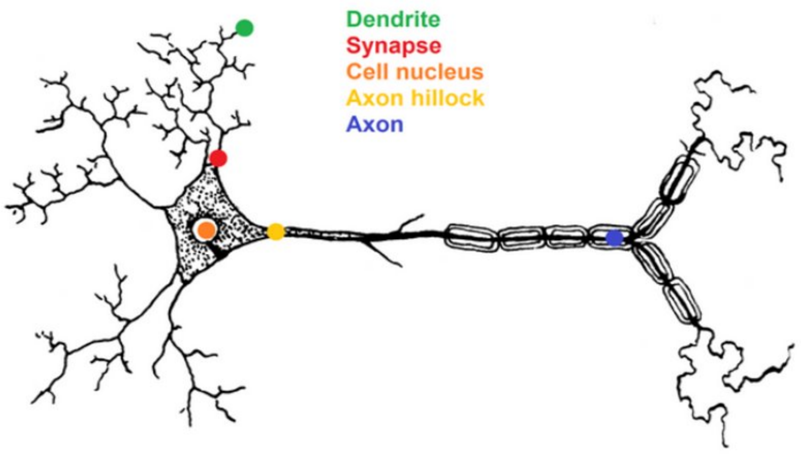


Figure 4: A biological neuron

Neuron are primarily information messengers. They collect information from other neurons on their dendrites via neurotransmitters. The information flows to the cell nuc-

[7] The source of the image was google, but it has been lost

leus and gets stored in the axon hillock. When enough information is gathered to excite the neuron it generates an action potential. Then the information travels down the axon, which is covered in myelin (layer that insulates the pathway so the signal won't loose it's strength. The signal reaches the axon terminals and the neuron emits neurotransmitters. Lastly the neuron resets to prepare to fire again.

1.4.2 Main parts of the brain

The brain consists of approximately 100 billion neurons [1.4.1]! There are many more parts in the brain than neurons. Synoptically the main parts of the brain are(fig.5[8]):

- Frontal lobe, is our cognitive center (controls speech, judgement, etc.)
- Parietal lobe, helps with sensory information
- Temporal lobe, is responsible for memory and hearing
- Occipital lobe, processes input coming from our eye retina
- Cerebellum, primary motor functions and balance
- Spinal cord, is what connects our brain with the rest of the body forming the CNS

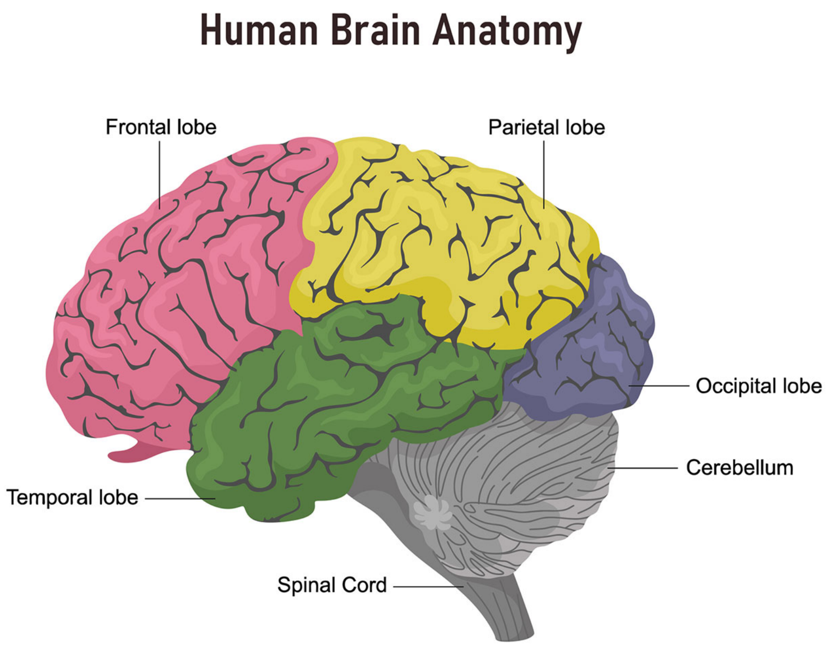


Figure 5: The brain's anatomy

[8] https://www.hopkinsmedicine.org/-/media/images/health/1_-conditions/brain/brain-lobes-anatomy.ashx

1.5 Cancer

Cancer is a genetic disease that is caused when cells in the human body disavow the natural cycle of their lives by refusing to die when they become too damaged or dictated to do so, or growing uncontrollably without being signaled to do so. This can happen anywhere in the body because as mentioned in [1.1] the human body averages over 30 trillion cells.

Cancerous cells that aren't intercepted by our immune system might form clumps that we call tumors. These tumors can be classified as:

- Benign, which is in general an overgrowth of human cells but may still pose a serious threat to ones life. These usually don't re-appear after being removed.
- Malign, where the tumor will start invading nearby tissue and start over consuming resources to the point that the further it expands, it's internal area dies from the lack of resources (oxygen, building blocks, etc.).

There are four distinct cancer stages and a preliminary stage:

- **Stage 0**: cancer is localized in the area that it started
- **Stage I**: cancer is localized to a small area and hasn't spread to lymph nodes or other tissues.
- **Stage II**: cancer has grown, but it hasn't spread.
- **Stage III**: cancer has grown larger and has possibly spread to lymph nodes or other tissues.
- **Stage IV**: cancer has spread to other organs or areas of the body. (**metastasis**)

Cancer is statistically likely to show up in our lives. A research facility in the UK claims that one in two people will develop cancer in their lifetime[9]. There are multiple major risk factors for cancer:

- Hereditary, if one or both parents had or develop cancer, the genes get passed down to the children
- Exposure to radiation (Atomic accidents like Chernobyl(1986), or UV sun rays) that causes destabilization of DNA which might lead to cancer
- Age, the older we get it's more likely for an error to occur while cell replication happens leading to cancerous cells

[9] https://www.cancerresearchuk.org/about-cancer/causes-of-cancer/age-and-cancer

1.5.1 Brain Cancer

Due to brain cancer not operating like other tumors (e.g. it's very rare for a brain tumor to metastasize outsize of the brain) a special grading system is used. The personnel in charge of diagnosing the grade will perform preliminary neurological tests to determine the impact of the tumor on basic functions (speech, motor function, etc.). The main factors used to asses the tumor include:

- Size, morphology and location
- Type of cells / tissue affected
- The possibility of the partial or full tumor volume being removed by surgery (resectability) or cauterization
- The spread of the cancer within the brain or spinal cord
- The possibility the cancer metastasized outside the brain area or the CNS

1.5.1.1 Gliomas

Gliomas make up about 33% of the brain cancers. Glioma is an umbrella term that denotes cancers found in the glial cells. Glial cells are responsible to clean up after neurons as well as resupply them with resources.

Usually the gliomas are named for the type of glial cell they resemble. The way we grade gliomas is how aggressive they are and how fast they grow:

- LGG: grade I & II
- HGG: grade III & IV

Often times it's not enough to grade a glioma by it's type. A low grade glioma can rise in grade if it shows excessive aggression or growth. It can also rise in grade if a gene analysis finds high correlation with already established high grade gliomas (e.g. GBM IDH wildtype is a grade IV glioma and currently the most aggressive brain cancer).

1.5.2 Diagnosis

Most cancers usually give a footprint signalling their existence. They might cause pain, discomfort, and a myriad of other symptoms. A doctor will perform standard physiological tests and look through the patients family medical record. In case they find something abnormal they might order lab tests(blood work), imaging tests(CT, MRI, PET, Ultrasound) or even a biopsy where tissue and fluid from the tumor is extracted surgically and tested in a lab.

The problem with brain cancer, especially LGGs (because of the low growth rate) is that the brain is encapsulated in our skulls. The brain itself does not have any pain receptors so brain cancer often times is very hard to diagnose. It will make itself known through various symptoms among others:

- Headaches coming in various frequencies and severities
- Problems with cognitive functions
- Motor functions operating abnormally
- Drastic changes in personality

1.5.2.1 Radiology

Radiology is a field in medical science that works with imaging techniques to let doctors see inside a patients body without invasive means. Despite the term containing the word radiation not all of Radiology is radiation based (e.g. MRI, Ultrasound).

Radiology can be broken down into two categories:

- Diagnostic, imaging within the body:
    - CT Scan
    - MRI
    - PET
    - Ultrasound
    - Mammography
- Interventional, when it’s used to guide a procedure, like incision, catheter placement, etc.

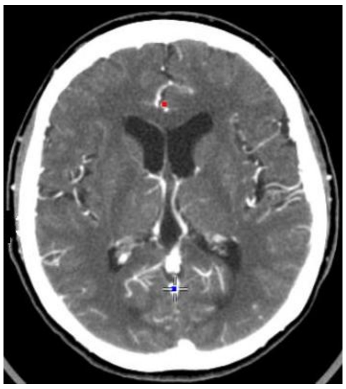


Figure 6: CT scan

The field that studies radiological data and extracts information in the forms of features is called radiomics [3.4].

1.5.2.1.1 MRI

MRI is an imaging technology that produces detailed anatomical images of internal body regions by non-invasive means.

It uses a giant magnet (Usually 1.5 or 3 Tesla but advancements in the field have proved that high tesla magnets increase the quality of the pictures taken(e.g. the 11.7 tesla magnet used in the Iseult Project[27])) to create a unified magnetic field around the patient.

When the patient enters the field the water molecules will align themselves with the magnetic field due to hydrogen atoms acting as magnets. Low energy water molecules also start spinning when we bombard them with a radio frequency waves by sapping the energy needed from the radio waves.

When the radio waves are interrupted these molecules discharge the energy and return to equilibrium state while the rest of the water molecules keep spinning in respects to the unified magnetic field. The MRI machine detects the movement of the low energy water molecules and then translates that into slices based on a gradient. By stacking these slices we obtain a 3D representation of the organ or we want to observe.

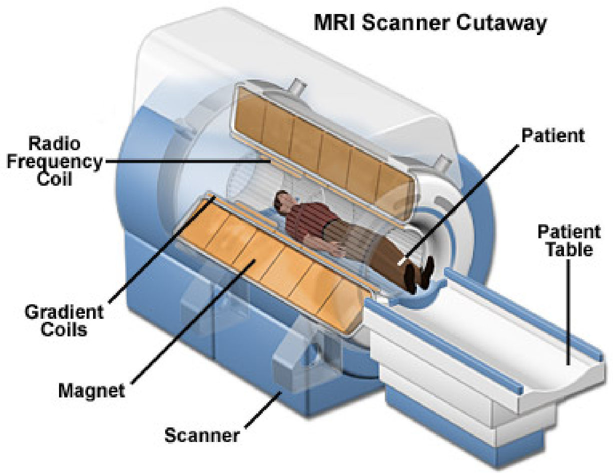


Figure 7: MRI Scanner

Because MRI doesn't use radiation like X-Rays or CT scans do, it's often the best type of imaging for frequent studies although the cost can be rather high in comparison to the aforementioned.

By changing the radio wave frequency and the gradient we obtain a different MRI [28]. These are called MRI sequences[10]. Using different sequences yields different tissue densities. Examples of sequences:

[10] https://www.wikidoc.org/index.php/MRI_sequences

- **T1** (longitudinal relaxation time)(fig. 8[11]):
  - Fat: bright
  - Muscle: gray
  - Fluid: dark
  - Moving blood: dark
  - Bone: dark
  - Air: dark
  - Brain:
    - Gray matter: gray
    - White matter: bright

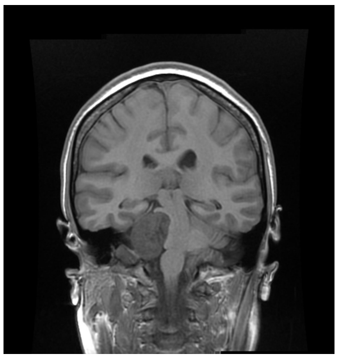


Figure 8: T1

T1 is best used in assessing the anatomy as the image resembles the tissue macroscopically.

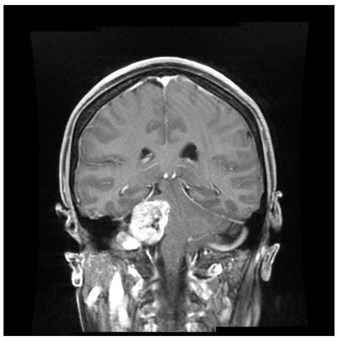


Figure 9: T1 CE

- **T1 CE**(or **GD**) (fig. 9[12]):

Practically the same as T1 with the difference that the patient is injected with GD. This is used to alter the moving blood density to bright. T1-CE is useful in assessing hypervascular lesions.

- **T2** (transverse relaxation time)(fig.10[13]):
  - Fat: bright
  - Muscle: gray
  - Fluid: dark
  - Moving blood: dark
  - Bone: dark
  - Air: dark
  - Brain:
    - Gray matter: gray
    - White matter: bright

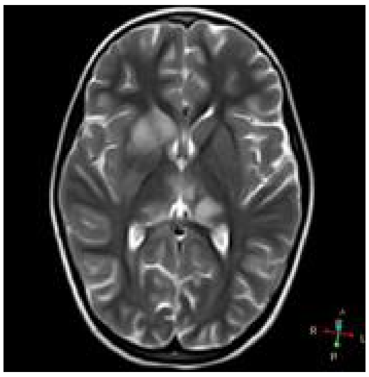


Figure 10: T2

Used mostly as supplementary to T1, to help with the lesion analysis.

[11] https://www.wikidoc.org/images/3/31/T1_acoustic-schwannoma-14.jpg
[12] https://www.wikidoc.org/images/c/c8/T1_c_acoustic-schwannoma-14.jpg
[13] https://prod-images-static.radiopaedia.org/images/3374474/17d9d073fda711fd52fd1522243594_thumb.jpg

- **FLAIR** (fig.11):
  - Fat: bright
  - Muscle: gray
  - Fluid: dark
  - Moving blood: dark
  - Bone: dark
  - Air: dark
  - Brain:
    - Gray matter: gray
    - White matter: darker than gray matter

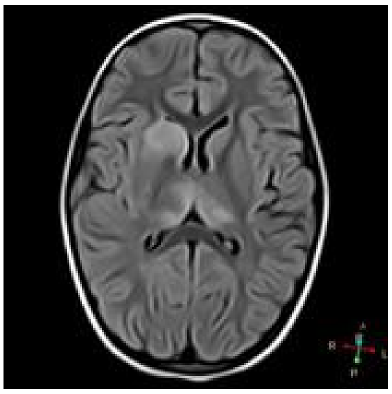


Figure 11: FLAIR

Useful in assessing lesions near ventricles, the lesion can easily be discriminated by cerebrospinal fluid.

1.5.3 Survival Rate

Survival rate is a metric that is used to calculate the life expectancy of a cancer patient based on previously recorded cases. It's often spread into three time frames:

- 1st year mark
- 5 year mark
- 10 year mark

In some special cases (GBM IDH wildtype), the cancer is aggressive enough that sub one year prediction metrics come into use[29]. Survival prediction isn't set in stone, one could argue that due to the older recordings of fatalities due to cancer the prediction can be biased by the time frame they were taken in respects to the diagnostic & technological level of the times. In recent years we've come to use survival rate as a metric to observe and document the results of cancer therapy. There are multiple factors that form this metric[14]:

- **Type** of cancer (glioma, lymphoma, etc.)
- **Stage** of cancer ([1.5])
- Available **treatment** (chemotherapy, radiation, etc.)
- **Age** & **gender**,

By adding more data types (radiomic, genomic, etc.) to the factor section we are able to create far more complex models to calculate the survival rate of a patient.

[14] https://www.wcrf.org/cancer-trends/cancer-survival-statistics/

2 Research Question

In this section we will discuss current problems related to cancer in multiple levels(social, diagnostic, treatment), efforts in the literature and give our proposed solution and define the scope and target of this dissertation.

2.1 Problems

According to WHO, cancer is one of the leading causes of death on the planet surpassing 10 million deaths in a year[15]! There are many problems associated with cancer, not all of them being the cancer itself:

- **Socioeconomic spectrum**: According to Anna Lewandowska[34] in a study involving 800 patients, they found out that cancer patients have a high level of unmet needs especially in terms of psychological support and medical information. Most of them find themselves in denial, despair and extreme anxiety. These states have a high effect on the decision making and clairvoyance of the situation the patient is in. On the same scope, a study in the UK [33] supports that patients would get treated for their symptoms without the idea of cancer being present based on socioeconomic and educational level factors.

- **Lack of data**: while there might be an influx of cancer caused deaths and new cancer cases[16]over the last decade, the lack of large multi type datasets and public data availability is impacting research teams across the globe and hinders design and creation and evolution of prognostic, diagnostic and treatment assessment tools.

- **Diagnostic**: Disavowing early signs. A Danish study [31] found out that the mortality rate, due to general symptoms being present in a multitude of other non-life threatening diseases therefore causing concern for a cancer diagnosis to be non existent or low, to be increased. This happens as a result of the low probability of the symptoms pointing to cancer.

[15] https://www.who.int/news-room/fact-sheets/detail/cancer
[16] https://www.cdc.gov/cancer/dcpc/data/index.htm

- **Biomarker complexity**: despite continuous efforts in the multi-omic biomedical field a lot of the biomarkers fail to complete their clinical evaluation trials due to the uniqueness of the cancers on a molecular level. A biomarker from it's discovery needs to be analytically validated and clinically evaluated before it can be implemented clinically [32],[37].

2.2 Current advancements

With a great deal of problems comes a lot of attempts to solve them, some solution implementations include:

- **Cancer patient pathway** (CPP): In many countries a "fast track" has been implemented as a system to shorten the interval between consultation, diagnosis and treatment in cases of suspected cancer [30].

- **Multi datatype banks**: efforts around the globe have started in the last decade to create public datasets that document cancer cases with as much information as possible (multi-omic data, medical imaging, patient metadata(background, medical history, etc.) in order to give researches the data availability to find deep structural patterns in various cancers as proposed by [36],[38].

- Precision medicine: we know that cancer varies from patient to patient in terms of it's uniqueness (genetic makeup, tissue it's effecting, etc) alongside a plethora of factors (patients health, demographics, etc). Precision medicine treatment comes into play with advancements in the multi-omic fields which lead to isolation of the genetic mutations of the tumors. This gives the medical professionals handles to perform targeted treatment (immunotherapy, cancer vaccine, etc) [35],[36].

2.3 Proposed solution

In our solution we are proposing a multi datatype classification of cancer with both radiomic features [4.4] and multi-omic data [4.7]. Furthermore we aim to bring radiomic extracted features (explained in [3.4]) and multi-omic related features together by calculating the accuracy of cancer survival rate predictors by utilizing ensembled models of weak learners into powerful regressors [4.6], [4.7])

2.3.1 Scope

The general purpose of the study is to peer into the usage of multi-type data for the purposes meta cancer analysis from a computational informatics perspective. The samples we obtained came from a vast number of institutes over the course of three years of competitions (BraTS datasets ‘18-’19-’20). These contain multi-grade gliomas. The gene & miRNA dataset came from a multi-omic benchmarking set [6]. From these we used only the data addressed to GBM.

The duration of the study happened over the course of two months. We will be discussing about data preprocessing, various supervised machine learning methods[3.3] (RFCs, gradient boosting, ANNs), class imbalance strategies and image features extraction and usage[3.4][4.5][4.6].

2.3.2 Target

Our target is to come up with a way to combine imaging data with multi-omic data in an effort to bolster classification of tumors and prediction of survival rates. In other words we’ll try proving that the use of radiomic and multi-omic (genomic & trascriptomic in this case) data can be used to have a more accurate classification of tumors alongside better survival predictions.

We aim to create a classifier / segmenter that locates and annotates the class ([4.2.1]) of a brain lesion (if a lesion exists) and extract it’s radiomic features based on the predicted mask. We will then use these features to train a regressor to try and approximate the survival rate of the patient.

We also aim to create a classifier that takes genomic and trasncriptomic data and is able to classify in between two classes ([4.2.2]). We will then use the multi-omic dataset to create a regressor to extract survival predictions based on a multi-omic sample input([4.7]).

3 Computational Literature

In this section we describe and analyze the informatics theoretical basics and various other needed components to give the theoretical background of our analysis.

3.1 Data preprocessing

Two major data types are used in this study:

- MRI, 3D anatomical image of our brain
- Multi-omic expressions, tabular data

3.1.1 MRI

We define a 3D image as a function:

$$I(i, j, k)$$

in an arbitrary 3D space with i, j, k denoting spatial coordinates where:

- $i = 0, \ldots, M-1$
- $j = 0, \ldots, N-1$
- $k = 0, \ldots, D-1$

Every (i, j, k) set translates to a voxel's location in the 3D image.

The way we get MRIs is by firstly acquiring a 2D slice and then stacking it on an axis. In MRIs a value is assigned to each of these voxels based on average magnetic resonance characteristics present in the tissue corresponding to that voxel[44].

3.1.1.1 Skull striping(fig.12[17])

MRIs of the brain come with a plethora of structures we don't need(CSF, neck, skull, eyes). Actions must be taken before we are left with just the brain tissue. A lot researchers have tackled the issue with a wide variety of ways [45]:

- **Morphology** based methods: these use the morphological erosion and dilation operations to separate the skull from the brain region
- **Intensity** based methods: these use the intensity values of the image pixels to separate brain and non brain regions
- **Deformable surface** based methods: these evolve and deform an active contour to fit the brain surface

[17] https://jerrylinew.github.io/cs188/public/front.png

- **Atlas**(or template) based methods: they rely on fitting an atlas on the MRI to separate brain from non brain matter.
- **Hybrid** methods: these use all of the above in order to counteract a specific methods' disadvantages as illustrated by Kalavathi et al[45].

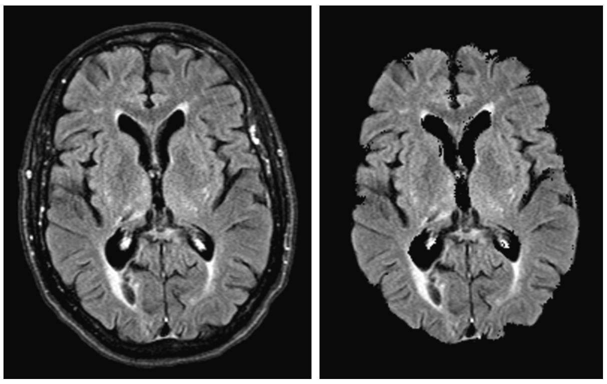


Figure 12: Skull Striping

3.1.1.2 Image Registration

Image registration(fig.13[18]) is the geometrical alignment of an N number of images depicting the same scene in different time intervals and maybe the use of different sensors (e.g. MRI sequences) [46].

These are important due to enabling healthcare professional from monitoring growth patterns on tumors. For a two image system:

- I_1, denotes the source image (the movable one)
- I_2, denotes the target image (the static one),

most of the registration methods will usually follow these steps:

- **Feature detection**, locating distinctive objects (edge, contours, corners, geometrical structures etc.)
- **Feature matching**, correlating detected features amongst the different source image and the target image
- **Transform model estimation**, calculating the type of the mapping functions that will help aligning the source image to the target image(translations, shears, scaling, etc.)
- **Resample and transformation**, performing the transformations from the model estimation and interpolates non integer spatial coordinates

[18] https://els-jbs-prod-cdn.jbs.elsevierhealth.com/cms/attachment/e51bb6c2-629f-4b55-8fea-677faf0299ae/gr2_lrg.jpg

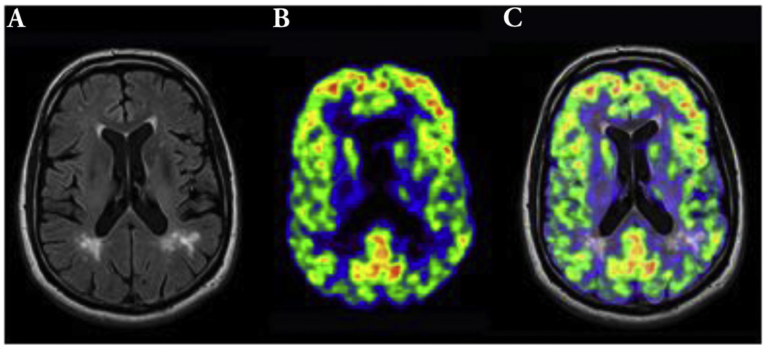


Figure 13: Image registration of MRI and fMRI

3.1.1.3 Denoising

Noise can cause tremendous amount of corruption to our data, causing errors in quantitative imaging with potential leading to miss diagnosis. There are multiple noise factors in the process of acquiring MRI data:

- **Thermal noise**, coming from the machine itself
- **Living noise**, which is caused by bio processes inside the brain or movement of the patient while inside the MRI machine

A standard way to filter MRIs has been proposed by Buades et al[47]. This uses the self spatial similarities that natural images have by using the redundancy of the neighbourhood pixels to remove the noise(fig.14[19]). A more detailed overview on various filters and methods be found in [48].

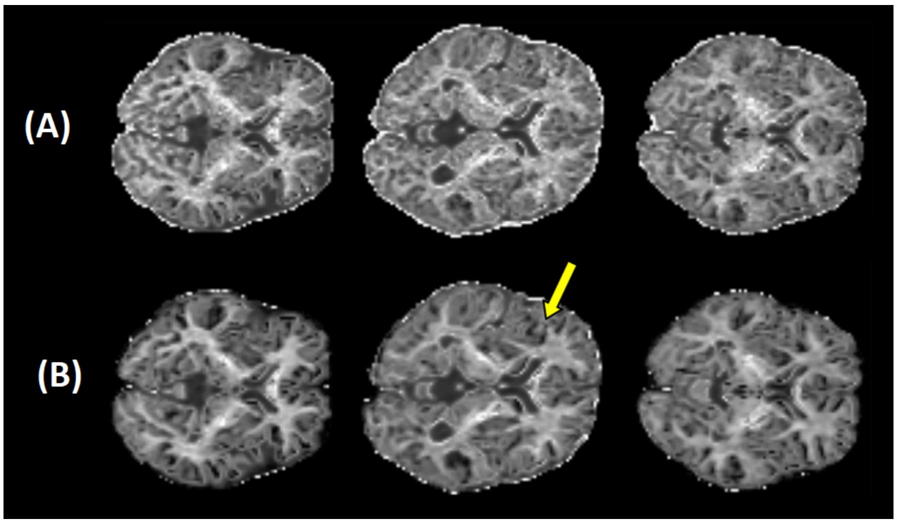


Figure 14: NLM filtering (A) noisy (B) filtered

[19] Edit of : https://journals.plos.org/plosone/article/file?id=10.1371/journal.pone.0116986.g006&type=large

3.1.1.4 Bias field correction

The bias field is a low-frequency artifact that causes a smooth signal intensity variation within tissue of the same physical properties[44]. This gets exacerbated in older MRI machines [49]. A very prominent way to fix this is by using an improved version of the famous non parametric nonuniform intensity normalization (N3) [50], dubbed "N4ITK" [51]. In short, it performs histogram normalization to vanish lightning defects that may be caused by the magnetic coils.

Both N3 and N4 corrections assume that the non-uniformity in the MRI is multiplicative. This means that the noisy image (I_a) we get is a multiplication of a corrected image (I_c) and a bias field (B) at each point. This is given by the equation:

$$I_a(r) = I_c(r) \text{ x } B(r)$$

Both the techniques theorize that the log of the bias field (B(r)) is a zero centered Gaussian distribution and so both of them operate in the log transformed space of image intensities. This transforms the above equation from multiplication to addition:

$$\log(I_a(r)) = \log(I_c(r)) + \log(B(r))$$

The process starts by masking the background. This happens in order to avoid areas in the image where the signal intensity approaches zero. Then begins an iterative process (usually this process has a function to break it out when demand is met but practically it's used with a set number of iterations (n_total)):

i. The bias field histogram is calculated to sharpen the image. This is achieved by using the Wiener deconvolution filter[20](it uses a Gaussian kernel)
ii. The estimation of the bias field is smoothed by fitting it with 3D B-spline field[21]
iii. Loops back to (i) until *iter* > n_total where the iterative process stops.

[20] https://homepages.inf.ed.ac.uk/rbf/CVonline/LOCAL_COPIES/VELDHUIZEN/node15.html#SECTION00533000000000000000

[21] https://en.wikipedia.org/wiki/B-spline

From there the bias field estimation is extrapolated over the entire field of view. Lastly the noisy image I_a is divided by the bias field estimation to give us the approximation of the corrected image I_c.

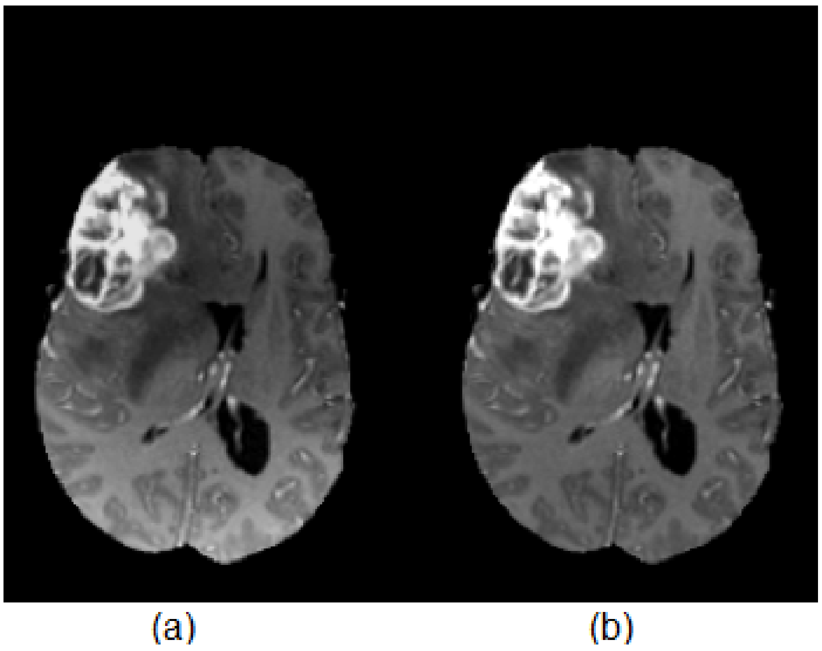


Figure 15: Example of N4ITK before(a) & after (b)

3.1.1.5 Normalization

We use normalization to bring the scale of image values to a range our neural network can utilize for learning without the fear of model corruption. Depending on our uses and targets the normalization might happen in two ranges:

- 0 … 255
- 0 … 1

The formula we use for normalization for any range [a, b] is given by the equation[22]:

$$x' = a + \frac{(x - \min(x))(b - a)}{\max(x) - \min(x)}$$

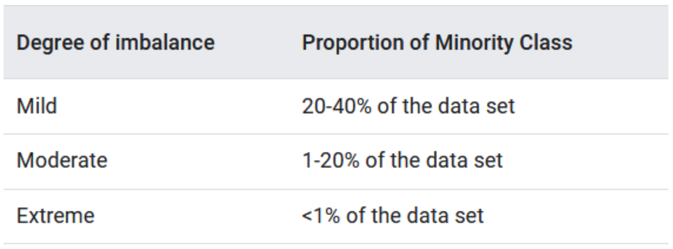
22 https://en.wikipedia.org/wiki/Feature_scaling#Rescaling_(min-max_normalization)

3.1.2 Tabular data

Tabular data are usually in csv files and are organized by rows and column, translating into samples and features respectively. The same formula as in [3.1.1.5] is used to normalize them.

3.1.2.1 Class Imbalance

Class imbalance occurs when we have a certain distribution of classes. In a two class system the imbalance degree would be given by these percentages[23]:



Degree of imbalance	Proportion of Minority Class
Mild	20-40% of the data set
Moderate	1-20% of the data set
Extreme	<1% of the data set

Table 1: Class imbalance tiers

There multiple ways to address class imbalance, some major concepts are:

- **Undersampling** the majority class:
    - **ClusterCentroids**(fig, where the majority class is undersampled by replacing a cluster of majority samples by the cluster centroid of a K-Means algorithm[24](fig.16[25]).

