OPTIMAL FEATURE SELECTION TO PREDICT RENAL CELL CARCINOMA USING DEEP LEARNING ALGORITHMS

A PROJECT REPORT

Submitted by

RAHUL S [211419104209]

SAI KUMAR M [211419104226]

TARUN V [211419104288]

in partial fulfillment for the award of the degree of

BACHELOR OF ENGINEERING

in

COMPUTER SCIENCE AND ENGINEERING

PANIMALAR ENGINEERING COLLEGE

(An Autonomous Institution, Affiliated to Anna University, Chennai)

APRIL 2023

PANIMALAR ENGINEERING COLLEGE

(An Autonomous Institution, Affiliated to Anna University, Chennai)

BONAFIDE CERTIFICATE

Certified that this project report **"OPTIMAL FEATURE SELECTION TO PREDICT RENAL CELL CARCINOMA USING DEEP LEARNING ALGORITHMS"** is the bonafide work of "**RAHUL S (211419104209), SAI KUMAR M (211419104226), TARUN V (211419104288)"** who carried out the project work under my supervision.

SIGNATURE

Dr.L.JABASHEELA,M.E.,Ph.D.,
PROFESSOR
HEAD OF THE DEPARTMENT

DEPARTMENT OF CSE,
PANIMALAR ENGINEERING COLLEGE,
NASARATHPETTAI,
POONAMALLEE,
CHENNAI-600 123.

SIGNATURE

Mr. M.MOHAN, B.E, M.Tech.,(Ph.D.),
SUPERVISOR
ASSISTANT PROFESSOR
GRADE 1

DEPARTMENT OF CSE,
PANIMALAR ENGINEERING COLLEGE,
NASARATHPETTAI,
POONAMALLEE,
CHENNAI-600 123.

Certified that the above candidate(s) was/ were examined in the End Semester Project Viva-Voce Examination held on.....

INTERNAL EXAMINER

EXTERNAL EXAMINER

DECLARATION BY THE STUDENT

We **RAHUL S (211419104209), SAI KUMAR M (211419104226), TARUN V (211419104288)** hereby declare that this project report titled "**OPTIMAL FEATURE SELECTION TO PREDICT RENAL CELL CARCINOMA USING DEEP LEARNING ALGORITHMS**" , under the guidance of **Mr.M.MOHAN, B.E, M.Tech.,(Ph.D.),** is the original work done by us and we have not plagiarized or submitted to any other degree in any university by us.

ACKNOWLEDGEMENT

We would like to express our deep gratitude to our respected Secretary and Correspondent **Dr.P.CHINNADURAI, M.A., Ph.D.** for his kind words and enthusiastic motivation, which inspired us a lot in completing this project.

We express our sincere thanks to our beloved Directors **Tmt.C.VIJAYA RAJESWARI**, **Dr.C.SAKTHI KUMAR,M.E.,Ph.D** and **Dr.SARANYASREE SAKTHI KUMAR B.E.,M.B.A.,Ph.D.,** for providing us with the necessary facilities to undertake this project.

We also express our gratitude to our Principal **Dr.K.MANI, M.E., Ph.D.** who facilitated us in completing the project.

We thank the Head of the CSE Department, **Dr. L. JABASHEELA , M.E.,Ph.D.,** for the support extended throughout the project.

We would like to thank my **Mr.M.MOHAN, B.E, M.Tech.,(Ph.D.),** and all the faculty members of the Department of CSE for their advice and encouragement for the successful completion of the project.

ABSTRACT

Kidney cancer is among the 10 most common cancers in both men and women. Overall, the lifetime risk for developing kidney cancer in men is about 1 in 46 (2.02%). Diagnosis of Kidney cancer at an early stage helps doctors to decide which treatment will work best. The natural history of renal cell carcinoma (RCC) is highly unpredictable and challenging, but it provides the best chance for cure. Current diagnosis of renal cancer consists of histopathologic examination of tissue sections and classification into tumor stages and grades of malignancy. We aimed to develop an improved cancer prediction system by using the deep learning techniques, which automates the classification of Renal Cell Carcinoma Levels and also for identification of features using histopathological images. We trained and validated Graph neural networks (GNN's) on whole-slide images to distinguish Renal Cell Carcinoma from normal tissue with comparatively improved classification accuracy. At a more fundamental level, it is evident that this system with deep learning techniques is helping to improve our basic understanding of cancer development and progression.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

ABBREVIATION	EXPANSION
CNN	Convolutional Neural Network
GNN	Graph Neural Network
CT	Computer Tomography
AHD CNN	Adaptive Hybridized Deep CNN
RELU	Rectified Linear Unit
ROC	Receiver Operating Curve
GAN	Graph Attention Network
CLI	Command Line Interface
IHC	ImmunoHistoChemistry
DFD	Data Flow Diagram
RUP	Rational Unified Process

CHAPTER 1
INTRODUCTION

1. INTRODUCTION

Renal cell carcinoma is one of the 10 most common types of cancers accounting for 2-3% of overall cancer diagnosed worldwide. Renal Cell Carcinoma is a group of heterogeneous tumors with different characteristics, molecular characteristics and clinical outcomes. The various types of renal cell carcinoma include Clear Cell, Papillary and chromophobe reasonable for 70-80%, 14-17% and 4-8% of renal cell carcinoma, respectively. Clear cell RCC (KIRC) and papillary RCC (KIRP) derive from cells inside the proximal convoluted tubules of the nephron5. KIRC is characterized by loss of chromosome 3p and mutation of the von Hippel–Lindau (VHL) gene while KIRP is characterized by trisomy of chromosomes and loss of chromosome 9p. Chromophobe RCC (KICH) originates from intercalated cells within the distal convoluted tubules and is characterized by loss of chromosomes. KIRC patients have an overall 5 year survival rate of 55-60% whereas for KIRP patients, it varies from 80- 90 and for KICH patients, it is 90. Due to such distinct biological and clinical behavior of subtypes, accurate detection of RCC and its subtypes is significant for the clinical management of patients. RCC subtypes are often detected radiologically that support the degree of tumor enhancement on multidetector computerized tomography (CT) or resonance imaging (MRI). Further, the microscopic examination of Haematoxylin and Eosin (H & E) stained slides of biopsies continue to be a valuable tool for pathologists and clinicians. Histological images contain markers of disease progression and phenotypic information which contains the diagnostic and predictive values. Major limitations within the examination of H & E images by pathologists are inter-observer discordance and time required to diagnose. The pathologist follows an easy decision tree approach utilizing only a fraction of obtainable information.

In an unaided approach to histopathology, complex morphological knowledge goes unused.

1.1 GNN

Graph Neural Networks (GNNs) are a class of neural networks that are designed to operate on graphs and other structured data. A graph is a mathematical representation of a set of objects (nodes) and their relationships (edges), and GNNs use this graph structure to perform computations and make predictions.

GNNs are particularly useful for applications where the data is naturally structured as a graph, such as social networks, protein structures, and chemical compounds. By leveraging the structure of the graph, GNNs can learn complex relationships between nodes and make predictions based on these relationships.

One of the key features of GNNs is their ability to perform message passing between nodes in the graph. During message passing, each node aggregates information from its neighbors and updates its own state based on this information. By iteratively passing messages between nodes, GNNs can build up a global representation of the graph that captures the relationships between all of the nodes.

GNNs have shown promise in a wide range of applications, including drug discovery, recommendation systems, and computer vision. However, there are still challenges to overcome in developing effective GNNs, such as dealing with large and complex graphs, handling missing data, and preventing overfitting. Nevertheless, as research in GNNs continues to advance, their potential for

modeling complex relationships in structured data is becoming increasingly clear.

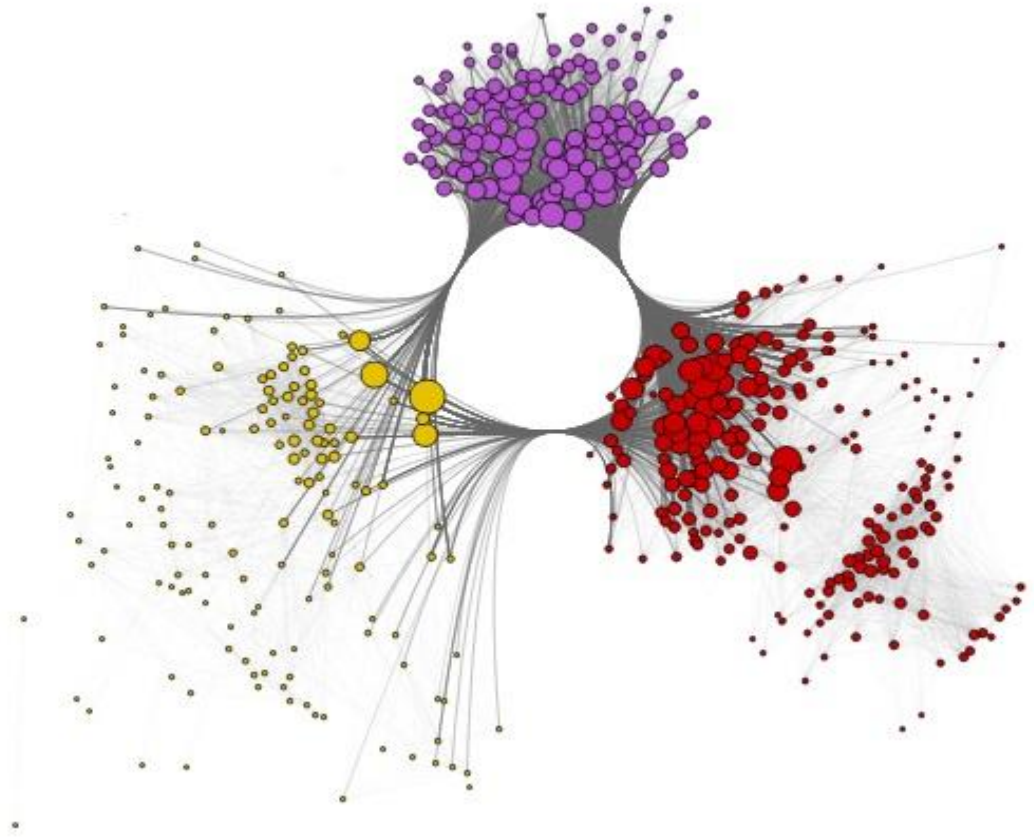


Figure 1.1 Graph Neural Network

Graph Neural Networks (GNNs) can be represented mathematically using the following notation:

G = (V, E) represents a graph, where V is the set of nodes and E is the set of edges between the nodes.

h_i^k represents the hidden state of node i at iteration k of the GNN. This hidden state is updated through message passing from neighboring nodes and aggregation of these messages.

x_i represents the feature vector for node i.

f represents a neural network that transforms the feature vector x_i into a hidden state h_i^0.

W^k represents the weight matrix for iteration k of the GNN.

a_{i,j} represents the adjacency matrix of the graph, which indicates whether there is an edge between node i and node j.

N(i) represents the set of neighboring nodes of node i.

Using this notation, the update rule for a GNN can be written as:

h_i^(l+1) = σ(Σ_j∈N(i) (1/√deg(i)deg(j)) h_j^(l)W^(l)h_i^(l))

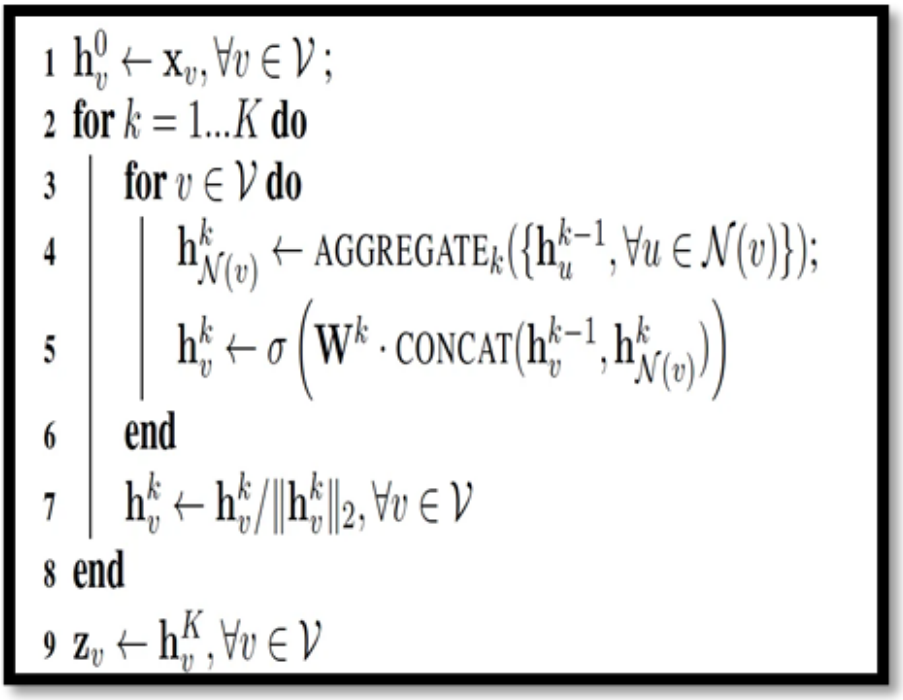


Figure 1.2 Algorithmic Notation for GNN

where deg(i) is the degree of node i, which is the number of edges incident to it. This equation updates the hidden state of node i by aggregating the hidden states of its neighbors and passing this information through the weight matrix W^k.

The output of a GNN can be obtained by applying a readout function to the final hidden states of all nodes in the graph. The readout function can be a simple sum or average of the hidden states, or it can be a more complex neural network that maps the hidden states to a final prediction.

1.2 Existing System

Using patient-derived Xenografts, several models were developed for renal cell carcinoma and support vector machines. Hence it has some certain limitations within. Some existing systems could not produce results with more accuracy which can lead to miscalculations.

DISADVANTAGES:

1. More costly
2. Rodent stroma
3. Immuno-deficient hosts
4. Non-natural site (sc)
5. Most tumor lines old
6. Genetic diversity limited

1.3 Prime Objectives

Early detection of cancer is important because when abnormal tissue is found early, it may be easier to treat. By the time symptoms appear, cancer may have begun to spread and be harder to treat. Therefore, this pre-operative assessment of

cancer grade malignancy is useful to apply the best treatment decisions. In recent years, deep learning-based image analysis has gained wide popularity in cancer prognosis and prediction. The goal of this system is to investigate the feasibility, stages of clear cell RCC histopathological images and also reviews the status of prognostic factors in RCC.

1.3.1 Scope

In areas with inadequate medical resources, some developing countries and rural areas, for example, the life expectancies and health conditions of residents are generally poor due to limited access to qualified healthcare providers. AI by its nature is suited to be applied in such areas, given that a single AI clinical decision support system has the capacity to serve a large population encompassed within its scope. Due to the high prevalence and low awareness of kidney diseases, early diagnosis and intervention are rarely possible, especially in resource-inadequate areas. With the help of AI, health practitioners in such areas will be able to screen potential kidney disease patients and make referrals according to their risk levels. In other situations where qualified healthcare professionals are scarce for a large population, auxiliary patient management tools will serve to reduce unsatisfactory waiting time, manage intelligent follow-up of patients, and educate patients with kidney disease.

1.4 Proposed System

The use of Histological images help us gain a better understanding of cell behavior and reproduction, making them simpler and understandable. Graph neural networks (GNN's) trained on whole-slide images to distinguish clear cell and chromophobe RCC from normal tissue with comparatively higher

classification accuracy. With the motto of enhancing the prediction accuracy of pan renal cancer using the images of abnormal cells, we're exclusively training a deep learning algorithm with a huge data set that could improve the quality of prediction compared to the existing system.

CHAPTER 2
LITERATURE SURVEY

LITERATURE SURVEY

There have been many works to digitally store and retrieve patient's medical prescription details. However there have been certain limitations in each of the techniques used.

Nathan Hadjiyski, Ann Arbor [1]et al Kidney Cancer Staging: Deep Learning Neural Network Based Approach (2020)

Kidney cancer is a common type of cancer that can be very deadly. Proper cancer staging is critical for the correct selection of treatment for the affected patients. Stage 1 kidney cancer is an important threshold for the treatment decision. The physicians usually have difficulty to determine the cancer stage correctly, which can result in under- or overtreatment. Deep Learning Neural Network (DLNN) was used to predict kidney cancer Stage 1 versus higher stages in order to allow for a more accurate kidney cancer stage assessment by physicians. Computer tomography (CT) scans from 227 patients with different stages of kidney cancer from the Cancer Imaging Archive TCIA database were used for training, validation, and testing of the DLNN. The kidney cancers were cropped from the 3D CT scans. The dataset was split into 48% training, 10% validation, and 42% test sets. Inception V3 DLNN with transfer learning was trained with the cropped kidney cancer training images. Area under the ROC curve (AUC) was used to estimate the classification accuracy. The AUC of 0.97 for training, 0.91 for validation and 0.90 for the test sets was obtained. This AI system shows promise for potentially assisting physicians in kidney cancer staging.

Guozhen Chen, Chenguang Ding, Yang Li, Xiaojun Hu, Xiao Li, Li Ren, Xiaoming Ding, Puxun Tian, Wujun Xue [2]et al Prediction of Chronic

Kidney Disease Using Adaptive Hybridized Deep Convolutional Neural Network on the Internet of Medical Things Platform (2020)

Chronic Kidney disease is a severe lifelong condition caused either by renal disease or by impaired functions of the kidneys. In the present area of research, Kidney cancer is one of the deadliest and crucial importance for the survival of the patients ' diagnosis and classification. Early diagnosis and proper therapy can stop or delay the development of this chronic disease into the final stage where dialysis or renal transplantation is the only way of saving the life of the patient. The development of automated tools to accurately identify subtypes of kidney cancer is, therefore, an urgent challenge in the recent past. In this paper, to examine the ability of various deep learning methods an Adaptive hybridized Deep Convolutional Neural Network (AHDCNN) has been proposed for the early detection of Kidney disease efficiently and effectively. Classification technology efficiency depends on the role of the data set. To enhance the accuracy of the classification system by reducing the feature dimension an algorithm model has been developed using CNN. These high-level properties help to build a supervised tissue classifier that discriminates between the two types of tissue. The experimental process on the Internet of medical things platform (IoMT)concludes, with the aid of predictive analytics, that advances in machine learning provide a promising framework for the recognition of intelligent solutions to prove their predictive capability beyond the field of kidney disease.