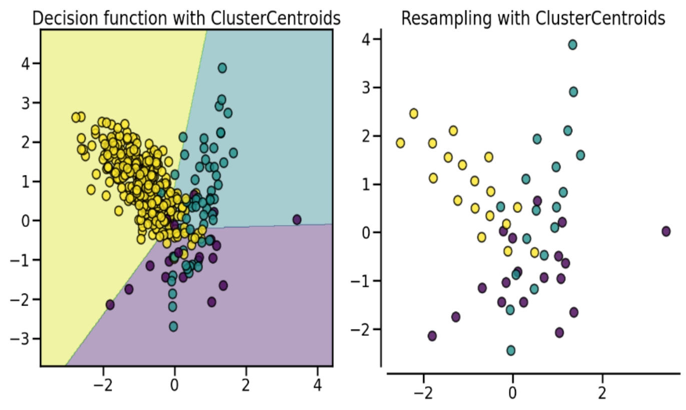


Figure 16: Undersampling of majority class with K-means clusters

- **Oversampling** the minority class **by augmentation**:

[23] https://developers.google.com/machine-learning/data-prep/construct/sampling-splitting/imbalanced-data

[24] https://imbalanced-learn.org/stable/references/generated/imblearn.under_sampling.ClusterCentroids.html

[25] https://imbalanced-learn.org/stable/_images/sphx_glr_plot_comparison_under_sampling_001.png

- **SMOTE** (fig.17[26])[52], suggests that data should be oversampled by generation of synthetic minority samples using the interpolating pairs of the minority classes original points

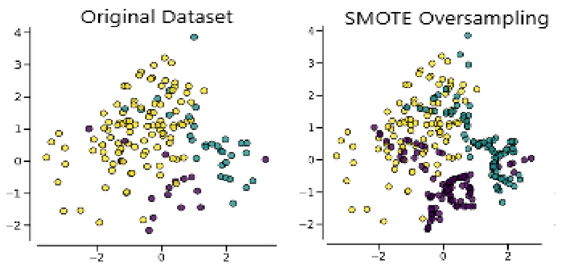


Figure 17: Oversampling with SMOTE

- **BorderlineSMOTE** (fig.18[27])[53], is a variant of SMOTE that enforces the synthetic minority samples to be at the border of the decision function between other classes

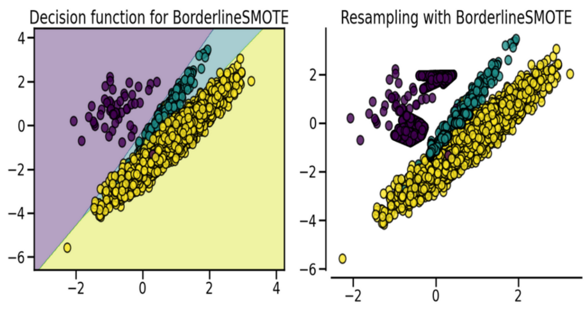


Figure 18: Oversampling with BorderlineSMOTE

- **ADASYN** (fig.19[26])[54], works the same as SMOTE with the only difference that it will address the samples that are difficult to get classified with a nearest neighbour rule, whereas SMOTE will be indifferent towards them

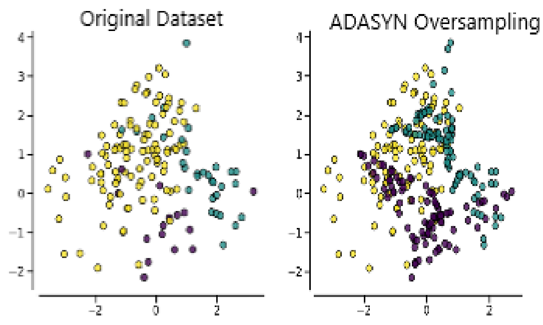


Figure 19: Oversampling with ADASYN

[26] https://imbalanced-learn.org/stable/_images/sphx_glr_plot_comparison_over_sampling_006.png
[27] https://imbalanced-learn.org/stable/_images/sphx_glr_plot_comparison_over_sampling_007.png

3.2 Machine Learning

In this section we explore one of, if not the most important, sector of modern informatics. ML is a subset of Artificial intelligence(fig.20[28])(AI is mostly boring mathematics and philosophy, but as soon as it moves to solving computational problems it leaves us all in awe).

As the title self explains, machine learning is when we use samples of experiences to teach a machine, so that it may analyse them and derive *knowledge* from them. It does this by approximations (in statistical machine learning) or by discovering deep mathematical structures within the data (deep learning).

The core things needed to start solving machine learning problems are:

- A problem in need of solution
- Data correlating to the problem that can yield usable results
- Performance metrics in order to evaluate the models created

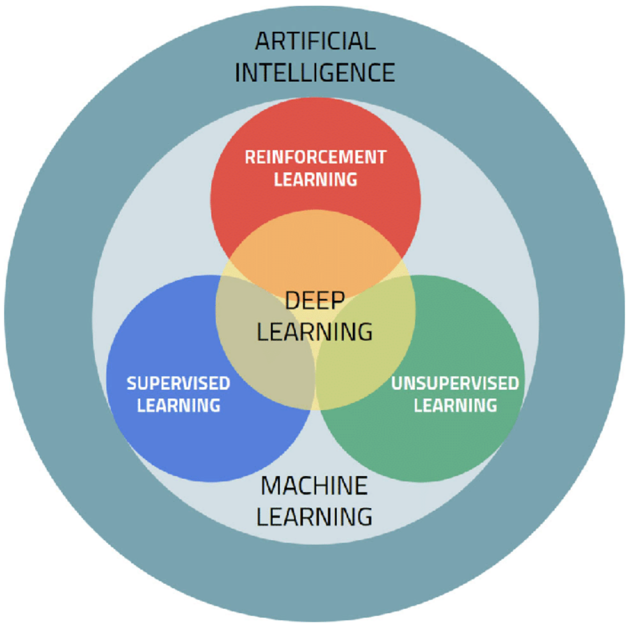


Figure 20: AI and it's Subsets

[28] https://www.researchgate.net/profile/Akshaya-Karthikeyan-2/publication/357234810/figure/fig2/AS:1103630992195584@1640137541038/Schematic-of-the-conventional-relationship-between-artificial-intelligence-AI-machine.png

3.2.1 Types of machine learning

There are three core denominations of machine learning and a fourth one that utilizes the best aspects of two of the main types (tools used by each can be found on fig.21[29]):

- **Supervised learning**, is when we are in full control of the training process. This includes having a clearly defined task as well as properly structured data with correct labels and tags. We train models based on authenticated data so when we feed the predictive model new data that it hasn't seen it might be able to come up with correct predictions.

An example of this would be object classification and the CIFAR-10[30] dataset contains 10 classes and 60000 images!

- **Unsupervised learning**, is when we have data or labels but lack any annotations. The results of this type varies and it's never a good idea to use this for practical models. It excels at exploratory operations due to it's nature, by giving us an idea of what the data looks like or what structures might lie underneath.

An example of this would be the use of K-means algorithm in any dataset to determine if there distinct classes exist.

- **Reinforcement learning**, is practically attempting to train a dog. The way this works is by having an agent explore the environment it's in and by taking any action, it either gets rewarded if it performs positively or punished(penalized) if it performs negatively. More professionally this means that it's trying to maximize it's reward function while at the same time trying to minimize it's loss function. Since the data here have no labels the agent is doomed to brute force the knowledge out of the data.

Bickering aside the example for this category would be an artificial dog. By letting it loose on a virtual field or a house you would be able to reward it positively for good behavior or punish it if it goes haywire and starts breaking the house.

[29] https://cdn-images-1.medium.com/max/800/1*rbaxTrB_CZCqbty_zv2bEg.png
[30] https://www.cs.toronto.edu/~kriz/cifar.html

- **Semi-supervised learning**, is the middle ground of supervised and unsupervised learning. Utilizing mostly unlabeled data and some data with a lot of noise it is able to reach a generalization faster. The models produced aren't as good as the models from supervised learning, but semi-supervised learning is a cheap way to reach a good point in both understanding your data alongside the scope of your task.

A good example of this would be the semi-supervised protein classification as proposed by Weston et al[57].

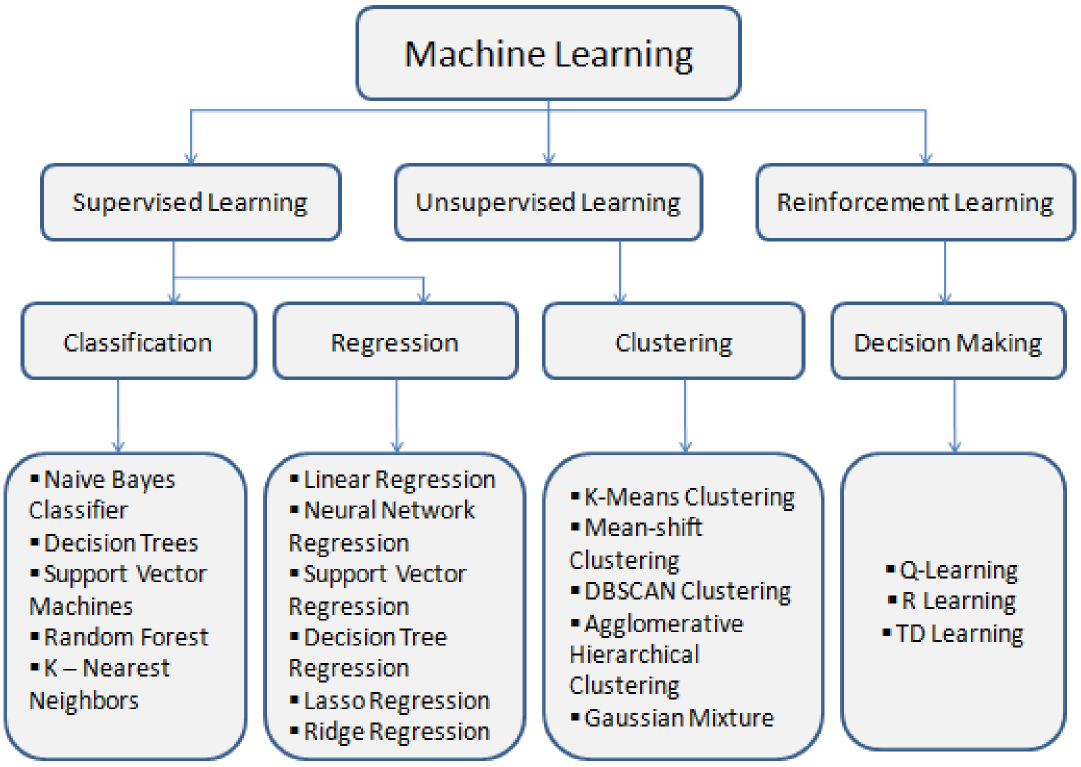


Figure 21: Tools used in ML

3.2.2 Random Forest Classifier

These derive from the ensemble of many Random Tree Classifiers(RTC) [58]. By taking a lot of weak learners, that we create by a random selection of features each time results in them accumulating their result and averaging out their prediction. This way they are able to beat the downside of a single RTC's high variance and achieve better generalization as showcased in fig.22[31].

[31] https://miro.medium.com/max/1200/1*hmtbIgxoflflJqMJ_UHwXw.jpeg

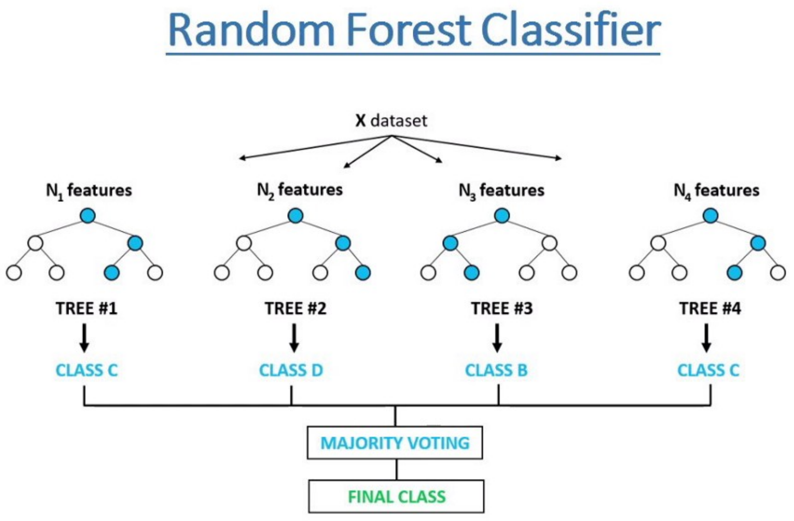


Figure 22: RFC breakdown

3.2.2.1 Boosting

Boosting[32] is an ensemble method where we build multiple weak learners one on top of another in order to increase the predictive capabilities of the final estimator. The main idea is that each new model added to the ensemble is attempting to fix the short-comings of its ancestor.

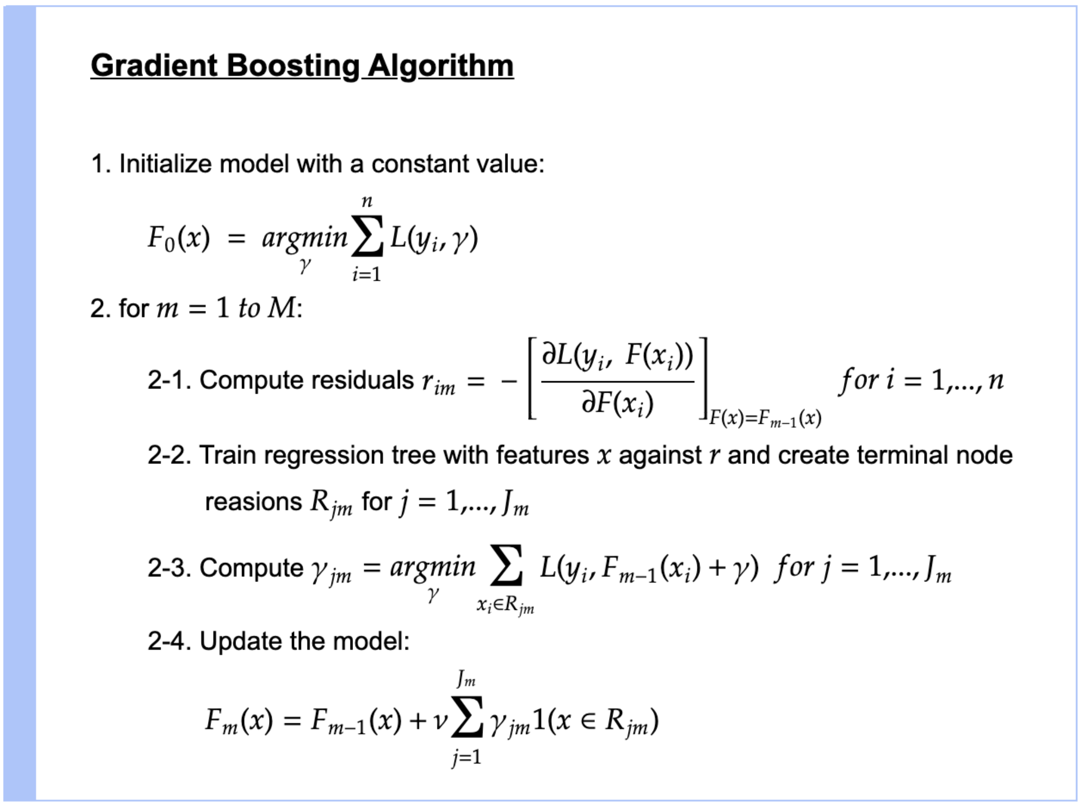
An example with RTCs would be that the first model we build, regardless it's accuracy would be used to train the second model, another RTC. The second model would then try to capitalize on the errors of the first one by focusing on learning the correct predictions for the miss predictions of the first model. This process repeats till a certain number of weak learners are conjoined in the ensemble or a certain threshold is reached.

3.2.2.2 Gradient Boosting

The difference of gradient boosting [59](fig. 23[33]) from normal boosting is that it focuses on the prediction error by factoring it in the next weak learner generation. It appends the error (residuals) into the dataset, but it scales it down by the learning rate

[32] https://en.wikipedia.org/wiki/Boosting_(machine_learning)
[33] https://miro.medium.com/max/1400/1*dIHrPFBT2fmXuTXMb-3_Xw.png

to intercept instances of over fitting. This process is iterative just like normal boosting until a set number of iterations has passed or a threshold of performance is reached.



Gradient Boosting Algorithm

1. Initialize model with a constant value:

$$F_0(x) = \underset{\gamma}{argmin} \sum_{i=1}^{n} L(y_i, \gamma)$$

2. for $m = 1$ to M:

2-1. Compute residuals $r_{im} = -\left[\frac{\partial L(y_i,\ F(x_i))}{\partial F(x_i)}\right]_{F(x)=F_{m-1}(x)}$ $for\ i = 1, \ldots, n$

2-2. Train regression tree with features x against r and create terminal node reasions R_{jm} for $j = 1, \ldots, J_m$

2-3. Compute $\gamma_{jm} = \underset{\gamma}{argmin} \sum_{x_i \in R_{jm}} L(y_i, F_{m-1}(x_i) + \gamma)$ $for\ j = 1, \ldots, J_m$

2-4. Update the model:

$$F_m(x) = F_{m-1}(x) + \nu \sum_{j=1}^{J_m} \gamma_{jm} 1(x \in R_{jm})$$

Figure 23: Friedman's Gradient Boosting Algorithm[59]

3.2.2.3 Metrics

For regression we use:

- **MSE**, which measures the squares of the error an estimator produces and the ground truth[34]:

$$\mathrm{MSE} = \frac{1}{n} \sum_{i=1}^{n} \left(Y_i - \hat{Y_i}\right)^2.$$

n being the quantity of predictions of a prediction vector

- **RMSE**, which measures the differences between values predicted by an estimator and the ground truth[35], theta being the estimator in question:

$$\mathrm{RMSD}(\hat{\theta}) = \sqrt{\mathrm{MSE}(\hat{\theta})}$$

[34] https://en.wikipedia.org/wiki/Mean_squared_error#cite_note-:1-1
[35] https://en.wikipedia.org/wiki/Root-mean-square_deviation

- **K-fold cross validation**, which is used to estimate the skill of an estimator on unseen data. The process that happens in is very simple[36]:
  1. If shuffle is set to true, the dataset is going to be shuffled
  2. The dataset is split into K groups
  3. For every K (or fold):
     1. Take that fold out as a test set
     2. Form the rest of the folds into a train set
     3. Fit the created training set and evaluate it on the test set
     4. Append the evaluation score to a list L and trash the model
  4. Sum the list L and divide it by K to figure out the mean score

Typically K-Fold cross validation returns a positive number, but if the log_neg is set to true it will return a negative.

For classification we use standard accuracy score:

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN}$$

Where TP = True positive, FP = False positive,
TN = True negative, FN = False negative

3.2.3 ANN

Following on the biological neuron presented in [1.4.1] early researchers decided to attempt to simulate the function of a biological neuron artificially. This gave birth to the first artificial neuron, by Warren McCulloch and Walter Pitts[55].

Fifteen years later, psychologist Frank Rosenblatt proposed the Perceptron(fig.24[37]) and gave birth to neural networks[56]. By taking Rosenblatts simple perceptron which acted as a linear classifier, and by stacking multiple of them into layers non linear functions could be solved[38].

[36] https://machinelearningmastery.com/k-fold-cross-validation/
[37] https://en.wikipedia.org/wiki/Perceptron
[38] https://analyticsindiamag.com/xor-problem-with-neural-networks-an-explanation-for-beginners/

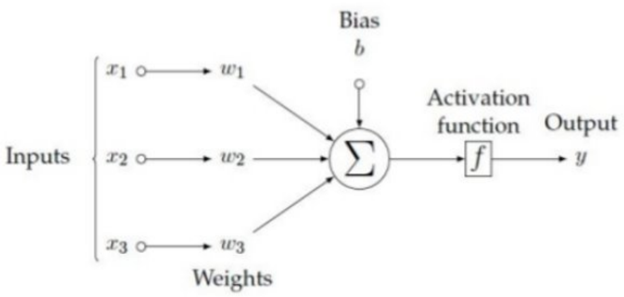


Figure 24: A perceptron

The way a single neuron works is as follows[39]:

- Set a static learning rate
- Initialize the weights (random small values or distributions)
- Sum all the weights multiplied by the inputs
- Add the bias factor by multiplying $Xj_{,0}$ * *bias*
- And pass the result through an activation function(fig. 25[40]) to get $Y_{predicted}$
- Update the weights via:

$$w_i(t+1) = w_i(t) + r \cdot (d_j - y_j(t))x_{j,i}, \text{ for all features } 0 \leq i \leq n$$

n is the total number of samples, r is the learning rate, and d is the ground truth

- Repeat the process till the model converges or epoch requirement are met

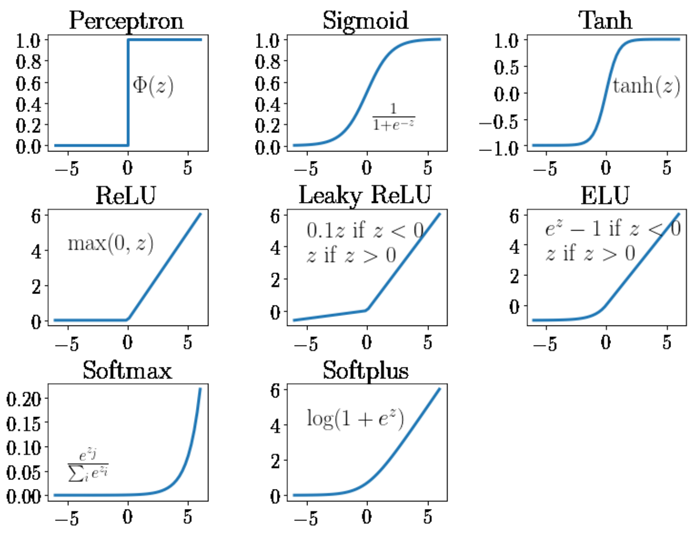


Figure 25: Activation Functions

[39] https://en.wikipedia.org/wiki/Perceptron#Learning_algorithm

[40] https://www.researchgate.net/publication/341310767/figure/fig7/AS:890211844255749@1589254451431/Common-activation-functions-in-artificial-neural-net-works-NNs-that-introduce.ppm

By stacking multiple of these perceptrons we obtain a basic neural network (fig.26[41]).

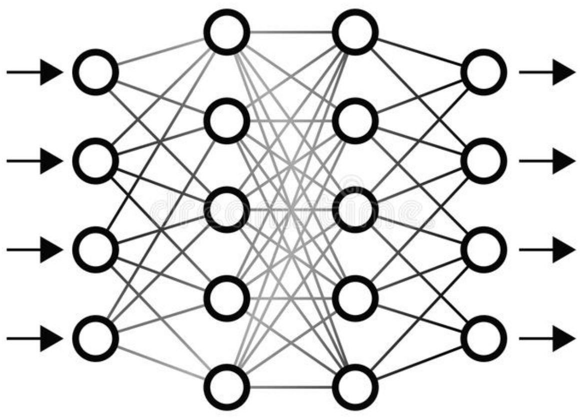


Figure 26: Basic Neural network, fully connected, multiple I/O

The neural network is spread in 3 parts (fig.27[42]):

1. Input layer
2. Hidden layer
3. Output layer

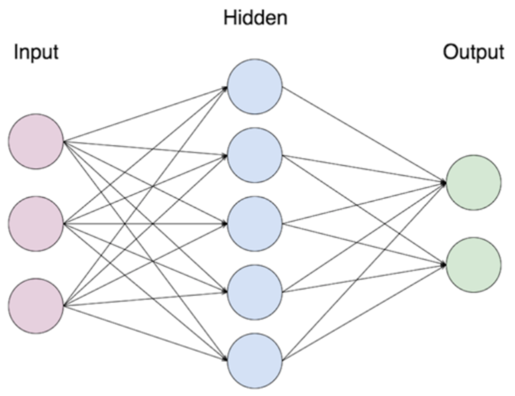


Figure 27: The three primary strips

Hyperparameters are global parameters that are set before the model is compiled. They are used to control the way the model trains. These include:

- **Learning rate**, determines how fast the model is going to learn from samples. Despite being a hyperparameter late literature has show that a decaying [61] or cyclical LR [60],[61]; derived from natural processes can help boost the models convergence.
- **Epochs**, determines how many times the entire training set is going to pass through the network

[41] https://thumbs.dreamstime.com/b/neural-network-illustration-vector-deep-learning-concept-neural-network-illustration-103427158.jpg
[42] https://miro.medium.com/max/1400/1*f9XlMlruW7TMF3EHbPDfYg.png

- **Hidden layers**, determines how many internal layers the unit will have(kind off deprecated considering we use high level APIs to build models now days)
- **Batch size**, determines how many samples are going in the model before a weight update
- **Dropout**, a percentage (let's call it dp) given to the neural network so it will null dp% of the neurons in order to not over fit the model

Back propagation is the method the model updates it's weights, among other it could be, we can see how these work on (fig. 28[43]):

- SGD (stochastic): where the weights update every sample
- Mini-batch: where the weights update after every batch of samples
- Batch: where the weights update once every epoch

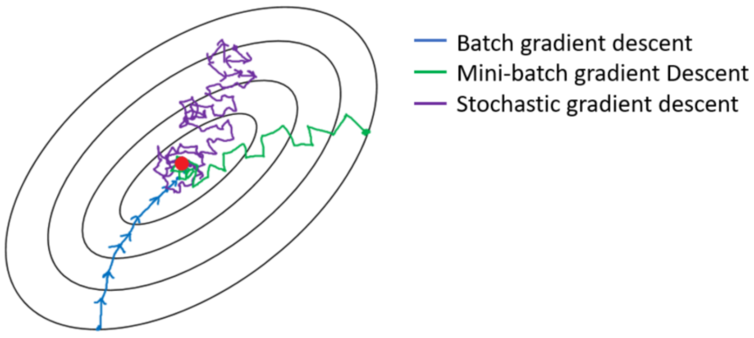


Figure 28: Weight updates with various GD

And finally, we need a way to properly update weights, possibly modify the LR, and minimize the loss function per update iteration. All this is an optimizers job[44]. Some optimizers may include:

- **ADAM**[62], standard deep learning optimizer
- **SGD** [63], is very slow and often times gets stuck on local minimas instead of reaching the global minima
- **AdaFAIR** [13], has the ability to alleviate discrimination against minority classes
- **AdaBoost**[64], is used for the creation of weak learners

[43] https://miro.medium.com/max/908/1*bKSddSmLDaYszWllvQ3Z6A.png
[44] https://www.analyticsvidhya.com/blog/2021/10/a-comprehensive-guide-on-deep-learning-optimizers/

3.2.3.1 CNN

CNNs introduce the idea of convolution.

Convolution is basically applying a kernel over an image where each pixels value is determined by the filter multiplied by the original image g(x, y) pixel values (fig29[45]).

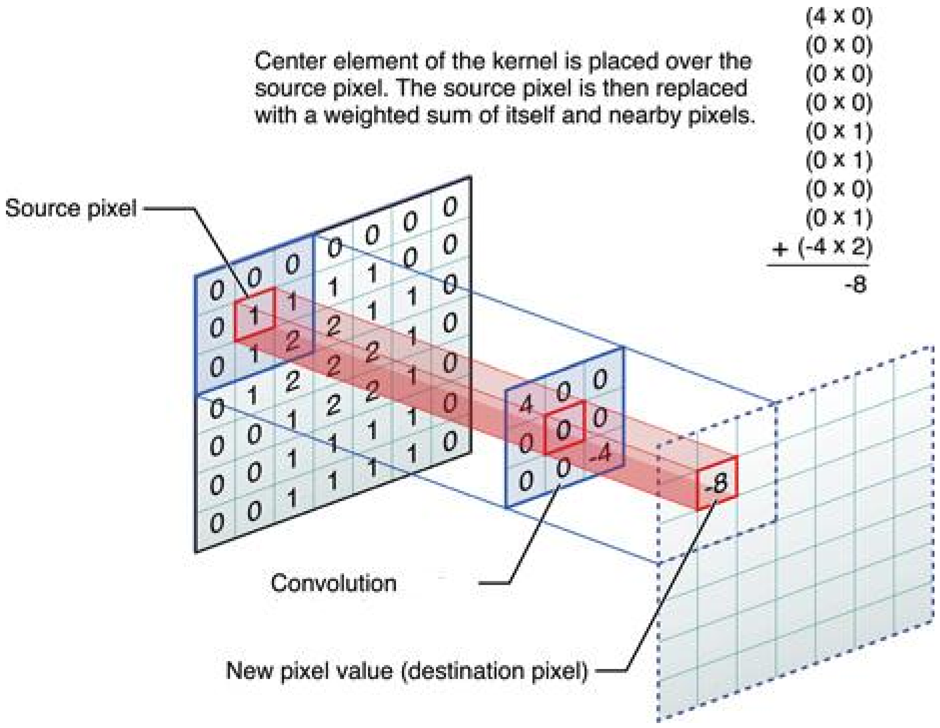

Center element of the kernel is placed over the source pixel. The source pixel is then replaced with a weighted sum of itself and nearby pixels.
(4 x 0)
(0 x 0)
(0 x 0)
(0 x 0)
(0 x 1)
(0 x 1)
(0 x 0)
(0 x 1)
+ (-4 x 2)
-8
Source pixel
Convolution
New pixel value (destination pixel)

Figure 29: Convolution

CNNs have a specific architecture(fig.30[46]) that enables them to learn from images[47].

This includes multiple level of convolutional layers followed by max pooling layers. The convolutional layers have the following parameters:

- **Number of filters**, how many random filters will it generate
- **Kernel size**, (how big the generated filter is going to be (this is a matrix shape)):
  - in 2D default kernels are: (2x2)
  - in 3D default kernels are: (3x3x3)
- **Stride**, which determines how many pixels the filter is going to move next
- **Kernel initialization**: a method to generate these filters (e.g. he_uniform)

[45] https://miro.medium.com/max/464/0*e-SMFTzO8r7skkpc
[46] https://cdn-images-1.medium.com/max/1600/1*g6qPMZTpO2Nl9Y2dxwgvCA.png
[47] https://aigents.co/data-science-blog/publication/introduction-to-convolutional-neural-networks-cnns

- The convolutional layer does the following:
  1. Takes an input image
  2. Pad the image in order to get all available information from the image
  3. Initialize a number of random filters and iterate through the image
  4. By convolution we reduce redundancy and leave the features flowing into the next layers to have more information yield.

After each convolutional layer a max pooling layer exist. This has multiple causes:

- The Convolved features amassed from the convolutional layer are too big, and they get scaled down to reduce computational cost via a dimensionality reduction process.
- It's very good for the dominant features because they are both positional and rotational invariant, so they are maintained

Lastly after multiple steps of convolution and max pooling we reach the classification layer. The usage of a FC (MLP like) layer of usually 3-5 layer size is a one way ticket to learn the purpose of the non linear combination of abstract level structures yielded from the convolutions. To do this we have to flatten the output of the last max pooling layer. The output layer is a softmax layer with one neuron for each class we're trying to classify.

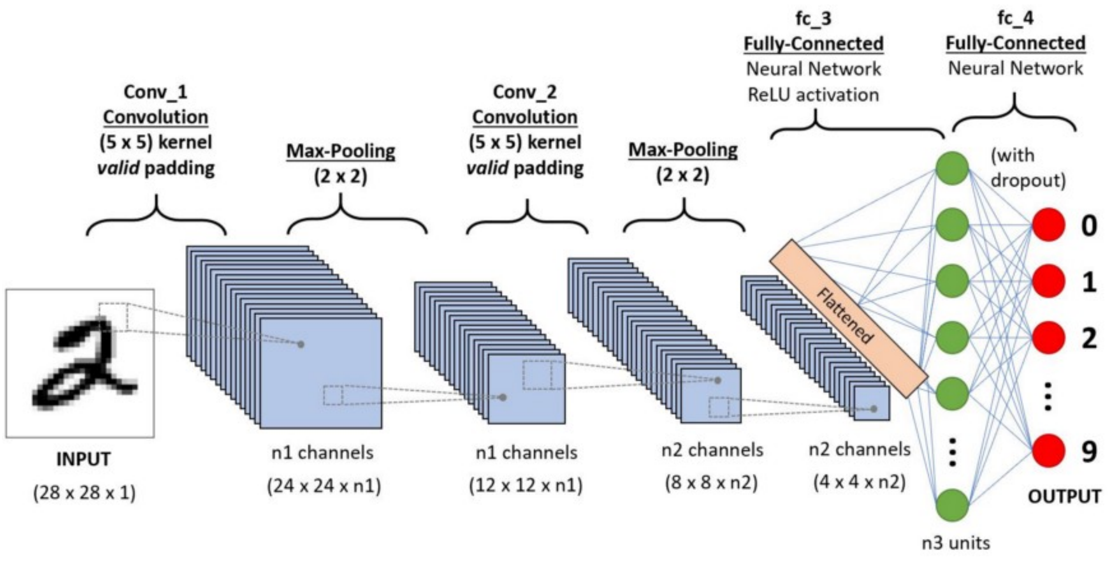


Figure 30: 2D CNN representation

3.2.3.1.1 Unet

A U-net is a CNN architecture based network. It was primarily developed to tackle the problem of biomedical image segmentation [65]. The architecture it's using attempts to maximize information yield on all levels of computation as well as utilizing all knowledge acquired from the model by propagating it throughout all levels of the model.

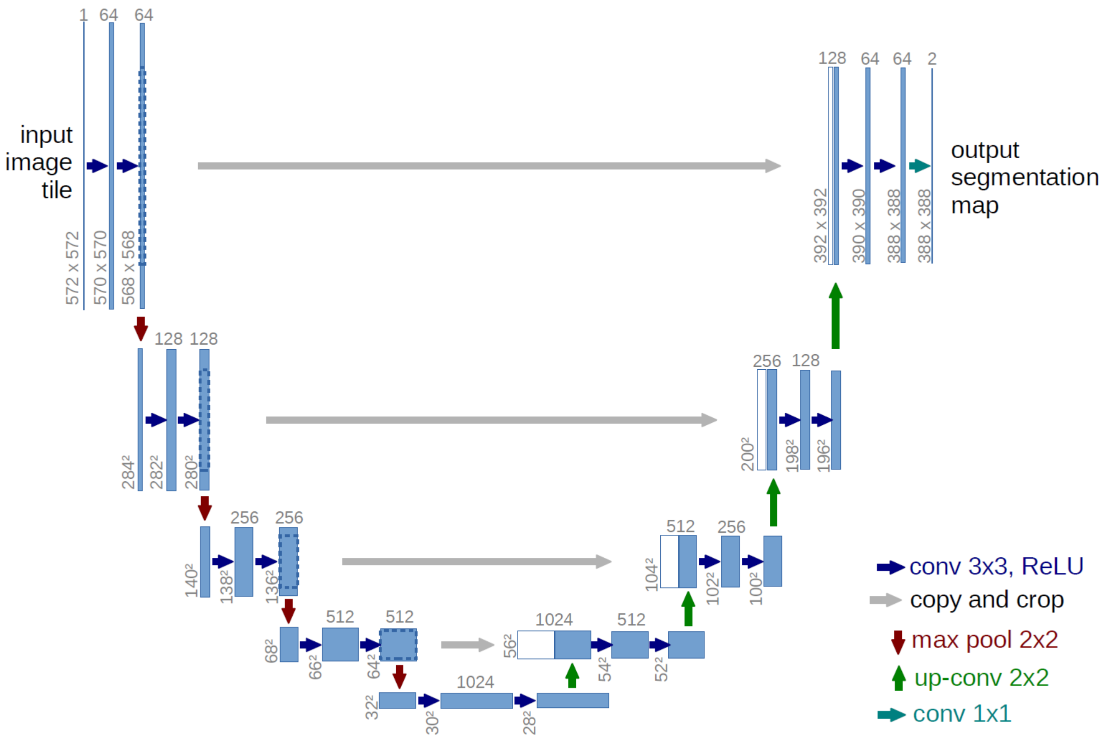


Figure 31: Original UNet Architecture

The model begins with a straight forward CNN classifier network (contraction path) and yields increasingly powerful imaging features with each convolutional operation.

On the end of the contraction the condensed features are stored inside a vector space. We'd like to add to this part, that this is how autoencoders(and their variations) are created by replacing the vector space with a FC layer of arbitrary amount of layers.

And then begins the expansion path, where high-resolution features from the contracting path are concatenated with the upscaled (deconvoluted) data.

All convolutions in the model are followed by a nonlinear function (ReLU) ensuring integrity by not having negative values. After each max pool operation the feature channels from the previous operation are doubled from the previous level. On the expansion path, this is reversed and they are divided by two.

3.2.3.1.2 Metrics

Metrics for neural networks can vary but in our case (image segmentation) we have three very powerful ones:

- **Categorical Accuracy**, this measures how often the model gets the prediction right. It generates two variables "total" & "count" that are used to store information in regards to how many times did the predicted Y match the ground truth[48].

- **Dice's coefficient** (DSC) [67],[68],[66], otherwise known as F1 score or "Sørensen–Dice index", it's given by the formula:

$$DSC = \frac{2|X \cap Y|}{|X| + |Y|}$$

- **Intersection over Union** (IoU), otherwise known as Jacards distance[69] that is given by the formula:

$$J(A,B) = \frac{|A \cap B|}{|A \cup B|} = \frac{|A \cap B|}{|A| + |B| - |A \cap B|}$$

3.2.3.1.3 Statistics or Deep learning?

As a data scientist it is very important to be able to recognize where each tool and methodology should be used. According to [42], they led a study on various tasks across 11 datasets for tabular data and found out that ensembled gradient boosted forests (XGB ensembles) still outperformed Deep Learning. They did however accept that deep learning models are still the leading methodology in complex operations like semantic segmentation in images. Therefore for our analysis in [4.4] we choose to use

[48] https://www.tensorflow.org/api_docs/python/tf/keras/metrics/CategoricalAccuracy

a modified 3D-Unet for our imaging data and since the rest of the data (both the ones we extracted and the ones we obtained are in tabular forms) to use XGBOOST.

3.3 Image Features

Image features represent characteristics that help us discriminate between ROIs and background in MRIs by providing us with imaging biomarkers. These can rely on shape based numerical measurements (like the total voxels present in the ROI) or quantitative visual appearance (like the neighbouring voxel intensity)[44]. In this section we analyse all the imaging features we extract from our imaging data as stated in [15] and used in [4.5]

3.3.1 First order statistics

First-order statistics describe the distribution of voxel intensities within the image region defined -by the mask through commonly used and basic metrics. These include:

- **Energy**: is a measure of the magnitude of voxel values in an image. A larger value implies a greater sum of the squares of these values
- **Total Energy**: is the value of Energy feature scaled by the volume of the voxel in cubic mm
- **Entropy**: it specifies the uncertainty/randomness in the image values. It measures the average amount of information required to encode the image values
- **Minimum**: the minimum gray level value within the ROI
- **10th percentile**: the 10th percentile of the gray level values within the ROI
- **90th percentile**: the 90th percentile of the gray level values within the ROI
- **Maximum**: the maximum gray level value within the ROI
- **Mean**: the average gray level intensity within the ROI
- **Median**: the median gray level intestate within the ROI
- **Interquartile Range**: 75th percentile minus the 25th percentile of the image array
- **Range**: Maximum – Minimum
- **Mean Absolute Deviation**(MAD): is the mean distance of all intensity values from the Mean Value of the image array

- **Robust Mean Absolute Deviation** (rMAD): is the mean distance of all intensity values from the Mean Value calculated on the subset of image array with gray levels in between, or equal to the 10th and 90th percentile
- **Root Mean Squared** (RMS): is the square-root of the mean of all the squared intensity values. It is another measure of the magnitude of the image values
- **Standard Deviation**: it measures the amount of variation or dispersion from the Mean Value
- **Skewness**: it measures the asymmetry of the distribution of values about the Mean value
- **Kurtosis**: is a measure of the ‘peakedness’ of the distribution of values in the image ROI
- **Variance**: is the mean of the squared distances of each intensity value from the Mean value
- **Uniformity**: is a measure of the sum of the squares of each intensity value

3.3.2 Shape Based (3D)

In this group of features we included descriptors of the three-dimensional size and shape of the ROI. These include:

- **Mesh Volume**: the volume of all the voxels in the ROI
- **Voxel Volume**: is approximated by multiplying the number of voxels in the ROI by the volume of a single voxel
- **Surface Area**: first the surface area of each triangle in the mesh is calculated. The total surface area is then obtained by taking the sum of all calculated subareas
- **Surface Area to Volume ratio**: Surface Area divided by Voxel Volume
- **Sphericity**: is a measure of the roundness of the shape of the tumor region relative to a sphere
- **Compactness 1**: is a measure of how compact the shape of the tumor is relative to a sphere (most compact)
- **Compactness 2**: is a measure of how compact the shape of the tumor is relative to a sphere (most compact)
- **Spherical Disproportion**: is the ratio of the surface area of the tumor region to the surface area of a sphere with the same volume as the tumor region

- **Maximum 3D diameter**(Feret Diameter): is the largest pairwise Euclidean distance between tumor surface mesh vertices
- **Maximum 2D diameter** (Slice): is the largest pairwise Euclidean distance between tumor surface mesh vertices in the row-column (generally the axial) plane
- **Maximum 2D diameter** (Column): is the largest pairwise Euclidean distance between tumor surface mesh vertices in the row-slice (usually the coronal) plane
- **Maximum 2D diameter** (Row): is the largest pairwise Euclidean distance between tumor surface mesh vertices in the column-slice (usually the sagittal) plane
- **Elongation**: it shows the relationship between the two largest principal components in the ROI shape
- **Flatness**: shows the relationship between the largest and smallest principal components in the ROI shape.

3.3.3 Gray Level Co-occurrence Matrix

A GLCM describes the second-order joint probability function of an image region constrained by the mask. This includes the features:

- **Autocorrelation**: is a measure of the magnitude of the fineness and coarseness of texture
- **Joint Average**: is the mean gray level intensity of the i distribution
- **Cluster Prominence**: is a measure of the skewness and asymmetry of the GLCM
- **Cluster Shade**: is a measure of the skewness and uniformity of the GLCM
- **Cluster Tendency**: is a measure of groupings of voxels with similar gray-level values
- **Contrast**: is a measure of the local intensity variation, favoring values away from the diagonal
- **Correlation**: is a value between 0 (uncorrelated) and 1 (perfectly correlated)
- **Difference Average**: it measures the relationship between occurrences of pairs with similar intensity values and occurrences of pairs with differing intensity values

- **Difference Entropy**: is a measure of the randomness/variability in neighborhood intensity value differences
- **Difference Variance**: is a measure of heterogeneity that places higher weights on differing intensity level pairs that deviate more from the mean
- **Joint Energy**: is a measure of homogeneous patterns in the image
- **Joint Entropy**: is a measure of the randomness/variability in neighborhood intensity values
- **Informational Measure of Correlation** (IMC) 1: it assesses the correlation between the probability distributions of i and j (quantifying the complexity of the texture), using mutual information I(x, y)
- **Informational Measure of Correlation** (IMC) 2: it also assesses the correlation between the probability distributions of i and j (quantifying the complexity of the texture)
- **Inverse Difference Moment** (IDM): is a measure of the local homogeneity of an image
- **Maximal Correlation Coefficient** (MCC): he Maximal Correlation Coefficient is a measure of complexity of the texture
- **Inverse Difference Moment Normalized** (IDMN): is a measure of the local homogeneity of an image
- **Inverse Difference** (ID): is another measure of the local homogeneity of an image
- **Inverse Difference Normalized** (IDN): is another measure of the local homogeneity of an image
- **Inverse Variance**
- **Maximum Probability**: is occurrences of the most predominant pair of neighboring intensity values
- **Sum Average**: measures the relationship between occurrences of pairs with lower intensity values and occurrences of pairs with higher intensity values
- **Sum Entropy**: is a sum of neighborhood intensity value differences
- **Sum of Squares**(Variance): is a measure in the distribution of neighboring intensity level pairs about the mean intensity level in the GLCM

3.3.4 Gray Level Run Length Matrix

A Gray Level Run Length Matrix (GLRLM) quantifies gray level runs, which are defined as the length in number of pixels, of consecutive pixels that have the same gray level value. This includes the features:

- **Short Run Emphasis** (SRE): is a measure of the distribution of short run lengths, with a greater value indicative of shorter run lengths and more fine textural textures
- **Long Run Emphasis** (LRE): is a measure of the distribution of long run lengths, with a greater value indicative of longer run lengths and more coarse structural textures
- **Gray Level Non-Uniformity** (GLN): measures the similarity of gray-level intensity values in the image, where a lower GLN value correlates with a greater similarity in intensity values
- **Gray Level Non-Uniformity Normalized** (GLNN): measures the similarity of gray-level intensity values in the image, where a lower GLNN value correlates with a greater similarity in intensity values
- **Run Length Non-Uniformity** (RLN): measures the similarity of run lengths throughout the image, with a lower value indicating more homogeneity among run lengths in the image
- **Run Length Non-Uniformity Normalized** (RLNN): measures the similarity of run lengths throughout the image, with a lower value indicating more homogeneity among run lengths in the image
- **Run Percentage** (RP): measures the coarseness of the texture by taking the ratio of number of runs and number of voxels in the ROI
- **Gray Level Variance** (GLV): measures the variance in gray level intensity for the runs
- **Run Variance** (RV): is a measure of the variance in runs for the run lengths
- **Run Entropy** (RE): measures the uncertainty/randomness in the distribution of run lengths and gray levels
- **Low Gray Level Run Emphasis** (LGLRE): measures the distribution of low gray-level values, with a higher value indicating a greater concentration of low gray-level values in the image

- **High Gray Level Run Emphasis** (HGLRE): measures the distribution of the higher gray-level values, with a higher value indicating a greater concentration of high gray-level values in the image
- **Short Run Low Gray Level Emphasis** (SRLGLE): measures the joint distribution of shorter run lengths with higher gray-level values
- **Short Run High Gray Level Emphasis** (SRHGLE): measures the joint distribution of shorter run lengths with higher gray-level values
- **Long Run Low Gray Level Emphasis** (LRLGLE): measures the joint distribution of long run lengths with lower gray-level values
- **Long Run High Gray Level Emphasis** (LRHGLE): measures the joint distribution of long run lengths with higher gray-level values

3.3.5 Gray Level Size Zone Matrix

A Gray Level Size Zone (GLSZM) quantifies gray level zones in an image. A gray level zone is defined as a the number of connected voxels that share the same gray level intensity. This includes the features:

- **Small Area Emphasis** (SAE): is a measure of the distribution of small size zones, with a greater value indicative of more smaller size zones and more fine textures
- **Large Area Emphasis** (LAE): is a measure of the distribution of large area size zones, with a greater value indicative of more larger size zones and more coarse textures
- **Gray Level Non-Uniformity** (GLN): measures the variability of gray-level intensity values in the image, with a lower value indicating more homogeneity in intensity values
- **Gray Level Non-Uniformity Normalized** (GLNN): measures the variability of gray-level intensity values in the image, with a lower value indicating a greater similarity in intensity values
- **Size-Zone Non-Uniformity** (SZN): measures the variability of size zone volumes in the image, with a lower value indicating more homogeneity in size zone volumes
- **Size-Zone Non-Uniformity Normalized** (SZNN): measures the variability of size zone volumes throughout the image, with a lower value indicating more homogeneity among zone size volumes in the image

- **Zone Percentage** (ZP): measures the coarseness of the texture by taking the ratio of number of zones and number of voxels in the ROI
- **Gray Level Variance** (GLV): measures the variance in gray level intensities for the zones
- **Zone Variance** (ZV): measures the variance in zone size volumes for the zones
- **Zone Entropy** (ZE): measures the uncertainty/randomness in the distribution of zone sizes and gray levels
- **Low Gray Level Zone Emphasis** (LGLZE): measures the distribution of lower gray-level size zones, with a higher value indicating a greater proportion of lower gray-level values and size zones in the image
- **High Gray Level Zone Emphasis** (HGLZE): measures the distribution of the higher gray-level values, with a higher value indicating a greater proportion of higher gray-level values and size zones in the image
- **Small Area Low Gray Level Emphasis** (SALGLE): measures the proportion in the image of the joint distribution of smaller size zones with lower gray-level values
- **Small Area High Gray Level Emphasis** (SAHGLE): measures the proportion in the image of the joint distribution of smaller size zones with higher gray-level values
- **Large Area Low Gray Level Emphasis** (LALGLE): measures the proportion in the image of the joint distribution of larger size zones with lower gray-level values
- **Large Area High Gray Level Emphasis** (LAHGLE): measures the proportion in the image of the joint distribution of larger size zones with higher gray-level values

3.3.6 Neighbouring Gray Tone Difference Matrix

A Neighbouring Gray Tone Difference Matrix (NGTDM) quantifies the difference between a gray value and the average gray value of its neighbours within distance δ. This includes the features:

- **Coarseness**: is a measure of average difference between the center voxel and its neighbourhood and is an indication of the spatial rate of change
- **Contrast**: is a measure of the spatial intensity change, but is also dependent on the overall gray level dynamic range

- **Busyness**: is a measure of the change from a pixel to its neighbour. A high value for busyness indicates a 'busy' image, with rapid changes of intensity between pixels and its neighbourhood
- **Complexity**: is considered complex when there are many primitive components in the image
- **Strength**: is a measure of the primitives in an image. Its value is high when the primitives are easily defined and visible

3.3.7 Gray Level Dependence Matrix

A Gray Level Dependence Matrix (GLDM) quantifies gray level dependencies in an image. A gray level dependency is defined as a the number of connected voxels within distance δ that are dependent on the center voxel. This includes the features:

- **Small Dependence Emphasis** (SDE): a measure of the distribution of small dependencies, with a greater value indicative of smaller dependence and less homogeneous textures
- **Large Dependence Emphasis** (LDE): a measure of the distribution of large dependencies, with a greater value indicative of larger dependence and more homogeneous textures
- **Gray Level Non-Uniformity** (GLN): measures the similarity of gray-level intensity values in the image, where a lower GLN value correlates with a greater similarity in intensity values
- **Dependence Non-Uniformity** (DN): measures the similarity of dependence throughout the image, with a lower value indicating more homogeneity among dependencies in the image
- **Dependence Non-Uniformity Normalized** (DNN): measures the similarity of dependence throughout the image, with a lower value indicating more homogeneity among dependencies in the image
- **Gray Level Variance** (GLV): measures the variance in grey level in the image.
- **Dependence Variance** (DV): measures the variance in dependence size in the image.
- **Low Gray Level Emphasis** (LGLE): measures the distribution of low gray-level values, with a higher value indicating a greater concentration of low gray-level values in the image

- **High Gray Level Emphasis** (HGLE): measures the distribution of the higher gray-level values, with a higher value indicating a greater concentration of high gray-level values in the image
- **Small Dependence Low Gray Level Emphasis** (SDLGLE): measures the joint distribution of small dependence with lower gray-level values
- **Small Dependence High Gray Level Emphasis** (SDHGLE): measures the joint distribution of small dependence with higher gray-level values
- **Large Dependence Low Gray Level Emphasis** (LDLGLE): measures the joint distribution of large dependence with lower gray-level values
- **Large Dependence High Gray Level Emphasis** (LDHGLE): measures the joint distribution of large dependence with higher gray-level values

4 Case Study

In all the previous sections we have analyzed core concepts from the domain of biology[1.] and informatics[3.]. We have established a problem, a scope and a target[2.]. In this section we present a strategy to yield survival rate predictions on cancer patients based on a two part strategy which includes the usage of 3D CNN (3D-Unet) and various weak learner approaches (RFC, XGBOOST).

4.1 Environment Info

We begin by giving a report of the packages used as well as the description of the machine where most computations took place.

Starting off with the machine description:

```
System: Windows
Release: 10
Version: 10.0.19044
Machine: AMD64

CPU:
  Intel(R) Core(TM) i7-6700K CPU @ 4.00GHz
    Base speed: 4,00 GHz
    Sockets:    1
    Cores:  4
    Logical processors: 8
    Virtualization: Enabled
    L1 cache:   256 KB
    L2 cache:   1,0 MB
    L3 cache:   8,0 MB
```

RAM:

Capacity: 16,0 GB

Speed: 2133 MHz

Slots used: 2 of 4

Form factor: DIMM

Total possible extension: 31,2 GB

GPU:

NVIDIA GeForce GTX 1060 6GB

Driver version: 31.0.15.1659

Driver date: 23/6/2022

DirectX version: 12 (FL 12.1)

DATA STORAGE:

SSD ADATA SP550 (240GB)

Continuing with a table of basic architecture of software and firmware:

Software / Firmware	Version
Anaconda Navigator	2.2.0
Jupyter Notebook	6.4.12
PyCharm Community Edition	2022.2
Python	3.9.12
CUDA	64_112
CUDA Computational Capabilities	sm_35, sm_50, sm_60, sm_70, sm_75, compute_80
CUDNN	64_8
Libre Office	64_7.3.3.2

Table 2: Machine software and firmware versions

Finishing off with a table of all the packages used in python excluding the basic python packages:

Package	Version
pandas	1.4.3
numpy	1.21.5
keras	2.9.0
matplotlib	3.5.2
sklearn	1.1.1

radiomics	3.0.1
nibabel	4.0.1
SimpleITK	2.1.1.2
imblearn	0.9.1
xgboost	1.6.1
tensorflow	2.9.1

Table 3: Python package versions

These were gathered through the use of Script [Rig information], except the rig information that were written down manually.

4.2 Datasets

In this section we give the overview of the data used in the experiment.

4.2.1 Image Data

The MRI images are acquired through the BraTS competitions datasets over the years. There are duplicate data that are carried over from year to year. These are removed based on the name mapping sheets that are given by the original data distributors. All data are using the compressed nifti[49] medical imaging protocol (including masks).

The data are accompanied by clinical metadata csv sheets supplying us with the age of the patient(float), days of survival(int or N/A) and resection status(String or N/A). The survival sheets format is identical across all three datasets.

BraTS19ID	Age	Survival	Resection Status
BraTS19_CBICA_AAB_1	60.4630137	289	GTR
BraTS19_CBICA_AAG_1	52.2630137	616	GTR
BraTS19_CBICA_AAL_1	54.30136986	464	GTR
BraTS19_TCIA02_331_1	84.84383562	187	N/A
BraTS19_CBICA_AAP_1	39.06849315	788	GTR

Table 4: Example of survival data

[49] https://radiopaedia.org/articles/nifti-file-format

Each patient folder contains a segmentation mask and four modalities:

- T1
- T1-CE
- T2
- FLAIR

Dataset	Type	Count
BraTS 2018[1][2][3][4]	HGG	210
	LGG	75
BraTS 2019[1][2][3][4][5]	HGG	259
	LGG	76
BraTS 2020[1][2][3][4][5]	Merged	369
Total		989

Table 5: Patient count for each dataset and type

All sets contain 3 distinct classes with labels:

Class	Label
Background	0
Necrotic core / Non Enhancing Tumor	1
Peritumoral Edema	2
Enhancing Tumor	4

Table 6: Image data class labels

4.2.2 Genomic and transcriptomic data

We obtain the gene expression and miRNA for GBM from Ron Shamir's lab[50][6]. The data are tabular and come in csv format. It needs to be mentioned that the clinical data csv has a fair amount of faulty lines, they are mentioned inside the genomics notebook.

- Gene Expression Data: 538 samples, 12042 genes expressed

Index(Patient_ID)	AACS	FSTL1	ELMO2	CREB3L1	RPS11
TCGA.02.0001.01	6.500551	8.729663	5.511362	4.882953	10.984784
TCGA.02.0003.01	6.539245	9.794400	6.213981	4.836276	10.811245
TCGA.02.0004.01	7.377848	12.059550	7.051738	6.112444	10.436374
TCGA.02.0007.01	7.186891	4.945053	5.230444	5.818606	10.477304
TCGA.02.0009.01	7.675038	10.840095	6.620676	5.333213	10.637267

Table 7: Gene expression data sample

50 http://acgt.cs.tau.ac.il/multi_omic_benchmark/download.html

- miRNA Data: 575 samples, 534 transcriptomes expressed

Index(Patient_ID)	ebv-miR-BART1-3p	ebv-miR-BART1-5p	ebv-miR-BART10
TCGA.02.0001.01	5.855126	5.799428	5.862059
TCGA.02.0003.01	5.801614	5.790478	5.818763
TCGA.02.0004.01	5.771332	5.758764	5.825401
TCGA.02.0006.01	5.763649	5.800184	5.831836
TCGA.02.0007.01	5.818828	5.800582	5.818181

Table 8: Transcriptome expression data sample

- Clinical Data: 629 samples, 137 columns

Column	Value
CDE_DxAge	44.3
CDE_survival_time	353.0
days_to_last_followup	279.0
CDE_vital_status	DECEASED

Table 9: Clinical data sample

4.3 Preprocessing

In this section we execute the preprocessing strategy as described in [3.2]. Multiple sanity checks are being made throughout the scripts to make sure everything is working as intended.

4.3.1 Images

We begin the preprocessing by pathing the image training folder and extracting recursively all files that end with a “nii.gz” suffix. We find that we have accumulated 4945 files. This is normal since we have five distinct file types:

Type	Suffix
Mask	_seg.nii.gz
FLAIR	_flair.nii.gz
T1	_t1.nii.gz
T1-CE (GD)	_t1ce.nii.gz
T2	_t2.nii.gz

Table 10: Raw file suffixes

From there we merge all the files in a single dataframe using the pandas library. We generate a new location mirroring the original folder structure to store the preprocessed data with minimal changes to the dataframe. We move the masks to the new locations as is with their new affixes (“_preprocessed”).

We begin the preprocess pipeline by taking the lines of the primary dataframe of locations one by one. Each image is getting loaded using the nibabel library and kept in memory to later use the affine matrix & object header, both important to guarantee data integrity for further usage of the preprocessed data in a full state.

From each of the four nifti object we extract the image array and we populate a list with four slots, each containing a 3D image of dimensions (240 x 240 x 155) of it's respective modality. Using the "grab_NSD" function we calculate the estimated sigmas with NaN intercept built in along the Z axis, giving us a single matrix with dimensions (155 x 1) for each 3D packet.

Using the estimated sigmas alongside the Z axis we initialize the denoising process through "denoise_process" and parallel cast it on four cores, each core handling a different modality. We apply the NLM filter from the skimage package on each 2D image alongside the Z axis and filter each of the 2D slices as mentioned in [3.2.1.3]. This will return each 3D packet with the Z axis on index 0 instead of index 2, meaning we have flipped the image matrix.

Continuing we initialize a global bias field corrector filter from SimpleITK library and pass each 3D flipped packet into "the bias_field_correction" function. We withdraw the image matrices yet again alongside the Z axis and cast them in Float32 (Real format) as it's a dependency of the corrector function. Performing aggressive multiple Otsu Threshold[51] for histogram_bins = 200, we yield a mask that we cast into uint8, again for dependency issues. Then we proceed by using the global corrector with the 2D image slice and the yielded mask. Then return the 3D packet once again.

Finally using the "data_nesting" function we grab the header and the affine matrix of each image and we remake a nifti image which we save to the modified primary location dataframe. Keep in mind that we use transpose from the numpy package on the image matrix so the axis return to their original locations.

Using random from the core python library we pick a random line from the processed dataframe and load it using "load_pack". Using random again to pick a slice

[51] https://scikit-image.org/docs/stable/auto_examples/segmentation/plot_multiotsu.html

number. Keep in mind for best visual results set the borders 40-50 from the start and at least -20 from the end, in this case low_bound = 50 and high_bound=100. And finish by plotting some pictures.

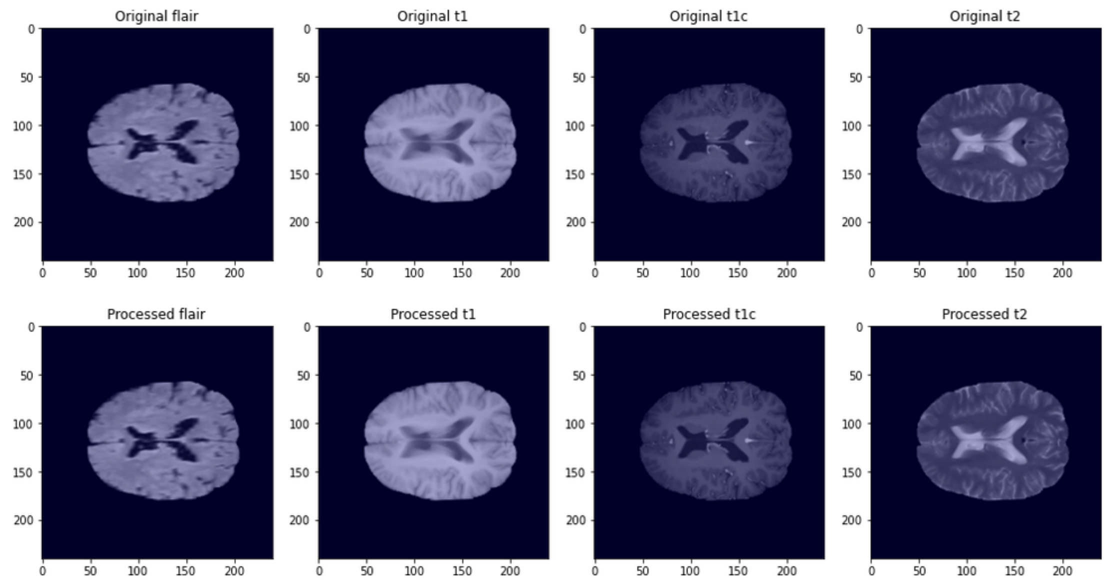


Figure 32: Randomly chosen package #914 and slice #87

The entire process was done using Script [Image Preprocess Pipeline].

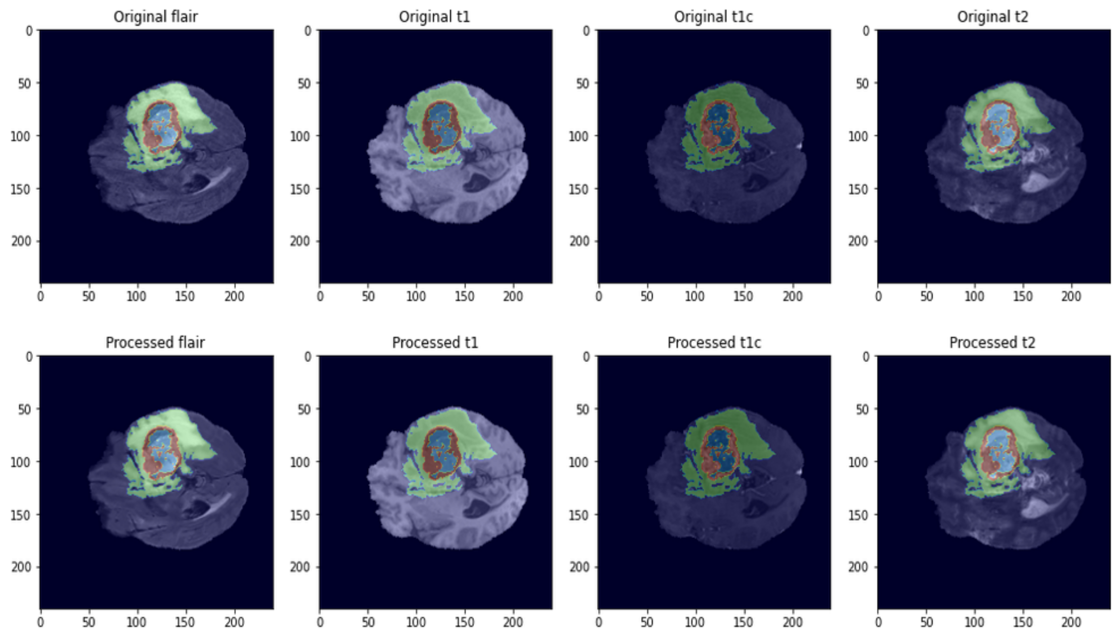


Figure 33: Randomly chosen package with visualized mask

Time for the main preprocess pipeline to run:

Time it took for one line (four 3D images)	80.45 sec
Time estimation for entire dataset	22.10 hours
Actual time for the entire dataset	23.51 hours

Table 11: Elapsed time for preprocess pipeline

4.3.1.1 Bundling

Since the 2D extraction of slices and generation of appropriately 750k 2D slices in order to use Jordan Colmans' modified 2D Unet[52][11] failed, due to computational infrastructure constraints(model was generating 74,6 million trainable parameters) which for the way they were feeding inputs (input being a multi modality image stack); required distributed strategy, we decided to follow technique from Dr. Sreenivas Bhattiprolu[53] were also a super stacked multi modality package is created by stacking the 3D modality blocks on top of each other as shown in Script [Data Bundling] for a slightly modified 3D Unet. Nifti images are normalized into 0-255 range. Furthermore the packets are pruned down to cubes of shape (128 x 128 x 128) for three reasons:

1. To **tackle class imbalance** by pruning the majority of non brain tissue background (black pixels)
2. To enable a 3D Unet to be able to perform **filter generation** without the need for asymmetrical paddings
3. To be **less computationally expensive** when passed through a 3D Unet

Our newly formed four dimensional MRI packets ([0-2]x 240 x 240 x 155) are ready but we are unable to save them as images of any kind, we proceed to save them as numpy arrays and drop them in a merged “data” folder. Masks are also loaded and pruned into (128 x 128 x 128) cubes. If a mask has less than 1% of valuable information on it's entirety the mask is dropped and the numpy array with the corresponding mask is purged. The masks that remain get the non existent class label nullified by turning the label “4” into label “3”. This happens to again help out the training process of the modified Unet by having a range of labels [0,1,2,3] without interrup-

[52] https://github.com/jordan-colman/DR-Unet104
[53] https://youtu.be/ScdCQqLtnis

tions. Lastly the masks get synced with the packets and are also moved in the merged data folder sharing a same numeric affix.

4.3.1.2 Dataset Split

As a last step to finalize the preprocess of the imaging dataset, we use the train_test_split function from the package sklearn.model_selection by passing it as input a dataframe with the locations of the paired mask and arrays for a 20% split into a validation set for the Unets' training. Then we run again the 80% dataset to get a 10% test set for the Unets' categorical accuracy. Our final data cluster ensures that we have tackled a plethora of issues that can arise from training a DNN with raw data that suffer from severe class imbalance, noise, etc. Our final data quantity is as follows:

Set	Quantity of numpy array objects
Train	662
Validation	166
Test	92

Table 12: Final Image data quantities

4.3.2 Multi-omic Data

The preprocessing strategy for the multi-omic data includes the separation of the dataset to two distinct classes which we extract from the clinical data through these tags:

a) CDE_Status: Living & days_to_last_followup > 100
b) CDE_Status: Deceased

Samples that don’t belong to either of these classes (containing N/A or second part of (a) not satisfied) are dropped from the dataset. Lastly, we take only the intersection of patient_IDs in both the gene expression data and miRNA. This generates:

Class	Quantity
CDE_Status == ’Living’ & days_to_last_followup > 100	103
CDE_Status == ‘Deceased’	387
Total	490

Table 13: Multi-omic class sample quantities

We observe that the classes are imbalanced. Copying the data to a new dataframe and by utilizing the package imblearn we initialize three different methods to balance our data distribution as shown in [3.2.2.4]:

- SMOTE
- BorderlineSMOTE
- ADASYN

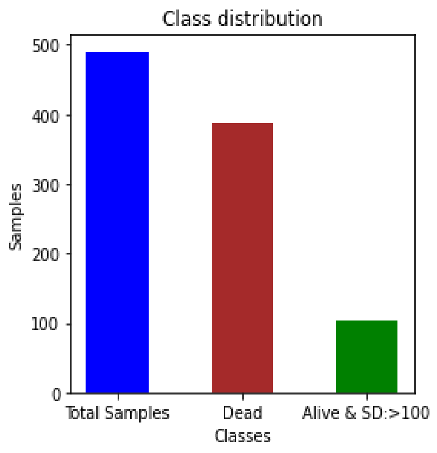


Figure 35: Original class distribution

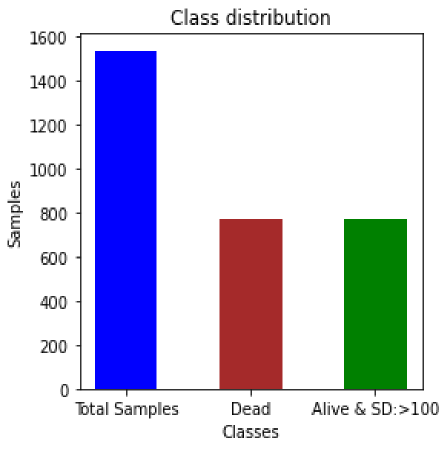


Figure 34: Distribution of classes after oversampling minority class

Creating a copy of the balanced dataframes and using the min_max_scaler from the package sklearn, we create a normalized version of the dataframe to compare with it's non normalized counterpart. Our four dataframes are now ready for the classifiers [5.6].

Going back to the original dataframes, we concatenate the gene expression data and the miRNA data into a singular dataframe alongside the index (we've already pruned non intersection members at the start of [5.3.2]. And finally append the survival (in days) of the patients as a feature column in the dataframe.