Ahmed Elmahy, Sherin Aly, Fayek Elkhwsky [3]et al Cancer Stage Prediction From Gene Expression Data Using Weighted Graph Convolution Network (2021)

The early detection of cancer stage is a crucial step for effective treatment. In contrast to traditional approaches, RNA -Seq is the current state of the art

technique for gene expression estimation. RNA -Seq data have been used in research and in production as input data for several classification and prediction models in many disease including cancer staging. We present a novel cancer stage prediction approach based on gene expression data. Our approach is based on Weighted Graph Convolution Networks (GCN). GCN is the application of deep learning back-propagation on graph structures. In this work, we used correlation between genes to generate a gene network graph. A neural network model with weighted graph convolution layer was trained to predict the cancer stage for cancer patients. We employed the Kidney Renal Clear Cell Carcinoma dataset (TCGA-KIRC) from the Human Cancer Genome Atlas (TCGA) to predict the cancer stage for each patient. The TCGA-KIRC dataset includes 4 cancer stages, I, II, III, and IV. We generated a binary classification problem where stages I and II are classified as "early cancer stage" and stages III and IV are classified as "late cancer stage". We compared our approach to the state of the art approaches such as random forest and support vector machine. Our approach achieved an accuracy of 82% which outperformed existing approaches with more than 3% increase.

Dong Liu; Jinkai Shao; Honggang Liu; Wei Cheng [4]et al Design on Early Warning System for Renal Cancer Recurrence Based on CNN-Based Internet of Things (2021)

Kidney cancer is a type of urinary system tumor. The incidence of kidney cancer, which is second only to bladder cancer, has shown an overall upward trend in recent years. However, the early judgment of kidney cancer is still in the imaging biomarker discovery stage. Early detection and treatment cannot be achieved. Based on the natural advantages of the Internet of Things in the medical field, we focused on an intelligent early warning model of renal cancer recurrence and built a renal cancer early warning system integrated with the Internet of

Things. We integrated the influencing factors of renal cell carcinoma, constructed a sample set, conducted data analysis and optimized the dataset. Aiming at the instability of renal cancer recurrence, five supervised learning prediction algorithms, including multiple linear regression, Bayesian ridge regression, gradient boosting tree, support vector regression, and convolutional neural network were used to develop a renal cancer recurrence prediction model. The predictive performance of these five algorithms were compared and discussed. Finally, the best renal cancer recurrence prediction model was established by combining a convolutional neural network with an Internet of Things medical framework. This design provided an intelligent early warning system to predict the recurrence time of renal cancer patients. In addition, the warning prompts provided in accordance with the model results can assist doctors in making preliminary judgments of the patient's condition and has a certain auxiliary effect on the clinical diagnosis and treatment of cancer and kidney cancers.

Christopher Ricketts, Aguirre Andres De Cubas, [5]et al The Cancer Genome Atlas Comprehensive Molecular Characterization of Renal Cell Carcinoma (2018)

Renal cell carcinoma (RCC) is not a single disease, but several histologically defined cancers with different genetic drivers, clinical courses, and therapeutic responses. The current study evaluated 843 RCC from the three major histologic subtypes, including 488 clear cell RCC, 274 papillary RCC, and 81 chromophobe RCC. Comprehensive genomic and phenotypic analysis of the RCC subtypes reveals distinctive features of each subtype that provide the foundation for the development of subtype-specific therapeutic and management strategies for patients affected with these cancers. Somatic alteration of BAP1, PBRM1, and PTEN and altered metabolic pathways correlated with subtype-specific decreased

survival, while CDKN2A alteration, increased DNA hypermethylation, and increases in the immune-related Th2 gene expression signature correlated with decreased survival within all major histologic subtypes. CIMP-RCC demonstrated an increased immune signature, and a uniform and distinct metabolic expression pattern identified a subset of metabolically divergent (MD) ChRCC that associated with extremely poor survival.

Nicolas Coudray, Paolo Santiago Ocampo, Theodore Sakellaropoulos, Navneet Narula, Matija Snuderl, David Fenyö, Andre L Moreira, Narges Razavian, Aristotelis Tsirigos [6]et al Classification and mutation prediction from non-small cell lung cancer histopathology images using deep learning (2018)

Visual inspection of histopathology slides is one of the main methods used by pathologists to assess the stage, type and subtype of lung tumors. Adenocarcinoma (LUAD) and squamous cell carcinoma (LUSC) are the most prevalent subtypes of lung cancer, and their distinction requires visual inspection by an experienced pathologist. In this study, we trained a deep convolutional neural network (inception v3) on whole-slide images obtained from The Cancer Genome Atlas to accurately and automatically classify them into LUAD, LUSC or normal lung tissue. The performance of our method is comparable to that of pathologists, with an average area under the curve (AUC) of 0.97. Our model was validated on independent datasets of frozen tissues, formalin-fixed paraffin-embedded tissues and biopsies. Furthermore, we trained the network to predict the ten most commonly mutated genes in LUAD. We found that six of them-STK11, EGFR, FAT1, SETBP1, KRAS and TP53-can be predicted from pathology images, with AUCs from 0.733 to 0.856 as measured on a held-out population. These findings

suggest that deep-learning models can assist pathologists in the detection of cancer subtype or gene mutations.

Jun Cheng, Jie Zhang [7]et alClassification and mutation prediction from non-small cell lung cancer histopathology images using deep learning (2017)

In cancer, both histopathologic images and genomic signatures are used for diagnosis, prognosis, and subtyping. However, combining histopathologic images with genomic data for predicting prognosis, as well as the relationships between them, has rarely been explored. In this study, we present an integrative genomics framework for constructing a prognostic model for clear cell renal cell carcinoma. We used patient data from The Cancer Genome Atlas (n = 410), extracting hundreds of cellular morphologic features from digitized whole-slide images and eigengenes from functional genomics data to predict patient outcome. The risk index generated by our model correlated strongly with survival, outperforming predictions based on considering morphologic features or eigengenes separately. The predicted risk index also effectively stratified patients in early-stage (stage I and stage II) tumors, whereas no significant survival difference was observed using staging alone. The prognostic value of our model was independent of other known clinical and molecular prognostic factors for patients with clear cell renal cell carcinoma. Overall, this workflow and the shared software code provide building blocks for applying similar approaches in other cancers.

Murari Lal Dhanetwal, Sonia Badwal, Gaurav Pratap Singh Gahlot, Kavita Sahai, A K Shukla [8]et al Immunohistological Diagnosis of Primary and Metastatic Renal Cell Carcinoma Using Panel of Immunohistochemical Markers (2019)

Tumour heterogeneity and lack of markers with high specificity makes diagnosis of renal cell carcinoma (RCC) challenging. The study was undertaken to evaluate panel of IHC markers to enable diagnosis and reproducible classification in primary and metastatic renal tumors. Methods: Descriptive Study wherein 100 cases of RCC and 25 trucut biopsies (20 metastatic and 5 primary renal tumors) were evaluated for morphology and immunostained by panel of immunohistochemical (IHC) markers consisting of CA-9, CD10, CK-7, AMACAR and TFE-3 with additional markers as required. Result: Morphologically tumors were grouped as clear cell and nonclear cell (eosinophilic and poorly differentiated). Clear cell RCCs (CCRCC), clear cell papillary RCC (CCPRCC) and multilocular cystic RCC (MCRNLMP) displayed strong statistical association of CA-9 immunostaining (p=50.00, x2-0.000). Inverse correlation was found between the intensity of the staining of CA-9 and tumor grade. (p=32.97, x2=0.000). CA-9 and CK-7 co-expression was evident in all cases of CCPRCC and MCRNLMP. Papillary RCC exhibited positive statistical correlation with CK-7 and AMACAR. E-cadherin and CD117 were required additionally to differentiate between oncocytoma and chromophobe RCC. CD10 and Pax 8 were most helpful in diagnosing metastatic RCCs Conclusion: IHC panel consisting of CA-9, CD10, CK7, AMACR and TFE3 helps triage RCCs with clear cell/eosinophilic cell / papillary/poorly differentiated pattern. In a setting of metastatic RCC, use of CD10 and Pax 8 together facilitate primary diagnosis of RCC when tissue available is limited.

Yun Liu, Krishna Gadepalli, Mohammad Norouzi, George E. Dahl, Timo Kohlberger, Aleksey Boyko, Subhashini Venugopalan, Aleksei Timofeev, Philip Q. Nelson, Greg S. Corrado, Jason D. Hipp, Lily Peng, Martin C.

Stumpe [9]et al Detecting Cancer Metastases on Gigapixel Pathology Images (2017)

Each year, the treatment decisions for more than 230,000 breast cancer patients in the U.S. hinge on whether the cancer has metastasized away from the breast. Metastasis detection is currently performed by pathologists reviewing large expanses of biological tissues. This process is labor intensive and error-prone. We present a framework to automatically detect and localize tumors as small as 100 x 100 pixels in gigapixel microscopy images sized 100,000 x 100,000 pixels. Our method leverages a convolutional neural network (CNN) architecture and obtains state-of-the-art results on the Camelyon16 dataset in the challenging lesion-level tumor detection task. At 8 false positives per image, we detect 92.4% of the tumors, relative to 82.7% by the previous best automated approach. For comparison, a human pathologist attempting exhaustive search achieved 73.2% sensitivity. We achieve image-level AUC scores above 97% on both the Camelyon16 test set and an independent set of 110 slides. In addition, we discover that two slides in the Camelyon16 training set were erroneously labeled normal. Our approach could considerably reduce false negative rates in metastasis detection.

[10] Pegah Khosravi, Ehsan Kazemi, Marcin Imielinski, Olivier Elemento, Iman Hajirasouliha [10]et al Deep Convolutional Neural Networks Enable Discrimination of Heterogeneous Digital Pathology Images (2018)

Pathological evaluation of tumor tissue is pivotal for diagnosis in cancer patients and automated image analysis approaches have great potential to increase precision of diagnosis and help reduce human error. In this study, we utilize several computational methods based on convolutional neural networks (CNN) and build a stand-alone pipeline to effectively classify different histopathology images across

different types of cancer. In particular, we demonstrate the utility of our pipeline to discriminate between two subtypes of lung cancer, four biomarkers of bladder cancer, and five biomarkers of breast cancer. In addition, we apply our pipeline to discriminate among four immunohistochemistry (IHC) staining scores of bladder and breast cancers. Our classification pipeline includes a basic CNN architecture, Google's Inceptions with three training strategies, and an ensemble of two state-of-the-art algorithms, Inception and ResNet. Training strategies include training the last layer of Google's Inceptions, training the network from scratch, and fine-tunning the parameters for our data using two pre-trained version of Google's Inception architectures, Inception-V1 and Inception-V3. We demonstrate the power of deep learning approaches for identifying cancer subtypes, and the robustness of Google's Inceptions even in presence of extensive tumor heterogeneity. On average, our pipeline achieved accuracies of 100%, 92%, 95%, and 69% for discrimination of various cancer tissues, subtypes, biomarkers, and scores, respectively.

CHAPTER 3
SYSTEM SPECIFICATIONS

3. SYSTEM SPECIFICATIONS

3.1 REQUIREMENT SPECIFICATIONS

3.1.1 MINIMUM SOFTWARE REQUIREMENTS

Operating System : Windows 2007 and above

Interface : Jupyter Notebook, Anaconda Distribution

Coding Language : Python

Dataset : Kaggle

3.1.2 MINIMUM HARDWARE REQUIREMENTS

Processor : Core2duo

Hard Disk : 128 GB

RAM : 2 GB or more

3.2 SOFTWARE SPECIFICATION

3.2.1 ANACONDA DISTRIBUTION

Anaconda is a distribution of the Python and R programming languages that seeks to streamline package management and deployment. Data-science packages for Windows, Linux, and macOS are included in the release. Anaconda, Inc., created in 2012 by Peter Wang and Travis Oliphant, is responsible for its development and upkeep. It is a product of Anaconda, Inc. and is also referred to as Anaconda Distribution or Anaconda Individual Edition, but the firm also produces Anaconda Team Edition and Anaconda Enterprise Edition, both of which are paid for. Conda, the package management system, controls Anaconda's versioning of packages.

The Anaconda distribution is preloaded with more than 250 packages, and there are over 7,500 open-source packages that can be installed from PyPI and managed through the conda package and virtual environment manager. Additionally, Anaconda Navigator, a graphical user interface, is included as an alternative to the command-line interface (CLI).

Anaconda's Python distribution provides several benefits as it comes bundled with not only Python but also around 600 additional Python packages that can be installed free of charge. These supplementary packages contain many of the frequently used Python packages for solving problems, making Anaconda a convenient choice for data analysis and scientific computing.

Anaconda provides an added benefit of simplified package management and virtual environments, making it much easier to handle. At the initial stage of downloading Anaconda, virtual environments and package handling might not seem like a significant aspect. In fact, if you are new to Anaconda, you may not even be aware of these concepts yet. However, as you begin to write Python programs and download additional modules from PyPI or conda-forge, managing packages and virtual environments become increasingly important. With Anaconda, these tasks are made more straightforward, enabling a smoother and more efficient workflow.

3.2.2 JUPYTER NOTEBOOK

A Jupyter notebook is a digital document that combines programming code with descriptive text. These notebooks have the ability to include media, such as charts, plots, images, videos, and links, and can be executed in a web browser like Google Chrome or Firefox. While Jupyter notebooks support numerous

programming languages, they frequently utilize Python code. The Python code included in a Jupyter notebook is the same type of code found in a standard .py file

The descriptive text portions of a Jupyter notebook use the markdown format to provide explanations and clarifications of the programming code. Markdown files use the .md extension and allow for text formatting, such as bold and italic text, tables, lists, code listings, and images. A Jupyter notebook can be thought of as a combination of a Python module .py file and a markdown .md file with the Python REPL (Read-Eval-Print Loop) interspersed between code sections.

In contrast to the Python REPL, where only one command can be entered at a time with only one line of output displayed, and a .py file, which runs the entire file line by line and produces output all at once, a Jupyter notebook can run code chunks individually and in any order, without running the entire notebook. Additionally, Jupyter notebooks can render markdown sections to display rich text with headings, formatting, and images. The three types of cells found in a Jupyter notebook are code cells, output cells, and markdown cells.

3.2.3 PYTHON 3.6

Python is a high-level, object-oriented, and interpreted programming language with dynamic semantics. Its built-in data structures, along with dynamic typing and binding, make it an appealing option for Rapid Application Development, as well as for use as a scripting or glue language to connect various components together. Python's syntax is easy to learn and understand, which emphasizes readability, and ultimately reduces the cost of program maintenance. Additionally, Python supports modules and packages, which encourages program modularity and code reuse. The

Python interpreter and extensive standard library are available in both source and binary form at no charge for all major platforms and can be distributed freely.

One of the reasons why programmers tend to favor Python is due to its increased productivity. Since there is no compilation step, the edit-test-debug cycle is exceptionally fast. Debugging Python programs is also straightforward, as errors are raised as exceptions instead of causing a segmentation fault. When the program does not catch an exception, the interpreter prints a stack trace. Furthermore, a source-level debugger allows inspection of both local and global variables, the evaluation of arbitrary expressions, setting breakpoints, and stepping through the code a line at a time, testifying to Python's introspective power. However, often the quickest way to debug a program is to add a few print statements to the source code, thanks to the fast edit-test-debug cycle, making this simple approach very effective.

3.2.4 KAGGLE

Kaggle is a valuable resource for cloud-based computing, offering up to 30 hours of GPU and 20 hours of TPU usage per week. Users can upload their own datasets and download others, as well as view and discuss other people's datasets and notebooks. The platform also includes a scoring system that rewards users for helping others and sharing information, and ranks them on a live leaderboard of 8 million Kaggle users.

It caters to a diverse audience, from beginners to experienced data scientists. The platform provides courses for beginners and a community of users at varying levels of expertise, offering opportunities to communicate with highly experienced data scientists. As users earn points and medals, they can demonstrate their

progress and potentially attract the attention of recruiters, unlocking new job opportunities.

This system describes the workflow for using 160,000 full sized histopathological images without kernel crash. This is accomplished by implementing a directory structure setup which uses generators to feed the data into our model so that the model can be trained, validated and also applied for the purpose of prediction. In this system, the model is trained using 144,000 images and validated on 16,000 images.

CHAPTER 4

SYSTEM ANALYSIS

4. SYSTEM ANALYSIS

4.1 FEASIBILITY STUDY:

A Cancer Prediction System using Deep learning Algorithms is built for kidney cancer prediction, assessment and its ability to accurately predict the (5 year) outcome of an incidence of cancer was assessed. A wide range of different histological images and classification methods were applied in order to test the feasibility of the model and also to find the best performing algorithms on any given dataset. A special Model with a variety of classification algorithms was examined to facilitate the implementation of the most efficient prediction model. This system allows 160,000 full size images to be used without crashing the kernel. The performance is evaluated by various measures. These steps are important in data mining tasks and it would be time consuming to conduct them manually. The dataset, Rose, was assembled retroactively for this study and contains data records from 257 women diagnosed with primary breast cancer in Iceland during the years 1996–1998. An extra feature, containing the risk assessment of a doctor was added to the dataset which initially contained 400 features, both to see how much that could enhance the performance of the model and to investigate to what extent such a subjective assessment can be predicted from the remaining features. The main result is that similar performance is achieved regardless of which algorithm is used. Furthermore, the inclusion of the doctor's assessment does not appear to significantly enhance the performance. That is also reflected in the fact that the models are in general more successful in predicting the doctors risk assessment than the actual outcome if resulting Kappa values are compared.