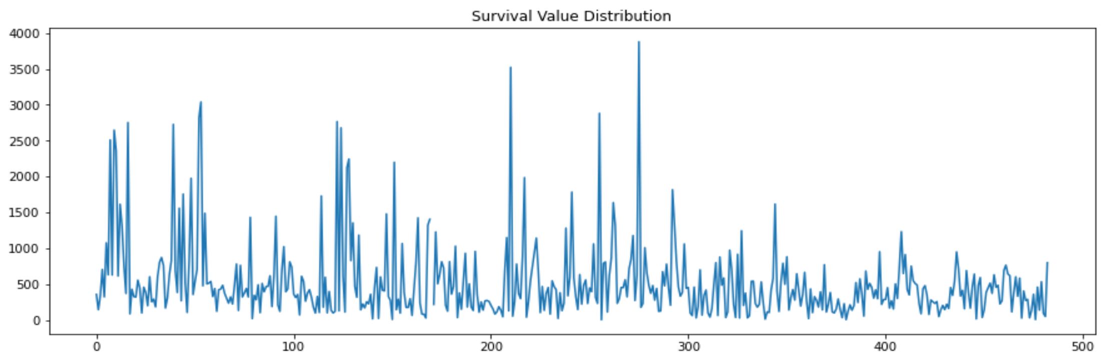


Figure 36: Survival value distribution

Observing multiple outlier cases we decide to prune the dataset by dropping the top 5% of the values (n=24) due to them holding over 45% of the value range upper limit, alongside their samples. Then we normalize the entire dataframe feature wise. The final dataframe shape is (457 x 12576). The finalized survival value distribution now looks smoother.

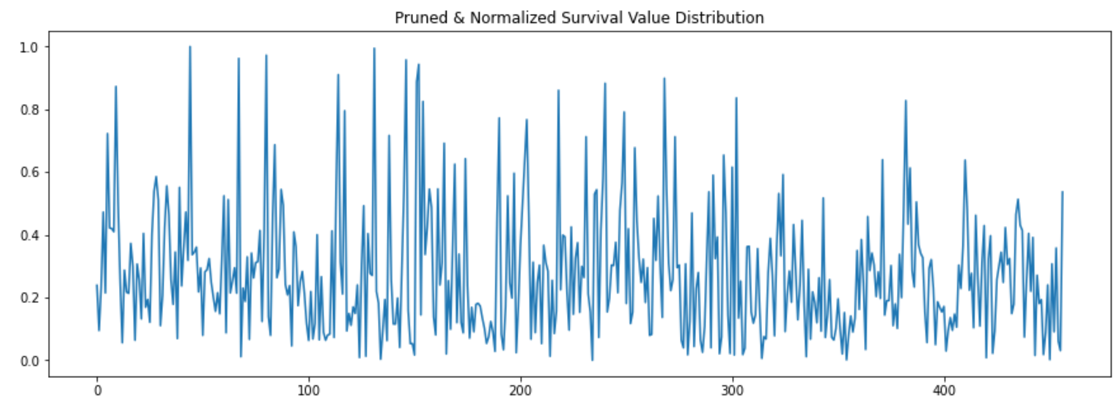
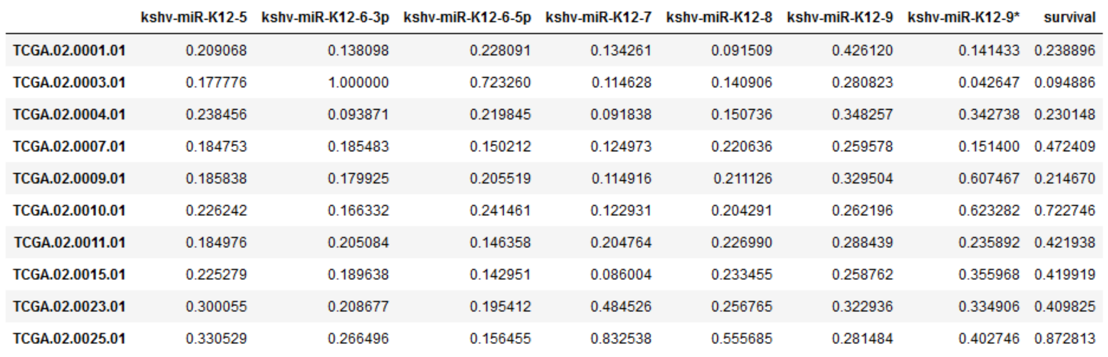


Figure 37: Pruned and normalized survival value distribution



	kshv-miR-K12-5	kshv-miR-K12-6-3p	kshv-miR-K12-6-5p	kshv-miR-K12-7	kshv-miR-K12-8	kshv-miR-K12-9	kshv-miR-K12-9*	survival
TCGA.02.0001.01	0.209068	0.138098	0.228091	0.134261	0.091509	0.426120	0.141433	0.238896
TCGA.02.0003.01	0.177776	1.000000	0.723260	0.114628	0.140906	0.280823	0.042647	0.094886
TCGA.02.0004.01	0.238456	0.093871	0.219845	0.091838	0.150736	0.348257	0.342738	0.230148
TCGA.02.0007.01	0.184753	0.185483	0.150212	0.124973	0.220636	0.259578	0.151400	0.472409
TCGA.02.0009.01	0.185838	0.179925	0.205519	0.114916	0.211126	0.329504	0.607467	0.214670
TCGA.02.0010.01	0.226242	0.166332	0.241461	0.122931	0.204291	0.262196	0.623282	0.722746
TCGA.02.0011.01	0.184976	0.205084	0.146358	0.204764	0.226990	0.288439	0.235892	0.421938
TCGA.02.0015.01	0.225279	0.189638	0.142951	0.086004	0.233455	0.258762	0.355968	0.419919
TCGA.02.0023.01	0.300055	0.208677	0.195412	0.484526	0.256765	0.322936	0.334906	0.409825
TCGA.02.0025.01	0.330529	0.266496	0.156455	0.832538	0.555685	0.281484	0.402746	0.872813

Table 14: Merged dataframe sample

The entire process took 2 minutes to complete, with Script [Multi-omics]

4.4 Image segmentation

In this section we analyse the modifications done to a standard CNN (3D Unet) as showcased in [3.3.4.1]. Then we briefly talk about hyperparameters and analyse the model and strategy used. Lastly, we present some training data and results. The theory is explained in [3.3.4]

4.4.1 Data Generators

Since we cannot use the internalized data generators from keras due to the nature of our data packs, keras will only support up to 3D representations and our data packets are in the fourth dimension due to a modality channel stack, we create custom generators. A custom generator is basically a function that instead of the "return" statement is using the "yield" statement. We do this by using Script [Data Generator].

Depending on our batch size, the generator will return a block of the numpy array objects as mentioned in [5.3.1.1]. In this case the batch_size is set to 1. This is because of the limited GPU memory that can only facilitate the model itself and one data packet at a time before running out of VRAM.

We create two data generators:

- Training data generator, which will feed the U-net
- Validation data generator, which will be used at the end of each epoch to assess the model

4.4.2 Hyperparameters, optimizer, and callbacks

In this section we briefly describe our optimizer, hyperparameter settings, the callbacks we use and how these function

4.4.2.1 Hyperparameters

- **Batch_size** = 1, reasons explained in [5.4.1]

- **Epochs**: depending on the model [0-6] different approaches were taken.
  From mini models of 5-10 epochs to the main model of 100 epochs.

- **LR**: default at 1e-4 but often times used 1e-3 or even 1e-2 to train[Train] and retrain[Retrain] models for 5 epochs with a LR_decay_rate = LR / Epochs to simulate cyclical LR[14], therefore skipping the need to tune it as a hyperparameter. This happens due to the nature of the model, each epoch takes approximately 15minutes and another 3 minutes to finish validation and update weights for a total of 18-20 minutes per epoch.

4.4.2.2 Optimizer

For the experiments in [4.4.3] a standard Adam optimizer (as showcased in [3.3.4]) was used. We are supplying it with a learning rate and a decay rate as showed in [4.4.2.1].

4.4.2.3 Callbacks

Model callbacks are functions that are called after every epoch. We use:

- **model_save**: used from tensorflow.keras.callbacks package to save the weights of the model every time it's prediction capability increases on the validation set.

- **Tensorboard**[54]:
  - Tracks and visualizes metrics such as loss and accuracy, etc.
  - Visualizes the model graph
  - Views histograms of weights, biases as they change over time

4.4.3 Model architecture

The model architecture is a modified version of a 3D Unet to facilitate the special data packages that we are feeding it as inputs. The only difference from a 3D Unet as showcased in [3.2.3.1.1] is that it has two more input channels cause the input shape to contain:

- **Batch_IDX**, self nulled due to current batch_size = 1
- **X** – Image Height = 128
- **Y** – Image Width = 128
- **Z** – Image Depth = 128
- **C** – Image Channels = 4, this is the range of the labels we want to segment

The input packet that goes into the network is an array containing:

[(none), (X, Y, Z), C]

The weights are initialized using he_uniform[55] transform from tf.keras. This means that it draws samples from a uniform distribution within [-limit, limit], where

54 https://www.tensorflow.org/tensorboard
55 https://www.tensorflow.org/api_docs/python/tf/keras/initializers/HeUniform

limit = $\sqrt{(6/qinput)}$,qinput is the number of input units in the weight tensor.

Overall the model is standing at 5,645 million parameters, all of them being trainable. We instantiate the model from Script [Train], the model itself is stored in Script [Unet].

4.4.4 Model Training

In this section we show graphs of metrics and losses of the two most prominent models. Overall six models were trained and two of them were retrained to boost categorical accuracy by utilizing cyclical LR simulation. Despite best efforts to avoid unfair prediction, the sparsity in the dataset is too great without some serious data augmentation.

- Model A (1):

  Trained for epochs = 100

  LR = 1e-4

  Decaying_LR = Yes

  Elapsed time: approximately 34hours

  MeanIoU over the test set: 82%

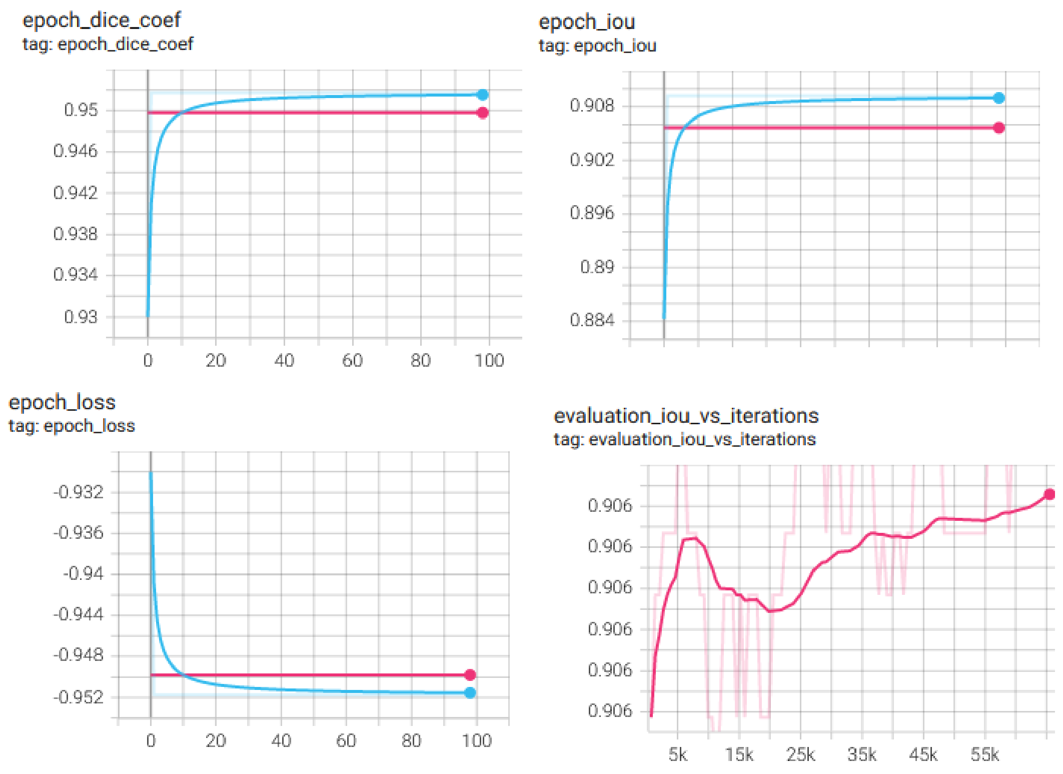


Figure 38: Tensorboard Graphs for model A

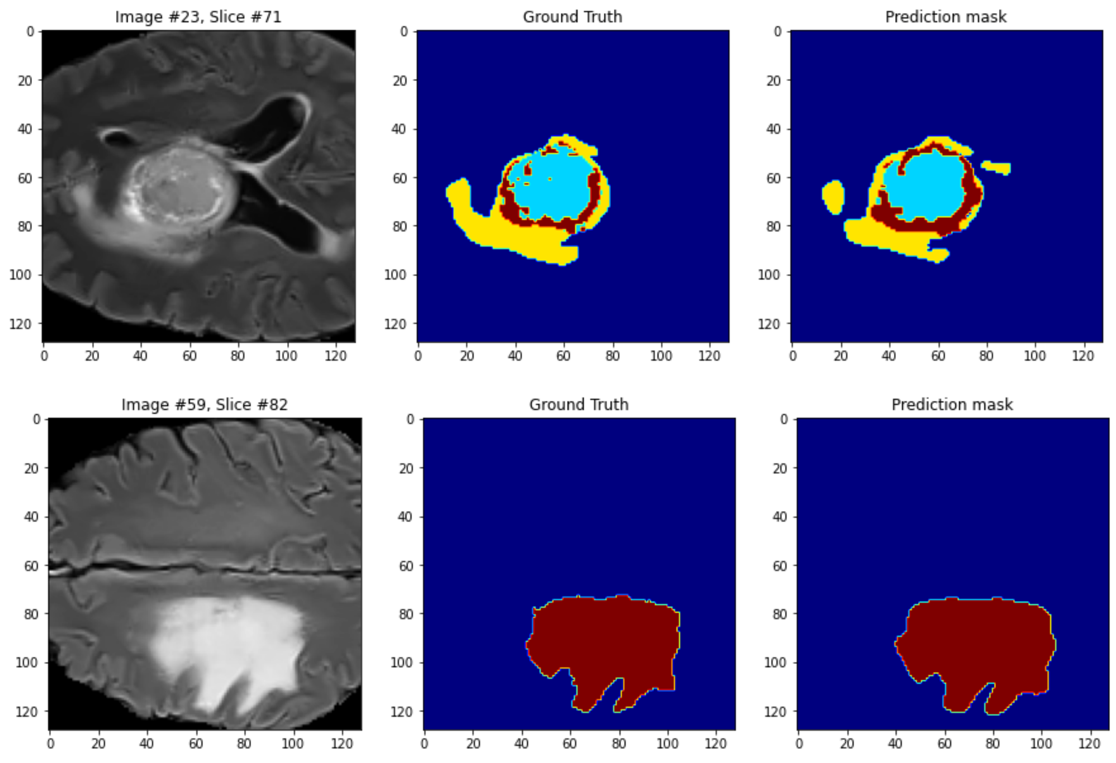


Figure 39: Predictions for model A

- Model B (3.1):

  Trained for epochs = 20, in segments of 5 to reset it's LR

  LR = 1e-3

  Decaying_LR = Yes

  Elapsed time: approximately 7hours

  MeanIoU over the test set: 78%

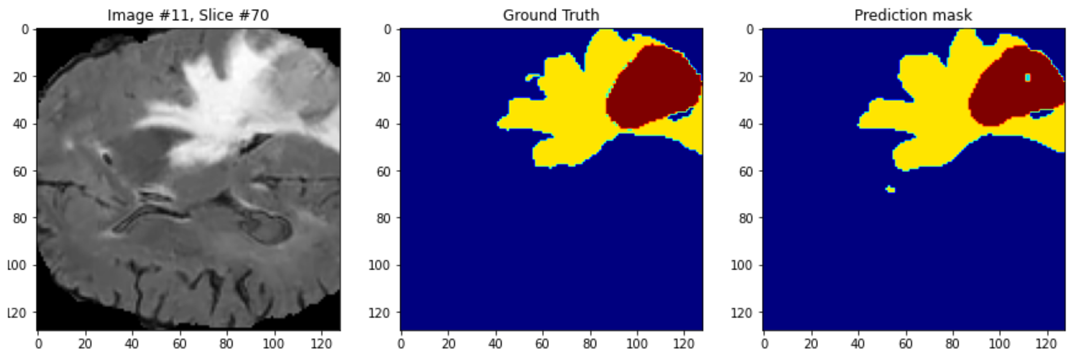


Figure 40: Prediction for model B

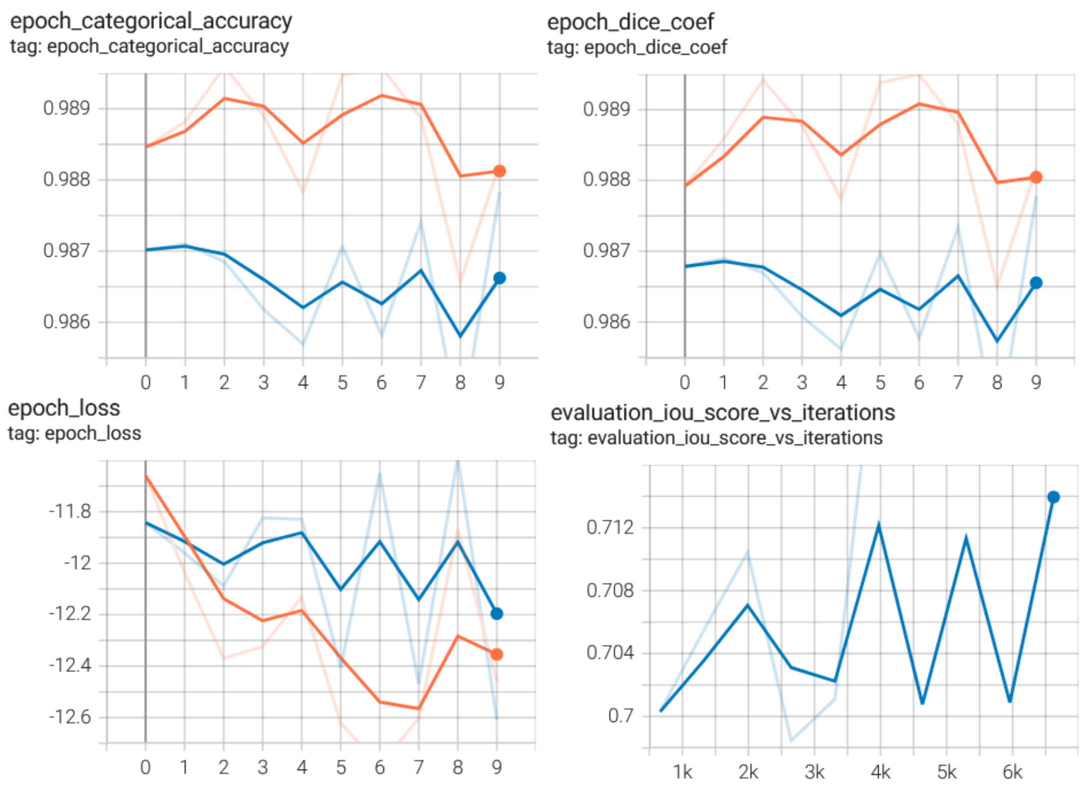
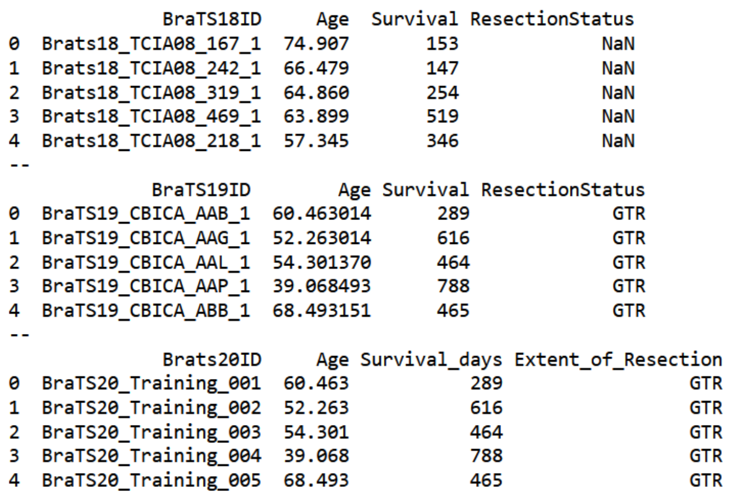


Figure 41: Tensorboard graphs for model B

4.5 Feature Extraction

To extract the imaging features that are thoroughly explained in [3.3] first we must yield the survival data from three different survival csv files. We begin by populating our location dataframe as we have done in previous chapters of this section. We then load the survival data of all three csv files.



```
                BraTS18ID     Age  Survival ResectionStatus
0  Brats18_TCIA08_167_1  74.907       153             NaN
1  Brats18_TCIA08_242_1  66.479       147             NaN
2  Brats18_TCIA08_319_1  64.860       254             NaN
3  Brats18_TCIA08_469_1  63.899       519             NaN
4  Brats18_TCIA08_218_1  57.345       346             NaN
--
               BraTS19ID        Age Survival ResectionStatus
0  BraTS19_CBICA_AAB_1  60.463014      289             GTR
1  BraTS19_CBICA_AAG_1  52.263014      616             GTR
2  BraTS19_CBICA_AAL_1  54.301370      464             GTR
3  BraTS19_CBICA_AAP_1  39.068493      788             GTR
4  BraTS19_CBICA_ABB_1  68.493151      465             GTR
--
                Brats20ID     Age Survival_days Extent_of_Resection
0  BraTS20_Training_001  60.463           289                 GTR
1  BraTS20_Training_002  52.263           616                 GTR
2  BraTS20_Training_003  54.301           464                 GTR
3  BraTS20_Training_004  39.068           788                 GTR
4  BraTS20_Training_005  68.493           465                 GTR
```

Figure 42: Sample of survival csv files

Using the patient IDs from the locations dataframe, we cross reference it with the survival sets to yield the survival days of each patient. If the patient does not exist inside the survival sets or the survival days are N/A, we place a NaN placeholder in the list, on it's hypothetical index to prune the data later.

After having completed the aforementioned, we count 381 inputs to be NaN. Utilizing dataframe operations we quickly drop these values and reset the dataframe index to intercept potential problems with iloc in the future. We perform frequent sanity checks to ensure data won't go missing. Finally from 989 lines we are left with 608 samples.

We instantiate four empty lists to facilitate each modality. Using the package pyradiomics[56], we enable all imaging features and we instantiate an extractor. We perform parallel casting of the extraction process including the T1 modality. It is acknowledged that the T1 modality is not used in training the models mentioned in [4.4.4] but in our judgement we would have one CPU core stall while the others worked, and the more data we extract the better it's going to be for our survival rate predictors in the next chapter.

Finally we create four dataframes, one for each modality, from the feature lists. We split the extra data (spatial information about the mask, tool versions, etc) from the feature (tabular) data and now we have two dataframes per modality that we save under /main/outputs/ as:

• Extra data	<modality>_extras.csv
• Feature data	<modality>_features.csv

Another csv is created to store the survival days of the 608 sample set and saved in the same folder as the above. For chapter [4.5] the Script [Image Feature Extraction] was used. Time usage for chapter [4.5]:

Operation	Time
Single loc dataframe line	7.01 seconds
Approximation of all data	4263.95 seconds (71.06 minutes)
True time for full feature extraction	4469.47 seconds (74.49 minutes)

Table 15: Time usage for feature extraction

[56] https://pyradiomics.readthedocs.io/en/latest/

4.6 Survival Prediction from Imaging Data

Having extracted the imaging features now we can perform analysis utilizing weak learners as showcased in [3.2.2]. We begin by loading our feature csv files for each modality and appending the survival days as a feature column at the end.

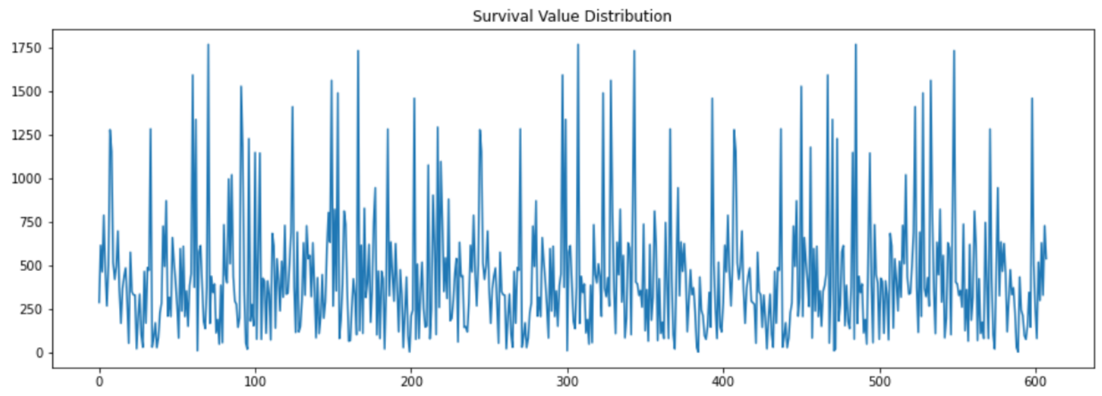


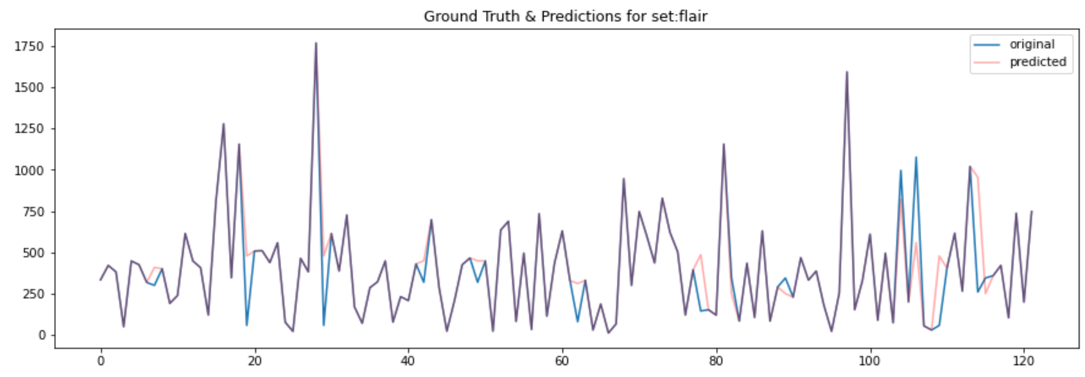
Figure 43: Survival data distribution of the BraTS data

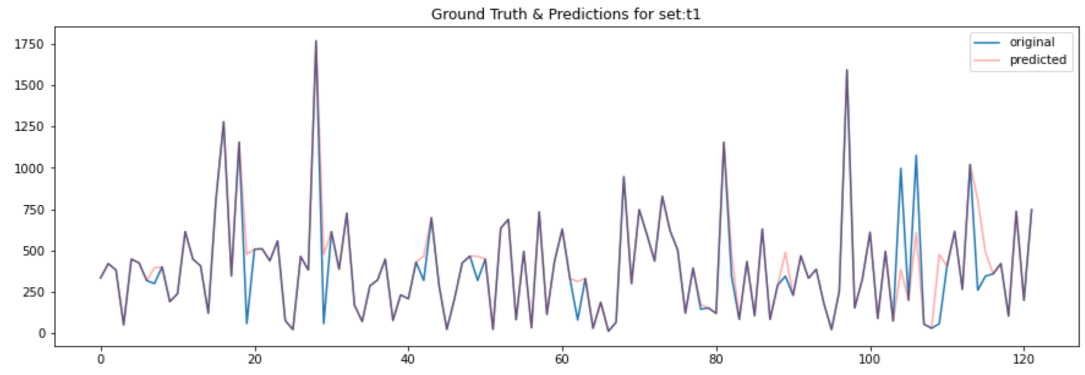
We use the train_test_split method from the sklearn package and split our data into 80% training and 20% test sets. Random state is set to 42. For the four modalities this works fine, but for the merged set we have to combine the previously split sets to not have values from the training set into the test set of the merged set.

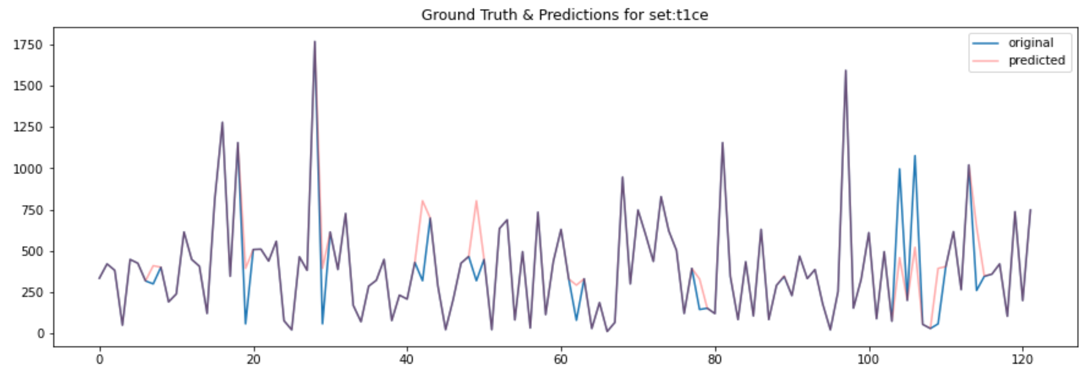
Initializing five XGBOOST Regressors with default settings. And fitting them to our train X_train, Y_train sets as taken from train_test_split.

```
The XGBRegressors took 1.52s to train.

Training score: 87.44% for set: flair
Training score: 87.20% for set: t1
Training score: 86.70% for set: t1ce
Training score: 88.59% for set: t2
Training score: 87.67% for set: merged

K-Fold cross val(n=10) score: 82.26 for set: flair
K-Fold cross val(n=10) score: 76.46 for set: t1
K-Fold cross val(n=10) score: 84.39 for set: t1ce
K-Fold cross val(n=10) score: 79.99 for set: t2
K-Fold cross val(n=10) score: 99.38 for set: merged

MSE: 12732.87 for set: flair
MSE: 12977.93 for set: t1
MSE: 13482.55 for set: t1ce
MSE: 11571.53 for set: t2
MSE: 12503.22 for set: merged
```

```
RMSE: 112.84 for set: flair
RMSE: 113.92 for set: t1
RMSE: 116.11 for set: t1ce
RMSE: 107.57 for set: t2
RMSE: 111.82 for set: merged
```







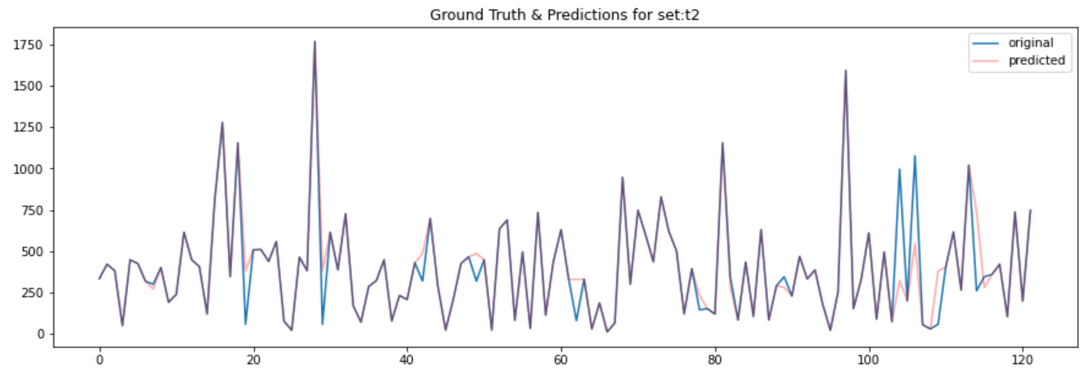


Figure 44: Prediction for non normalized data modality data

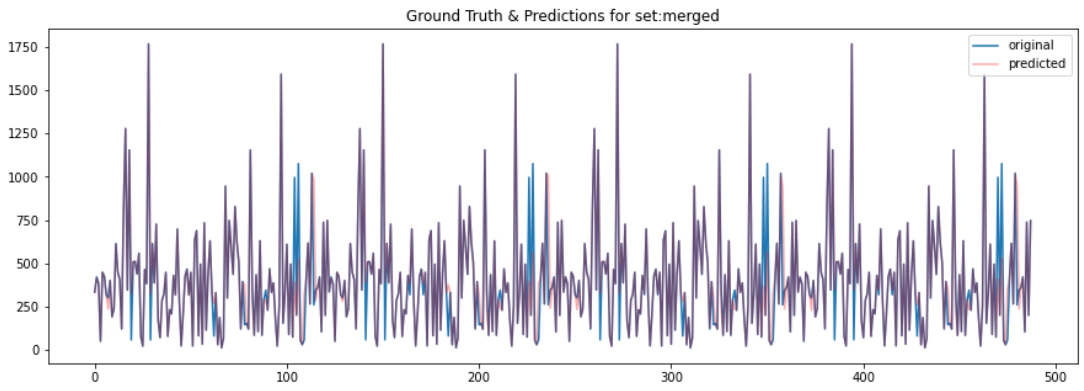


Figure 45: Prediction for non normalized merged modality set

It’s normal for the MSE / RMSE scores to look absurd because the data are not normalized. This is done in order to look for significant changes in the prediction in normalized and non normalized datasets.