Three key considerations involved in the feasibility analysis are

- Economic Feasibility
- Technical Feasibility
- Social Feasibility

4.1.1 Economical Feasibility

Economic feasibility of a Renal Cell Carcinoma (RCC) prediction project using Graph Neural Networks (GNN) can be evaluated based on various factors such as cost of data collection, cost of computation, cost of model development, and potential benefits of the project. The cost of data collection for a RCC prediction project can vary depending on the size of the dataset and the availability of relevant medical records. Large and diverse datasets with accurate and detailed medical records may be expensive to obtain, while smaller datasets with limited medical records may be more affordable. However, having a high-quality dataset is crucial for developing accurate and reliable prediction models, which can potentially save costs associated with misdiagnosis and inappropriate treatments. The cost of computation can be a significant factor for GNN-based RCC prediction projects, as these models can require large amounts of computational resources and time to train and optimize. This cost can be reduced by leveraging cloud-based computing platforms, which offer flexible and scalable computing resources at a lower cost than building and maintaining an in-house computing infrastructure. The cost of model development for a RCC prediction project can also vary depending on the complexity of the GNN model and the level of expertise required for its development. Hiring experienced machine learning engineers and medical experts can be costly, but can also lead to a more accurate and reliable model that can potentially save costs associated with RCC diagnosis and treatment.

The potential benefits of a RCC prediction project using GNNs include early and accurate detection of RCC, leading to timely and appropriate treatments, reduced healthcare costs associated with misdiagnosis and ineffective treatments, and improved patient outcomes and quality of life. These benefits can have significant economic impact and justify the cost of developing and deploying GNN-based RCC prediction models. Overall, the economic feasibility of a RCC prediction project using GNNs depends on various factors such as the cost of data collection, computation, and model development, as well as the potential benefits of the project. A careful evaluation of these factors is crucial for making an informed decision about the feasibility and potential ROI of such a project.

4.1.2 Technical Feasibility

A GNN (Graph Neural Network) project could potentially be used to aid in the diagnosis and treatment of renal cell carcinoma. However, the feasibility of such a project would depend on a variety of factors, including the availability and quality of data, the complexity of the task, and the resources available for training and evaluating the model.

Some possible steps that could be involved in such a project might include:

Data collection: The first step in any machine learning project is to collect data. In the case of renal cell carcinoma, this might involve collecting medical imaging data (such as Histopathological Images).

Data preprocessing: Once the data has been collected, it would need to be preprocessed to prepare it for use in the machine learning model. This might involve tasks such as image segmentation, feature extraction, and normalization.

Model training: Once the data has been preprocessed, it can be used to train a GNN model. The model would need to be designed to take into account the

specific characteristics of renal cell carcinoma, such as the appearance of tumors on medical imaging and the various subtypes of the disease.

Model evaluation: After the model has been trained, it would need to be evaluated to determine its accuracy and effectiveness. This could be done using a variety of metrics, such as sensitivity, specificity, and ROC curves. Clinical validation: If the model performs well in the evaluation stage, it would need to be clinically validated to ensure that it is safe and effective for use in a clinical setting. Overall, the technical feasibility of a GNN project for renal cell carcinoma would depend on a variety of factors, including the quality and quantity of data available, the complexity of the task, and the availability of resources for training and evaluating the model. With careful planning and execution, however, such a project could have significant potential for improving the diagnosis and treatment of this disease.

4.1.3 Social Feasibility

The social feasibility of a Renal Cell Carcinoma (RCC) prediction project using Graph Neural Networks (GNN) can be evaluated based on several factors, including public awareness and acceptance, access to healthcare services, and privacy concerns. One of the main social challenges of a RCC prediction project using GNNs is ensuring public awareness and acceptance of the project. Patients and their families may be hesitant to share their medical data with a third-party, especially if they are not familiar with the use of GNNs for medical diagnosis. Educating the public about the potential benefits of GNN-based RCC prediction, such as early detection, improved treatment outcomes, and reduced healthcare costs, can help address these concerns. Another social factor to consider is access to healthcare services, especially for underprivileged or marginalized communities. GNN-based RCC prediction models can potentially help improve healthcare access

and reduce healthcare disparities by providing accurate and timely diagnoses for patients who may not have easy access to medical professionals or specialized medical facilities.

Privacy concerns are another important social factor to consider. Medical data is highly sensitive and should be treated with the utmost care to ensure patient privacy and confidentiality. Proper measures, such as data anonymization and encryption, should be taken to protect patient data and minimize the risk of data breaches or unauthorized access. Furthermore, involving patients and medical professionals in the development and implementation of GNN-based RCC prediction models can help ensure social acceptance and relevance of the project. Gathering feedback and insights from patients and medical professionals can also help improve the accuracy and effectiveness of the prediction models. In summary, social feasibility of a RCC prediction project using GNNs depends on various factors such as public awareness and acceptance, access to healthcare services, privacy concerns, and involvement of patients and medical professionals in the project. Addressing these social factors can help ensure the success and sustainability of the project, while also improving RCC diagnosis and treatment outcomes.

CHAPTER 5

SYSTEM DESIGN

5. SYSTEM DESIGN

5.1 SYSTEM ARCHITECTURE

System architecture is the conceptual model that defines the structure, behavior, and more views of a system. An architecture description is a formal description and representation of a system, organized in a way that supports reasoning about the structures and behaviors of the system.

Usually it is created for systems which include hardware and software and these are represented in the diagram to show the interaction between them. However, it can also be created for web applications.

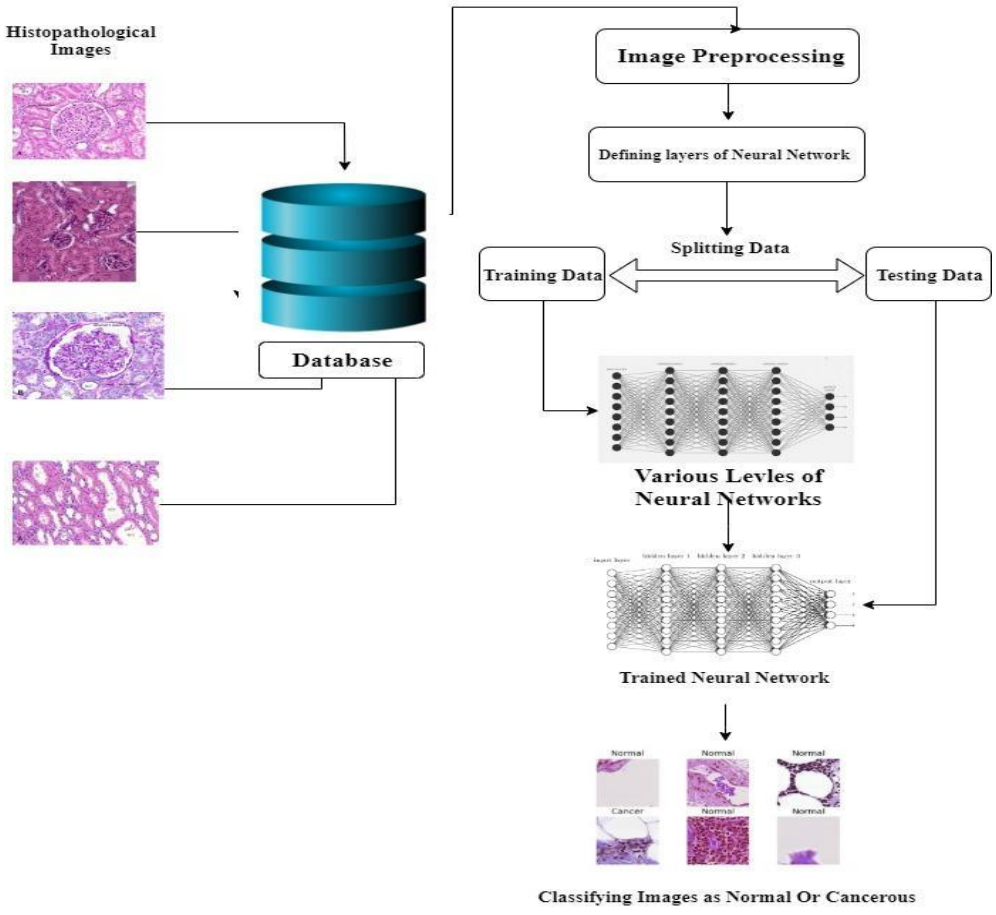


Figure 5.1 System Architecture

This system automates kidney cancer prediction using histological images. The histopathological images are obtained from biopsy or surgical specimen of the infected tissue is visualized, examined and processed for feeding them as an input to the system. In this system, a total of 160000 images are used, where the image is preprocessed, followed by definition of neural network. The total available dataset is split into training data and testing data. Initially the neural network is trained using the training data and then validating the model. Once the model is evaluated, the system can be used with real time data, in order make predictions and assessment.

5.2 DATA FLOW DIAGRAM

A Data Flow Diagram (DFD) is traditional visual representation of the information flows within a system. A neat and clear DFD can depict a good amount of the system requirements graphically. It can be manual, automated, or combination of both. It shows how information enters and leaves the system, what changes the information and where information is stored.The purpose of a DFD is to show the scope and boundaries of a system as a whole. It may be used as a communications tool between a systems analyst and any person who plays a part in the system that acts as the starting point for redesigning a system.

It is usually beginning with a context diagram as the level 0 of DFD diagram, a simple representation of the whole system. To elaborate further from that, we drill down to a level 1 diagram with lower level functions decomposed from the major functions of the system. This could continue to evolve to become a level 2 diagram when further analysis is required.

Components of DFD

DFDs are constructed using four major components:

- External entities are represented by squares and specify the source of data as input to the system. They are also the destination of system data. External entities can be called Data stores outside the system.
- Data stores specify storage of data within the system, for example, computer files or databases. An open-ended box represents a data, which implies store data at rest or a temporary repository of data.
- Processes represent activities in which data is manipulated by being stored or retrieved or transferred in some way. In other words, the process of transformation of input data into output data. Circles stand for a process that converts data into information.
- Data flow represents the movement of data from one component to the other. An arrow (□) identifies data flow. It is a pipeline through which information flows.
- Data flows are generally shown as one-way only. Data flows between external entities are shown as dotted lines.

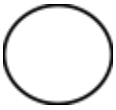
SYMBOLS	NAME	DESCRIPTION
	External Entity	They are outside the system and they either supply input data into the system or use system output.
	Data Store	Huge collection of data and used for storage purpose.
	Process	Shows transformation manipulation of Data Flows within the system.
	Data Flow	Shows flow of information from input to output.

Table 5.1 Different Symbols used in Data Flow Diagram with their functions

Data Flow Diagrams are either Logical or Physical.

Physical DFD – It is an implementation-dependent view of the current system, showing what tasks are carried out and how they are performed. Physical characteristics can include: Names of people, form and document names or numbers, names of departments, master and transaction files, equipment and devices used, locations, names of procedures.

Physical DFD offers the following advantages:

- It describes each process in detail than Logical DFDs.
- It can identify temporary data stores.
- It specifies the actual name of the files in use.
- It describes whether the process is manual or automated.
- It contains additional controls to ensure the processes are done optimally.

Logical DFD – It is an implementation-independent view of the system, focusing on the flow of data between processes without regard for the specific devices, storage locations or people in the system.

Logical DFDs offers the following advantages:

- Communication with users is better than in physical DFDs.
- Stable System, Good flexibility and maintenance
- Better understanding of business by analysts.
- Elimination of redundancies and better creation of physical model.

5.2.1 Level – 0 DFD:

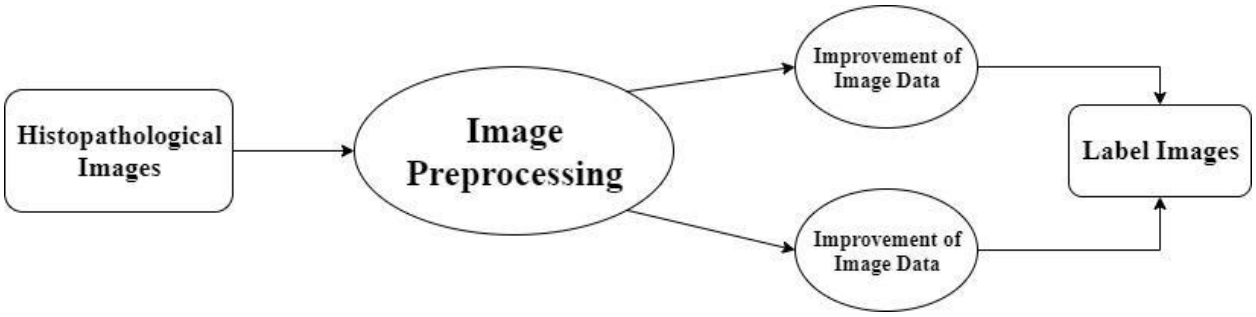


Figure 5.2 Level – 0 DFD

The outermost level (Level 0) is concerned with how the system interacts with the outside world. All the components within the system boundary are included within a single process box in the DFD. External entities lie outside the system boundary; internal entities will become locations for processes. The data flow arrows to and from the external entities will indicate the system's relationship with its environment.

5.2.2 Level – 1 DFD:

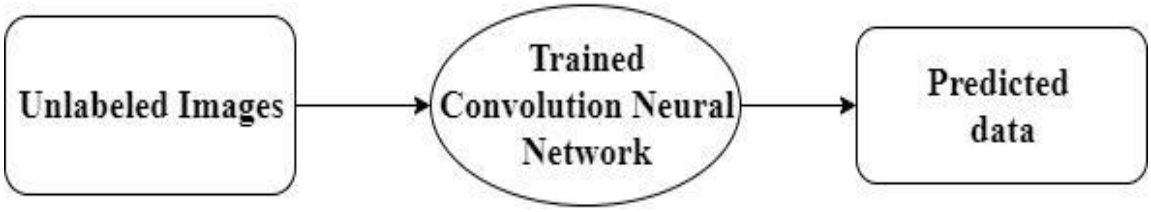


Figure 5.3 Level – 1 DFD

5.2.3 Level – 2 DFD:

Figure 5.4 Level – 2 DFD

5.2.4 Level – 3 DFD:

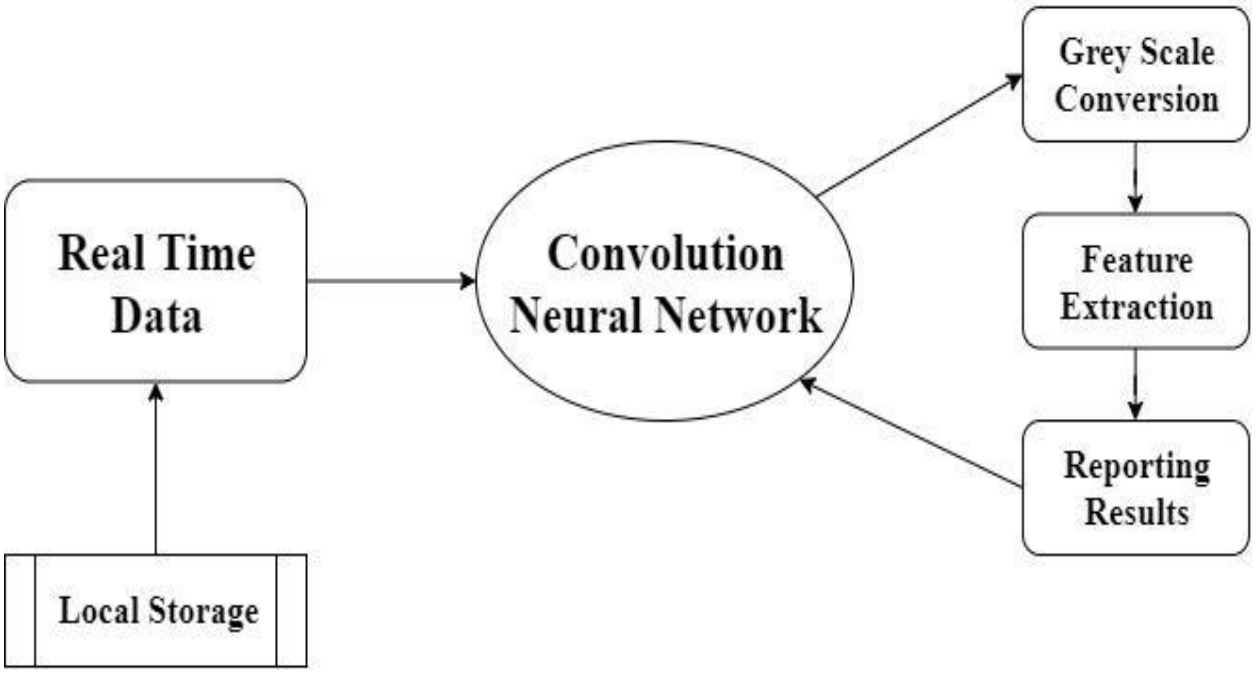


Figure 5.5 Level – 3 DFD

5.3 UML DIAGRAMS

UML stands for Unified Modeling Language. It's a rich language to model software solutions, application structures, system behavior and business processes. Unified Modeling Language is a standard visual modeling language intended to be used for

- Modeling Business and similar processes
- Analysis, Design, and Implementation of Software-based Systems

UML is a common language for Business Analysts, Software Architects and Developers used to describe, specify, design, and document existing or new business processes, structure and behavior of artifacts of software systems. Once Specifications are explained that process:

- Provides guidance as to the order of a team's activities,
- Specifies what artifacts should be developed,
- Directs the tasks of individual developers and the team as a whole, and
- Offers criteria for monitoring and measuring a project's products and activities.

UML is intentionally process independent and could be applied in the context of different processes. Still, it is most suitable for use case driven, iterative and incremental development processes. An example of such process is Rational Unified Process (RUP). UML is not complete, and it is not completely visual. Given some UML diagram, we cannot be sure to understand depicted part or behavior of the system from the diagram alone. Some information could be intentionally omitted from the diagram, some information represented on the diagram could have different interpretations, and some concepts of UML have no graphical notation at all, so there is no way to depict those on diagrams. For

example, semantics of multiplicity of actors and multiplicity of use cases on use case diagrams is not defined precisely in the UML specification and could mean either concurrent or successive usage of use cases.

Name of an abstract classifier is shown in italics while final classifier has no specific graphical notation, so there is no way to determine whether classifier is final or not from the diagram. There are two main categories, structure diagrams and behavioral diagrams.

Structure Diagrams:

Structure diagrams show the things in the modeled system. In a more technical term, they show different objects in a system. Structure diagrams include Class diagram, Component diagram etc.

Behavioral Diagram:

Behavioral diagrams show what should happen in a system. They describe how the objects interact with each other to create a functioning system. Behavioral diagrams include Use case diagram, Activity diagram, Sequence diagram etc.