Moving forward we perform feature (column) wise normalization. Note that we do not remove any high values due to the dataset being somewhat balanced. After we’re done we repeat the process with new XGB Regressors.

```
XGBRegressors took 1.27s to train.

Training score: 83.03% for set: flair
Training score: 84.75% for set: t1
Training score: 80.42% for set: t1ce
Training score: 86.73% for set: t2
Training score: 85.25% for set: merged

K-Fold cross validation took 12.45s to estimate.

K-Fold cross val(n=10) score: 75.73 for set: flair
K-Fold cross val(n=10) score: 76.63 for set: t1
K-Fold cross val(n=10) score: 87.22 for set: t1ce
K-Fold cross val(n=10) score: 89.00 for set: t2
K-Fold cross val(n=10) score: 98.96 for set: merged

MSE: 0.01 for set: flair
MSE: 0.00 for set: t1
MSE: 0.01 for set: t1ce
MSE: 0.00 for set: t2
MSE: 0.00 for set: merged

RMSE: 0.07 for set: flair
RMSE: 0.07 for set: t1
RMSE: 0.08 for set: t1ce
RMSE: 0.07 for set: t2
RMSE: 0.07 for set: merged
```

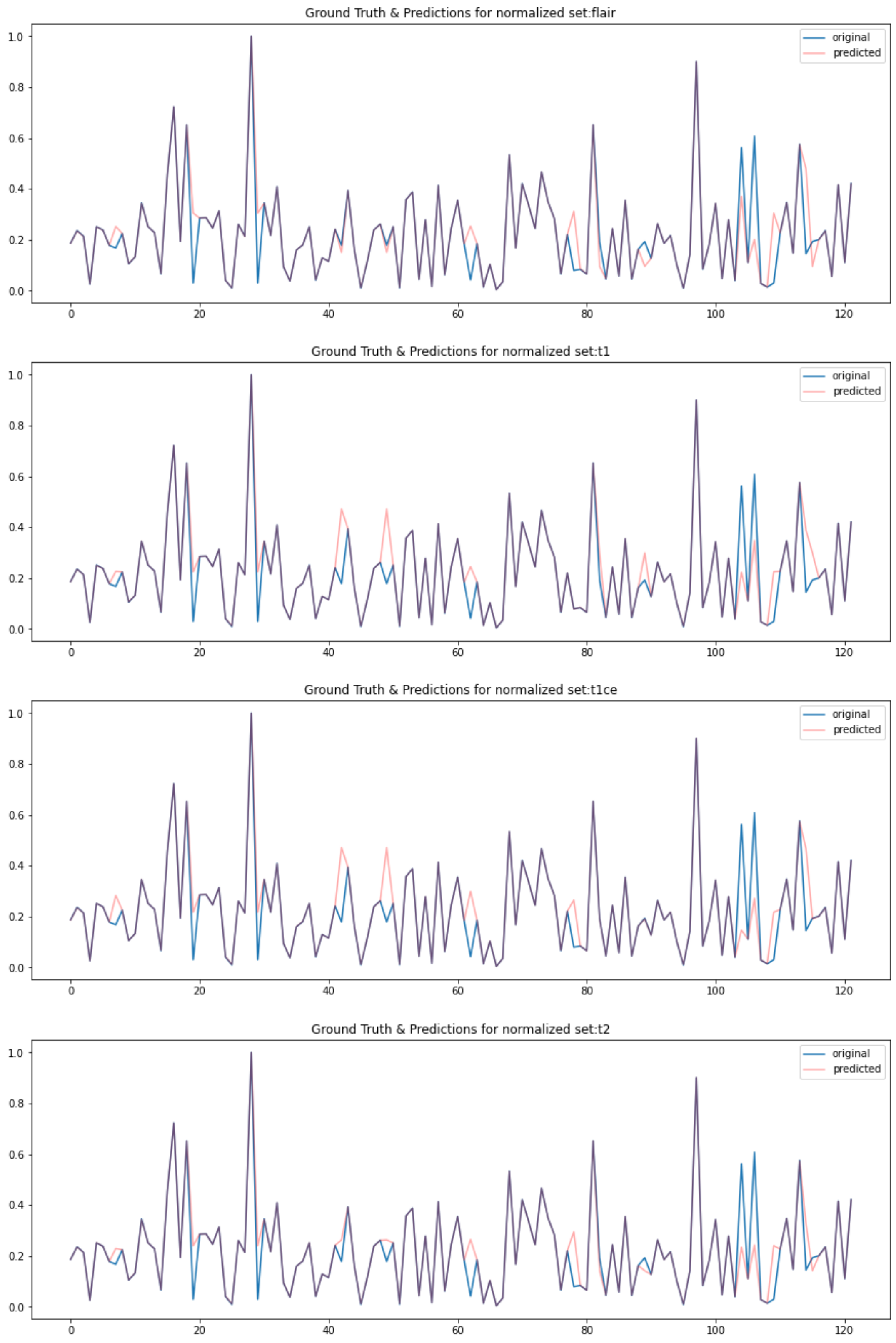


Figure 46: Predictions for normalized modality data

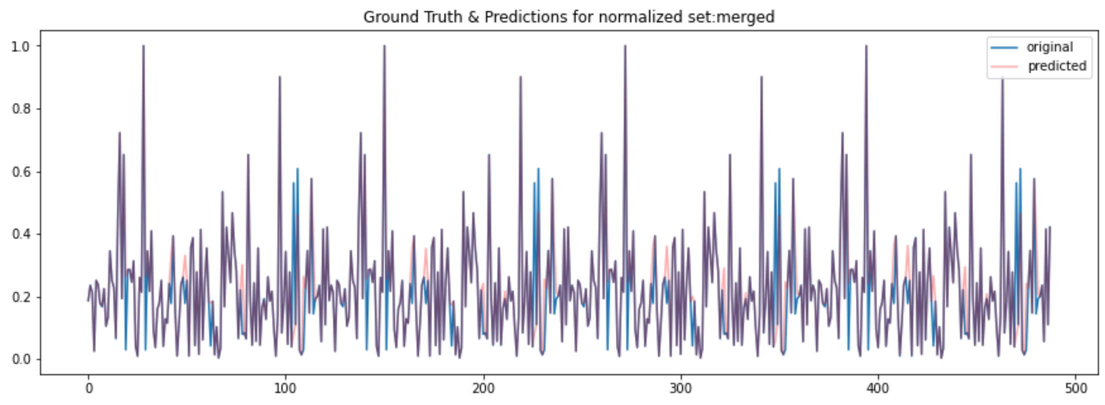


Figure 47: Prediction for normalized merged data

This chapter was completed using Script [Image Feature Survival Prediction]. Time elapsed was approximately 1 minute.

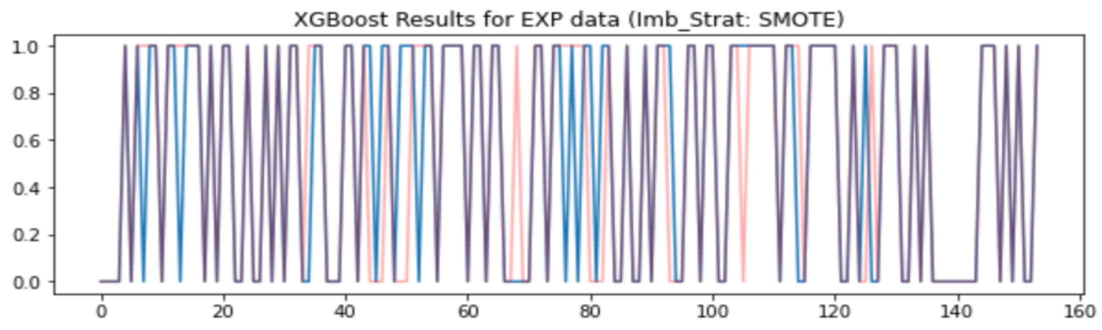
4.7 Multi-omic Data analysis

Lastly, we use the preprocessed data we created in [4.3.2] to create multiple classifiers and finally a regressor for survival prediction on the merged multi-omic set. Multiple classifiers are used to compare between class imbalance control strategies and two weak learners: RFC and XGBOOST for non normalized and normalized data. Script used is still [Multi-omics]

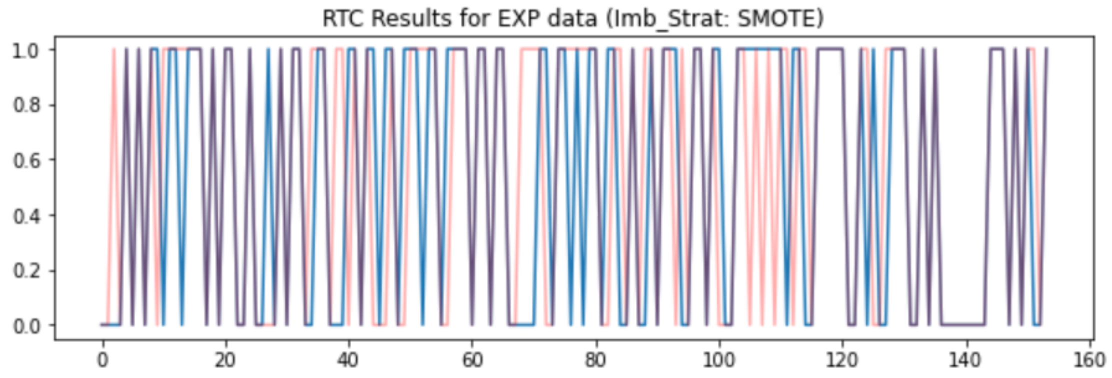
We split all our sets into 80% train and 20% test with random state = 42. Then we initialize for each data set (gene_exp, miRNA) and each oversampling strategy (SMOTE, BorderlineSMOTE, ADASYN) a RFC and a XGB Classifier. Bellow are the results of each one. To avoid having parts of the graph hidden, the legend is turned off.

Starting off with the non normalized sets: (Legend: Original(Blue), Predicted(Red)) Purple means that the prediction matches the ground truth:

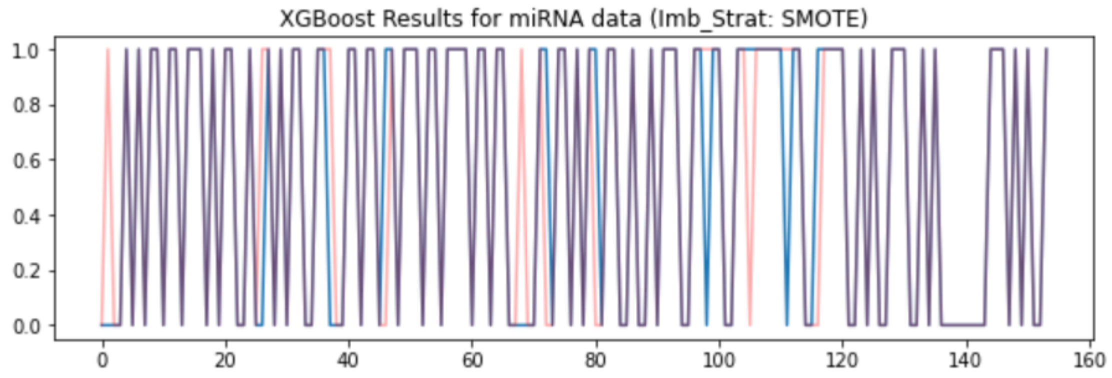
```
Accuracy for XGBOOST on exp: 88.31%
```



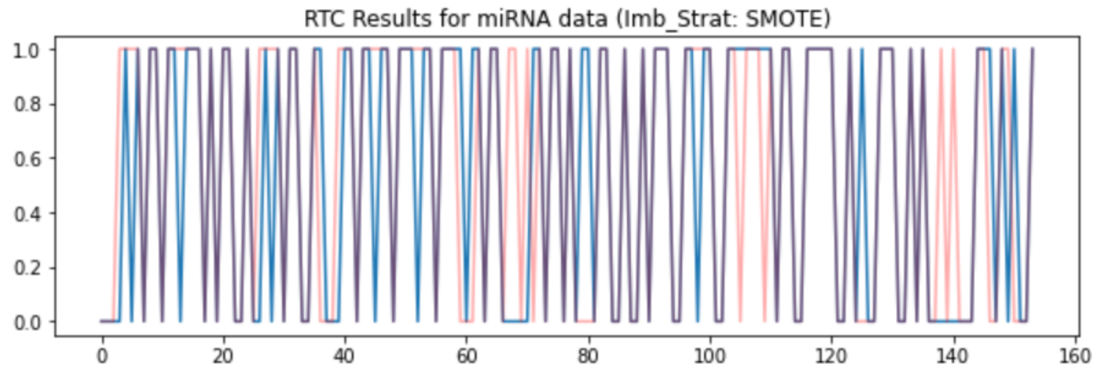
```
Accuracy for RTC on exp: 76.62%
```



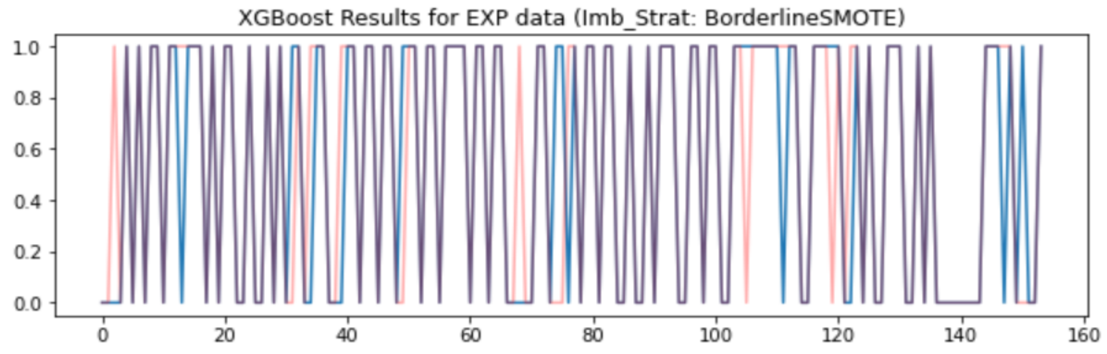
```
Accuracy for XGBOOST on mirna: 92.86%
```



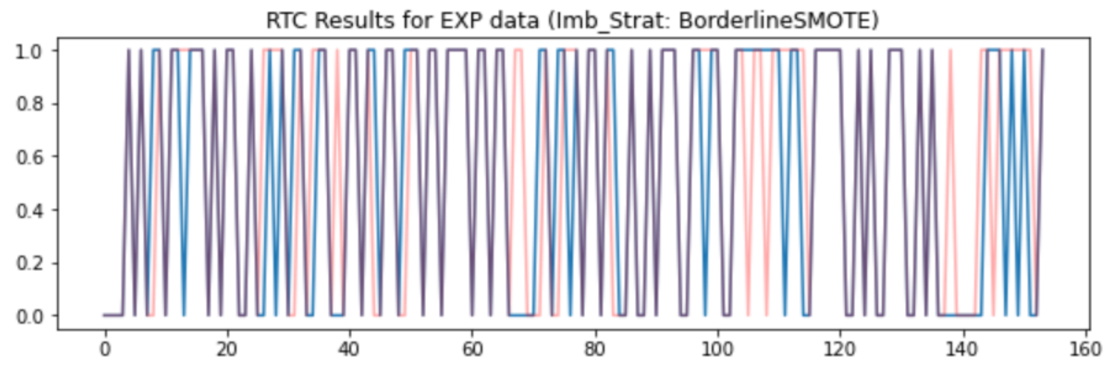
```
Accuracy for RTC on mirna: 83.12%
```



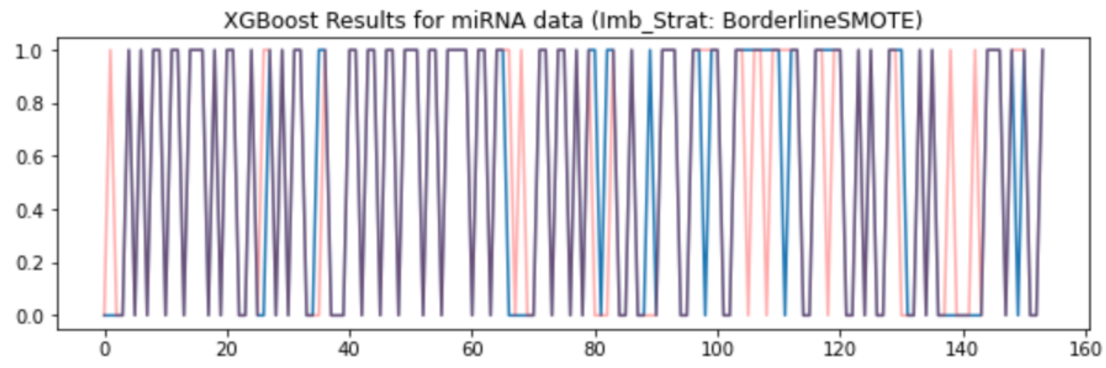
```
Accuracy for XGBOOST on exp: 89.61%
```



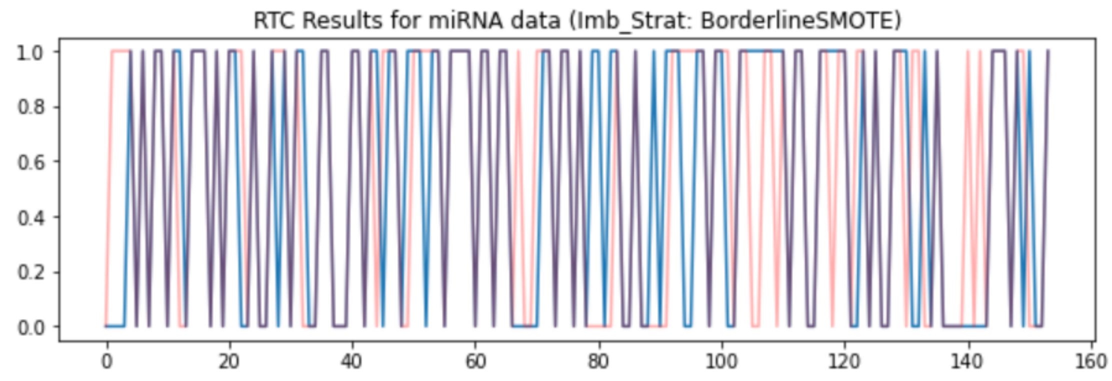
```
Accuracy for RTC on exp: 83.12%
```



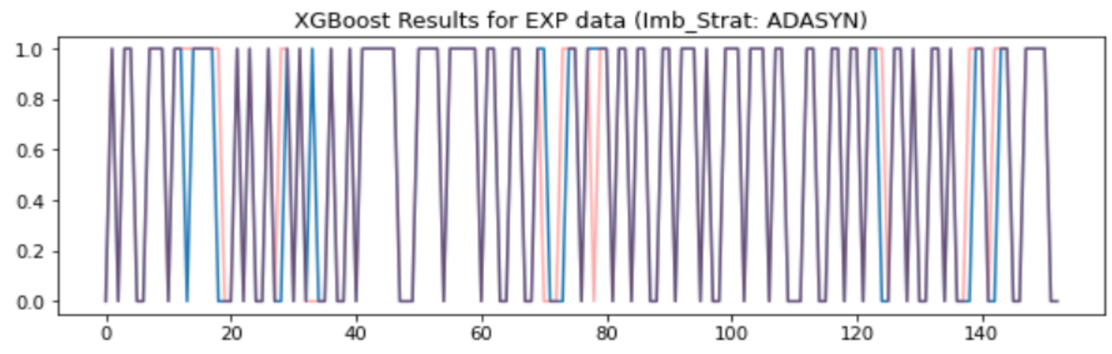
```
Accuracy for XGBOOST on mirna: 88.96%
```



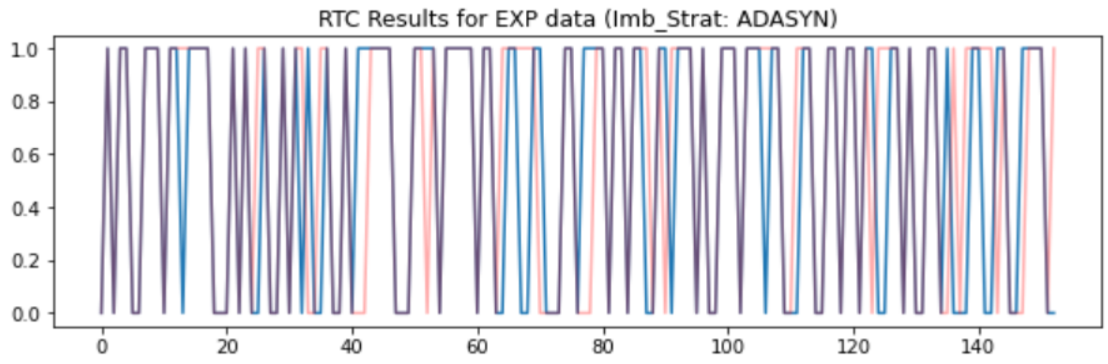
```
Accuracy for RTC on mirna: 77.92%
```



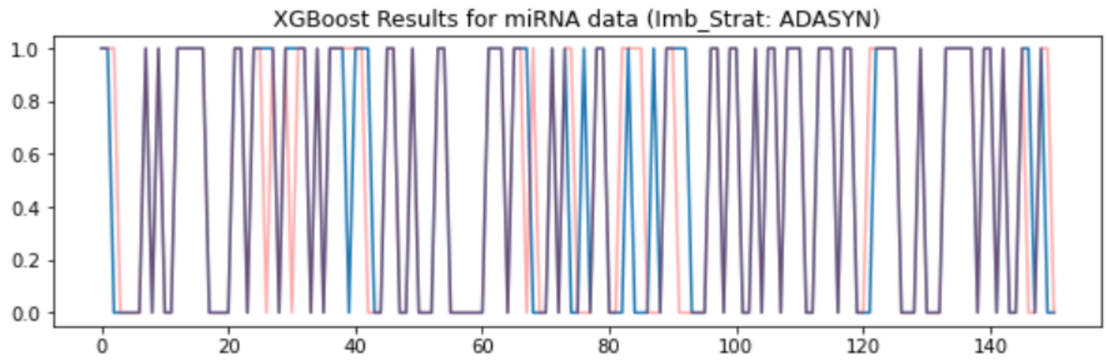
```
Accuracy for XGBOOST on exp: 93.46%
```



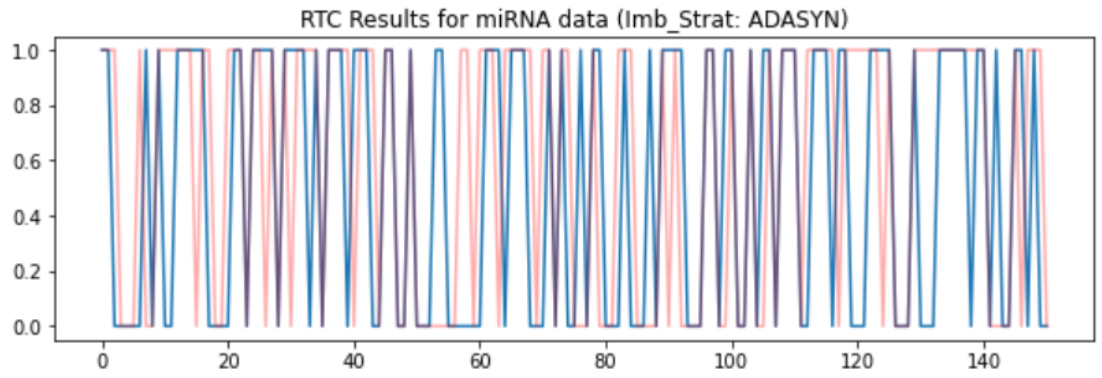
```
Accuracy for RTC on exp: 80.39%
```



```
Accuracy for XGBOOST on mirna: 88.08%
```

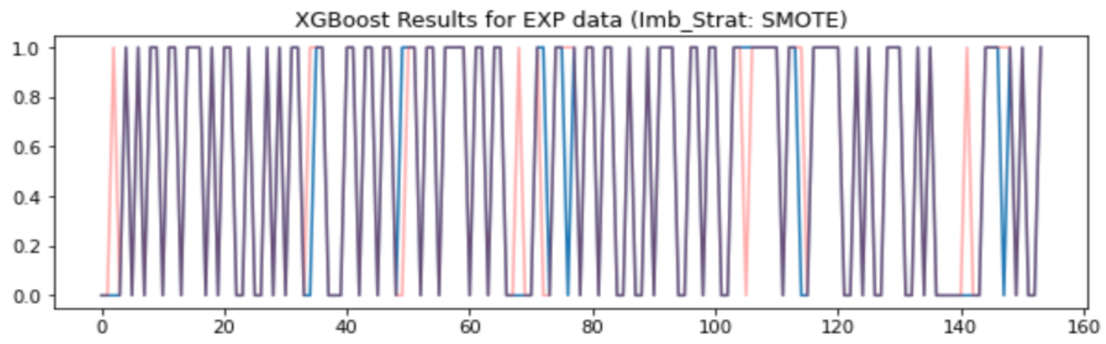


```
Accuracy for RTC on mirna: 68.21%
```

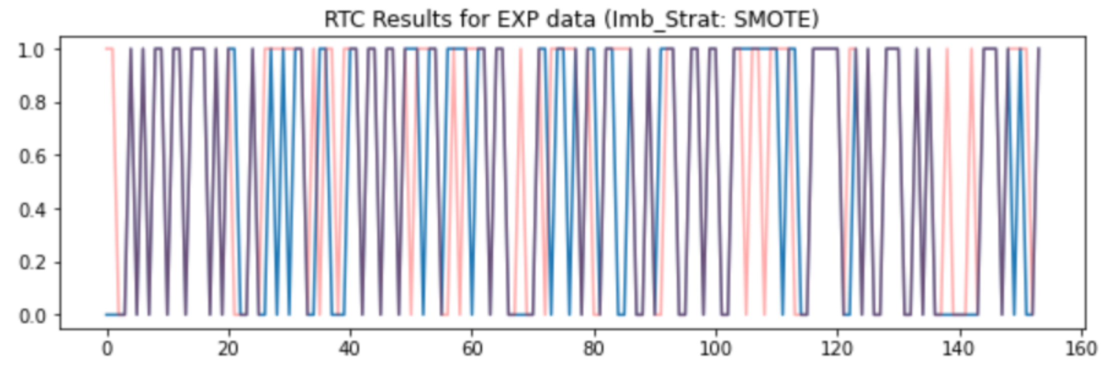


And now we repeat the same process for new classifiers on normalized data.

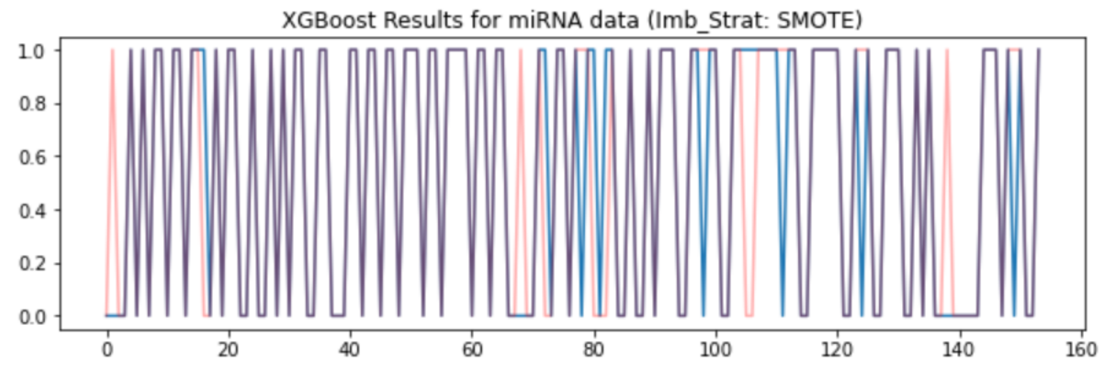
```
Accuracy for XGBOOST on exp: 93.51%
```



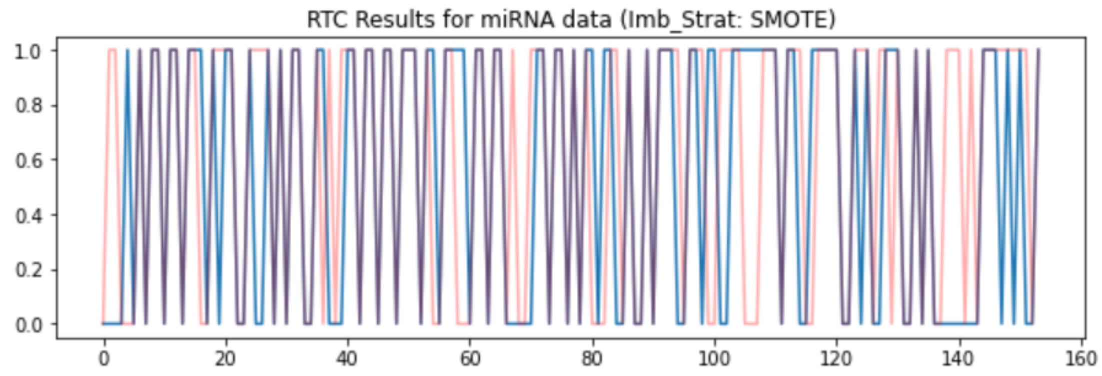
```
Accuracy for RTC on exp: 79.22%
```



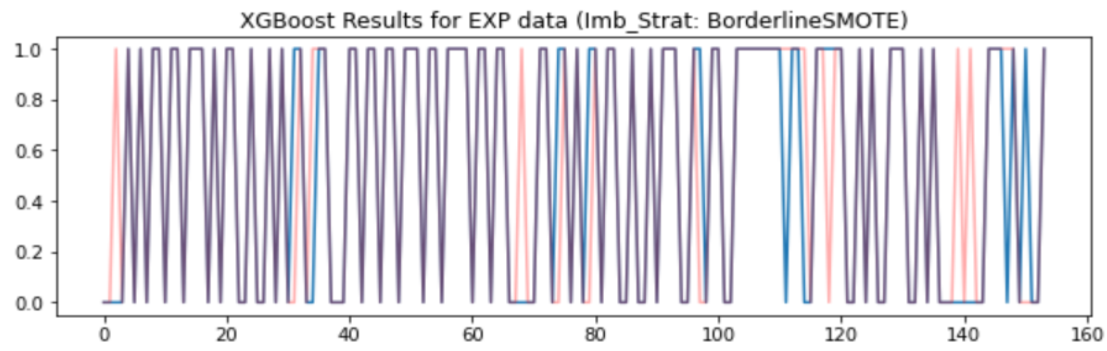
```
Accuracy for XGBOOST on mirna: 90.91%
```



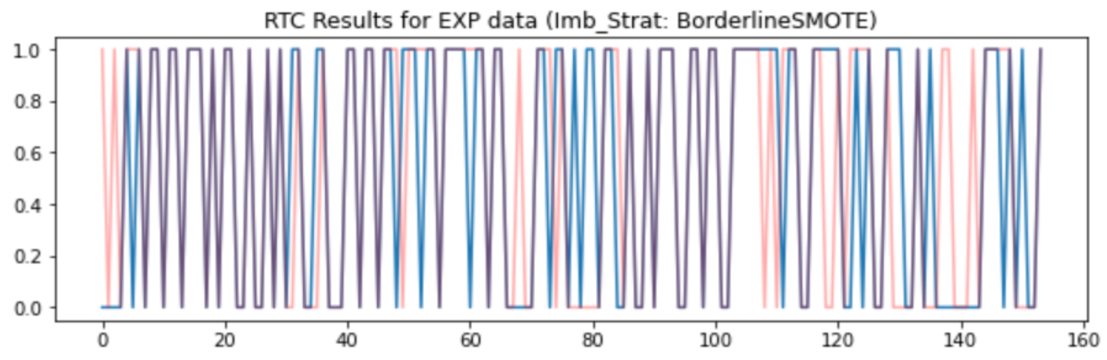
```
Accuracy for RTC on mirna: 74.68%
```



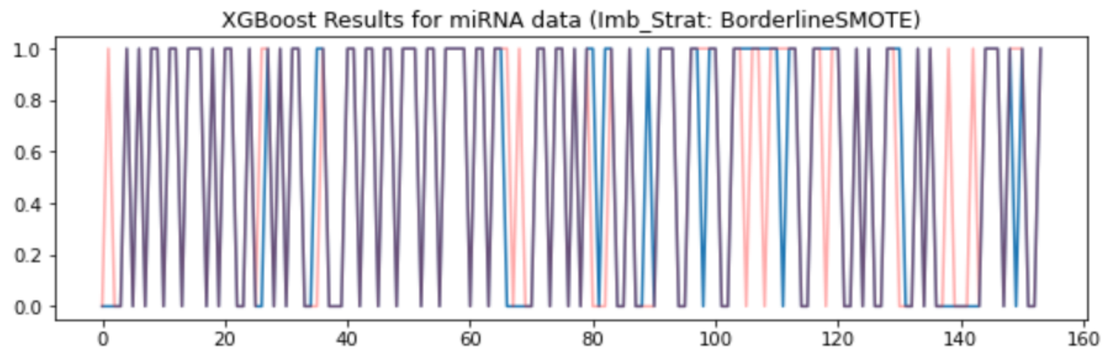
```
Accuracy for XGBOOST on exp: 90.91%
```



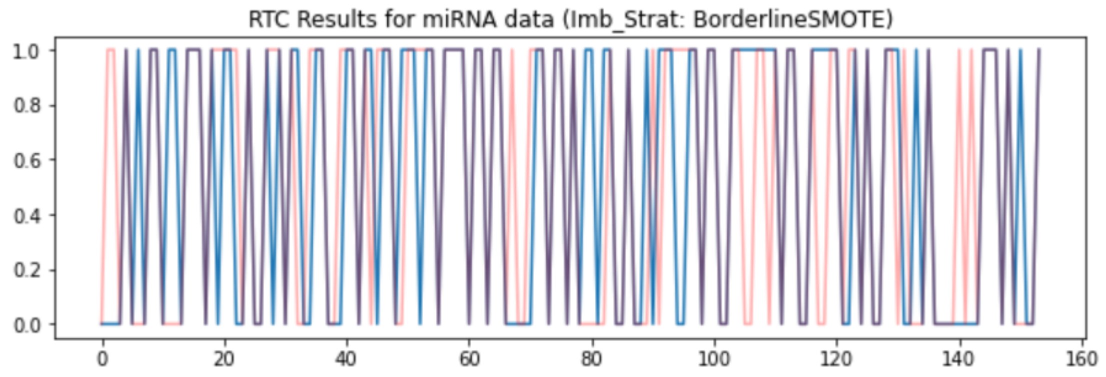
```
Accuracy for RTC on exp: 79.87%
```



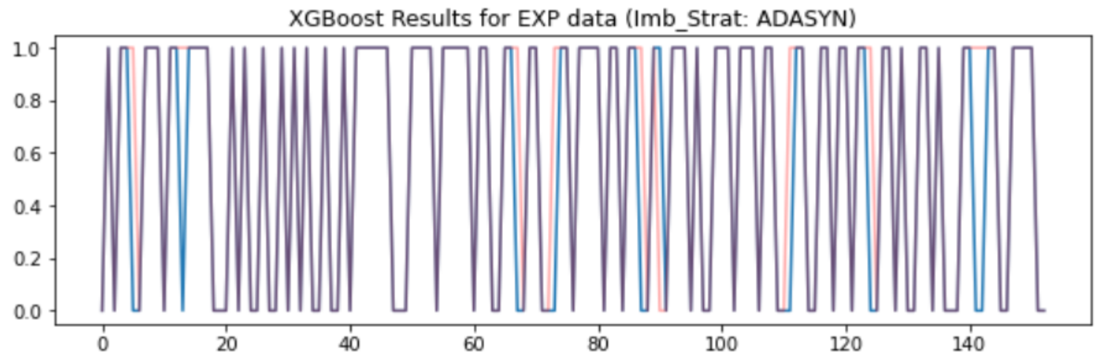
```
Accuracy for XGBOOST on mirna: 88.96%
```



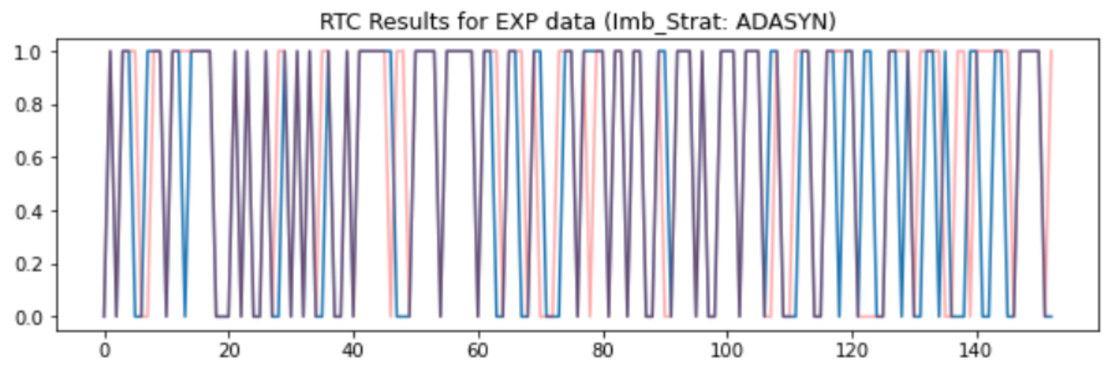
```
Accuracy for RTC on mirna: 75.97%
```



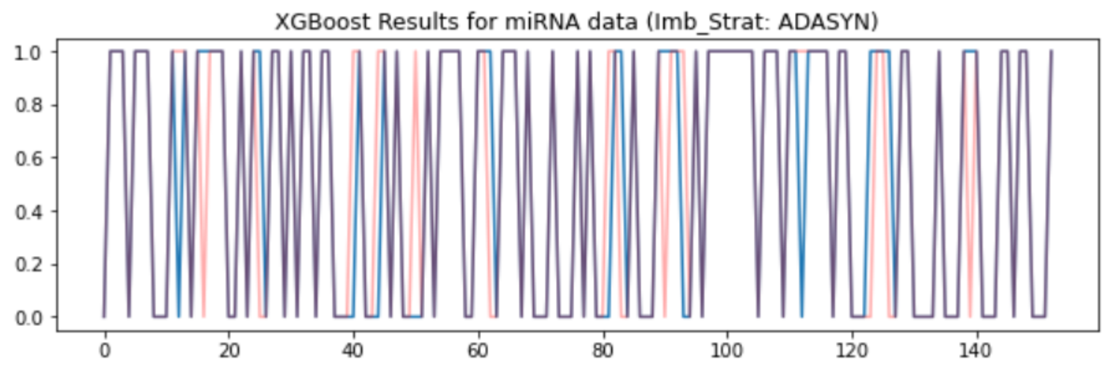
```
Accuracy for XGBOOST on exp: 93.46%
```



```
Accuracy for RTC on exp: 80.39%
```



```
Accuracy for XGBOOST on mirna: 90.20%
```



```
Accuracy for RTC on mirna: 75.16%
```

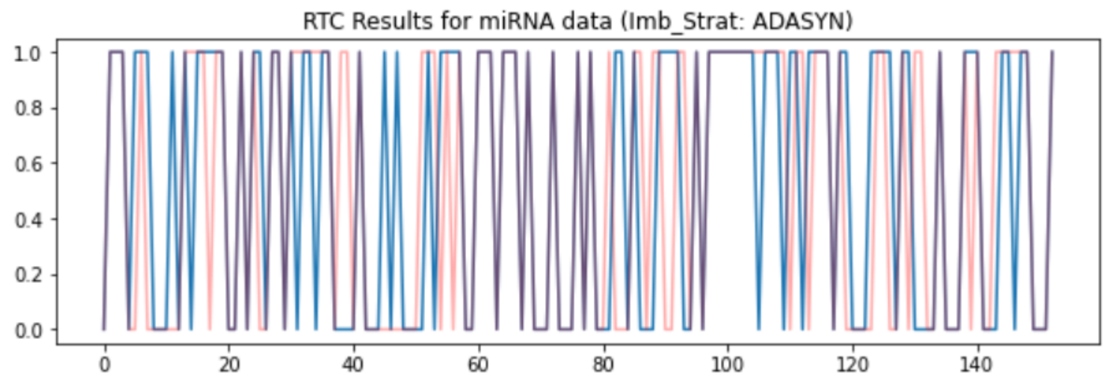


Figure 48: XGB Classifiers predictions

Total time elapsed to train all 24 classifiers was about two minutes.

We now merge the two main datasets (gene_exp & miRNA) and only keep the samples that are in the intersection of the two sets. Furthermore we drop any sample that we don't have the survival days and append the survival days as a feature column at the end of the new dataframe. We are left with 457 samples containing 12577 features.

We perform multiple tests and use K-Fold cross validation[3.2.2.3] with number of splits = 10 and enabled shuffling. Our initial random state is set to 42. Tests performed:

- Raw survival days, normalized data:
  K-fold cross validation took 231.16s with a score of: 55.90%

- Normalized survival days, normalized data:
  K-fold cross validation took 211.29s with a score of: 57.99%

- Normalized & pruned(n=10) survival days, normalized data
  K-fold cross validation took 197.23s with a score of: 27.87%

- Normalized & pruned(n=5%) survival days, normalized data
  K-fold cross validation took 249.02s with a score of: 21.14%

- Normalized & pruned(n=5%) survival days, normalized data, CRS[57] (rt=5)
  K-fold cross validation took 230.84s with a score of: 28.56%

- Normalized survival days, normalized data, RRS[58], CTS[59] (n=.33)
  K-fold cross validation took 213.03s with a score of: 25.07%

Overall time elapsed for the classifiers training was approximately 25 minutes.

```
MSE:  0.05
RMSE: 0.21
```

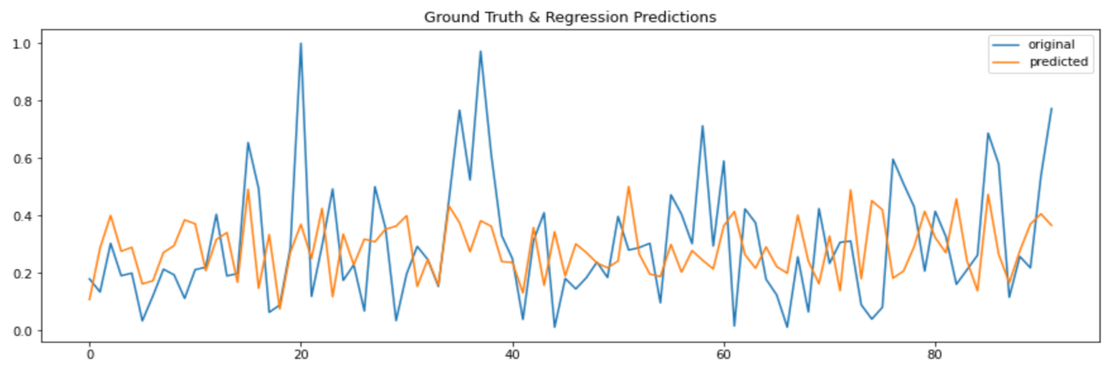


Figure 49: Best Classifier Predictions

[57] CRS = Changed Random State
[58] RRS = Removed Random State
[59] CTS = Changed Test Split

5 Results

Though lengthy our experiment yielded some interesting results.

Total time to run all computations per department:

Operation(s)	Total Time
Preprocessing	24,5 hours
Train 3D CNNs	52-55 hours
Evaluate models	1 hour
Extract radiomic features	75 minutes
Image survival predictor (train & eval)	1 minute
Multi-omic classifier (train & eval)	2 minutes
Multi-omic survival predictor (train & eval)	23 minutes
Total Elapsed	~ 82 hours

Table 16: Computational time for the entire project

The results of our imaging classifiers:

Type	Accuracy
3D CNN (1st model)	82%
3D CNN (2nd model)	78%
3D CNN (3rd model)	75%
3D CNN (4th model)	69%
3D CNN (5th model)	68%
3D CNN (6th model)	42%

Table 17: CNN models categorical accuracy

The results of our imaging survival predictors for non normalized data:

Modality	Train score	K-FCV	MSE	RMSE
FLAIR	87.40%	82.26%	12732.87	112.84
T1	87.20%	76.46%	12977.93	113.92
T1-CE	86.70%	84.39%	13482.55	116.11
T2	88.59%	79.99%	11571.53	107.57
Merged	87.67%	99.38%	12503.22	111.82

Table 18: Imaging survival predictor scores (non normalized)

The results of our imaging survival predictors for normalized data:

Modality	Train score	K-FCV	MSE	RMSE
FLAIR	83.03%	75.73%	0.01	0.07
T1	84.75%	76.63%	<0.01	0.07
T1-CE	80.42%	87.22%	0.01	0.08
T2	86.73%	89.00%	<0.01	0.07
Merged	85.25%	98.96%	<0.01	0.07

Table 19: Imaging survival predictor scores (normalized)

The result of our non normalized multi-omic classifiers:

Set	Classifier	Oversampling Type	Accuracy score
Gene Expression	RFC	SMOTE	76.62%
Gene Expression	RFC	BorderlineSMOTE	83.12%
Gene Expression	RFC	ADASYN	80.39%
Gene Expression	XGBOOST	SMOTE	88.31%
Gene Expression	XGBOOST	BorderlineSMOTE	89.61%
Gene Expression	XGBOOST	ADASYN	**93.46%**
miRNA	RFC	SMOTE	83.12%
miRNA	RFC	BorderlineSMOTE	77.92%
miRNA	RFC	ADASYN	68.21%
miRNA	XGBOOST	SMOTE	92.86%
miRNA	XGBOOST	BorderlineSMOTE	88.96%
miRNA	XGBOOST	ADASYN	88.08%

Table 20: Accuracy score of non normalized multi-omic classifiers

The result of our normalized multi-omic classifiers:

Set	Classifier	Oversampling Type	Accuracy score
Gene Expression	RFC	SMOTE	79.22%
Gene Expression	RFC	BorderlineSMOTE	79.87%
Gene Expression	RFC	ADASYN	80.39%
Gene Expression	XGBOOST	SMOTE	**93.51%**
Gene Expression	XGBOOST	BorderlineSMOTE	90.91%
Gene Expression	XGBOOST	ADASYN	93.46%
miRNA	RFC	SMOTE	74.68%
miRNA	RFC	BorderlineSMOTE	75.97%
miRNA	RFC	ADASYN	75.16%
miRNA	XGBOOST	SMOTE	90.91%
miRNA	XGBOOST	BorderlineSMOTE	88.96%
miRNA	XGBOOST	ADASYN	90.20%

Table 21: Accuracy score of normalized multi-omic classifiers

The K-FCV result of our merged set (gene_exp & miRNA)

Normalized Days	Special	Prune	Score
No	-	No	55.90%
Yes	-	No	**57.99%**
Yes	-	Yes(n=10)	27.87%
Yes	-	Yes(n=5%)	21.14%
Yes	CRS(rt=5)	Yes(n=5%)	28.56%
Yes	RRS & CTS(n=.33)	No	25.07%

Table 22: K-FCV of our merged multi-omic regressor

To summarize, we achieved:

- Cancer diagnosis and semantic segmentation of the tumor at a categorical accuracy of **82%** and **78%** with multiple different approaches utilizing modified 3D Unets and training six models.

- Survival type classification (classes from [4.3.2]) based on gene expression data and miRNA data with our best score coming from the gene expression data classified by a XGB Classifier after the data got normalized and balanced with the use of SMOTE [4.3.2], [4.7] for a score of **93.51%**.

- Merged omic set attempts only yielded a maximum of a score of **57.99%** with **MSE** of **0.05** and **RMSE of 0.21**. Given the oppressive amount of features the dataset had against the amount of samples, we find this normal yet underwhelming.

- And bringing it all together we created survival rate predictors for both the imaging data (by extracting their imaging features and performing analysis with weak learners) reaching a **MSE** of **less than 0.01** in imaging [4.6] and **0.05** in merged multi-omic data analysis [4.7]

6 Discussion & Outlook

The objective of this thesis was to prove that it is possible to merge data obtained by multiple scientific fields (biology & medicine) that are brought together by the field of informatics in order to reach the goal of cancer diagnostic tools such as [4.4.4] and [4.7] as well as survival rate predictions as shown in [4.6] and [4.7]. It also proved that you can approximate a pretty accurate result in survival rates if you have multiple type data sources(imaging, multi-omics, etc.).

Despite our poor results[5.] due to us still being naive to the grand scheme of 3D semantic segmentation (we got a good slap from the danning krueger effect[40]) optimal deep learning with neural networks is by far the bleeding edge in terms of semantic segmentation in medical imaging with scoring as much a 0.95 dice score[39]!

For the multi-omics part on the other hand, some ANN based approaches fall behind [41]. Despite DNNs being able to discover structures inside big data, problems arise when our features far exceed our samples. It's causing the model to over fit resulting in the model loosing predictive capabilities. This is why XGBOOST usually outperforms standard ANNs in tabular data classification & regression[42]. Despite all that in respects to multi-omics as a whole, given the sheer volume of data that exists and continues generating; DNNs will surely play a very important role in the year to come.

Recent scientific literature indicates that quantitative features extracted from multimodal imaging data (CT, MRI, X-Ray) can be used as imaging biomarkers to characterize a lesion. Added to this imaging arsenal, multi-omics data acquisition being performed alongside imaging data acquisition with the result of a slow but steadily increasing quantity of data. Soon we will be able to engage into large scale research with deep learning models in the field of radiogenomics[43]

For our closing remarks we'd like to state that we started this thesis with a serious limitation on time, knowledge and sense of time but despite that still choose to fully embrace the idea behind a thesis (by learning new things and applying them in order to solve a problem[2.1],[2.3]) we have to say that we are positively surprised by the turn of events (in respect to our results[6.]).

Overall, parts of our knowledge in machine learning techniques and bioinformatics have been re-established through trial and error and have set the stage for much more knowledge to come and experiments to be made. For that, we are grateful.

7 Future Prospects

Moving forward in the brave new world of bioinformatics, we would like to further our experimentation on multiple levels. These include but are not limited to preprocessing, training models and data augmentation. Our focus is to expand our research horizons by asking increasingly more complex questions to further understand problems inside the domain of bioinformatics and systems biology as a whole with the aim to figure out solutions.

In the preprocessing part, we would like to experiment with ANN/DNN based methods for denoising as presented in papers like [7], [8] and bias field correction as presented in papers like [9], [10].

In the segmentation model part, we would like to experiment with multiple neural networks, as mentioned already in the aftermath of the failed attempt mentioned in [4.3.1.1]. Another approach we would like to look into is ensemble learning by utilizing different types of neural networks and attempt to make a super accurate segmenter networks with the ability to generalize in brain gliomas. Subsequently, we would also like to use different optimizers(AdaFair [13]) and hyperparameter tuning strategies.

In the regression and classification part we would like to take the time and analyze our features. Due to our limited time, resources and knowledge some important steps were not used that would have yielded optimal or at least better results. Example of this would be [4.7], [4.6] where the entire datasets were used instead of proper feature weighting and massive feature drops.

Lastly, we understand the diversity and sparsity of datasets due to problems mentioned in [3.1]. Therefore we would also tap into data augmentation with the usage of conventional methods like random affine, elastic and pixel wise transformations on our currently available datasets, as well as utilizing deep GANs in order to create new and unseen data as showcased in [12].

The End.

8 Appendix – Scripts

8.1 Rig information

```
#!/usr/bin/env python

print(f'Python version: {platform.python_version()}')


sys_details = tf.sysconfig.get_build_info()
print(f'CUDA version: {sys_details["cuda_version"]}')
print(f'CUDA computational capabilities:
{sys_details["cuda_compute_capabilities"]}')
print(f'CUDNN version: {sys_details["cudnn_version"]}')


tools = ['pandas','numpy','keras','matplotlib','sklearn','radiomics',
         'nibabel','SimpleITK','imblearn','xgboost','tensorflow']
packages = {}

for i in range(len(tools)):
    try:
        packages[tools[i]] = import_module(tools[i])
        temp = packages[tools[i]].__version__
        if i < 3:
            print(f'Package: {tools[i]}\t\t is in version:\t {temp}')
        else:
            print(f'Package: {tools[i]}\t is in version:\t {temp}')
    except (PackageNotFound, NameError):
        print(f'Package: {tools[i]}\t is in version:\t {version(tools[i])}')
```

8.2 Image Preprocess Pipeline

```
def grab_NSD(img_data, depth_len):
    """
    Simple function to grab sigma estimates from MRI slices with NaN intercept
    img_data : i x j x z image data matrix
    depth_len : slice count (Z)
    """
    temp_list_a = []

    for i in range(depth_len):
        X = np.mean(estimate_sigma(img_data[:, :, i]))
        if np.isnan(X):
            temp_list_a.append(1)
        else:
            temp_list_a.append(X)

    return np.float64(temp_list_a)

def denoise_img_data(temp_list_A, temp_list_B):
    """
    Clutter control function to generate NLM denoised images
    temp_list_A: a 3D MRI image
    temp_list_B: the estimated sigma values for each Z depth slice of temp_list_A
    """
    a = []

    options = dict(fast_mode=True,  # true for non gaussian
                   patch_size=5,  # 5x5 patches
                   patch_distance=6,  # 13x13 search area
                   multichannel=False)

    for i in range(len(temp_list_B)):
        a.append(denoise_nl_means(temp_list_A[:, :, i], h=1.15 * temp_list_B[i],
**options))

    return a

def denoise_process(i):
    """
    Denoise function to assist with
    Parallel error intercept due to i/o stream going ballistic
    i: counter for delayed
    """
    stream = getattr(sys, "stdout")
    f = denoise_img_data((temp_list_A[i]), temp_list_B[i])
    stream.flush()

    return f

def bias_field_correction(i):
    """
    Bias field correction function to assist with
    Parallel error intercept due to i/o stream going ballistic
    i: counter for delayed
    """
    stream = getattr(sys, "stdout")

    f = []

    for j in range(Z_depth):
        # cast image to Real ITK format
        obj_f = sitk.GetImageFromArray(temp_list_C[i][j])
        obj_f = sitk.Cast(obj_f, sitk.sitkFloat32)
        # Cast mask to uint8 format
        mask_image = sitk.OtsuMultipleThresholds(obj_f, 0, 1, 200)
        mask_image = sitk.Cast(mask_image, sitk.sitkUInt8)
        # note that both casts are done as a dependency to the correction execute seq , it's
an inconvenience
        # but my time management is BAD so this will have to do for now , might change
it later

        bias_corrected_img = corrector.Execute(obj_f, mask_image)

        # return the image slice to original 240x240 dimensions and drop it on the
Z_depth stack
        f.append(sitk.GetArrayFromImage(bias_corrected_img))

    stream.flush()

    return f

def data_nesting(x):
    """
    data saving function to assist with
    Parallel error intercept due to i/o stream going ballistic
    i: counter for delayed
    """
    stream = getattr(sys, "stdout")

    hdr = object_nifti[x].header
    aff = object_nifti[x].affine

    finalized_nifti_img = nib.Nifti1Image(np.transpose(np.array(temp_list_D[x]),
axes=(1, 2, 0)), aff, hdr)
    nib.save(finalized_nifti_img, finalized_locs.iloc[i][x + 1])

    stream.flush()
```

```
def create_dir_tree_without_files(src, dst):
    # src https://www.geeksforgeeks.org/python-copy-directory-structure-without-files/

    # getting the absolute path of the source
    # directory
    src = os.path.abspath(src)

    # making a variable having the index till which
    # src string has directory and a path separator
    src_prefix = len(src) + len(os.path.sep)

    # making the destination directory
    os.makedirs(dst)

    # doing os walk in source directory
    for root, dirs, files in os.walk(src):
        for dirname in dirs:
            # here dst has destination directory,
            # root[src_prefix:] gives us relative
            # path from source directory
            # and dirname has folder names
            dirpath = os.path.join(dst, root[src_prefix:], dirname)

            # making the path which we made by
            # joining all of the above three
            os.mkdir(dirpath)

def load_pack(index, slice_index):
    """
    simple data grabber

    index: number that indicates which image set will be grabbed from the location
dataframes
```

```
    slice_index: number that indicates which Z-depth slice is gonna get grabbed
    """
    # mask
    mask = nib.load(finalized_locs.iloc[index][0]).get_fdata()[:, :, slice_index]

    # a raw data sample
    raw_img = [(nib.load(raw_data_loc.iloc[index][x]).get_fdata())[:, :, slice_index] for
x in range(1, 5)]

    # a preprocessed data sample
    prep_img = [(nib.load(finalized_locs.iloc[index][x]).get_fdata())[:, :, slice_index]
for x in range(1, 5)]

    return mask, raw_img, prep_img



def plot_pack(mask, raw_img, prep_img):
    """
    simple data plot

    mask: 240x240 segmentation mask
    raw_img : list[0-4] of 240x240 images
    prep_img : same as above
    """
    names = ["flair", "t1", "t1c", "t2"]

    plt.figure(figsize=(17, 17))
    for i in range(4):
        plt.subplot(1, 4, i + 1)
        plt.title("Original " + names[i])
        plt.imshow(raw_img[i], cmap='gray')
        plt.imshow(mask, cmap='jet', alpha=.33)
```

```
  plt.figure(figsize=(17, 17))
  for i in range(4):
    plt.subplot(1, 4, i + 1)
    plt.title("Processed " + names[i])
    plt.imshow(prep_img[i], cmap='gray')
    plt.imshow(mask, cmap='jet', alpha=.33)
    plt.show()

## ~~ *** ~~~ *** ~~~ *** ~~~ ##

# Initial data loc grab of the BraTS datasets
files = glob('X:\Datasets\BraTS\DATA\DATA_Training\**\*.nii.gz', recursive=True)
train_files_masks = glob('X:\Datasets\BraTS\DATA\DATA_Training\**\*seg.nii.gz',
recursive=True)

train_files_scans = [fn for fn in (filter(lambda x: not x.__contains__("seg"), files))]
print(f'Found masks :{len(train_files_masks)} and scans:{len(train_files_scans)}.')

# separating scan pairs and merging data locations
flair = []
t1ce = []
t1 = []
t2 = []
for x in train_files_scans:
  if "t1ce.nii.gz" in x:
    t1ce.append(x)
  elif "t1.nii.gz" in x:
    t1.append(x)
  elif "t2" in x:
    t2.append(x)
  elif "flair.nii.gz" in x:
    flair.append(x)
  else:
```

```
        print("Something funny happened.")
        break

print(f'Accumulated -> Flair:{len(flair)}, T1:{len(t1)}, T1c:{len(t1ce)}, T2:{len(t2)}\n')

temp_a = list(zip(train_files_masks, flair, t1, t1ce, t2))
temp_b = ["mask", "flair", "t1", "t1c", "t2"]

raw_data_loc = pd.DataFrame(temp_a, columns=temp_b)
null = [print(raw_data_loc.iloc[0][i]) for i in range(5)]
#raw_data_loc.head()

# Generate Mirror Locations for post process storing

# Uncomment if you need to recreate directory
create_dir_tree_without_files('D:\Datasets\BraTS\DATA\DATA_Training',
                              'D:\Datasets\BraTS\DATA\Processed_DATA_Training')

finalized_locs = raw_data_loc.copy()

for i in range(finalized_locs.shape[0]):
    for j in range(finalized_locs.shape[1]):
        temp = raw_data_loc.iloc[i][j]
        temp = temp[:23] + "Processed_" + temp[23:]
        finalized_locs.iloc[i][j] = temp

# Uncomment if you need to transfer the masks , if they are there already this should
return an error
for i in range(finalized_locs.shape[0]):
    shutil.copyfile(raw_data_loc.iloc[i][0], finalized_locs.iloc[i][0])

# Preprocess Pipeline
```

```
temp_time_start = time()
warnings.filterwarnings("ignore")
Z_depth = 155

for i in range(len(raw_data_loc)):
    object_nifti = []  # keep these for header & affine affix copy
    temp_list_A = []  # line tuple from raw_data
    temp_list_B = []  # estimated sigmas for tuple data

    # grab data tuple
    for j in range(4):
        object_nifti.append(nib.load(raw_data_loc.iloc[i][j + 1]))  # a nifti image object
        temp_list_A.append(object_nifti[j].get_fdata())  # grab image_data #dim: 
240x240x155
        temp_list_B.append(grab_NSD(temp_list_A[j], Z_depth))  # grab estimated 
sigmas , dim: 155x

    temp_list_C = Parallel(n_jobs=4, backend="threading")(delayed(denoise_process)
(x) for x in range(4))

    corrector = sitk.N4BiasFieldCorrectionImageFilter()  # generate global filter
    temp_list_D = Parallel(n_jobs=4, backend="threading")
(delayed(bias_field_correction)(x) for x in range(4))

    x = Parallel(n_jobs=4, backend="threading")(delayed(data_nesting)(x) for x in 
range(4))

temp_time_stop = time()
print(f'Time elasped for preprocessing: {(temp_time_stop - temp_time_start)} sec.')

pack_index = np.random.randint(low=0, high=len(raw_data_loc))
slice_index = np.random.randint(low=50, high= 100)
```

```
mask, raw_img, prep_img = load_pack(pack_index, slice_index)
plot_pack(mask, raw_img, prep_img)
```

8.3 Data Bundling

```
t1_list = glob('X:/Datasets/BraTS/DATA/Processed_DATA_Training/**/*t1.nii.gz',
recursive=True)
t2_list = glob('X:/Datasets/BraTS/DATA/Processed_DATA_Training/**/*t2.nii.gz',
recursive=True)
t1ce_list = glob('X:/Datasets/BraTS/DATA/Processed_DATA_Training/**/
*t1ce.nii.gz', recursive=True)
flair_list = glob('X:/Datasets/BraTS/DATA/Processed_DATA_Training/**/
*flair.nii.gz', recursive=True)
mask_list = glob('X:/Datasets/BraTS/DATA/Processed_DATA_Training/**/
*seg.nii.gz', recursive=True)

c = 0
scaler = MinMaxScaler()

for i in range(len(t1_list)):
    print(f"\rCurrent: {i}")

    #temp_image_t1 = nib.load(t1_list[i]).get_fdata()
    #temp_image_t1 = scaler.fit_transform(temp_image_t1.reshape(-1,
temp_image_t1.shape[-1])).reshape(temp_image_t1.shape)

    temp_image_t2 = nib.load(t2_list[i]).get_fdata()
    temp_image_t2 = scaler.fit_transform(temp_image_t2.reshape(-1,
temp_image_t2.shape[-1])).reshape(temp_image_t2.shape)

    temp_image_t1ce = nib.load(t1ce_list[i]).get_fdata()
    temp_image_t1ce = scaler.fit_transform(temp_image_t1ce.reshape(-1, temp_im-
age_t1ce.shape[-1])).reshape(temp_image_t1ce.shape)

    temp_image_flair = nib.load(flair_list[i]).get_fdata()
    temp_image_flair = scaler.fit_transform(temp_image_flair.reshape(-1, temp_im-
age_flair.shape[-1])).reshape(temp_image_flair.shape)

    temp_mask = nib.load(mask_list[i]).get_fdata()
    temp_mask = temp_mask.astype(np.uint8)
    # 3 has no representation in the entire dataset so we replace it with 4
    temp_mask[temp_mask == 4] = 3

    #add temp_image_t1 in the stack if you want to save it too
    temp_combined_images = np.stack([temp_image_flair, temp_image_t1ce,
temp_image_t2], axis=3)

    # cropping down to 128x128x128 patches
    temp_combined_images = temp_combined_images[56:184, 56:184, 13:141]
    temp_mask = temp_mask[56:184, 56:184, 13:141]

    val, counts = np.unique(temp_mask, return_counts=True)
    # if the useful information on the picture is less than 1%, drop the image
    if (1 - (counts[0] / counts.sum())) > 0.01:
        temp_mask = to_categorical(temp_mask, num_classes=4)
        np.save('X:/Data/3D_Blocks/images/stack_' + str(i) + '.npy', temp_combined_im-
ages)
        np.save('X:/Data/3D_Blocks/classes/mask_' + str(i) + '.npy', temp_mask)
    else:
        c += 1

print(f'Out of {len(t1_list)} 3D stacks, {c} didn't have enough information')
```