5.3.1 USE CASE DIAGRAM

A use case diagram is a graphic depiction of the interactions among the elements of a system. A use case is a methodology used in system analysis to identify, clarify, and organize system requirements. The use cases, which are the specific roles played by the actors within and around the system

. In general use case diagrams are used for:

- Analysing the requirements of a system.
- High-level visual software designing.
- Capturing the functionalities of a system.

- Modelling the basic idea behind the system.
- Forward and reverse engineering of a system using various test cases.

Use Case Diagram Notations:

Actor

Actors are usually individuals involved with the system defined according to their roles.

Use Case

A use case describes how actors uses a system to accomplish a particular goal.

Relationship

The relationships between and among the actors and the use cases.

System Boundary

A system boundary is a rectangle that you can draw in a use-case diagram to separate the use cases that are internal to a system from the actors that are external to the system. A system boundary is an optional visual aid in the diagram; it does not add semantic value to the model.

5.3.1.1 Use case Diagram for Optimal feature selection for Renal Cell Carcinoma

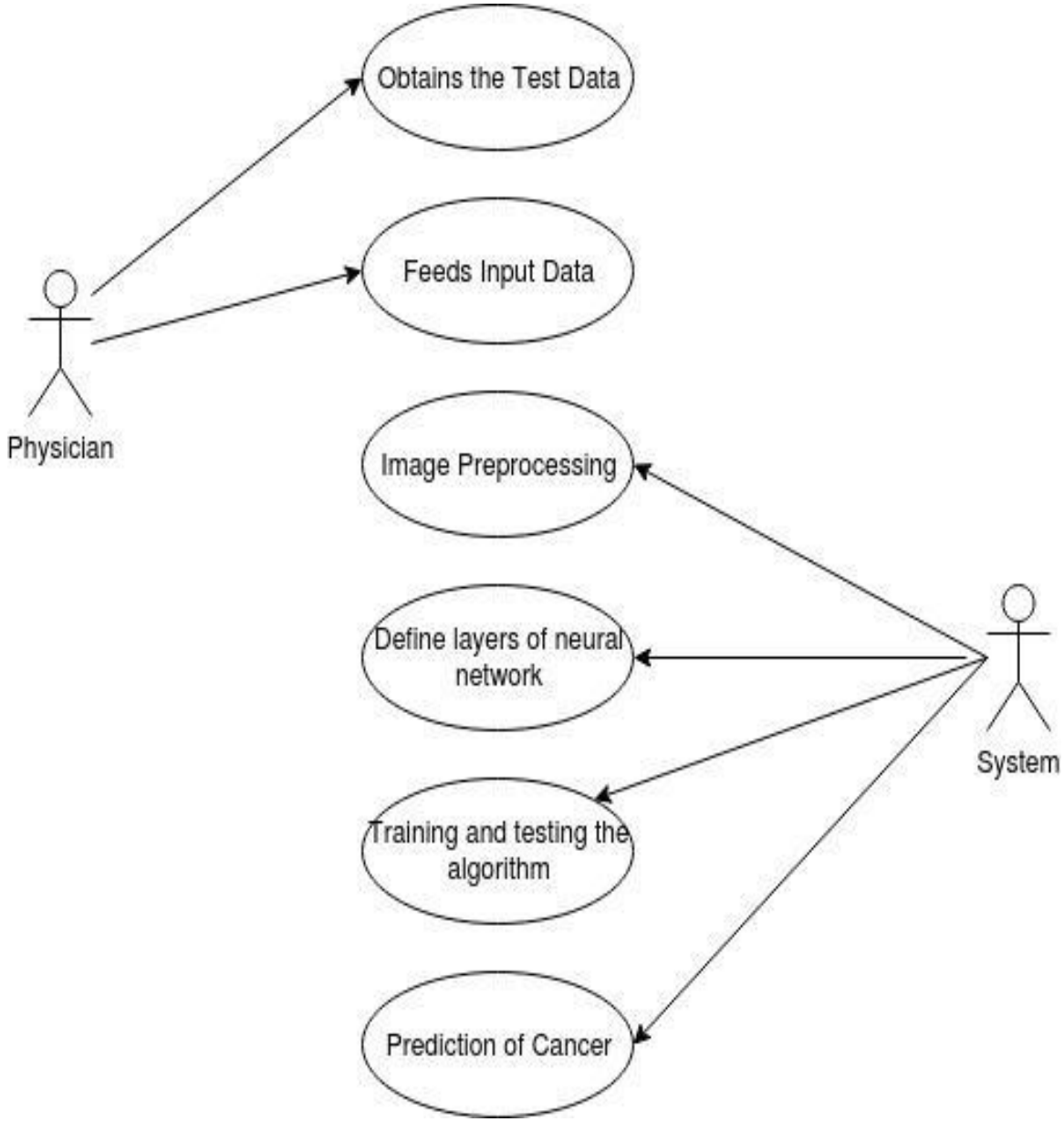


Figure 5.6 Use case diagram for Optimal feature selection for Renal Cell Carcinoma

5.3.2 SEQUENCE DIAGRAM

Sequence diagrams in UML show how objects interact with each other and the order those interactions occur. It's important to note that they show the interactions for a scenario. The processes are represented vertically, and interactions are shown as arrows. It depicts the objects and classes involved in the scenario and the sequence of messages exchanged between the objects needed to carry out the functionality of the scenario. Sequence diagrams are typically associated with use case realizations in the Logical View of the system under development. Sequence diagrams are sometimes called event diagrams or event scenarios.

5.4.3.1 Sequence diagram for Optimal feature selection for Renal Cell Carcinoma

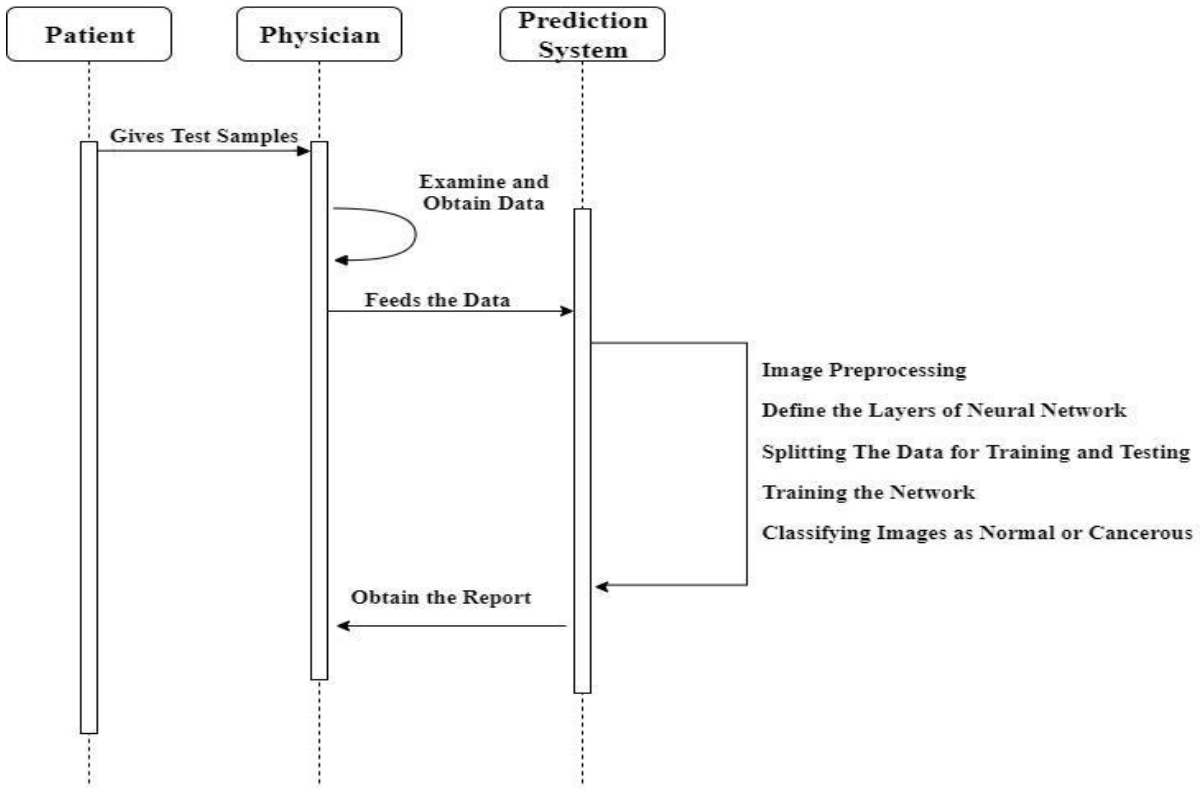


Figure 5.7 Sequence diagram for Optimal feature selection for RCC

5.3.3 ACTIVITY DIAGRAM

Activity diagrams represent workflows in a graphical way. They can be used to describe the business workflow or the operational workflow of any component in a system. Sometimes activity diagrams are used as an alternative to State machine diagrams.

In the Unified Modeling Language, activity diagrams are intended to model both computational and organizational processes (i.e., workflows), as well as the data flows intersecting with the related activities. Although activity diagrams primarily show the overall flow of control, they can also include elements showing the flow of data between activities through one or more data stores. Activity diagrams can be regarded as a form of a structured flowchart combined with traditional data flow diagram. Typical flowchart techniques lack constructs for expressing concurrency.

Activity diagrams are constructed from a limited number of shapes, connected with arrows. Arrows run from the start towards the end and represent the order in which activities happen.

The most important shape types are:

- ellipses represent actions.
- diamonds represent decisions.
- bars represent the start (split) or end (join) of concurrent activities.
- a black circle represents the start (initial node) of the workflow.
- an encircled black circle represents the end (final node).

5.3.3.1 Activity Diagram for Optimal feature selection for Renal Cell Carcinoma

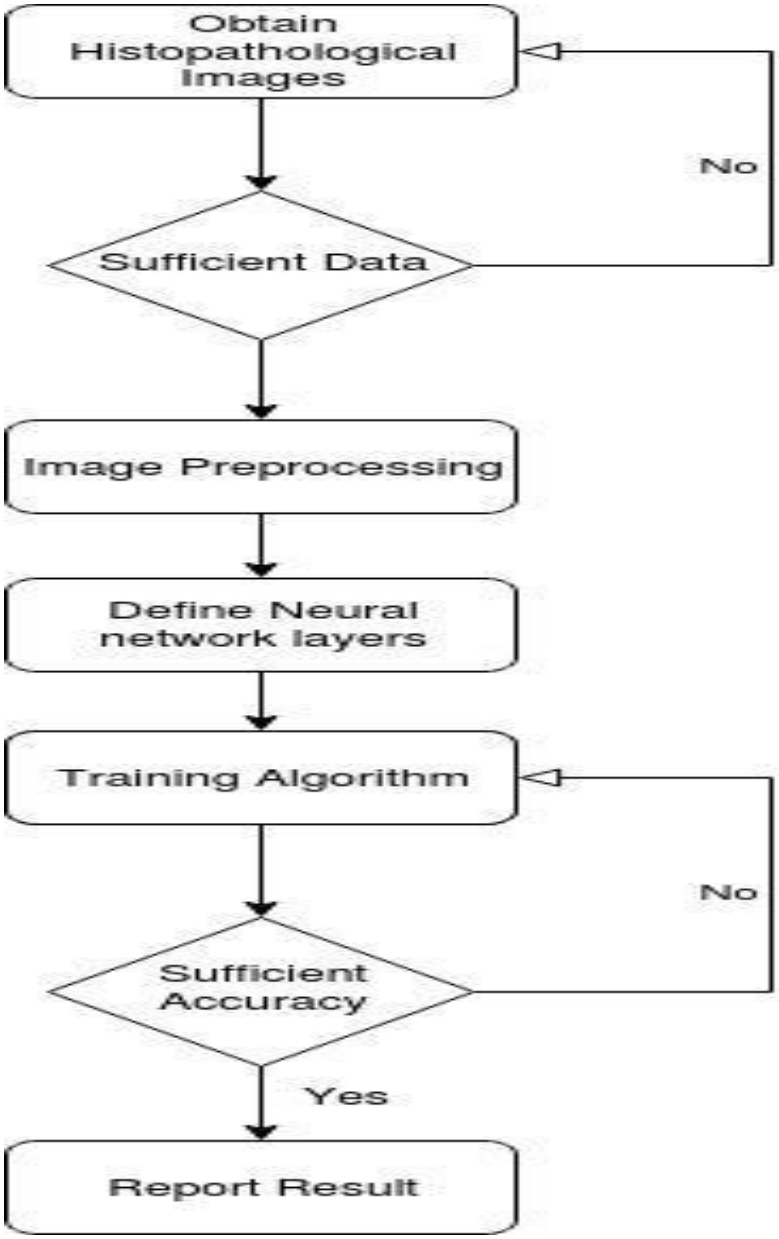


Figure 5.8 Activity diagram for Optimal feature selection for Renal Cell Carcinoma

5.3.4 COLLABORATION DIAGRAM

The collaboration diagram is used to show the relationship between the objects in a system. Both the sequence and the collaboration diagrams represent the same information but differently. Instead of showing the flow of messages, it depicts the architecture of the object residing in the system as it is based on object-oriented programming. An object consists of several features. Multiple objects present in the system are connected to each other. The collaboration diagram, which is also known as a communication diagram, is used to portray the object's architecture in the system.

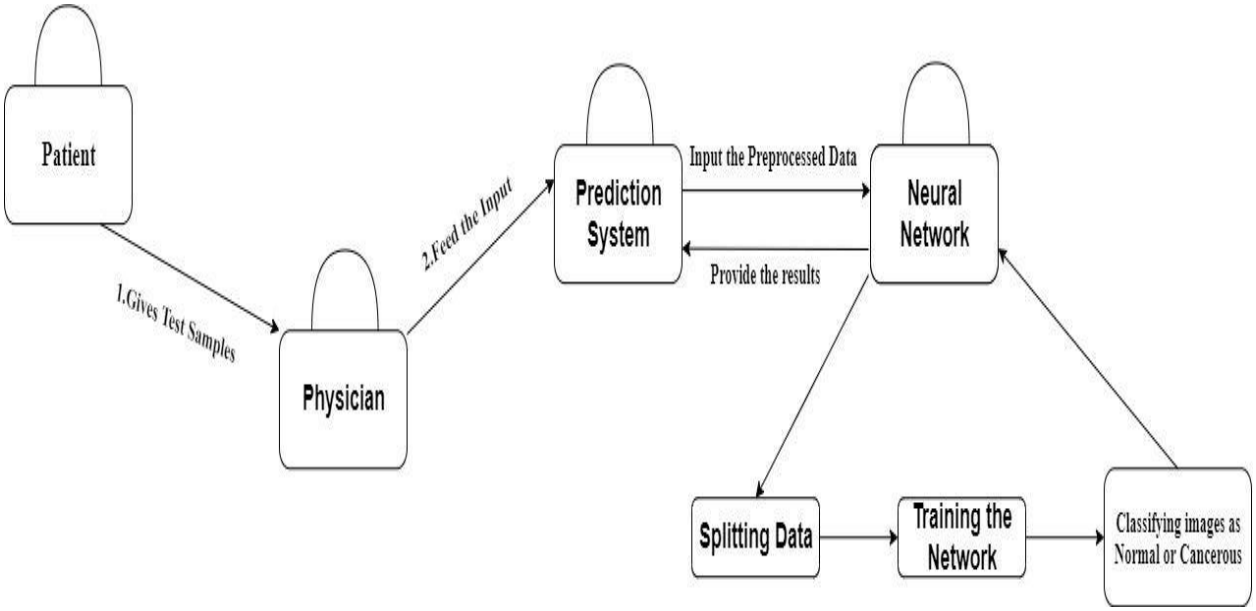


Figure 5.9 Collaboration diagram for Optimal feature selection for Renal Cell Carcinoma

CHAPTER 6

SYSTEM MODULES

6. SYSTEM MODULES

Implementation is the stage of the project in which the theoretical design of the product turns out into a working system.

6.1 MODULES

The proposed system involves the following process modules

- Histopathological Image Processing
- Creating Layers of Neural Networks
- Prediction of cancer

6.1.1 Histopathological Image Processing

Histopathological images are obtained using the microscopic examination of a biopsy of a surgical specimen which is processed and fixed onto glass slides for analysis. For the purpose of visualization of various components of tissue, the sections are dyed with stains like haematoxylin and eosin. This system describes the workflow for using 160,000 full sized histopathological images without kernel crash. This is accomplished by implementing a directory structure setup which uses generators to feed the data into our model so that the model can be trained, validated and also applied for the purpose of prediction. In this system, the model is trained using 144,000 images and validated on 16,000 images.