8.4 Multi-omics

```
#!/usr/bin/env python

    # initial data grab
    # CLINICAL DATA GBM IS RIDDEN WITH DELIMITER ERRORS ON THESE
LINES*
    #*:we get rid of them but noting them nontheless in case we can repair / yield some
information from them
    # x = [66, 110, 111, 117, 119,
    #     120, 126, 128, 138, 145,
    #     163, 165, 167, 227, 277,
    #     300, 304, 306, 349, 373,
    #     431, 468, 485, 499, 585]

    x_loc = 'X:/Data/Extras/Genomics/'  # >SE , g_loc , L:D
    #x_loc = 'D:/thesis_movable/Genomics/' # >SE , g_loc , L:L

    #currently indexing per sample name
    exp_data = pd.read_csv(x_loc+'exp', index_col=0)
    mirna_data = pd.read_csv(x_loc+'mirna', index_col=0)
    survival_data = pd.read_csv(x_loc+'survival', index_col=0)
    clinical_data = pd.read_csv(x_loc + 'clinical_gbm',
                        index_col=0,
                        delimiter='\t',
                        on_bad_lines='skip')

    # Taking a look to retain sanity points
    print(f'Exp data size: {exp_data.T.shape}.\n--')
    print(exp_data.iloc[0:5,0:5].T)
    print('\n\n***\n\n')

    print(f'miRNA data size: {mirna_data.T.shape}.\n--')
    print(mirna_data.iloc[0:3,0:5].T)
    print('\n\n***\n\n')

    print(f'Survival data size {survival_data.shape}.\n--')
    print(survival_data.head())
    print('\n\n***\n\n')

    print(f'Clinical data size {clinical_data.shape}.\n--')
    pd.set_option('display.max_rows', len(clinical_data.iloc[0]))
    print(clinical_data.iloc[0].T)
    pd.reset_option('display.max_rows')
    print('\n\n***\n\n')

    # define classes and grab usable datasets
    c1 = clinical_data.query('CDE_vital_status == "DECEASED"')
    c2 = clinical_data.query('CDE_vital_status == "LIVING" & days_to_last_followup
> 100')

    print(f'Classes:\t\t[Dead: {len(c1)}]  [Alive & DTLF>100: {len(c2)}]  [Total
Samples: {len(c1)+len(c2)}]')

    # obtain subject tags
    c1_tags = c1.index.str.replace("-",".").to_list()
    c2_tags = c2.index.str.replace("-",".").to_list()
    exp_tags = exp_data.T.index.to_list()
    mirna_tags = mirna_data.T.index.to_list()

    #check if they have exp & mirna data and purge if they dont
    for x in c1_tags:
       if x in exp_tags and x in mirna_tags:
          continue
       else:
          c1_tags.remove(x)
```

```
for x in c2_tags:
    if x in exp_tags and x in mirna_tags:
        continue
    else:
        c2_tags.remove(x)

class_dead_exp_data = exp_data.T.query(f'index in {c1_tags}')
class_dead_mirna_data = mirna_data.T.query(f'index in {c1_tags}')

class_alive_exp_data = exp_data.T.query(f'index in {c2_tags}')
class_alive_mirna_data = mirna_data.T.query(f'index in {c2_tags}')

print(f'Classes after pruning:  [Dead: {len(c1_tags)}]  [Alive & DTLF>100:
{len(c2_tags)}]  [Total Samples: {len(c1_tags)+len(c2_tags)}]\n')

plt_data = {'Total Samples': len(c1_tags)+len(c2_tags), 'Dead': len(c1_tags), 'Alive
& SD:>100':len(c2_tags)}
classes = list(plt_data.keys())
values = list(plt_data.values())
colors = ['blue','brown','green']
plt.figure(figsize=(4,4))

plt.bar(classes, values, width=.5, color=colors)
plt.title('Class distribution')
plt.xlabel("Classes")
plt.ylabel('Samples')
plt.show()

print('Majority class: %.2f' % (abs(1-(len(c2_tags) / len(c1_tags)))))
print('Minority class: %.2f' % (len(c2_tags) / len(c1_tags) ))

# true sets
```

```
    u_X_exp = pd.concat([class_dead_exp_data, class_alive_exp_data],
verify_integrity=True)
    u_Y_exp = np.zeros(len(u_X_exp), dtype=np.uint8)
    u_Y_exp[len(class_dead_exp_data):] = 1

    u_X_mirna = pd.concat([class_dead_mirna_data, class_alive_mirna_data],
verify_integrity=True)
    u_Y_mirna = np.zeros(len(u_X_mirna), dtype=np.uint8)
    u_Y_mirna[len(class_dead_mirna_data):] = 1

    print(f"\nTrue sets:\n\tEXP:  {u_X_exp.shape}\n\tMIRNA:{u_X_mirna.shape}")

    # three way oversampling of C2
    rt = 42

    s1 = sm(random_state=rt)
    s2 = bsm(random_state=rt)
    s3 = ada(random_state=rt)

    # SMOTE
    V1_X_exp_res, V1_Y_exp_res = s1.fit_resample(u_X_exp, u_Y_exp)
    V1_X_mirna_res, V1_Y_mirna_res = s1.fit_resample(u_X_mirna, u_Y_mirna)

    # BorderlineSMOTE
    V2_X_exp_res, V2_Y_exp_res = s2.fit_resample(u_X_exp, u_Y_exp)
    V2_X_mirna_res, V2_Y_mirna_res = s2.fit_resample(u_X_mirna, u_Y_mirna)

    # ADASYN
    V3_X_exp_res, V3_Y_exp_res = s3.fit_resample(u_X_exp, u_Y_exp)
    V3_X_mirna_res, V3_Y_mirna_res = s3.fit_resample(u_X_mirna, u_Y_mirna)

    # reploting
    plt_data = {'Total Samples': len(V1_Y_exp_res)+len(V1_Y_mirna_res),
            'Dead': len(V1_Y_exp_res),
            'Alive & SD:>100':len(V1_Y_mirna_res)}
    classes = list(plt_data.keys())
    values = list(plt_data.values())
    colors = ['blue','brown','green']
    plt.figure(figsize=(4,4))

    plt.bar(classes, values, width=.5, color=colors)
    plt.title('Class distribution')
    plt.xlabel("Classes")
    plt.ylabel('Samples')
    plt.show()

    # Chop suey
    rt = 7

    exp_X_trains = []
    exp_Y_trains = []
    exp_X_tests = []
    exp_Y_tests = []

    mirna_X_trains = []
    mirna_Y_trains = []
    mirna_X_tests = []
    mirna_Y_tests = []

    V1expXtrain, V1expXtest, V1expYtrain, V1expYtest =
train_test_split(V1_X_exp_res, V1_Y_exp_res, test_size=.2, random_state=rt)
    V1mirnaXtrain, V1mirnaXtest, V1mirnaYtrain, V1mirnaYtest =
train_test_split(V1_X_mirna_res, V1_Y_mirna_res, test_size=.2, random_state=rt)
    V2expXtrain, V2expXtest, V2expYtrain, V2expYtest =
train_test_split(V2_X_exp_res, V2_Y_exp_res, test_size=.2, random_state=rt)
```

```
    V2mirnaXtrain, V2mirnaXtest, V2mirnaYtrain, V2mirnaYtest =
train_test_split(V2_X_mirna_res, V2_Y_mirna_res, test_size=.2, random_state=rt)
    V3expXtrain, V3expXtest, V3expYtrain, V3expYtest =
train_test_split(V3_X_exp_res, V3_Y_exp_res, test_size=.2, random_state=rt)
    V3mirnaXtrain, V3mirnaXtest, V3mirnaYtrain, V3mirnaYtest =
train_test_split(V3_X_mirna_res, V3_Y_mirna_res, test_size=.2, random_state=rt)


    for i in range(3):
        affix = 'V' + str(i+1)

        exp_X_trains.append(eval(affix + 'expXtrain'))
        exp_Y_trains.append(eval(affix + 'expYtrain'))
        exp_X_tests.append(eval(affix + 'expXtest'))
        exp_Y_tests.append(eval(affix + 'expYtest'))

        mirna_X_trains.append(eval(affix + 'mirnaXtrain'))
        mirna_Y_trains.append(eval(affix + 'mirnaYtrain'))
        mirna_X_tests.append(eval(affix + 'mirnaXtest'))
        mirna_Y_tests.append(eval(affix + 'mirnaYtest'))

    # init classfiers ( W L ) 4x3 = 12 classfiers
    xgboost_exp_cls = []
    xgboost_mirna_cls = []

    dtc_exp_cls = []
    dtc_mirna_cls = []

    for i in range(3):
        xgboost_exp_cls.append(xgb.XGBClassifier())
        dtc_exp_cls.append(dtc())

        xgboost_mirna_cls.append(xgb.XGBClassifier())
        dtc_mirna_cls.append(dtc())

    # training classfiers
    start_t = time()

    for i in range(3):
        xgboost_exp_cls[i].fit(exp_X_trains[i], exp_Y_trains[i])
        dtc_exp_cls[i].fit(exp_X_trains[i], exp_Y_trains[i])

        xgboost_mirna_cls[i].fit(mirna_X_trains[i], mirna_Y_trains[i])
        dtc_mirna_cls[i].fit(mirna_X_trains[i], mirna_Y_trains[i])

    print(f'Training all classifiers took {int(time()-start_t)}sec.')

    # results
    c_type = ['SMOTE', 'BorderlineSMOTE', 'ADASYN']

    for i in range(3):
        print(f'Printing accuracy results for imbalance correction method: {c_type[i]}.\
n-')

        # predictions
        a_pred = xgboost_exp_cls[i].predict(exp_X_tests[i])
        b_pred = dtc_exp_cls[i].predict(exp_X_tests[i])
        c_pred = xgboost_mirna_cls[i].predict(mirna_X_tests[i])
        d_pred = dtc_mirna_cls[i].predict(mirna_X_tests[i])

        # accuracy stuff & plots
        exp_x_ax = range(len(exp_Y_tests[i]))
        mirna_x_ax = range(len(mirna_X_tests[i]))

        a_acc = acc(exp_Y_tests[i], a_pred)
        print('Accuracy for XGBOOST on exp: %.2f%%' %(a_acc*100))
```

```
plt.figure(figsize=(10,3))
plt.title(f'XGBoost Results for EXP data (Imb_Strat: {c_type[i]})')
plt.plot(exp_x_ax, exp_Y_tests[i], label='original')
plt.plot(exp_x_ax, a_pred, label='predicted', color='r', alpha=.33)
plt.show()

b_acc = acc(exp_Y_tests[i], b_pred)
print('Accuracy for RFC on exp: %.2f%%' %(b_acc*100))
plt.figure(figsize=(10,3))
plt.title(f'RFC Results for EXP data (Imb_Strat: {c_type[i]})')
plt.plot(exp_x_ax, exp_Y_tests[i], label='original')
plt.plot(exp_x_ax, b_pred, label='predicted', color='r', alpha=.33)
plt.show()

c_acc = acc(mirna_Y_tests[i], c_pred)
print('Accuracy for XGBOOST on mirna: %.2f%%' %(c_acc*100))
plt.figure(figsize=(10,3))
plt.title(f'XGBoost Results for miRNA data (Imb_Strat: {c_type[i]})')
plt.plot(mirna_x_ax, mirna_Y_tests[i], label='original')
plt.plot(mirna_x_ax, c_pred, label='predicted', color='r', alpha=.33)
plt.show()

d_acc = acc(mirna_Y_tests[i], d_pred)
print('Accuracy for RFC on mirna: %.2f%%' %(d_acc*100))
plt.figure(figsize=(10,3))
plt.title(f'RFC Results for miRNA data (Imb_Strat: {c_type[i]})')
plt.plot(mirna_x_ax, mirna_Y_tests[i], label='original')
plt.plot(mirna_x_ax, d_pred, label='predicted', color='r', alpha=.33)
plt.show()

# post
print ('\n**')
```

```
    # save model cause it's cool to keep nice things
    #f_loc = 'D:/thesis_movable/main/saved_models/'  # global , S_E
    f_loc = 'C:/Users/delta/my_thesis/main/saved_models/'   # global , S_E

    prefix = 'non_normal_'

    model_name = 'GEN_model_XGBClassifier_' +
datetime.datetime.now().strftime("%Y%m%d-%H%M%S")

    suffix = '.json'

    f_name =  f_loc + prefix  + model_name + suffix

    xgboost_exp_cls[2].save_model(f_name)

    # full dataset normalization attempt
    scaler1 = MinMaxScaler()
    scaler2 = MinMaxScaler()

    scaler1.fit(u_X_exp)
    scaler2.fit(u_X_mirna)

    u_X_exp = scaler1.transform(u_X_exp)
    u_X_mirna = scaler2.transform(u_X_mirna)

    # three way oversampling of C2
    rt = 42

    s1 = sm(random_state=rt)
    s2 = bsm(random_state=rt)
    s3 = ada(random_state=rt)

    # SMOTE
    f1_X_exp_res, f1_Y_exp_res = s1.fit_resample(u_X_exp, u_Y_exp)
    f1_X_mirna_res, f1_Y_mirna_res = s1.fit_resample(u_X_mirna, u_Y_mirna)

    # BorderlineSMOTE
    f2_X_exp_res, f2_Y_exp_res = s2.fit_resample(u_X_exp, u_Y_exp)
    f2_X_mirna_res, f2_Y_mirna_res = s2.fit_resample(u_X_mirna, u_Y_mirna)

    # ADASYN
    f3_X_exp_res, f3_Y_exp_res = s3.fit_resample(u_X_exp, u_Y_exp)
    f3_X_mirna_res, f3_Y_mirna_res = s3.fit_resample(u_X_mirna, u_Y_mirna)

    # Chop suey
    rt = 7

    fxp_X_trains = []
    fxp_Y_trains = []
    fxp_X_tests = []
    fxp_Y_tests = []

    firna_X_trains = []
    firna_Y_trains = []
    firna_X_tests = []
    firna_Y_tests = []

    f1fxpXtrain, f1fxpXtest, f1fxpYtrain, f1fxpYtest = train_test_split(f1_X_exp_res,
f1_Y_exp_res, test_size=.2, random_state=rt)
    f1firnaXtrain, f1firnaXtest, f1firnaYtrain, f1firnaYtest =
train_test_split(f1_X_mirna_res, f1_Y_mirna_res, test_size=.2, random_state=rt)
    f2fxpXtrain, f2fxpXtest, f2fxpYtrain, f2fxpYtest = train_test_split(f2_X_exp_res,
f2_Y_exp_res, test_size=.2, random_state=rt)
    f2firnaXtrain, f2firnaXtest, f2firnaYtrain, f2firnaYtest =
train_test_split(V2_X_mirna_res, V2_Y_mirna_res, test_size=.2, random_state=rt)
```

```
    f3fxpXtrain, f3fxpXtest, f3fxpYtrain, f3fxpYtest = train_test_split(f3_X_exp_res,
f3_Y_exp_res, test_size=.2, random_state=rt)
    f3firnaXtrain, f3firnaXtest, f3firnaYtrain, f3firnaYtest =
train_test_split(f3_X_mirna_res, f3_Y_mirna_res, test_size=.2, random_state=rt)


    for i in range(3):
        affix = 'f' + str(i+1)

        fxp_X_trains.append(eval(affix + 'fxpXtrain'))
        fxp_Y_trains.append(eval(affix + 'fxpYtrain'))
        fxp_X_tests.append(eval(affix + 'fxpXtest'))
        fxp_Y_tests.append(eval(affix + 'fxpYtest'))

        firna_X_trains.append(eval(affix + 'firnaXtrain'))
        firna_Y_trains.append(eval(affix + 'firnaYtrain'))
        firna_X_tests.append(eval(affix + 'firnaXtest'))
        firna_Y_tests.append(eval(affix + 'firnaYtest'))

    # init classfiers ( W L ) 4x3 = 12 classfiers
    fgboost_exp_cls = []
    fgboost_mirna_cls = []

    ftc_exp_cls = []
    ftc_mirna_cls = []

    for i in range(3):
        fgboost_exp_cls.append(xgb.XGBClassifier())
        ftc_exp_cls.append(dtc())

        fgboost_mirna_cls.append(xgb.XGBClassifier())
        ftc_mirna_cls.append(dtc())
```

```
# training classfiers
start_t = time()

for i in range(3):
    fgboost_exp_cls[i].fit(fxp_X_trains[i], fxp_Y_trains[i])
    ftc_exp_cls[i].fit(fxp_X_trains[i], fxp_Y_trains[i])

    fgboost_mirna_cls[i].fit(firna_X_trains[i], firna_Y_trains[i])
    ftc_mirna_cls[i].fit(firna_X_trains[i], firna_Y_trains[i])

print(f'Training all classifiers took {int(time()-start_t)}sec.')

# results
c_type = ['SMOTE', 'BorderlineSMOTE', 'ADASYN']

for i in range(3):
    print(f'Printing accuracy results for imbalance correction method: {c_type[i]}.\
n-')

    # predictions
    fa_pred = fgboost_exp_cls[i].predict(fxp_X_tests[i])
    fb_pred = ftc_exp_cls[i].predict(fxp_X_tests[i])
    fc_pred = fgboost_mirna_cls[i].predict(firna_X_tests[i])
    fd_pred = ftc_mirna_cls[i].predict(firna_X_tests[i])

    # accuracy stuff & plots
    fxp_x_ax = range(len(fxp_Y_tests[i]))
    firna_x_ax = range(len(firna_X_tests[i]))

    fa_acc = acc(fxp_Y_tests[i], fa_pred)
    print('Accuracy for XGBOOST on exp: %.2f%%' %(fa_acc*100))
    plt.figure(figsize=(10,3))
    plt.title(f'XGBoost Results for EXP data (Imb_Strat: {c_type[i]})')
    plt.plot(fxp_x_ax, fxp_Y_tests[i], label='original')
    plt.plot(fxp_x_ax, fa_pred, label='predicted', color='r', alpha=.33)
    plt.show()

    fb_acc = acc(fxp_Y_tests[i], fb_pred)
    print('Accuracy for RFC on exp: %.2f%%' %(fb_acc*100))
    plt.figure(figsize=(10,3))
    plt.title(f'RFC Results for EXP data (Imb_Strat: {c_type[i]})')
    plt.plot(fxp_x_ax, fxp_Y_tests[i], label='original')
    plt.plot(fxp_x_ax, fb_pred, label='predicted', color='r', alpha=.33)
    plt.show()

    fc_acc = acc(firna_Y_tests[i], fc_pred)
    print('Accuracy for XGBOOST on mirna: %.2f%%' %(fc_acc*100))
    plt.figure(figsize=(10,3))
    plt.title(f'XGBoost Results for miRNA data (Imb_Strat: {c_type[i]})')
    plt.plot(firna_x_ax, firna_Y_tests[i], label='original')
    plt.plot(firna_x_ax, fc_pred, label='predicted', color='r', alpha=.33)
    plt.show()

    fd_acc = acc(firna_Y_tests[i], fd_pred)
    print('Accuracy for RFC on mirna: %.2f%%' %(fd_acc*100))
    plt.figure(figsize=(10,3))
    plt.title(f'RFC Results for miRNA data (Imb_Strat: {c_type[i]})')
    plt.plot(firna_x_ax, firna_Y_tests[i], label='original')
    plt.plot(firna_x_ax, fd_pred, label='predicted', color='r', alpha=.33)
    plt.show()

    # post
    print ('\n**')

# save model cause it's cool to keep nice things
#f_loc = 'D:/thesis_movable/main/saved_models/'  # global , S_E
```

```
    f_loc = 'C:/Users/delta/my_thesis/main/saved_models/'   # global , S_E

    model_name = 'GEN_model_XGBClassifier_' + 
datetime.datetime.now().strftime("%Y%m%d-%H%M%S")

    prefix = 'normal_'

    suffix = '.json'

    f_name = f_loc + prefix + model_name + suffix

    fgboost_exp_cls[2].save_model(f_name)

    # merging dataframes
    u_X_merged = pd.merge(u_X_exp, u_X_mirna, left_index=True, 
right_index=True, how='outer')

    #eliminate non intersection participants
    u_X_merged.dropna(inplace=True)

    print(f'Merged set shape: {u_X_merged.shape}.')

    # grab survival days
    survival_days = [survival_data.loc[x[:-3]][0] for x in 
u_X_merged.index.str.lower()]

    # inject it in the end of the merged dataset
    u_X_merged['survival'] = np.array(survival_days)

    # ploting survival
    x_ax = range(len(u_X_merged))
    plt.figure(figsize=(15,5))
    plt.title('Survival Value Distribution')
    plt.plot(x_ax, u_X_merged['survival'])
    plt.show()

    # getting rid of some outliers
    u_X_merged.drop(u_X_merged['survival'].nlargest(25).index, inplace=True)
    # 24 is the 5% of the dataset, aka getting rid of the top 5% of the dataset

    # normalize dataset
    u_X_merged = (u_X_merged-u_X_merged.min())/(u_X_merged.max()-
u_X_merged.min())

    # eliminate errors
    u_X_merged.replace([np.inf, -np.inf], np.nan, inplace=True)
    u_X_merged.dropna(inplace=True)

    # ploting pruned survival
    x_ax = range(len(u_X_merged))
    plt.figure(figsize=(15,5))
    plt.title('Pruned & Normalized Survival Value Distribution')
    plt.plot(x_ax, u_X_merged['survival'])
    plt.show()

    print(f'Finalized merged set shape: {u_X_merged.shape}.')
    u_X_merged.iloc[:10,-8:] # S_C
    # to address data leakage perform line 10 after you've split the dataset & normalize
them INDIVIDUALY , S_E - R_5 ?!

    # splits
    rt=42
    mX_train, mX_test, mY_train, mY_test = train_test_split(u_X_merged.iloc[:,:-1],
u_X_merged.iloc[:,-1], test_size=.2, random_state=rt)
    print(f'Sets:\n\t Train:{len(mX_train)}, Test:{len(mX_test)}.')
    # model init
    reg_model = xgb.XGBRegressor()
    a_time = time()

    # fit it
    reg_model.fit(mX_train, mY_train)
    print(f'XGB regressor took {time()-a_time} to train.')

    # measure it
    score = reg_model.score(mX_train, mY_train)
    print(f"Training score: {score}")
    score2 = reg_model.score(mX_test, mY_test)
    print(f"Testing score: {score2}")

    b_time = time()

    kfold = KFold(n_splits=10, shuffle=True)
    kf_cv_scores = cross_val_score(reg_model, mX_train, mY_train, cv=kfold )
    print(f"K-fold cross validation took {format(time()-b_time,'.2f')}s with a score of :
{format(kf_cv_scores.mean(),'.2f')}")

    # predictions
    mY_pred = reg_model.predict(mX_test)
    mse_score = mse(mY_test, mY_pred)

    print("MSE:  %.2f" % mse_score)
    print("RMSE: %.2f" % (mse_score**(1/2.0)))

    # plots
    x_ax = range(len(mY_test))
    plt.figure(figsize=(15,5))
    plt.title('Ground Truth & Predictions')
    plt.plot(x_ax, mY_test, label="original")
    plt.plot(x_ax, mY_pred, label="predicted")
    plt.legend()
    plt.show()

    # save model cause it's cool to keep nice things
    #f_loc = 'D:/thesis_movable/main/saved_models/'  # global , S_E
    f_loc = 'C:/Users/delta/my_thesis/main/'   # global , S_E

    model_name = 'GEN_model_XGBRegressor_' +
datetime.datetime.now().strftime("%Y%m%d-%H%M%S")

    suffix = '.json'

    f_name = f_loc + model_name + suffix
    reg_model.save_model(f_name)
```

8.5 Data Generator

```
def load_img(img_dir, img_list):
    images = []
    for i, image_name in enumerate(img_list):
        if (image_name.split('.')[1] == 'npy'):
            image = np.load(img_dir + image_name)

            images.append(image)
    images = np.array(images)

    return (images)

def imageLoader(img_dir, img_list, mask_dir, mask_list, batch_size=1):
    L = len(img_list)

    while True:

        batch_start = 0
        batch_end = batch_size

        while batch_start < L:
            limit = min(batch_end, L)

            X = load_img(img_dir, img_list[batch_start:limit])
            Y = load_img(mask_dir, mask_list[batch_start:limit])

            yield (X, Y)

            batch_start += batch_size
            batch_end += batch_size
```

8.6 Unet

```
kernel_initializer = 'he_uniform'

def simple_unet_model(IMG_HEIGHT, IMG_WIDTH, IMG_DEPTH,
IMG_CHANNELS, num_classes):
    # Build the model
    inputs = Input((IMG_HEIGHT, IMG_WIDTH, IMG_DEPTH,
IMG_CHANNELS))
    s = inputs

    # Contraction path
    c1 = Conv3D(16, (3, 3, 3), activation='relu', kernel_initializer=kernel_initializer,
padding='same')(s)
    c1 = Dropout(0.1)(c1)
    c1 = Conv3D(16, (3, 3, 3), activation='relu', kernel_initializer=kernel_initializer,
padding='same')(c1)
    p1 = MaxPooling3D((2, 2, 2))(c1)

    c2 = Conv3D(32, (3, 3, 3), activation='relu', kernel_initializer=kernel_initializer,
padding='same')(p1)
    c2 = Dropout(0.1)(c2)
    c2 = Conv3D(32, (3, 3, 3), activation='relu', kernel_initializer=kernel_initializer,
padding='same')(c2)
    p2 = MaxPooling3D((2, 2, 2))(c2)

    c3 = Conv3D(64, (3, 3, 3), activation='relu', kernel_initializer=kernel_initializer,
padding='same')(p2)
    c3 = Dropout(0.2)(c3)
    c3 = Conv3D(64, (3, 3, 3), activation='relu', kernel_initializer=kernel_initializer,
padding='same')(c3)
    p3 = MaxPooling3D((2, 2, 2))(c3)
    c4 = Conv3D(128, (3, 3, 3), activation='relu',
kernel_initializer=kernel_initializer, padding='same')(p3)
    c4 = Dropout(0.2)(c4)
    c4 = Conv3D(128, (3, 3, 3), activation='relu',
kernel_initializer=kernel_initializer, padding='same')(c4)
    p4 = MaxPooling3D(pool_size=(2, 2, 2))(c4)

    c5 = Conv3D(256, (3, 3, 3), activation='relu',
kernel_initializer=kernel_initializer, padding='same')(p4)
    c5 = Dropout(0.3)(c5)
    c5 = Conv3D(256, (3, 3, 3), activation='relu',
kernel_initializer=kernel_initializer, padding='same')(c5)

    # Expansive path
    u6 = Conv3DTranspose(128, (2, 2, 2), strides=(2, 2, 2), padding='same')(c5)
    u6 = concatenate([u6, c4])
    c6 = Conv3D(128, (3, 3, 3), activation='relu',
kernel_initializer=kernel_initializer, padding='same')(u6)
    c6 = Dropout(0.2)(c6)
    c6 = Conv3D(128, (3, 3, 3), activation='relu',
kernel_initializer=kernel_initializer, padding='same')(c6)

    u7 = Conv3DTranspose(64, (2, 2, 2), strides=(2, 2, 2), padding='same')(c6)
    u7 = concatenate([u7, c3])
    c7 = Conv3D(64, (3, 3, 3), activation='relu', kernel_initializer=kernel_initializer,
padding='same')(u7)
    c7 = Dropout(0.2)(c7)
    c7 = Conv3D(64, (3, 3, 3), activation='relu', kernel_initializer=kernel_initializer,
padding='same')(c7)

    u8 = Conv3DTranspose(32, (2, 2, 2), strides=(2, 2, 2), padding='same')(c7)
    u8 = concatenate([u8, c2])
    c8 = Conv3D(32, (3, 3, 3), activation='relu', kernel_initializer=kernel_initializer,
padding='same')(u8)
    c8 = Dropout(0.1)(c8)
    c8 = Conv3D(32, (3, 3, 3), activation='relu', kernel_initializer=kernel_initializer,
padding='same')(c8)

    u9 = Conv3DTranspose(16, (2, 2, 2), strides=(2, 2, 2), padding='same')(c8)
    u9 = concatenate([u9, c1])
    c9 = Conv3D(16, (3, 3, 3), activation='relu', kernel_initializer=kernel_initializer,
padding='same')(u9)
    c9 = Dropout(0.1)(c9)
    c9 = Conv3D(16, (3, 3, 3), activation='relu', kernel_initializer=kernel_initializer,
padding='same')(c9)

    outputs = Conv3D(num_classes, (1, 1, 1), activation='softmax')(c9)

    model = Model(inputs=[inputs], outputs=[outputs])
    return model
```

8.7 Train

```
gpus = tf.config.experimental.list_physical_devices('GPU')
tf.config.experimental.set_memory_growth(gpus[0], True)

smooth=100
def dice_coef(y_true, y_pred):
    y_truef = K.flatten(y_true)
    y_predf = K.flatten(y_pred)
    And = K.sum(y_truef* y_predf)
    return ((2* And + smooth) / (K.sum(y_truef) + K.sum(y_predf) + smooth))

#def dice_coef_loss(y_true, y_pred):
#    return -dice_coef(y_true, y_pred)

def iou(y_true, y_pred):
    intersection = K.sum(y_true * y_pred)
    sum_ = K.sum(y_true + y_pred)
    jac = (intersection + smooth) / (sum_ - intersection + smooth)
    return jac

# data locs
train_img_dir = "X:/Data/3D_Blocks/Sets/train/train/"
train_mask_dir = "X:/Data/3D_Blocks/Sets/train/class/"
val_img_dir = "X:/Data/3D_Blocks/Sets/val/train/"
val_mask_dir = "X:/Data/3D_Blocks/Sets/val/class/"

train_img_list = os.listdir(train_img_dir)
train_mask_list = os.listdir(train_mask_dir)
val_img_list = os.listdir(val_img_dir)
val_mask_list = os.listdir(val_mask_dir)

# model params
```

```
    batch_size = 1 #( 1 x [3x3D image & seg_mask]) >> this cannot be higher due to
hardware constraints
    steps_per_epoch = len(train_img_list) // batch_size
    val_steps_per_epoch = len(val_img_list) // batch_size

    model_params = dict(IMG_HEIGHT=128,
                IMG_WIDTH=128,
                IMG_DEPTH=128,
                IMG_CHANNELS=3,
                num_classes=4)

    # opt , adafair was having trouble cause i'd have to downgrade everything for it to
work
    #we're gonna leave that for a later date
    l_r=1e-4
    v_epochs = 25
    decay_rate = l_r/ v_epochs
    optimizer = tf.optimizers.Adam(learning_rate=l_r,
                        decay=decay_rate,
                        amsgrad=False)
    #model number
    model_num = 5

    #callbacks
    model_save =
tf.keras.callbacks.ModelCheckpoint(f'C:/Users/delta/my_thesis/main/saved_models/
model_{str(model_num)}.hdf5', verbose=1,save_best_only=True)

    log_dir = f"logs/logs_{str(model_num)}/" + "fit" +
datetime.datetime.now().strftime("%Y%m%d-%H%M%S")
    tensorboard_callback = tf.keras.callbacks.TensorBoard(log_dir=log_dir,
histogram_freq=1)
```

```
    # data generators
    train_img_datagen = data_gen.imageLoader(train_img_dir, train_img_list,
                                             train_mask_dir, train_mask_list, batch_size)

    val_img_datagen = data_gen.imageLoader(val_img_dir, val_img_list,
                                           val_mask_dir, val_mask_list, batch_size)

    #init casual 3D unet
    model = simple_unet_model(**model_params)
    model.summary()

    model.compile(optimizer=optimizer,loss=tf.keras.losses.CategoricalCrossentropy(),
metrics=['CategoricalAccuracy',iou, dice_coef])



    timer_a = time()

    history = model.fit(train_img_datagen,
                        steps_per_epoch=steps_per_epoch,
                        epochs=v_epochs,
                        validation_data=val_img_datagen,
                        validation_steps=val_steps_per_epoch,
                        callbacks=[model_save, tensorboard_callback])

    print(f'Took {time()-timer_a} to finish all training.')

    #class_weight={0:0.26, 1:22.53, 2:22.53, 3:26.21},
    #save history to load it to the evaluation script // not needed since we use
tensorboard but you could do it nontheless
    np.save(f"saved_models/history_{str(model_num)}.npy", history.history)
```

8.8 Retrain

```
gpus = tf.config.experimental.list_physical_devices('GPU')
tf.config.experimental.set_memory_growth(gpus[0], True)

#functs
smooth=1.

def dice_coef(y_true, y_pred):
    y_truef = K.flatten(y_true)
    y_predf = K.flatten(y_pred)
    And = K.sum(y_truef* y_predf)
    return ((2* And + smooth) / (K.sum(y_truef) + K.sum(y_predf) + smooth))

def iou(y_true, y_pred):
    intersection = K.sum(y_true * y_pred)
    sum_ = K.sum(y_true + y_pred)
    jac = (intersection + smooth) / (sum_ - intersection + smooth)
    return jac

#model load
model =
load_model('C:/Users/delta/my_thesis/main/saved_models/model_3.1.hdf5',
                  custom_objects={'iou': iou, 'dice_coef':dice_coef})

# data locs
train_img_dir = "X:/Data/3D_Blocks/train/train/"
train_mask_dir = "X:/Data/3D_Blocks/train/class/"
val_img_dir = "X:/Data/3D_Blocks/val/train/"
val_mask_dir = "X:/Data/3D_Blocks/val/class/"

train_img_list = os.listdir(train_img_dir)
train_mask_list = os.listdir(train_mask_dir)
val_img_list = os.listdir(val_img_dir)
```

```
    val_mask_list = os.listdir(val_mask_dir)

    # model params
    batch_size = 1 #( 1 x [3x3D image & seg_mask]) >> this cannot be higher due to
hardware constraints
    steps_per_epoch = len(train_img_list) // batch_size
    val_steps_per_epoch = len(val_img_list) // batch_size

    wt0, wt1, wt2, wt3 = 0.26, 22.53, 22.53, 26.21 # taken from contextual analysis of
mask pixels
    dice_loss = sm.losses.DiceLoss(class_weights=np.array([wt0, wt1, wt2, wt3]))
    focal_loss = sm.losses.CategoricalFocalLoss()
    total_loss = dice_loss + (1 * focal_loss)

    l_r= 1e-3
    v_epochs = 25
    decay_rate = l_r/ v_epochs
    optimizer = tf.optimizers.Adam(learning_rate=l_r,
                            decay=decay_rate,
                            amsgrad=False)

    #model number
    model_num = 3.2

    #callbacks
    model_save =
tf.keras.callbacks.ModelCheckpoint(f'C:/Users/delta/my_thesis/main/saved_models/
model_{str(model_num)}.hdf5',
                                        verbose=1,
                                        save_best_only=True)

    log_dir = f"logs/logs_{str(model_num)}/" + "fit" +
datetime.datetime.now().strftime("%Y%m%d-%H%M%S")
```

```
    tensorboard_callback = tf.keras.callbacks.TensorBoard(log_dir=log_dir,
histogram_freq=1)

    # data generators
    train_img_datagen = data_gen.imageLoader(train_img_dir, train_img_list,
                                  train_mask_dir, train_mask_list, batch_size)

    val_img_datagen = data_gen.imageLoader(val_img_dir, val_img_list,
                                val_mask_dir, val_mask_list, batch_size)

    model.summary()
    metrics = ['Accuracy','CategoricalAccuracy', sm.metrics.IOUScore(threshold=0.5),
dice_coef]

    model.compile(optimizer=optimizer, loss=total_loss, metrics=metrics)


    timer_a = time()

    history = model.fit(train_img_datagen,
                    steps_per_epoch=steps_per_epoch,
                    epochs=v_epochs,
                    validation_data=val_img_datagen,
                    validation_steps=val_steps_per_epoch,
                    callbacks=[model_save, tensorboard_callback])

    print(f'Took {time()-timer_a} to finish retraining.')

    #class_weight={0:0.26, 1:22.53, 2:22.53, 3:26.21},
    #save history to load it to the evaluation script // not needed since we use
tensorboard but you could do it nontheless
    np.save(f"saved_models/history_{str(model_num)}.npy", history.history)
```