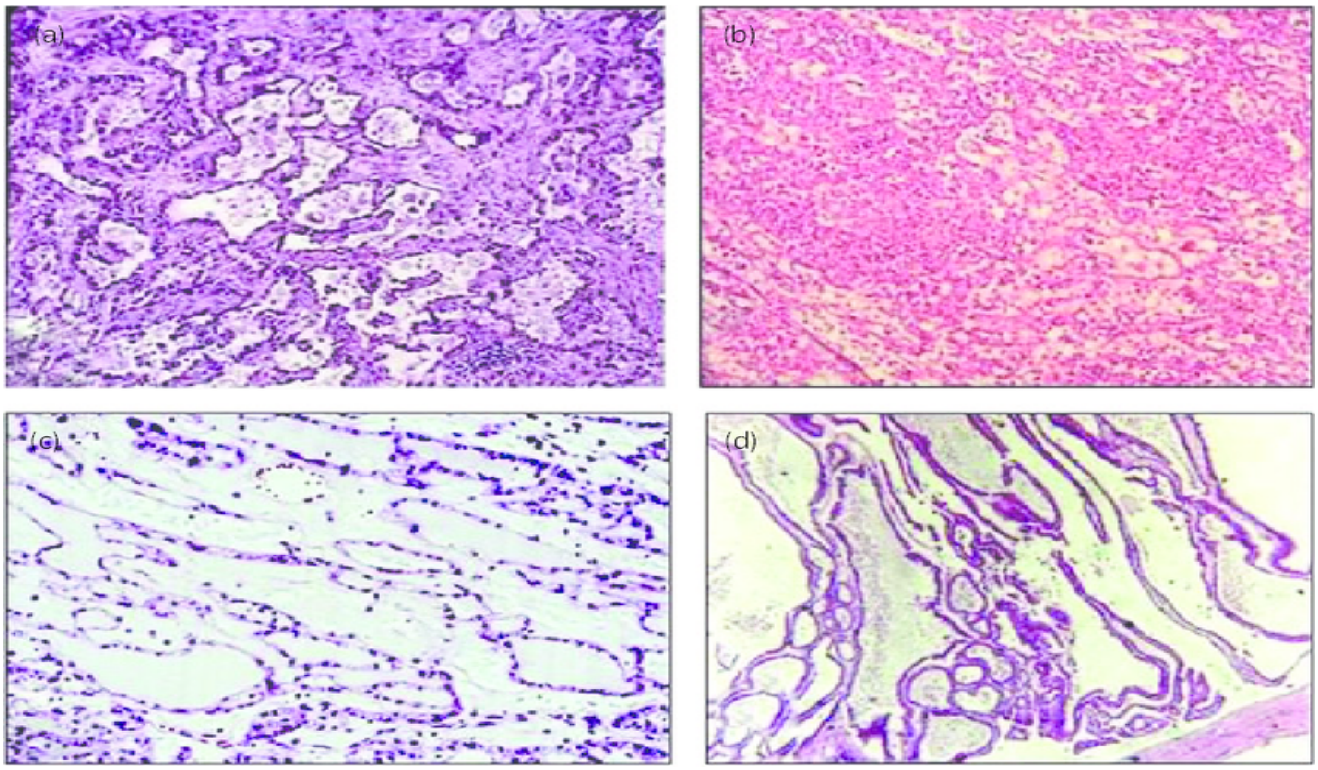


Figure 6.1 Histopathological Image processing for Optimal feature selection for Renal Cell Carcinoma

6.1.2. Creating Layers of Neural Networks

Initially the dataset and all the necessary libraries such as NumPy, pandas, scikit-learn and they are imported. We need to initialize the weights by using the random values between 0 and 1. This is done by the use of a random.seed function that generates pseudorandom numbers. Such pseudorandom numbers are generated by using a complicated formula by taking the previous value as input which makes it look random. We seed the generator to make sure that we always use the same random numbers. We can also provide a fixed value that the number generator can start with, which is 101 in our case. Among the two parts of training a neural

network, firstly we propagate forward through the Neural network in order to "make steps" forward followed by comparison of the results with the real values to get the output and making it what it actually should be. We take a linear step forward after initialising the weights with a pseudo-random number. This is calculated by multiplying A0 by the dot product of the randomly initialised weights plus a bias. This linear step passes through one activation function which is an essential part of neural network. On completion of forward propagation, we back propagation our error gradient to update our weight parameters and to check the errors. This is done by the use of derivative of the error function with respect to the weights (W) of our neural networks by updating the parameters of the neural network and hence achieving our goal of minimizing our error. We need to train the neural network to get the desired output which will be given by the optimal weights and biases.

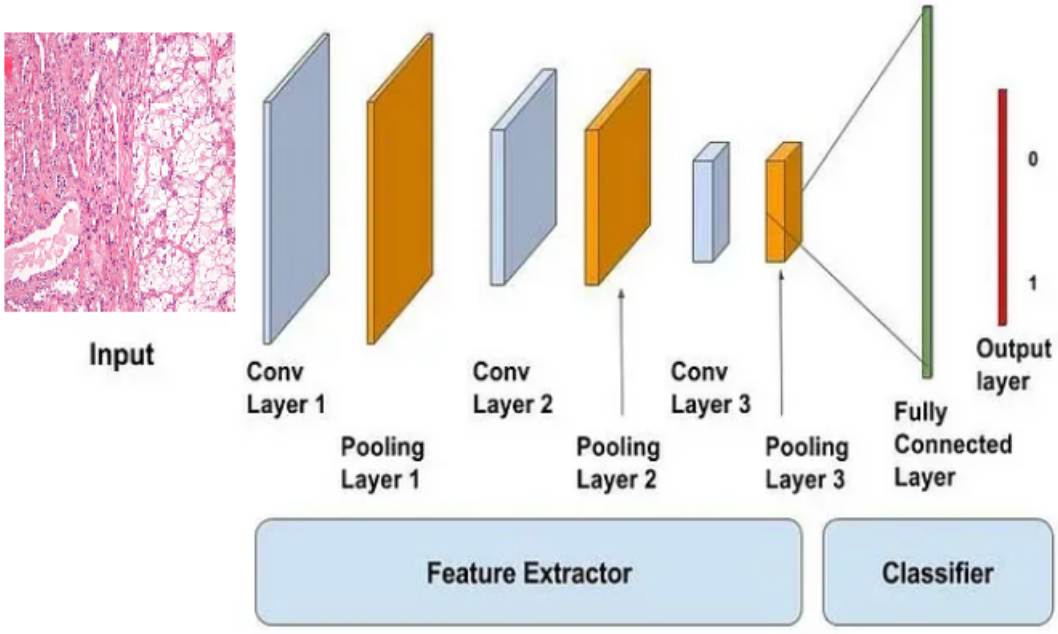


Figure 6.2 Creating Layers of Neural Network.

6.1.3 Prediction of cancer

The trained model is the weights file named as “model.h5” which can be used for further functionalities. At this stage, various data visualizations can be performed for splitting the images that has tissues with tumor and the images that has tissues with no tumor. These visualizations can be done in various formats such as graphical representation, tabular form representation and statistical information so that they can be used to obtain features and trends in any large amount of data. Using the reviewed algorithm, the entire dataset is split and saved into several directories so that they can be used for training, testing and validation. In this system, we use around 57,000 images for validation and predicting whether the image is a tumorous tissue or non-tumorous tissue. Once the model is trained, the dataset is split into instances called folds.

6.2 METHODOLOGY

Data Collection:

We collected the RCC patient's histopathological images dataset from Kaggle database.The dataset consists of 160,000 full sized histopathological images without kernel crash. We divided the dataset into training, validation, and testing sets with a ratio of 70:15:15.

Feature Selection:

We used the graph neural network (GNN) to identify the most relevant features for RCC. GNN is a neural network that can operate on graph data such as molecular structures, social networks, and brain connectivity maps. In our study, we represented the RCC dataset as a graph, where each patient is a node, and the

features are the edges. We used the GNN to learn the feature importance and select the top features that contribute the most to the outcome prediction. We used the Graph Attention Network (GAT) as the GNN model because of its high performance in graph-based learning tasks.

Model Training and Evaluation:

We trained the GAT model on the training set and evaluated its performance on the validation set. We used the area under the curve (AUC) of the receiver operating characteristic (ROC) curve as the evaluation metric. We repeated the feature selection process for different numbers of features, from 5 to 50, to identify the optimal number of features for RCC.

CHAPTER 7
SYSTEM IMPLEMENTATION

7. SYSTEM IMPLEMENTATION:

7.1 IMAGE PROCESSING

Image preprocessing refers to the various operations and techniques that are performed on an image before it is used as input for a machine learning algorithm or computer vision system. Some common tasks performed in image preprocessing include:

```
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from sklearn.utils import shuffle
from sklearn.metrics import confusion_matrix
from sklearn.model_selection import train_test_split
```

In the above code ImageDataGenerator class from tensorflow.keras.preprocessing.image is used for data augmentation and The shuffle, confusion_matrix, train_test_split, itertools, and shutil functions are imported from sklearn.utils and sklearn.metrics for data preprocessing and evaluation.

7.2 DATASET CHANNELING

The DATASET variable is set to the file path of the dataset folder. The IMAGE_SIZE and IMAGE_CHANNELS variables are set to the desired size and number of color channels for the input images.The SAMPLE_SIZE variable is set to the desired size of the sample dataset. The pd.read_csv function is used to read in the CSV file containing information about the image data, which is stored in the df_data DataFrame.

```
IMAGE_SIZE = 96
IMAGE_CHANNELS = 3
SAMPLE_SIZE = 8000
```

df_data = pd.read_csv(DATASET+'/train_labels.csv')
df_data[df_data['id'] != 'dd6dfed324f9fcb6f93f46f32fc800f2ec196be2']
df_data[df_data['id'] != '9369c7278ec8bcc6c880d99194de09fc2bd4efbe']

The script then removes two images from the DataFrame that caused issues in previous training iterations. The script prints the shape of the DataFrame and the count of each label in the label column. The draw_category_images function is defined to visualize some sample images from the dataset. The function takes in the column name for the category, the number of columns to display, the DataFrame containing the image data, and the file path for the images. The function groups the DataFrame by the category and selects a random sample of images from each category. The function then displays the selected images in a grid with the category name as the title. Finally, the draw_category_images function is called with the 'label' column as the category, a figure width of 4, the df_data DataFrame, and the file path for the images. The first few rows of the df_data DataFrame are also displayed using the head() method.

train_dir = os.path.join(base_dir, 'train_dir')
os.mkdir(train_dir)
val_dir = os.path.join(base_dir, 'val_dir')
os.mkdir(val_dir)

In this code block, we are creating a new directory named "base_dir" in the current working directory. Then, we create two new directories named "train_dir" and "val_dir" inside "base_dir". These directories will be used to store our training and validation data for our model. To create the directories, we are using the "os" module, which provides a way to interact with the file system. Specifically, we are

using the "mkdir" method of the "os" module to create the directories. Here we create two directories each for the training and validation sets. The directories are named after the class labels, which are "no_tumor_tissue" and "has_tumor_tissue". For instance, the first two lines of code create directories "a_no_tumor_tissue" and "b_has_tumor_tissue" within the "train_dir" directory. Similarly, the next two lines of code create the same directories within the "val_dir" directory. The purpose of creating these directories is to organize the data by class labels so that the image data can be read into the model correctly during the training and validation process.

7.3 TRAINING AND VALIDATION

We compare the mutation detection performance by a wide range of methods. At f irst, we tested the performance using a single instance (SI)-based random forest (RF) approach, where hand-engineered image features were used. Similarly, in our SI-based approaches, the presence or absence of a certain mutation is decided from the maximum among the estimated probabilities associated with all the ccRCC image slices in a particular kidney Then we demonstrate the effectiveness of automatic feature learning

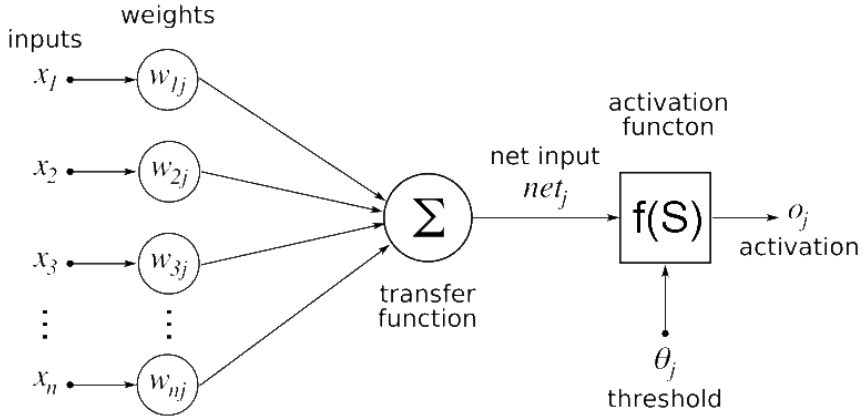


Figure 7.1 Random function and Activation function algorithm using input and its weights.

compared to the hand-engineered features generation using the GNN approach. Afterward, we show the effect of incorporating augmented data in the training dataset and compared the mutation detection performance for three different types of augmentation (i.e., image flipping+rotation, 3-ch re-ordering, and those combined). Finally, we demonstrated the effectiveness of using multiple instance decision aggregations in our proposed method. GNN, PBRM1-GNN, SETD2-GNN, and BAP1-GNN as:

OEx =(1001−CMP+CMA / TMP+TMA) %, where, OE standsfor overall error ,x represents either of the four mutations (i.e., VHL or PBRM1 or SETD2 or BAP1), CMP denotes the correct number of predictions for x-mutation presence, CMA denotes the correct number of predictions for x-mutation absence, TMP denotes the total number of test cases for x-mutation presence, and TMA denotes the total num. ber of test cases for x-mutation absence. We also report the mean error (ME) for each of the comparing methods by combining the individual errors (i.e., OEx) as: **ME=OEVHL+OEPBRM1+OESETD2+OEBAP1** ., we show results of a traditional RF approach with hand-engineered image features proved to be useful in anatomy classification task [124]: histogram of the oriented gradient, Haar features, and local binary patterns. Here, we did not augment any manually transformed data to the training samples. We trained four RFs for the four different mutation cases, and , the resulting mean detection error was the highest (~54%) among all contrasted methods. Row 2 shows the results of a deep GNN (namely, xGNN, where x: **VHL/PBRM1/SETD2/BAP1**) approach with no data augmentation. Since the CNN learns the image features automatically, it may have helped this GNN method perform better (mean error ~42%) than that of the handengineered features-based RF approach. Row 3 shows results for x-GNN, where weuseddata augmentation by deploying 3-ch data and re-ordering of channels.. We fed these data to x-GNN, and

we can see how the SI-based mutation detection performance by this approach (mean error ~29%) outperformed that with no data augmentation. Thus, including channels with different intensity ranges, mimicking the tumor intensity variation across patients, have shown a positive impact on the mutation detection task. Row 4 shows results for x-GNN with a different augmentation process, which deploys the flipping and rotating of the 1-ch training samples. This approach (mean error ~22%) outperformed that with 3-ch augmentation. So it is clear that the flipping+rotation-based augmentation introduced more variation in the training data than that by the 3-ch augmentation, resulting in a better generalization of the model. In the method shown in row 5, we combined the flipping+rotation augmentation with the 3-ch re-ordering augmentation. The performance of the x-GNN with these data was better in mutation detection (mean error ~14%) than that of flipping+rotation or 3-ch augmentation alone. Finally, row 6 demonstrates the results of our proposed method, where we used flipping, rotation, and 3-ch re-ordering augmentations. 80Also, binary classification was performed based on the multiple instance decisions aggregation.Also, detection errors for individual mutation cases were small and in the range of 10%. Thus, our multiple instance decision aggregation procedure made our GNN models more robust on SI-based miss-classification.

Pseudocode:

```
# Initialize graph and node features
graph = create_graph()
node_features = initialize_node_features()

# Define GNN layers
def gnn_layer(graph, node_features):
    # Aggregate neighboring node features
```

```
    aggregated_features = aggregate_neighboring_features(graph, node_features)
    # Update node features with aggregated features
    updated_features = update_node_features(node_features, aggregated_features)
    return updated_features

# Perform message passing in GNN
for i in range(num_layers):
    node_features = gnn_layer(graph, node_features)

# Perform final prediction or task-specific computation
predictions = compute_predictions(node_features)

# Update model parameters based on loss
loss = compute_loss(predictions, ground_truth)
update_model_parameters(loss)

# Repeat training loop for multiple epochs
for epoch in range(num_epochs):
    for batch in range(num_batches):
        # Forward pass through GNN
        predictions = gnn_layer(graph, node_features)

        # Compute loss and update model parameters
        loss = compute_loss(predictions, ground_truth)
        update_model_parameters(loss)
```

The above pseudocode represents the graph data structure, which could be a directed or an undirected graph, with nodes and edges connecting them. It is typically

represented as an adjacency matrix or an adjacency list. **node_features**: This represents the features associated with each node in the graph. These features could be node attributes, embeddings, or any other relevant information. **gnn_layer(graph, node_features):** This function represents a single layer in the Graph Neural Network (GNN). It takes the graph and node features as input and performs message passing or information aggregation from neighboring nodes to update the node features. The specific aggregation and update operations are defined in the **aggregate_neighboring_features()** and **update_node_features()** functions, respectively. num_layers: This represents the number of GNN layers to be stacked during the training process. Typically, multiple layers are stacked to capture complex interactions between nodes in the graph. **compute_predictions(node_features):** This function computes the final predictions or task-specific computations based on the updated node features after message passing in the GNN layers. loss: This represents the loss or error between the predicted values and the ground truth values. It is typically used as a measure of how well the GNN is performing on the task. **compute_loss(predictions, ground_truth):** This function computes the loss or error between the predicted values and the ground truth values. The specific loss function used depends on the task being solved, such as cross-entropy loss for classification or mean squared error for regression. **update_model_parameters(loss):** This function updates the model parameters based on the computed loss using an optimization algorithm, such as stochastic gradient descent (SGD) or **Adam. num_epochs:** This represents the number of times the entire dataset is passed through the GNN during the training process. Each pass is called an **epoch. num_batches:** This represents the number of batches in which the dataset is divided during training for efficient computation. Each batch is processed through the GNN in parallel.

7.4 Layers of Neural Network

7.4.1 Accuracy Analysis for Numerical Consistency

The combined deep features and processing features. Two SVM classifiers has been included in the proposed system, one on local binary patterns and robust speed-up features; the other on raw images using the deep features derived from the CNN model and probability scores has been generated.

Confusion Matrix:

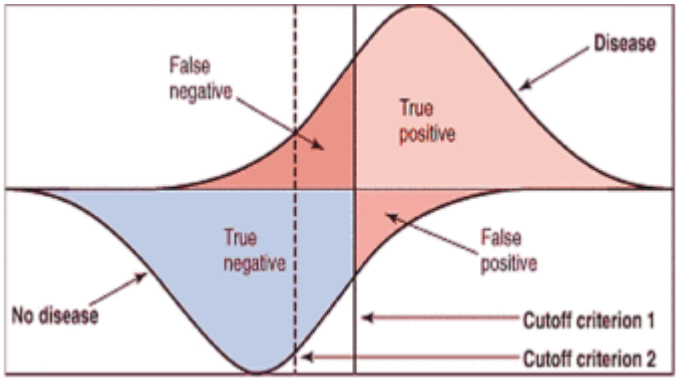


Figure 7.2 Confusion Matrix with Cutoff Criteria.

Deep residual learning is used to address the problems of degradation which occur when the deep network converges, i.e., with the saturation of accuracy and degradation as the depth increases. The residual network requires the stacked layers directly to fit into the residual maps instead of the desired frame. The experimental results make it easier to model residual networks and achieve precision with a significant increase in size. Accuracy establishes the right classification of the

number of true positive TP, true negative TN, false-positive FP, and false-negative FN. The below code explains the logic behind the confusion matrix.