8.9 Image Feature Extraction

```
#!/usr/bin/env python

  # functions
  def feature_extraction(x):
    col_space = x+1

    for i in range(len(raw_data_loc)):
      # load label mask
      mask = sitk.ReadImage(raw_data_loc.iloc[i][0])
      mask_array = sitk.GetArrayFromImage(mask)

      # uniform mask
      for x in (2,3,4):
        mask_array[mask_array == x] = 1

      # apply original spatial data
      mask_merged = sitk.GetImageFromArray(mask_array)
      mask_merged.CopyInformation(mask)

      # extract featrues
      features = extractor.execute(raw_data_loc.iloc[i][col_space], mask_merged,
label=1)

      # store the data in  their respective list
      if col_space == 1:
        flair.append(features)
      elif col_space == 2:
        t1.append(features)
      elif col_space == 3:
        t1ce.append(features)
      elif col_space == 4:
        t2.append(features)
```

```
# saving loc
os.chdir('C:/Users/delta/my_thesis/main/__outputs/')

# Initial data loc grab of the BraTS Preprocessed datasets
files = glob('X:\Datasets\BraTS\DATA\Processed_DATA_Training\**\*.nii.gz',
recursive=True)
train_files_masks = glob('X:\Datasets\BraTS\DATA\Processed_DATA_Training\**\
*seg.nii.gz', recursive=True)

train_files_scans = [fn for fn in (filter(lambda x: not x.__contains__("seg"),
files))]
print(f'Found masks :{len(train_files_masks)} and scans:{len(train_files_scans)}.')

# separating scan pairs and merging data locations
flair = []
t1ce = []
t1 = []
t2 = []
for x in train_files_scans:
    if "t1ce.nii.gz" in x:
        t1ce.append(x)
    elif "t1.nii.gz" in x:
        t1.append(x)
    elif "t2" in x:
        t2.append(x)
    elif "flair.nii.gz" in x:
        flair.append(x)
    else:
        print("Something funny happened, you should check the sys log.")
        break;
print(f'Accumulated -> Flair:{len(flair)}, T1:{len(t1)}, T1c:{len(t1ce)}, T2:{len(t2)}\
n')

temp_a = list(zip(train_files_masks, flair, t1, t1ce, t2))
temp_b = ["mask", "flair", "t1", "t1c", "t2"]

raw_data_loc = pd.DataFrame(temp_a, columns=temp_b)
[print(raw_data_loc.iloc[0][i]) for i in range(5)]
raw_data_loc.head()

# load the survival datasets and assign targets to the raw_locs
data_origin = []
for x in raw_data_loc['mask']:
    if str(x).__contains__('MICCAI_BraTS2020'):
        data_origin.append(2)
    if str(x).__contains__('MICCAI_BraTS_2019'):
        data_origin.append(1)
    if str(x).__contains__('MICCAI_BraTS_2018'):
        data_origin.append(0)

s_d = glob(r'X:\Datasets\BraTS\DATA\DATA_Training\**\*survival*.csv',
recursive=True)
sd_2018 = pd.read_csv(s_d[0], delimiter=',')
sd_2019 = pd.read_csv(s_d[1], delimiter=',')
sd_2020 = pd.read_csv(s_d[2], delimiter=',')

print(f'{sd_2018.head()}\n--\n{sd_2019.head()}\n--\n{sd_2020.head()}')

# grab patient ids
patients = [str(raw_data_loc.iloc[i][0]).split('\\')[-1].split('_seg')[:-1] for i in
range(len(raw_data_loc))]

# grab survival data for the entire merged set
```

```
t_survival = []
err_count = 0
for count, x in enumerate(data_origin):
    try:
        if x == 0:
            t_survival.append(int(sd_2018[sd_2018['BraTS18ID']==patients[count][0]]
['Survival']))
        elif x == 1:
            t_survival.append(int(sd_2019[sd_2019['BraTS19ID']==patients[count][0]]
['Survival']))
        elif x == 2:
            t_survival.append(int(sd_2020[sd_2020['Brats20ID']==patients[count][0]]
['Survival_days']))

    except (ValueError, TypeError):
        # Error catch for non existing rows with that name
        # Also includes NA values , we force a nan so we can identify the positions
        err_count += 1
        t_survival.append(np.nan)

print(f'Found {err_count} inputs that don\'t exist in the survival datasets or are nan')

# drop locations from the feature extraction
for count, surv in enumerate(t_survival):
    if np.isnan(surv):
        raw_data_loc.drop(count, inplace=True)

# reset df index to establish index flow
raw_data_loc.reset_index(drop=True, inplace=True)

# yield true survival set
survival = [x for x in t_survival if np.isnan(x) == False]
```

```
    # sanity check
    print(f'Packets:{len(raw_data_loc)}, Survivals:{len(survival)}')

    survival_csv = pd.DataFrame(survival, columns=['survival_days'])
    survival_csv.to_csv('survival.csv', index=False)

    #initialize feature lists
    flair = []
    t1 = []
    t1ce = []
    t2 = []

    # initialize a global extractor
    extractor = featureextractor.RadiomicsFeatureExtractor()
    setVerbosity(40) #
https://pyradiomics.readthedocs.io/en/latest/radiomics.html#radiomics.setVerbosity
    extractor.enableAllFeatures() #instead of this we can yield specific features from
    # https://pyradiomics.readthedocs.io/en/latest/features.html, but the more data the
better

    time_a = time()

    null = Parallel(n_jobs=4, backend="threading")(delayed(feature_extraction)(x) for
x in range(4))
    ##    CPU : Intel(R) Core(TM) i7-6700K CPU @ 4.00GHz
    ##    Time for 1 line of the raw_data to be processed : 7.013089895248413 seconds
    ##    Time estimation for all data (608 lines) : 71.06597760518392 minutes

    print(f'Time elasped: {time()-time_a} sec')

    # convert lists to dataframes
    flair_features = pd.DataFrame.from_dict(flair)
    t1_features = pd.DataFrame.from_dict(t1)
    t1ce_features = pd.DataFrame.from_dict(t1ce)
    t2_features = pd.DataFrame.from_dict(t2)

    # saving feature exports per modality 1st csv is the data 2nd csv are the diagnostic
data
    flair_features.to_csv('flair_features.csv',index=False,columns=flair_features.colum
ns[22:])
    flair_features.to_csv('flair_extras.csv',index=False,columns=flair_features.columns
[:22])

    t1_features.to_csv('t1_features.csv',index=False,columns=t1_features.columns[22:]
)
    t1_features.to_csv('t1_extras.csv',index=False,columns=t1_features.columns[:22])

    t1ce_features.to_csv('t1ce_features.csv',index=False,columns=t1ce_features.colum
ns[22:])
    t1ce_features.to_csv('t1ce_extras.csv',index=False,columns=t1ce_features.columns
[:22])

    t2_features.to_csv('t2_features.csv',index=False,columns=t2_features.columns[22:]
)
    t2_features.to_csv('t2_extras.csv',index=False,columns=t2_features.columns[:22])
```

8.10 Image Feature Survival Prediction

```
#!/usr/bin/env python
    # grab data
    flair, t1, t1ce, t2 = glob(os.getcwd()+'/*features.csv', recursive=True)
    survival = glob(os.getcwd()+'/*survival.csv', recursive=True)[0]

    flair_data = pd.read_csv(flair)
    t1_data = pd.read_csv(t1)
    t1ce_data = pd.read_csv(t1ce)
    t2_data = pd.read_csv(t2)
    survival_data = pd.read_csv(survival)

    # append survival days to the datasets
    flair_data['survival_days'] = np.array(survival_data)
    t1_data['survival_days'] = np.array(survival_data)
    t1ce_data['survival_days'] = np.array(survival_data)
    t2_data['survival_days'] = np.array(survival_data)

    # create merged set
    merged_modalities = [flair_data, t1_data, t1ce_data, t2_data]
    merged_data = pd.concat(merged_modalities)

    x_ax = range(len(survival_data))
    plt.figure(figsize=(15,5))
    plt.title('Survival Value Distribution')
    plt.plot(x_ax, survival_data)
    plt.show()

    # split sets
    g_time = time()
    rt=42
    sets = ['flair', 't1', 't1ce', 't2', 'merged']
    # chow swaay splits #+1 for the merged
    X_train = list(range(5))
    X_test = list(range(5))
    Y_train = list(range(5))
    Y_test = list(range(5))

    X_train[0], X_test[0], Y_train[0], Y_test[0] = train_test_split(flair_data.iloc[:,:-1],
flair_data.iloc[:,-1],
                                                    test_size=.2, random_state=rt)
    X_train[1], X_test[1], Y_train[1], Y_test[1] = train_test_split(t1_data.iloc[:,:-1],
t1_data.iloc[:,-1],
                                                    test_size=.2, random_state=rt)
    X_train[2], X_test[2], Y_train[2], Y_test[2] = train_test_split(t1ce_data.iloc[:,:-1],
t1ce_data.iloc[:,-1],
                                                    test_size=.2, random_state=rt)
    X_train[3], X_test[3], Y_train[3], Y_test[3] = train_test_split(t2_data.iloc[:,:-1],
t2_data.iloc[:,-1],
                                                    test_size=.2, random_state=rt)
    """
    for the merged set we have to merge the previous sets
     this happens because if we attempt to merge them and split the data
     we'll cause values in the test set to exist in the train set , voiding the model
    """
    X_train[4] = pd.concat([X_train[0],X_train[1],X_train[2],X_train[3]])
    X_test[4] = pd.concat([X_test[0],X_test[1],X_test[2],X_test[3]])
    Y_train[4] = pd.concat([Y_train[0],Y_train[1],Y_train[2],Y_train[3]])
    Y_test[4] = pd.concat([Y_test[0],Y_test[1],Y_test[2],Y_test[3]])

    #initialize regressors
    regressors = [xgb.XGBRegressor() for x in range(5)]

    timer_a = time()
    #fit regressors for non normalized data
    for i in range(5):
        regressors[i].fit(X_train[i],Y_train[i])

    print(f'XGBRegressors took {format(time()-timer_a, ".2f")}s to train.\n--')

    # measure them
    score = list(range(5))
    for i in range(5):
        score[i] = regressors[i].score(X_test[i], Y_test[i])
    null = [print(f"Training score: {format(score[i]*100,'.2f')}% for set: {sets[i]}") for i
in range(5)]

    timer_b = time()

    KFolds = [KFold(n_splits=10, shuffle=True) for i in range(5)]
    KF_CV_scores = [cross_val_score(regressors[i], X_train[i], Y_train[i],
cv=KFolds[i]) for i in range(5)]
    print(f'\nK-Fold cross validation took {format(time()-timer_b,".2f")}s to estimate.\
n--')
    null = [print(f'K-Fold cross val(n=10) score:
{format(KF_CV_scores[i].mean()*100,".2f")} for set: {sets[i]}') for i in range(5)]

    # calculate predictions
    Y_pred = [regressors[i].predict(X_test[i]) for i in range(5)]
    mse_scores = [mse(Y_test[i],Y_pred[i]) for i in range(5)]
    print('\n--')
    null = [print(f'MSE: {format(mse_scores[i],".2f")} for set: {sets[i]} ') for i in
range(5)]
    print('\n--')
    null = [print(f'RMSE: {format(mse_scores[i]**(1/2.0),".2f")} for set: {sets[i]} ') for
i in range(5)]
```

```
    # plots
    for i in range(5):
        x_ax = range(len(Y_pred[i]))
        plt.figure(figsize=(15,5))
        plt.title(f'Ground Truth & Predictions for set:{sets[i]}')
        plt.plot(x_ax, Y_test[i], label='original')
        plt.plot(x_ax, Y_pred[i], label='predicted', color='r', alpha=.33)
        plt.legend()
        plt.show()

    print(f'Total runtime {format(time()-g_time,".2f")}sec.')

    # normalize all sets
    normalized_flair_data = (flair_data-flair_data.min())/(flair_data.max()-
flair_data.min())
    normalized_t1_data = (t1_data-t1_data.min())/(t1_data.max()-t1_data.min())
    normalized_t1ce_data = (t1ce_data-t1ce_data.min())/(t1ce_data.max()-
t1ce_data.min())
    normalized_t2_data = (t2_data-t2_data.min())/(t2_data.max()-t2_data.min())
    normalized_merged_data = (merged_data-
merged_data.min())/(merged_data.max()-merged_data.min())

    # split sets
    g_time = time()
    rt=42
    sets = ['flair', 't1', 't1ce', 't2', 'merged']

    # chow swaay splits #+1 for the merged
    X_train = list(range(5))
    X_test = list(range(5))
    Y_train = list(range(5))
    Y_test = list(range(5))
    X_train[0], X_test[0], Y_train[0], Y_test[0] =
train_test_split(normalized_flair_data.iloc[:,:-1],
                                                  normalized_flair_data.iloc[:,-1],
                                                  test_size=.2, random_state=rt)
    X_train[1], X_test[1], Y_train[1], Y_test[1] =
train_test_split(normalized_t1_data.iloc[:,:-1],
                                                  normalized_t1_data.iloc[:,-1],
                                                  test_size=.2, random_state=rt)
    X_train[2], X_test[2], Y_train[2], Y_test[2] =
train_test_split(normalized_t1ce_data.iloc[:,:-1],
                                                  normalized_t1ce_data.iloc[:,-1],
                                                  test_size=.2, random_state=rt)
    X_train[3], X_test[3], Y_train[3], Y_test[3] =
train_test_split(normalized_t2_data.iloc[:,:-1],
                                                  normalized_t2_data.iloc[:,-1],
                                                  test_size=.2, random_state=rt)
    """
    for the merged set we have to merge the previous sets
     this happens because if we attempt to merge them and split the data
     we'll cause values in the test set to exist in the train set , voiding the model
    """
    X_train[4] = pd.concat([X_train[0],X_train[1],X_train[2],X_train[3]])
    X_test[4] = pd.concat([X_test[0],X_test[1],X_test[2],X_test[3]])
    Y_train[4] = pd.concat([Y_train[0],Y_train[1],Y_train[2],Y_train[3]])
    Y_test[4] = pd.concat([Y_test[0],Y_test[1],Y_test[2],Y_test[3]])

    #initialize regressors
    regressors = [xgb.XGBRegressor() for x in range(5)]

    timer_a = time()

    #fit regressors for non normalized data
    for i in range(5):
        regressors[i].fit(X_train[i],Y_train[i])

    print(f'XGBRegressors took {format(time()-timer_a, ".2f")}s to train.\n--')

    # measure them
    score = list(range(5))
    for i in range(5):
        score[i] = regressors[i].score(X_test[i], Y_test[i])
    null = [print(f"Training score: {format(score[i]*100,'.2f')}% for set: {sets[i]}") for i
in range(5)]

    timer_b = time()

    KFolds = [KFold(n_splits=10, shuffle=True) for i in range(5)]
    KF_CV_scores = [cross_val_score(regressors[i], X_train[i], Y_train[i],
cv=KFolds[i]) for i in range(5)]
    print(f'\nK-Fold cross validation took {format(time()-timer_b,".2f")}s to estimate.\
n--')
    null = [print(f'K-Fold cross val(n=10) score:
{format(KF_CV_scores[i].mean()*100,".2f")} for set: {sets[i]}') for i in range(5)]

    # calculate predictions
    Y_pred = [regressors[i].predict(X_test[i]) for i in range(5)]
    mse_scores = [mse(Y_test[i],Y_pred[i]) for i in range(5)]
    print('\n--')
    null = [print(f'MSE: {format(mse_scores[i],".2f")} for set: {sets[i]} ') for i in
range(5)]
    print('\n--')
    null = [print(f'RMSE: {format(mse_scores[i]**(1/2.0),".2f")} for set: {sets[i]} ') for
i in range(5)]

    # plots
    for i in range(5):
```

```
    x_ax = range(len(Y_pred[i]))
    plt.figure(figsize=(15,5))
    plt.title(f'Ground Truth & Predictions for normalized set:{sets[i]}')
    plt.plot(x_ax, Y_test[i], label='original')
    plt.plot(x_ax, Y_pred[i], label='predicted', color='r', alpha=.33)
    plt.legend()
    plt.show()
print(f'Total runtime {format(time()-g_time,".2f")}sec.')
```

9 Bibliography

9.1 Online Sources

[1] https://www.cancer.gov/publications/dictionaries/cancer-terms/def/cell

[2] https://en.wikipedia.org/wiki/DNA

[3] https://www.news-medical.net/life-sciences/History-of-Genomics.aspx

[4] https://www.genome.gov/genetics-glossary/RNA-Ribonucleic-Acid

[5] https://en.wikipedia.org/wiki/Oncomir

[6] https://en.wikipedia.org/wiki/MicroRNA

[7] https://www.youtube.com/watch?v=h4t-fhvAorA

[8] https://1drv.ms/p/s!Ah1DKWJIS3ebgZR_YODNnQ8do71UYw?e=pH4fvi

[9] https://flexbooks.ck12.org/cbook/ck-12-middle-school-life-science-2.0/section/3.6/primary/lesson/rna-ms-ls/

[10] https://www.youtube.com/watch?v=6qS83wD29PY

[11] https://www.cancerresearchuk.org/about-cancer/what-is-cancer/body-systems-and-cancer/the-immune-system-and-cancer#fight

[12] https://www.cancer.gov/about-cancer/understanding/what-is-cancer

[13] https://my.clevelandclinic.org/health/diseases/12194-cancer

[14] https://www.cancercenter.com/cancer-types/brain-cancer/grades

[15] https://www.mdanderson.org/cancerwise/glioma-vs--glioblastoma--what-is-the-difference-in-these-brain-tumors-treatment-diagnosis.h00-159537378.html

[16] https://www.hopkinsmedicine.org/health/conditions-and-diseases/brain-tumor#cancer

[17] https://www.mayoclinic.org/diseases-conditions/brain-tumor/symptoms-causes/syc-20350084

[18] https://en.wikipedia.org/wiki/Radiology

[19] https://www.cancer.org/treatment/understanding-your-diagnosis/tests/imaging-radiology-tests-for-cancer.html

[20] https://www.cancer.org/treatment/understanding-your-diagnosis/tests/mri-for-cancer.html

[21] https://www.javatpoint.com/basic-concepts-in-machine-learning

[22] https://www.analyticsvidhya.com/blog/2021/03/everything-you-need-to-know-about-machine-learning/

[23] https://www.computerworld.com/article/2591759/artificial-neural-network-s.html

[24] https://apothesis.lib.hmu.gr/bitstream/handle/20.500.12688/9883/PanagiotakisGeorgios2021.pdf?sequence=1&isAllowed=y

9.2 References

[1] Menze BH, et al. "The Multimodal Brain Tumor Image Segmentation Benchmark (BRATS)", IEEE Transactions on Medical Imaging 34(10), 1993-2024 (2015) DOI: 10.1109/TMI.2014.2377694

[2] Bakas S, et al. "Advancing The Cancer Genome Atlas glioma MRI collections with expert segmentation labels and radiomic features", Nature Scientific Data, 4:170117 (2017) DOI: 10.1038/sdata.2017.117

[3] Bakas S, et al. "Segmentation Labels and Radiomic Features for the Pre-operative Scans of the TCGA-GBM collection", The Cancer Imaging Archive, 2017. DOI: 10.7937/K9/TCIA.2017.KLXWJJ1Q

[4] Bakas S, et al. "Segmentation Labels and Radiomic Features for the Pre-operative Scans of the TCGA-LGG collection", The Cancer Imaging Archive, 2017. DOI: 10.7937/K9/TCIA.2017.GJQ7R0EF

[5] S. Bakas, M. Reyes, A. Jakab, S. Bauer, M. Rempfler, A. Crimi, et al., "Identifying the Best Machine Learning Algorithms for Brain Tumor Segmentation, Progression Assessment, and Overall Survival Prediction in the BRATS Challenge", arXiv preprint arXiv:1811.02629 (2018)

[6] Nimrod Rappoport, Ron Shamir, Multi-omic and multi-view clustering algorithms: review and cancer benchmark, *Nucleic Acids Research*, Volume 46, Issue 20, 16 November 2018, Pages 10546–10562, https://doi.org/10.1093/nar/gky889

[7] Jose V. Manjon, Pierrick Coupé. MRI Denoising Using Deep Learning. International Workshop on Patch-based Techniques in Medical Imaging (MICCAI), Sep 2018, Granada, Spain. pp.12 – 19, 10.1007/978-3-030-00500-9_2. hal-01918437

[8] Sanqian, Li & Zhou, Jinjie & Liang, Dong & Liu, Qiegen. (2020). MRI denoising using progressively distribution-based neural network. Magnetic Resonance Imaging. 71. 10.1016/j.mri.2020.04.006.

[9] Xu, Yan & Hu, Shunbo & Du, Yuyue. (2021). Deep Convolutional Neural Networks for Bias Field Correction of Brain Magnetic Resonance Images. 10.21203/rs.3.rs-853699/v1.

[10] T. Goldfryd, S. Gordon and T. R. Raviv, "Deep Semi-Supervised Bias Field Correction Of Mr Images," 2021 IEEE 18th International Symposium on Biomedical Imaging (ISBI), 2021, pp. 1836-1840, doi: 10.1109/ISBI48211.2021.9433889.

[11] Colman, J., Zhang, L., Duan, W., Ye, X. (2021). DR-Unet104 for Multimodal MRI Brain Tumor Segmentation. In: Crimi, A., Bakas, S. (eds) Brainlesion: Glioma, Multiple Sclerosis, Stroke and Traumatic Brain Injuries. BrainLes 2020. Lecture Notes in Computer Science(), vol 12659. Springer, Cham. https://doi.org/10.1007/978-3-030-72087-2_36

[12] Nalepa J, Marcinkiewicz M and Kawulok M (2019) Data Augmentation for Brain-Tumor Segmentation: A Review. *Front. Comput. Neurosci.* 13:83. doi: 10.3389/ fncom.2019.00083

[13] Vasileios Iosifidis and Eirini Ntoutsi. 2019. AdaFair: Cumulative Fairness Adaptive Boosting. In Proceedings of the 28th ACM International Conference on Information and Knowledge Management (CIKM '19). Association for Computing Machinery, New York, NY, USA, 781–790. https://doi.org/10.1145/3357384.3357974

[14] Smith, Leslie. (2017). Cyclical Learning Rates for Training Neural Networks. 464-472. 10.1109/WACV.2017.58.

[15] *van Griethuysen, J. J. M., Fedorov, A., Parmar, C., Hosny, A., Aucoin, N., Narayan, V., Beets-Tan, R. G. H., Fillon-Robin, J. C., Pieper, S., Aerts, H. J. W. L. (2017). Computational Radiomics System to Decode the Radiographic Phenotype. Cancer Research, 77(21), e104–e107. `https://doi.org/10.1158/0008-5472.CAN-17-0339 <https://doi.org/10.1158/0008-5472.CAN-17-0339>`_*

[16] WATSON, J., CRICK, F. Molecular Structure of Nucleic Acids: A Structure for Deoxyribose Nucleic Acid. *Nature* **171**, 737–738 (1953). https://doi.org/10.1038/171737a0

[17] Collins FS, Fink L. The Human Genome Project. Alcohol Health Res World. 1995;19(3):190-195. PMID: 31798046; PMCID: PMC6875757.

[18] Bartel DP. Metazoan MicroRNAs. Cell. 2018 Mar 22;173(1):20-51. doi: 10.1016/j.-cell.2018.03.006. PMID: 29570994; PMCID: PMC6091663.

[19] Khodadadian A, Darzi S, Haghi-Daredeh S, Sadat Eshaghi F, Babakhanzadeh E, Mirabutalebi SH, Nazari M. Genomics and Transcriptomics: The Powerful Technologies in Precision Medicine. Int J Gen Med. 2020 Sep 17;13:627-640. doi: 10.2147/IJGM.S249970. PMID: 32982380; PMCID: PMC7509479.

[20] *Hammond, SM. (Nov 2006). "RNAi, microRNAs, and human disease". Cancer Chemother Pharmacol. 58 Suppl 1: s63–8. doi:10.1007/s00280-006-0318-2. PMID 17093929. S2CID 682108.*

[21] *"Detect microRNAs most commonly found in Cancer". SBI. Retrieved 14 February 2013.*

[22] *Võsa U, Vooder T, Kolde R, Fischer K, Välk K, Tõnisson N, Roosipuu R, Vilo J, Metspalu A, Annilo T (October 2011). "Identification of miR-374a as a prognostic marker for survival in patients with early-stage nonsmall cell lung cancer". Genes, Chromosomes & Cancer.* ***50*** *(10): 812–22. doi:10.1002/gcc.20902. PMID 21748820. S2CID 9746594.*

[23] *Akçakaya P, Ekelund S, Kolosenko I, Caramuta S, Ozata DM, Xie H, Lindforss U, Olivecrona H, Lui WO (August 2011). "miR-185 and miR-133b deregulation is associated with overall survival and metastasis in colorectal cancer". International Journal of Oncology.* ***39*** *(2): 311–8. doi:10.3892/ijo.2011.1043. PMID 21573504.*

[24] Zhang, J., “Basic Neural Units of the Brain: Neurons, Synapses and Action Potential”, <i>arXiv e-prints</i>, 2019.

[25] Mesfin FB, Al-Dhahir MA. Gliomas. 2022 Jun 4. In: StatPearls [Internet]. Treasure Island (FL): StatPearls Publishing; 2022 Jan–. PMID: 28722904.

[26] *Herman GT (14 July 2009). Fundamentals of Computerized Tomography: Image Reconstruction from Projections (2nd ed.). Springer. ISBN 978-1-84628-723-7*

[27] Quettier, Lionel & Aubert, Guy & Belorgey, Jean & Berriaud, Christophe & Bourquard, Alex & Bredy, Philippe & Dubois, Olivier & Gilgrass, G. & Juster, F.P. & Lannou, Herve & Molinie, Frederic & Nusbaum, Marc & Nunio, Francois & Payn, Alain & Schild, T. & Schweitzer, Michel & Scola, Loris & Sinanna, Armand & Stepanov, Vadim & Vedrine, Pierre. (2016). Iseult/INUMAC Whole Body 11.7 T MRI Magnet. IEEE Transactions on Applied Superconductivity. PP. 1-1. 10.1109/TASC.2016.2627501.

[28] Gaillard, F., Baba, Y. MRI sequences (overview). Reference article, Radiopaedia.org. (accessed on 19 Sep 2022) https://doi.org/10.53347/rID-37346

[29] Yusuke Tabei, Keiichi Kobayashi, Kuniaki Saito, Saki Shimizu, Kaori Suzuki, Nobuyoshi Sasaki, Yoshiaki Shiokawa, Motoo Nagane, Survival in patients with glioblastoma at a first progression does not correlate with *isocitrate dehydrogenase (IDH)1* gene mutation status, *Japanese Journal of Clinical Oncology*, Volume 51, Issue 1, January 2021, Pages 45–53, https://doi.org/10.1093/jjco/hyaa162

[30] Holtedahl K. Challenges in early diagnosis of cancer: the fast track. Scand J Prim Health Care. 2020 Sep;38(3):251-252. doi: 10.1080/02813432.2020.1794415. PMID: 32791936; PMCID: PMC7470137.

[31] Tørring ML, Frydenberg M, Hansen RP, Olesen F, Vedsted P. Evidence of increasing mortality with longer diagnostic intervals for five common cancers: a cohort

study in primary care. Eur J Cancer. 2013 Jun;49(9):2187-98. doi: 10.1016/j.ejca.2013.01.025. Epub 2013 Feb 27. PMID: 23453935.

[32] Mishra A, Verma M. Cancer biomarkers: are we ready for the prime time? Cancers (Basel). 2010 Mar 22;2(1):190-208. doi: 10.3390/cancers2010190. PMID: 24281040; PMCID: PMC3827599.

[33] Macleod U, Mitchell ED, Burgess C, Macdonald S, Ramirez AJ. Risk factors for delayed presentation and referral of symptomatic cancer: evidence for common cancers. Br J Cancer 2009. Dec;101(Suppl 2):S92-S101. 10.1038/sj.bjc.6605398

[34] Lewandowska A, Rudzki G, Lewandowski T, Rudzki S. The Problems and Needs of Patients Diagnosed with Cancer and Their Caregivers. Int J Environ Res Public Health. 2020 Dec 24;18(1):87. doi: 10.3390/ijerph18010087. PMID: 33374440; PMCID: PMC7795845.

[35]Krzyszczyk P, Acevedo A, Davidoff EJ, Timmins LM, Marrero-Berrios I, Patel M, White C, Lowe C, Sherba JJ, Hartmanshenn C, O'Neill KM, Balter ML, Fritz ZR, Androulakis IP, Schloss RS, Yarmush ML. The growing role of precision and personalized medicine for cancer treatment. Technology (Singap World Sci). 2018 Sep-Dec;6(3-4):79-100. doi: 10.1142/S2339547818300020. Epub 2019 Jan 11. PMID: 30713991; PMCID: PMC6352312.

[36] Ginsburg GS, Phillips KA. Precision Medicine: From Science To Value. Health Aff (Millwood). 2018 May;37(5):694-701. doi: 10.1377/hlthaff.2017.1624. PMID: 29733705; PMCID: PMC5989714.

[37] Goossens N, Nakagawa S, Sun X, Hoshida Y. Cancer biomarker discovery and validation. Transl Cancer Res. 2015 Jun;4(3):256-269. doi: 10.3978/j.issn.2218-676X.2015.06.04. PMID: 26213686; PMCID: PMC4511498.

[38] Heo YJ, Hwa C, Lee GH, Park JM, An JY. Integrative Multi-Omics Approaches in Cancer Research: From Biological Networks to Clinical Subtypes. Mol Cells. 2021 Jul 31;44(7):433-443. doi: 10.14348/molcells.2021.0042. PMID: 34238766; PMCID: PMC8334347.

[39] https://bmcbioinformatics.biomedcentral.com/articles/10.1186/s12859-022-04794-9

[40] *"Dunning-Kruger effect". www.britannica.com. Archived from the original on 30 November 2021. Retrieved 7 December 2021.*

[41] Koumakis L. Deep learning models in genomics; are we there yet? Comput Struct Biotechnol J. 2020 Jun 17;18:1466-1473. doi: 10.1016/j.csbj.2020.06.017. PMID: 32637044; PMCID: PMC7327302.

[42] Ravid Shwartz-Ziv and Amitai Armon. 2022. Tabular data: Deep learning is not all you need. Inf. Fusion 81, C (May 2022), 84–90. https://doi.org/10.1016/j.inffus.2021.11.011

[43] Saxena S, Jena B, Gupta N, Das S, Sarmah D, Bhattacharya P, Nath T, Paul S, Fouda MM, Kalra M, Saba L, Pareek G, Suri JS. Role of Artificial Intelligence in Radiogenomics for Cancers in the Era of Precision Medicine. *Cancers*. 2022; 14(12):2860. https://doi.org/10.3390/cancers14122860

[44] Despotović I, Goossens B, Philips W. MRI segmentation of the human brain: challenges, methods, and applications. Comput Math Methods Med. 2015;2015:450341. doi: 10.1155/2015/450341. Epub 2015 Mar 1. PMID: 25945121; PMCID: PMC4402572.

[45] Kalavathi P, Prasath VB. Methods on Skull Stripping of MRI Head Scan Images-a Review. J Digit Imaging. 2016 Jun;29(3):365-79. doi: 10.1007/s10278-015-9847-8. PMID: 26628083; PMCID: PMC4879034.

[46] B. Zitov ́a and J. Flusser, “Image registration methods: a survey,”Image and Vision Computing, vol. 21, no. 11, pp. 977–1000, 2003.

[47] Buades, A., Coll, B., and Morel, J. (2005). “A Non-local Algorithm for Image Denoising,” in 2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR’05), 60–65.

[48] Moreno López M, Frederick JM and Ventura J (2021) Evaluation of MRI Denoising Methods Using Unsupervised Learning. *Front. Artif. Intell.* 4:642731. doi: 10.3389/frai.2021.642731

[49] Juntu, J., Sijbers, J., Van Dyck, D., Gielen, J. (2005). Bias Field Correction for MRI Images. In: Kurzyński, M., Puchała, E., Woźniak, M., żołnierek, A. (eds) Computer Recognition Systems. Advances in Soft Computing, vol 30. Springer, Berlin, Heidelberg. https://doi.org/10.1007/3-540-32390-2_64

[50] Sled JG, Zijdenbos AP, Evans AC. A nonparametric method for automatic correction of intensity nonuniformity in MRI data. IEEE Trans Med Imaging. 1998 Feb;17(1):87-97. doi: 10.1109/42.668698. PMID: 9617910.

[51] Tustison NJ, Avants BB, Cook PA, Zheng Y, Egan A, Yushkevich PA, Gee JC. N4ITK: improved N3 bias correction. IEEE Trans Med Imaging. 2010 Jun;29(6):1310-20. doi: 10.1109/TMI.2010.2046908. Epub 2010 Apr 8. PMID: 20378467; PMCID: PMC3071855.

[52] N. V. Chawla, K. W. Bowyer, L. O.Hall, W. P. Kegelmeyer, "SMOTE: synthetic minority over-sampling technique," Journal of artificial intelligence research, 321-357, 2002

[53] H. Han, W. Wen-Yuan, M. Bing-Huan, "Borderline-SMOTE: a new over-sampling method in imbalanced data sets learning," Advances in intelligent computing, 878-887, 2005.

[54] Haibo He, Yang Bai, E. A. Garcia and Shutao Li, "ADASYN: Adaptive synthetic sampling approach for imbalanced learning," *2008 IEEE International Joint Conference on Neural Networks (IEEE World Congress on Computational Intelligence)*, 2008, pp. 1322-1328, doi: 10.1109/IJCNN.2008.4633969.

[55] Palm, G. (1986). Warren McCulloch and Walter Pitts: A Logical Calculus of the Ideas Immanent in Nervous Activity. In: Palm, G., Aertsen, A. (eds) Brain Theory. Springer, Berlin, Heidelberg. https://doi.org/10.1007/978-3-642-70911-1_14

[56] Rosenblatt, F. (1958). The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review, 65*(6), 386–408. https://doi.org/10.1037/h0042519

[57] Weston J, Leslie C, Ie E, Zhou D, Elisseeff A, Noble WS. Semi-supervised protein classification using cluster kernels. Bioinformatics. 2005 Aug 1;21(15):3241-7. doi: 10.1093/bioinformatics/bti497. Epub 2005 May 19. PMID: 15905279.

[58] Breiman, L. Random Forests. *Machine Learning* **45**, 5–32 (2001). https://doi.org/10.1023/A:1010933404324

[59] Friedman, Jerome. (2000). Greedy Function Approximation: A Gradient Boosting Machine. The Annals of Statistics. 29. 10.1214/aos/1013203451.

[60] Smith, L. N., "Cyclical Learning Rates for Training Neural Networks", <i>arXiv e-prints</i>, 2015.

[61]]You, K., Long, M., Wang, J., and Jordan, M. I., "How Does Learning Rate Decay Help Modern Neural Networks?", <i>arXiv e-prints</i>, 2019.

[62] Kingma, Diederik & Ba, Jimmy. (2014). Adam: A Method for Stochastic Optimization. International Conference on Learning Representations.

[63] Ruder, S., “An overview of gradient descent optimization algorithms”, <i>arXiv e-prints</i>, 2016.

[64] Schapire, R. E. (2013). Explaining adaboost. In *Empirical inference* (pp. 37–52). Springer.

[65] *Ronneberger O, Fischer P, Brox T (2015). "U-Net: Convolutional Networks for Biomedical Image Segmentation". arXiv:1505.04597*

[66] *Dice, Lee R. (1945). "Measures of the Amount of Ecologic Association Between Species". Ecology.* ***26*** *(3): 297–302. doi:10.2307/1932409. JSTOR 1932409*

[67] *Carass, A.; Roy, S.; Gherman, A.; Reinhold, J.C.; Jesson, A.; et al. (2020). "Evaluating White Matter Lesion Segmentations with Refined Sørensen-Dice Analysis". Scientific Reports.* ***10*** *(1): 8242. Bibcode:2020NatSR..10.8242C. doi:10.1038/s41598-020-64803-w. ISSN 2045-2322. PMC 7237671. PMID 32427874*

[68] *Sørensen, T. (1948). "A method of establishing groups of equal amplitude in plant sociology based on similarity of species and its application to analyses of the vegetation on Danish commons". Kongelige Danske Videnskabernes Selskab.*