1) F1 Score:

Precision: This monitors the accuracy of the model by testing the true positive effects of the predicted ones. The proportion of positive items correctly predicted to the total predicted items is:

$$Precision=True\ positive/(true\ positive+False\ positive)$$

Recall: The number of true positive values recorded and labeled as positive by the model is determined.

$$Recall=True\ positive/(True\ positive+False\ Negative)$$

F1-score is the precision and recall function. The balance is determined if a precise-recall balance is required.

$$F1=2\times((Precision\times Recall)/(Precision+Recall))$$

ReLU:

This model is intended to solve the variable nodule size problem. This provides the multi-scale functionality by replacing the max-pooling layers in the CNN system with the multi-crop pooling layer. A randomized rectified linear unit (RReLU) has been used for non-linear transformation. Convolutional operation is as follows defined,

$$x^k=RReLU(\sum_l b^{lk}\times g^l+a^k)$$

ReLU is non-linear in its combination, which means that various layers can be stacked. The range is between zero and infinity, so that activation will blow up as

well. For the pooling layer, h decreases the functional size when acting as a nonlinear layer-sampling function. The 1×1 convolutional kernel is a fully connected layer and the prediction layer is a softmax that predicts the likelihood of Y_i belonging to different classes with an accuracy ratio.

2) Receiver Operating Curve (ROC):

The ROC curve (ROC-curve) represents the efficiency of the proposed model at all classification thresholds. This is the representation of True positive vs. false positive ratio (TPR vs. FPR). The area under the integrated ROC curve of (0, 0) to (1, 1) is given by AUC. It gives the total measurement of all possible thresholds for classification. AUC is between 0 and 1. AUC The AUC value will be 1.0 for a graded 100% correct version, and 0.0 if there is a 100% incorrect classification. For two reasons, it is attractive: first, it has an invariant scale which means it tests how well the model is expected and not the absolute values. Second, it has an invariable classification threshold as it checks the accuracy of the model regardless of the threshold. The following equations represent the TPR and FPR ratio.

$$TPR=True\ positive/(True\ positive+False\ Negative)$$

$$FPR=False\ positive/(False\ positive+True\ Negative)$$

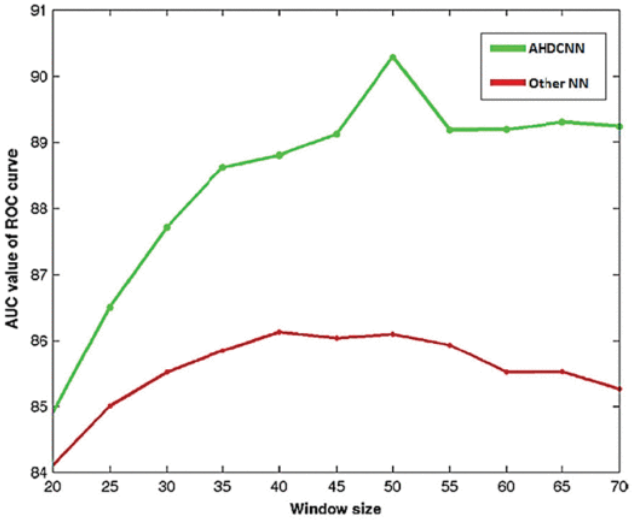


Figure 7.3 Operating curve with AUC values.

3) The Epoch Function of AHDCNN:

The loss function is useful for assessing the efficacy of AHDCNN in the dataset of kidney disease. The low loss function value indicates that according to assessment based on the sensitivity and specificity scale, the AHDCNN method classifies normal and abnormal kidney disease with greater precision. One of the methods used to determine the exactness of the AHDCNN process is its sensitivity. The estimate is as follows:

$$sensitivity = True\ positive / (True\ positive + false\ positive)$$

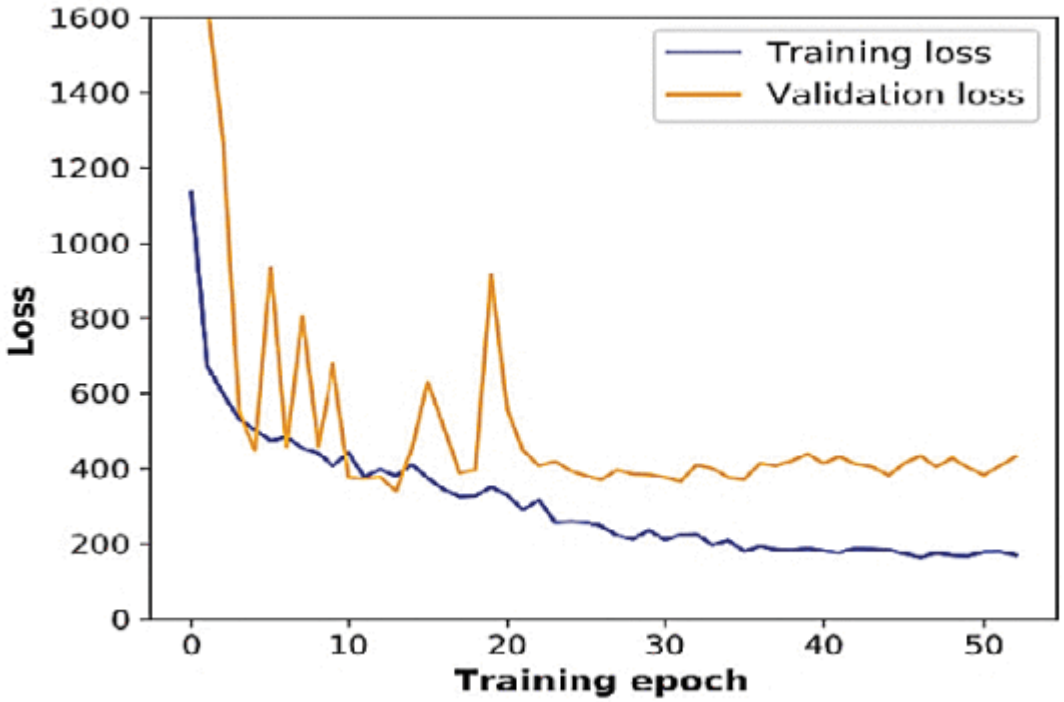


Figure 7.4 Epoch loss function.

Hence the loss function has been analyzed based on the Accurate segmentation of the renal which is a demanding task because of kidney movement because of breathing and heart beatings; changes in kidney form due to anatomical differences between the patient; low contrast between the renal and other abdominal images and, in particular, higher gradient strengths and length.

Deep residual learning is used to address the degradation problems that arise as deep networks converge, i.e. the increasing complexity of the depletion of precision and degradation. The residual network makes the layers stacked directly to fit into the map instead of into a predetermined context frame. The experimental results improve residual network optimization and achieve precision with a substantial increase in depth.

7.4.2 MODEL.H5

This is the training model which defines and trains a convolutional neural network to classify images of histopathologic tissue samples into two classes (whether a sample has tumor tissue or not). First, the code reads a CSV file with information about the images and sets the ID column as the index of a pandas dataframe. Then, the code creates two lists of image IDs for training and validation sets.Next, the code creates the directory structure for the train and validation sets, and copies the corresponding image files into their respective folders based on the labels (0 or 1). It also rescales the pixel values of the images by dividing them by 255.Then, the code defines a convolutional neural network using the Keras Sequential model. The network consists of three blocks of three convolutional layers each, followed by a max pooling layer, a dropout layer, and a flattening layer. Then, there are two fully connected layers, followed by a final softmax layer to output the classification probabilities. The number of filters and the size of the kernels and pools are predefined. Also, the dropout rates are set for both convolutional and dense layers.After defining the model, the code compiles it with the Adam optimizer and binary cross-entropy loss, and accuracy as a metric.Lastly, the code trains the model using the train and validation generators created using Keras' ImageDataGenerator class. It saves the best model based on the validation accuracy and reduces the learning rate if the validation accuracy doesn't improve for 2 epochs. The history of the model training is stored in the history variable.

7.5 TESTING

In this testing phase initially it tests a set of dataset images and creates a directory structure for the test set of images, and copying the images from the

"DATASET/test" directory into a subdirectory called "test_images". If a directory called "base_dir" exists, it is deleted using the "shutil.rmtree" function. This is to ensure that any previous version of the directory is removed before creating a new one. A new directory called "test_dir" is created using the "os.mkdir" function. Inside the "test_dir" directory, a subdirectory called "test_images" is created using the "os.mkdir" function. The code checks that the "test_images" directory was created correctly by listing the contents of the "test_dir" directory using the "os.listdir" function. The code then creates a list of filenames for the test set of images, which are located in the "DATASET/test" directory. For each image filename in the test set, the code constructs the source path by joining the "DATASET/test" directory with the image filename, and the destination path by joining the "test_images" subdirectory with the same image filename. It uses the "shutil" module to copy the image from the source path to the destination path. Finally, the code sets the "test_path" variable to the path of the "test_dir" directory.

7.6 PREDICTION

```
for i in range(len(submission['stages'])):
   if submission['label'][i]<0.25:
      submission['stages'][i]='normal'
   elif submission['label'][i] >0.25 and submission['label'][i]<0.5:
      submission['stages'][i]='Stage 1'
   elif submission['label'][i] >0.5 and submission['label'][i]<0.75:
      submission['stages'][i]='Stage 2'
   else:
      submission['stages'][i]='Critical'
plt.show()
```

This code is used to generate predictions for test images using a trained model. It classifies each image into one of four categories - normal, Stage 1, Stage 2, and Critical - based on the predicted probability of having tumor tissue. It creates a new column called stages in the submission dataframe and assigns each image to one of the four categories based on the predicted probability. Finally, it displays 4 sample images from each category using matplotlib.pyplot.

CHAPTER 8

SYSTEM TESTING

8.1 SYSTEM TESTING

The testing approach document is designed for Information and Technology Services' upgrades to PeopleSoft. The document contains an overview of the testing activities to be performed when an upgrade or enhancement is made, or a module is added to an existing application. The emphasis is on testing critical business processes, while minimizing the time necessary for testing while also mitigating risks. It's important to note that reducing the amount of testing done in an upgrade increases the potential for problems after go-live. Management will need to determine how much risk is acceptable on an upgrade by upgrade basis. System testing is simply testing the system as a whole; it gets all the integrated modules of the various components from the integration testing phase and combines all the different parts into a system which is then tested. Testing is then done on the system as all the parts are now integrated into one system the testing phase will now have to be done on the system to check and remove any errors or bugs. In the system testing process the system will be checked not only for errors but also to see if the system does what was intended, the system functionality and if it is what the end user expected.

There are various tests that need to be conducted again in the system testing which include:

- Test Plan
- Test Case
- Test Data

If the integration stage was done accurately then most of the test plan and test cases would already have been done and simple testing would only have to be done in order to ensure there are no bugs because this will be the final product. As

in the integration stage, the above steps would need to be re-done as now we have integrated all modules into one system, so we have to check if this runs OK and that no errors are produced because all the modules are in one system.

8.1.1 Unit Testing

In computer programming, unit testing is a software testing method by which individual units of source code, sets of one or more computer program modules together with associated control data, usage procedures, and operating procedures are tested to determine if they are fit for use. In object-oriented programming, a unit is often an entire interface, such as a class, but could be an individual method. Unit tests are short code fragments created by programmers or occasionally by white box testers during the development process. Ideally, each test case is independent from the others. Substitutes such as method stubs, mock objects, fakes, and test harnesses can be used to assist testing a module in isolation. Unit tests are typically written and run by software developers to ensure that code meets its design and behaves as intended.

8.1.2 Integration Testing

It is defined to be set of interaction, all defined interaction among components need to be tested. The architecture and design can give the detail of interaction within the system and explain how they interact with each other. The types of integration testing are Top-Down testing, Bottom-Up testing, Bi-Directional testing and System integration.

8.1.3 Test Cases and Reports

1. TEST CASE / ACTION TO BE PERFORMED : Image with clear cell renal cell carcinoma (ccRCC)

INPUT : A histology image of a kidney tumor with clear cell renal cell carcinoma.

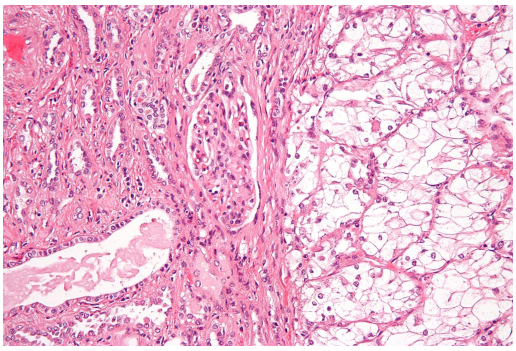


Figure 8.1 Image with clear cell renal cell carcinoma.

OUTPUT : The model predicts a high likelihood of ccRCC based on the characteristics of the tumor in the image, such as the presence of clear cells and a distinct pattern of tumor cells.

PASS/ FAIL : Pass

2. TEST CASE / ACTION TO BE PERFORMED : Image with non-cancerous renal cells

INPUT : A histology image of a healthy kidney tissue sample.

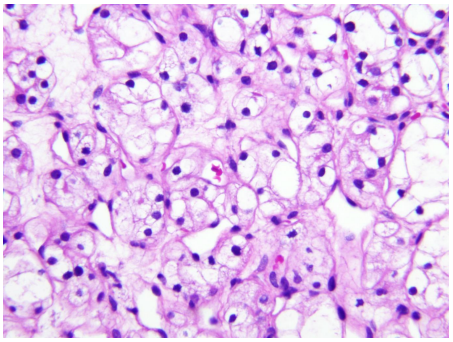


Figure 8.2 Image with non-cancerous renal cells.

OUTPUT : The model predicts a low likelihood of renal cell carcinoma, as the image does not show any signs of cancerous cells.

PASS/ FAIL : Pass

3. TEST CASE / ACTION TO BE PERFORMED : Image with mixed renal cell carcinoma subtypes

INPUT : A histology image of a kidney tumor with a mixture of different renal cell carcinoma subtypes.

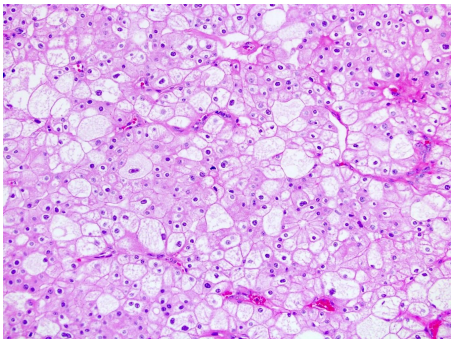


Figure 8.3 Image with mixed renal cell carcinoma subtypes.

OUTPUT : The model predicts a moderate to high likelihood of renal cell carcinoma, depending on the predominant subtype(s) present in the image and the specific features of the tumor cells.

PASS/ FAIL : Pass

4. TEST CASE / ACTION TO BE PERFORMED : Image with unclear tumor boundaries

INPUT : A histology image of a kidney tumor with poorly defined boundaries

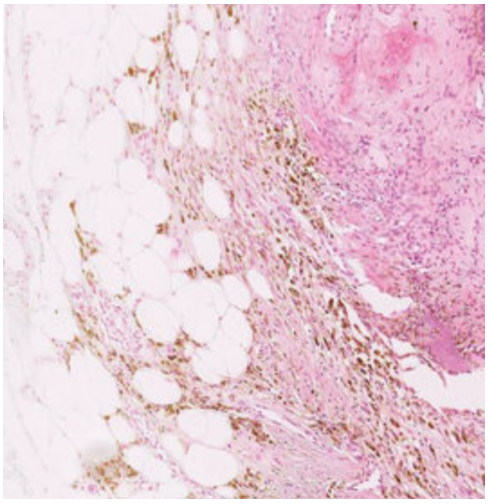


Figure 8.4 Image with unclear tumor boundaries.

OUTPUT : The model may have difficulty accurately predicting the likelihood of renal cell carcinoma in this case, as the boundaries between the tumor and surrounding tissue are unclear.

PASS/ FAIL : Pass

5. TEST CASE / ACTION TO BE PERFORMED : Image with artifacts or staining anomalies

INPUT : A histology image of a kidney tumor with staining artifacts or other anomalies that could affect image quality

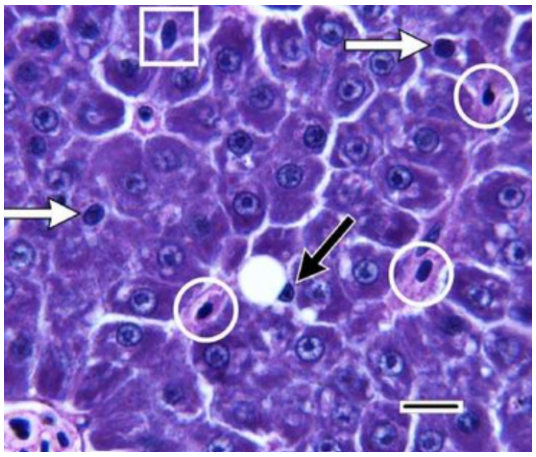


Figure 8.5 Image with artifacts or staining anomalies.

OUTPUT : The model may produce inaccurate results if the artifacts or anomalies in the image interfere with its ability to accurately identify and classify renal cell carcinoma cells.

PASS/ FAIL : Pass

CHAPTER 9

CONCLUSION

9. CONCLUSION

This system presented a heuristic approach using deep learning algorithms so as to build a classification model for monitoring the tumor stage progression in RCC based on histopathological images. This system works based on an optimized algorithm which is generally not programmed with any task-specific rules. A Neural network model is constructed to perform the diagnosis of renal cell carcinoma progression in patients. The major advantage of this system is that it is responsible for analyzing the data with preloaded input to detect the accurate result with greater performance and consume a less amount of time in processing and predicting the results. Hence it helps doctors in starting the treatments early for the patients and also it helps in diagnosing more patients within a shorter period of time. We anticipate that these prediction models based on the histopathological images could contribute to the optimal management of patients in ccRCC.

CHAPTER 10

APPENDIX

10.1 SOURCE CODE

```
from numpy.random import seed

seed(101)

import tensorflow

tensorflow.random.set_seed(101)

import pandas as pd

import numpy as np

import tensorflow as tf

from tensorflow import keras

from tensorflow.keras.preprocessing.image import ImageDataGenerator

from tensorflow.keras.layers import Conv2D, MaxPooling2D

from tensorflow.keras.layers import Dense, Dropout, Flatten, Activation

from tensorflow.keras.models import Sequential

from tensorflow.keras.callbacks import EarlyStopping, ReduceLROnPlateau,
ModelCheckpoint

from tensorflow.keras.optimizers import Adam

import os

import cv2
```

```
from sklearn.utils import shuffle
from sklearn.metrics import confusion_matrix
from sklearn.model_selection import train_test_split
import itertools
import shutil
import matplotlib.pyplot as plt
%matplotlib inline
DATASET = "E:\Project\data"
IMAGE_SIZE = 96
IMAGE_CHANNELS = 3
SAMPLE_SIZE = 8000
df_data = pd.read_csv(DATASET+'/train_labels.csv')

# removing this image because it caused a training error previously
df_data[df_data['id'] != 'dd6dfed324f9fcb6f93f46f32fc800f2ec196be2']

# removing this image because it's black
df_data[df_data['id'] != '9369c7278ec8bcc6c880d99194de09fc2bd4efbe']
```

```
print(df_data.shape)
df_data['label'].value_counts()
def draw_category_images(col_name,figure_cols, df, IMAGE_PATH):
    categories = (df.groupby([col_name])[col_name].nunique()).index
    f, ax = plt.subplots(nrows=len(categories),ncols=figure_cols,
                         figsize=(4*figure_cols,4*len(categories))) # adjust size here
    # draw a number of images for each location
    for i, cat in enumerate(categories):
        sample = df[df[col_name]==cat].sample(figure_cols) # figure_cols is also the
sample size
        for j in range(0,figure_cols):
            file=IMAGE_PATH + sample.iloc[j]['id'] + '.tif'
            im=cv2.imread(file)
            ax[i, j].imshow(im, resample=True, cmap='gray')
            ax[i, j].set_title(cat, fontsize=16)
    plt.tight_layout()
    plt.show()
IMAGE_PATH = DATASET+'/train/'
draw_category_images('label',4, df_data, IMAGE_PATH)
df_data.head()
```

 # take a random sample of class 0 with size equal to num samples in class 1

df_0 = df_data[df_data['label'] == 0].sample(SAMPLE_SIZE, random_state = 101)

 # filter out class 1

df_1 = df_data[df_data['label'] == 1].sample(SAMPLE_SIZE, random_state = 101)

 # concat the dataframes

df_data = pd.concat([df_0, df_1], axis=0).reset_index(drop=True)

 # shuffle

df_data = shuffle(df_data)

df_data['label'].value_counts()

df_data.head()

 # train_test_split

 # stratify=y creates a balanced validation set.

y = df_data['label']

df_train, df_val = train_test_split(df_data, test_size=0.10, random_state=101, stratify=y)

```
print(df_train.shape)

print(df_val.shape)

df_train['label'].value_counts()

df_val['label'].value_counts()


# Create a new directory

base_dir = 'base_dir'

os.mkdir(base_dir)


# create a path to 'base_dir' to which we will join the names of the new folders

# train_dir

train_dir = os.path.join(base_dir, 'train_dir')

os.mkdir(train_dir)


# val_dir

val_dir = os.path.join(base_dir, 'val_dir')

os.mkdir(val_dir)
```

 # [CREATE FOLDERS INSIDE THE TRAIN AND VALIDATION FOLDERS]

 # Inside each folder we create seperate folders for each class

 # create new folders inside train_dir

```
no_tumor_tissue = os.path.join(train_dir, 'a_no_tumor_tissue')

os.mkdir(no_tumor_tissue)

has_tumor_tissue = os.path.join(train_dir, 'b_has_tumor_tissue')

os.mkdir(has_tumor_tissue)
```

 # create new folders inside val_dir

```
no_tumor_tissue = os.path.join(val_dir, 'a_no_tumor_tissue')

os.mkdir(no_tumor_tissue)

has_tumor_tissue = os.path.join(val_dir, 'b_has_tumor_tissue')

os.mkdir(has_tumor_tissue)
```

 # check that the folders have been created

```
os.listdir('base_dir/train_dir')

df_data
```

```
# Set the id as the index in df_data
df_data.set_index('id', inplace=True)

# Get a list of train and val images
train_list = list(df_train['id'])
val_list = list(df_val['id'])

# Transfer the train images
for image in train_list:

  # the id in the csv file does not have the .tif extension therefore we add it here
    fname = image + '.tif'

  # get the label for a certain image
    target = df_data.loc[image,'label']
    # these must match the folder names
    if target == 0:
        label = 'a_no_tumor_tissue'
    if target == 1:
        label = 'b_has_tumor_tissue'
```

```
    # source path to image
    src = os.path.join(DATASET+'\\train', fname)
    # destination path to image
    dst = os.path.join(train_dir, label, fname)
    # copy the image from the source to the destination
    shutil.copyfile(src, dst)

# Transfer the val images
for image in val_list:

    # the id in the csv file does not have the .tif extension therefore we add it here
    fname = image + '.tif'
    # get the label for a certain image
    target = df_data.loc[image,'label']

    # these must match the folder names
    if target == 0:
        label = 'a_no_tumor_tissue'
```

```
    if target == 1:
        label = 'b_has_tumor_tissue'

    # source path to image
    src = os.path.join(DATASET+'\\train', fname)

    # destination path to image
    dst = os.path.join(val_dir, label, fname)

    # copy the image from the source to the destination
    shutil.copyfile(src, dst)

# check how many train images we have in each folder
print(len(os.listdir('base_dir/train_dir/a_no_tumor_tissue')))
print(len(os.listdir('base_dir/train_dir/b_has_tumor_tissue')))

# check how many val images we have in each folder
print(len(os.listdir('base_dir/val_dir/a_no_tumor_tissue')))
print(len(os.listdir('base_dir/val_dir/b_has_tumor_tissue')))
train_path = 'base_dir/train_dir'
```

```
valid_path = 'base_dir/val_dir'

test_path = '../input/test'

num_train_samples = len(df_train)

num_val_samples = len(df_val)

train_batch_size = 10

val_batch_size = 10

train_steps = np.ceil(num_train_samples / train_batch_size)

val_steps = np.ceil(num_val_samples / val_batch_size)

datagen = ImageDataGenerator(rescale=1.0/255)

train_gen = datagen.flow_from_directory(train_path,

                                        target_size=(IMAGE_SIZE,IMAGE_SIZE),

                                        batch_size=train_batch_size,

                                        class_mode='categorical')

val_gen = datagen.flow_from_directory(valid_path,

                                        target_size=(IMAGE_SIZE,IMAGE_SIZE),

                                        batch_size=val_batch_size,

                                        class_mode='categorical')


# Note: shuffle=False causes the test dataset to not be shuffled

test_gen = datagen.flow_from_directory(valid_path,
```

```
                    target_size=(IMAGE_SIZE,IMAGE_SIZE),
                    batch_size=1,
                    class_mode='categorical',
                    shuffle=False)
kernel_size = (3,3)
pool_size= (2,2)
first_filters = 32
second_filters = 64
third_filters = 128
dropout_conv = 0.3
dropout_dense = 0.3
model = Sequential()
model.add(Conv2D(first_filters, kernel_size, activation = 'relu', input_shape = (96, 96, 3)))
model.add(Conv2D(first_filters, kernel_size, activation = 'relu'))
model.add(Conv2D(first_filters, kernel_size, activation = 'relu'))
model.add(MaxPooling2D(pool_size = pool_size))
model.add(Dropout(dropout_conv))
```

```
model.add(Conv2D(second_filters, kernel_size, activation ='relu'))
model.add(Conv2D(second_filters, kernel_size, activation ='relu'))
model.add(Conv2D(second_filters, kernel_size, activation ='relu'))
model.add(MaxPooling2D(pool_size = pool_size))
model.add(Dropout(dropout_conv))
model.add(Conv2D(third_filters, kernel_size, activation ='relu'))
model.add(Conv2D(third_filters, kernel_size, activation ='relu'))
model.add(Conv2D(third_filters, kernel_size, activation ='relu'))
model.add(MaxPooling2D(pool_size = pool_size))
model.add(Dropout(dropout_conv))
model.add(Flatten())
model.add(Dense(256, activation = "relu"))
model.add(Dropout(dropout_dense))
model.add(Dense(2, activation = "softmax"))
model.summary()
model.compile(Adam(lr=0.0001), loss='binary_crossentropy',
        metrics=['accuracy'])
```

 # Get the labels that are associated with each index

```
print(val_gen.class_indices)
```

```
filepath = "model.h5"
checkpoint = ModelCheckpoint(filepath, monitor='val_acc', verbose=1,
                             save_best_only=True, mode='max')
reduce_lr = ReduceLROnPlateau(monitor='val_acc', factor=0.5, patience=2,
                              verbose=1, mode='max', min_lr=0.00001)
callbacks_list = [checkpoint, reduce_lr]
history = model.fit_generator(train_gen, steps_per_epoch=train_steps,
                    validation_data=val_gen,
                    validation_steps=val_steps,
                    epochs=20, verbose=1,
                   callbacks=callbacks_list)

# get the metric names so we can use evaulate_generator
model.metrics_names

# Here the best epoch will be used.
model.load_weights(DATASET+'/../model.h5')
val_loss, val_acc = \
model.evaluate_generator(test_gen,
                         steps=len(df_val))
```

```
print('val_loss:', val_loss)

print('val_acc:', val_acc)


# make a prediction

predictions = model.predict_generator(test_gen, steps=len(df_val), verbose=1)


# This is how to check what index keras has internally assigned to each class.

test_gen.class_indices

df_preds = pd.DataFrame(predictions, columns=['no_tumor_tissue',
'has_tumor_tissue'])

df_preds.head()


# Get the true labels

y_true = test_gen.classes


# Get the predicted labels as probabilities

y_pred = df_preds['has_tumor_tissue']

from sklearn.metrics import roc_auc_score

roc_auc_score(y_true, y_pred)

def plot_confusion_matrix(cm, classes,
```

```
                    normalize=False,
                    title='Confusion matrix',
                    cmap=plt.cm.Blues):
if normalize:
    cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
    print("Normalized confusion matrix")
else:
    print('Confusion matrix, without normalization')
print(cm)
plt.imshow(cm, interpolation='nearest', cmap=cmap)
plt.title(title)
plt.colorbar()
tick_marks = np.arange(len(classes))
plt.xticks(tick_marks, classes, rotation=45)
plt.yticks(tick_marks, classes)
fmt = '.2f' if normalize else 'd'
thresh = cm.max() / 2.
for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
    plt.text(j, i, format(cm[i, j], fmt),
             horizontalalignment="center",
```

```
            color="white" if cm[i, j] > thresh else "black")
    plt.ylabel('True label')
    plt.xlabel('Predicted label')
    plt.tight_layout()

# Get the labels of the test images.
test_labels = test_gen.classes
test_labels.shape

# argmax returns the index of the max value in a row
cm = confusion_matrix(test_labels, predictions.argmax(axis=1))

# Print the label associated with each class
test_gen.class_indices
cm_plot_labels = ['no_tumor_tissue', 'has_tumor_tissue']
plot_confusion_matrix(cm, cm_plot_labels, title='Confusion Matrix')
from sklearn.metrics import classification_report
y_pred_binary = predictions.argmax(axis=1)
report = classification_report(y_true, y_pred_binary,
target_names=cm_plot_labels)
```

```
print(report)

# deleting the base_dir File
if os.path.isdir(base_dir):
    shutil.rmtree('base_dir')

# create test_dir
test_dir = 'test_dir'
os.mkdir(test_dir)

# create test_images inside test_dir
test_images = os.path.join(test_dir, 'test_images')
os.mkdir(test_images)

# check that the directory we created exists
os.listdir('test_dir')

# Transfer the test images into image_dir
test_list = os.listdir(DATASET+'\\test')
for image in test_list:
```

```
    fname = image
    # source path to image
    src = os.path.join(DATASET+'\\test', fname)
    # destination path to image
    dst = os.path.join(test_images, fname)
    # copy the image from the source to the destination
    shutil.copyfile(src, dst)
len(os.listdir('test_dir/test_images'))
test_path ='test_dir'

# Here we change the path to point to the test_images folder.
test_gen = datagen.flow_from_directory(test_path,
                                       target_size=(IMAGE_SIZE,IMAGE_SIZE),
                                       batch_size=1,
                                       class_mode='categorical',
                                       shuffle=False)
num_test_images = len(os.listdir('test_dir/test_images'))

# make sure we are using the best epoch
model.load_weights('model.h5')
```

```
predictions = model.predict_generator(test_gen, steps=num_test_images,
verbose=1)

import pandas as pd

len(predictions)


# Put the predictions into a dataframe

df_preds = pd.DataFrame(predictions, columns=['no_tumor_tissue',
'has_tumor_tissue'])

len(df_preds)

test_filenames = test_gen.filenames

print(len(test_filenames))

df_preds['file_names'] = test_filenames

df_preds.head()

def extract_id(x):
    # split into a list
    a = x.split('\\')
    # split into a list
    b = a[1].split('.')
    extracted_id = b[0]
    return extracted_id

df_preds['id'] = df_preds['file_names'].apply(extract_id)
```

```
df_preds.head()
y_pred = df_preds['has_tumor_tissue']


# get the id column
image_id = df_preds['id']
submission = pd.DataFrame({'id':image_id,
                           'label':y_pred,
                          })
submission.to_csv('patch_preds.csv')
submission.head()
import warnings
warnings.filterwarnings('ignore')
if os.path.isdir(test_dir):
    shutil.rmtree('test_dir')
submission['stages']=1
for i in range(len(submission['stages'])):
    if submission['label'][i]<0.25:
        submission['stages'][i]='normal'
    elif submission['label'][i] >0.25 and submission['label'][i]<0.5:
        submission['stages'][i]='Stage 1'
```

```
    elif submission['label'][i] >0.5 and submission['label'][i]<0.75:
        submission['stages'][i]='Stage 2'
    else:
        submission['stages'][i]='Critical'
submission['stages'].value_counts()
f, ax = plt.subplots(nrows=4,ncols=4,figsize=(16,20))
categories=['normal','Stage 1','Stage 2','Critical']
for i ,cat in enumerate(categories):
    sample=submission[submission['stages'] == cat].sample(4)
    for j in range(0,4):
        file=DATASET+'/test/'+sample.iloc[j]['id']+'.tif'
        #print(file)
        image=cv2.imread(file)
        #print(type(image))
        plt.imshow(image)
        ax[i,j].imshow(image,resample=True,cmap='gray')
        ax[i,j].set_title(cat,fontsize=16)
plt.show()
```

10.2 SCREENSHOTS

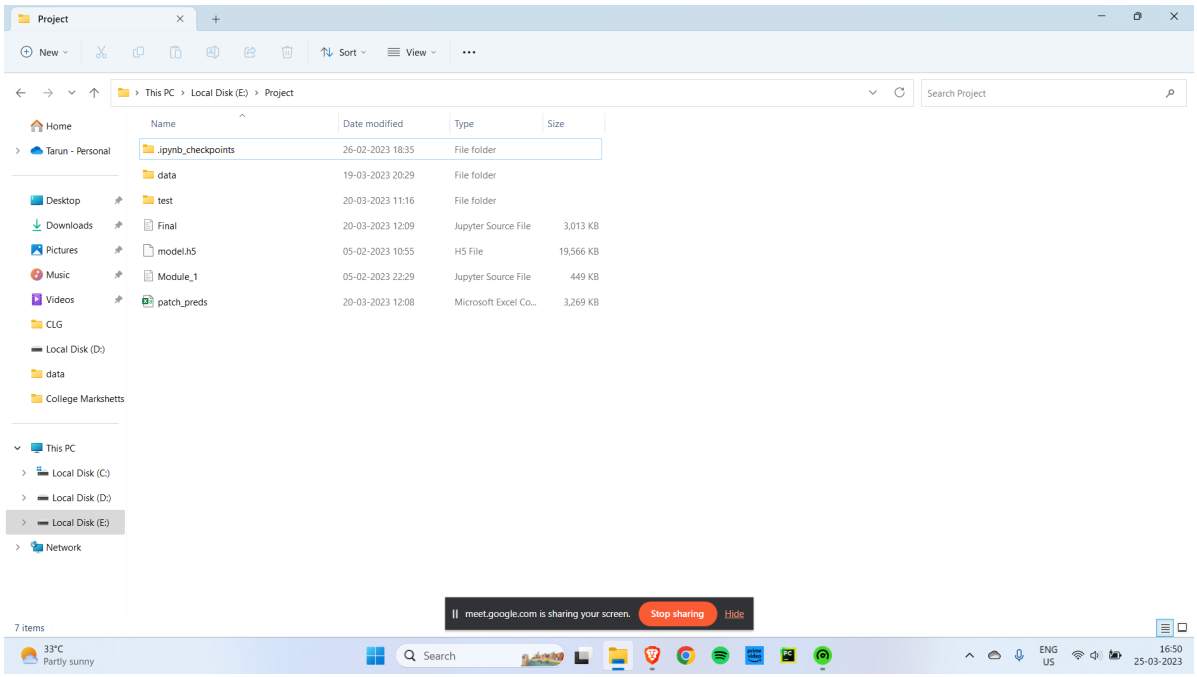


Figure A.1.1 Source code folder

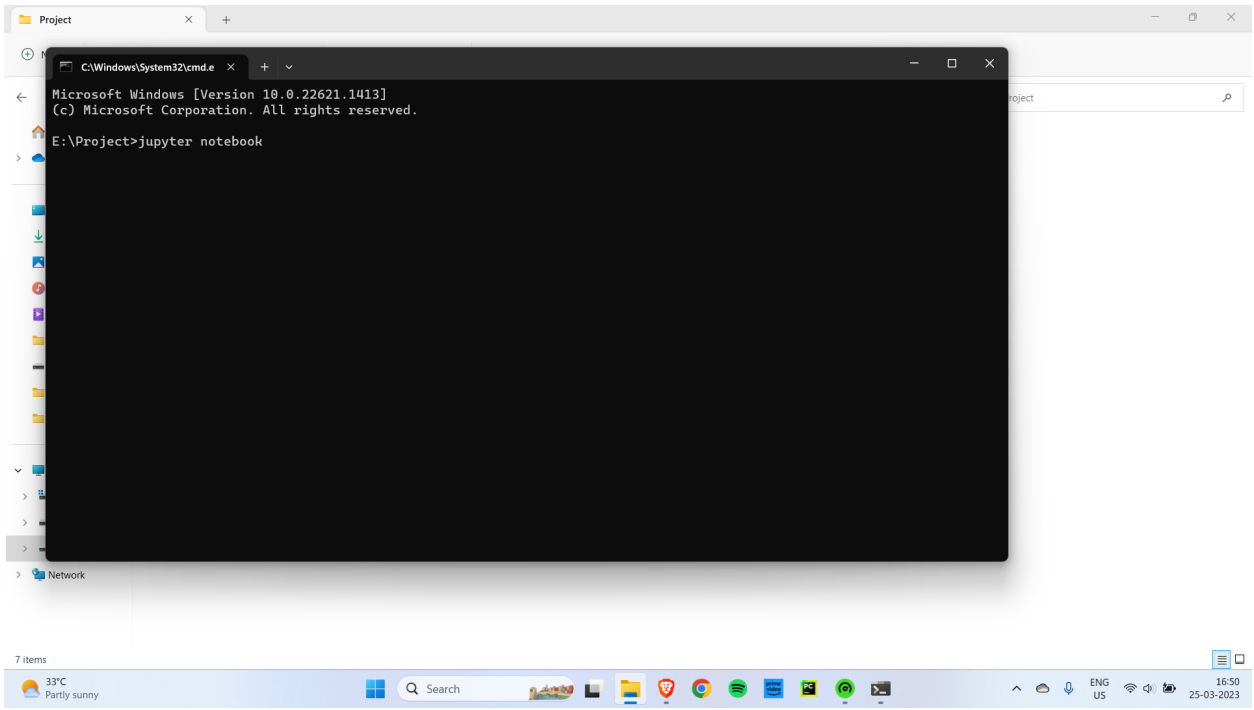


Figure A.1.2 Running the project through command prompt

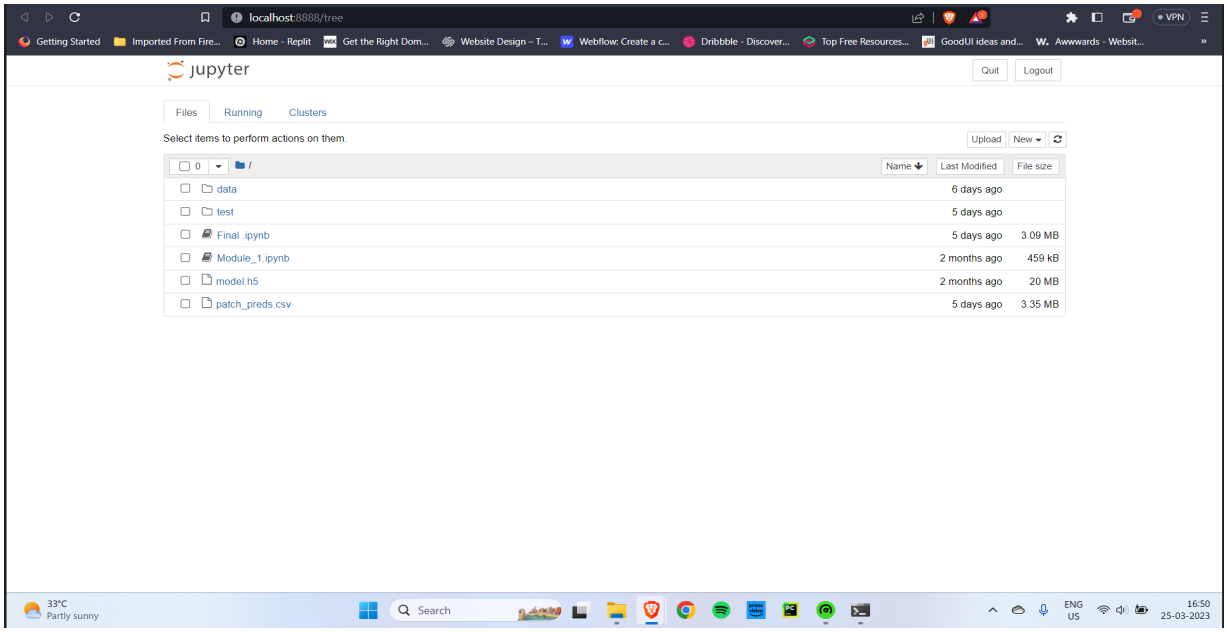


Figure A.1.3 Initializing the project with Jupyter notebook

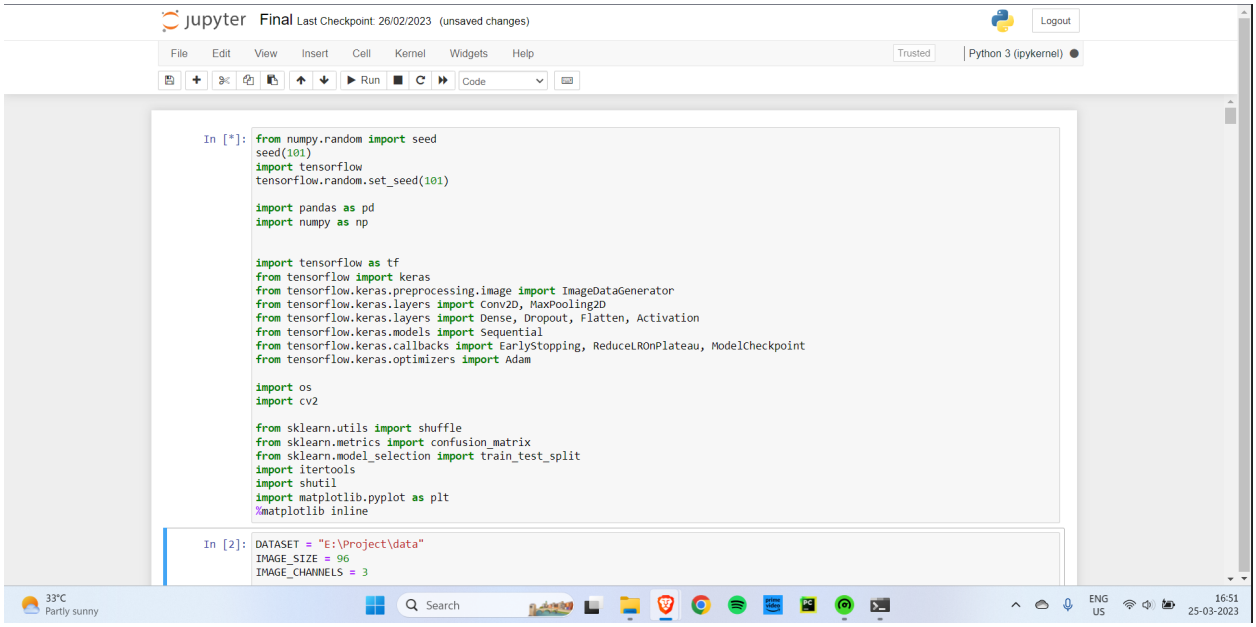


Figure A.1.4 Image of the main code

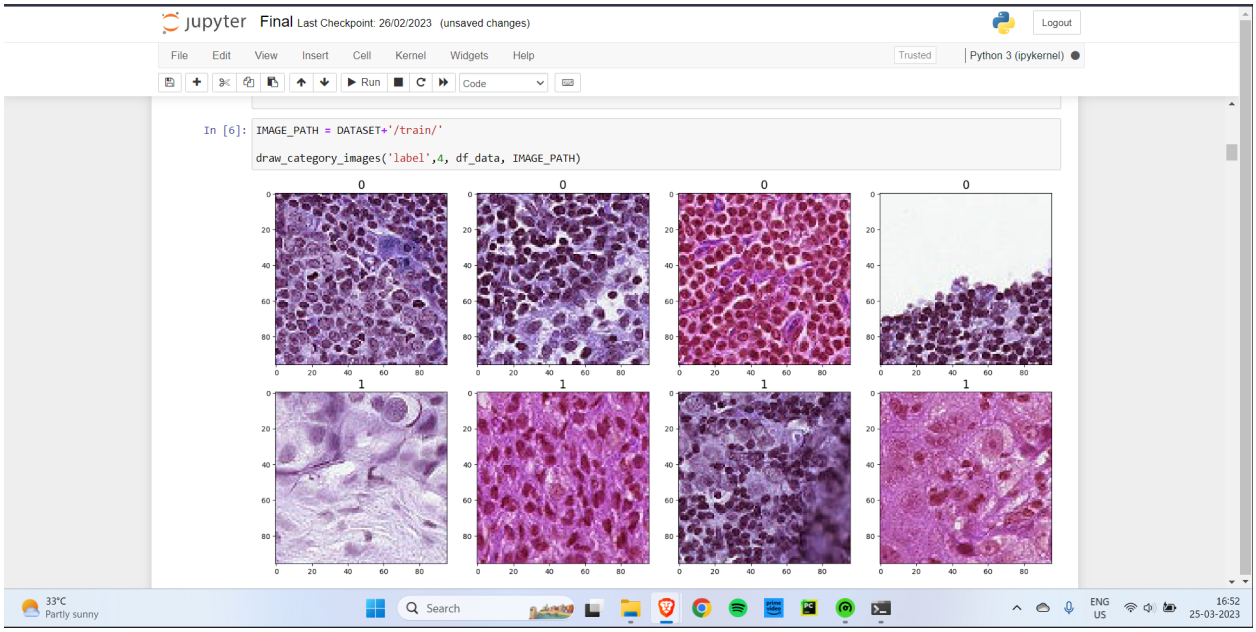


Figure A.1.5 Input data

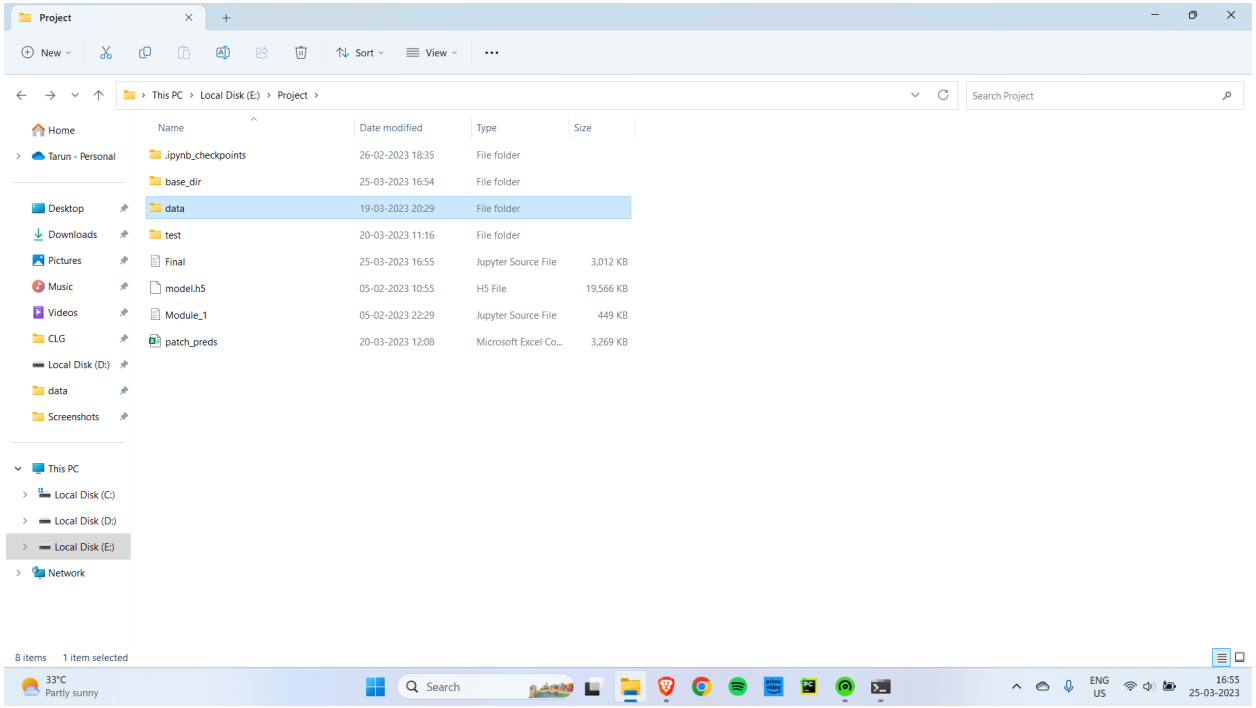


Figure A.1.6 Dataset Images

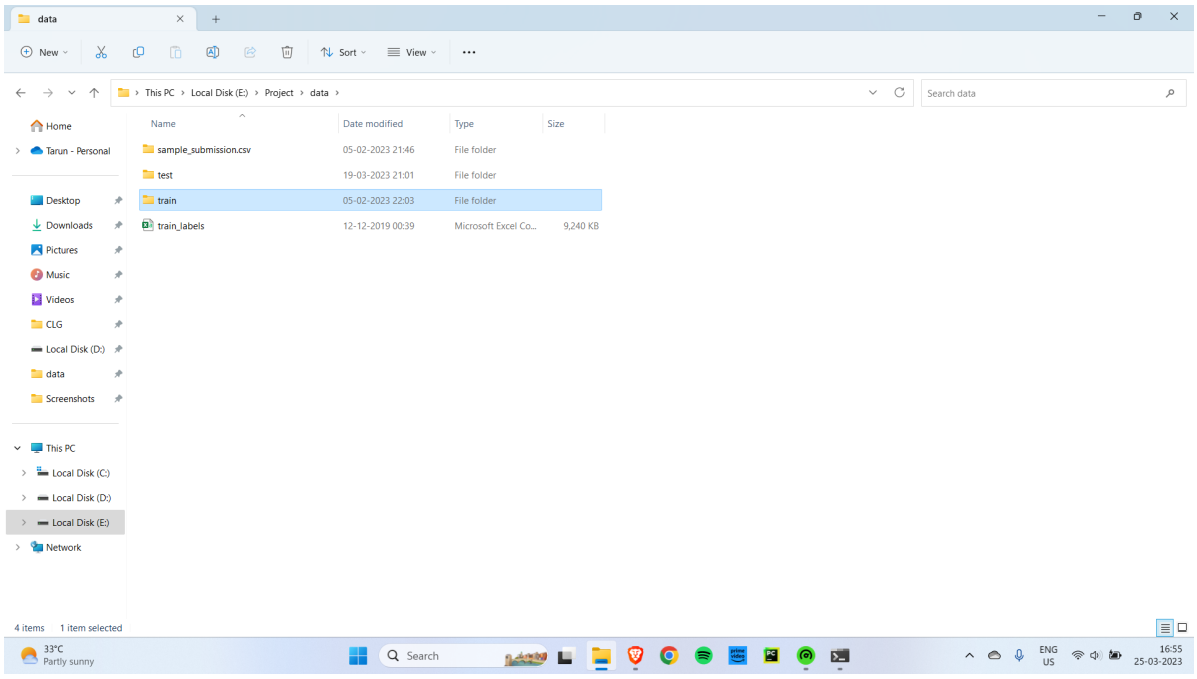


Figure A.1.7 Training dataset folder

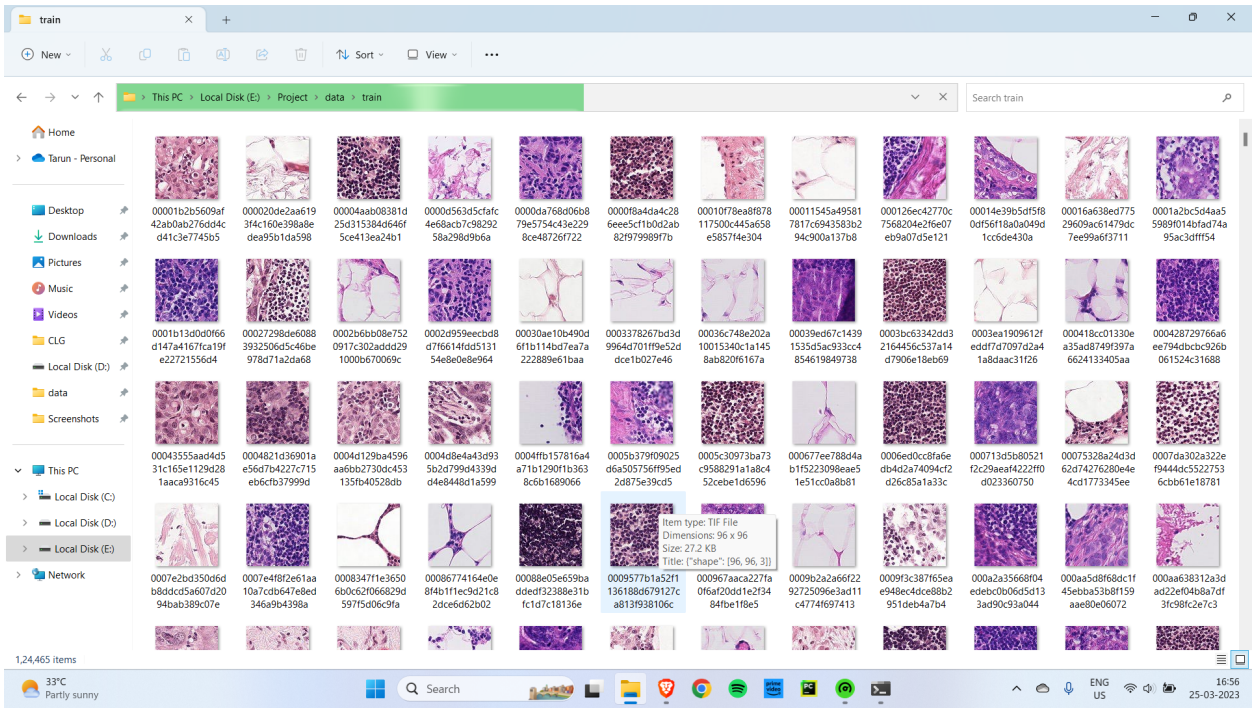


Figure A.1.8 Training dataset images

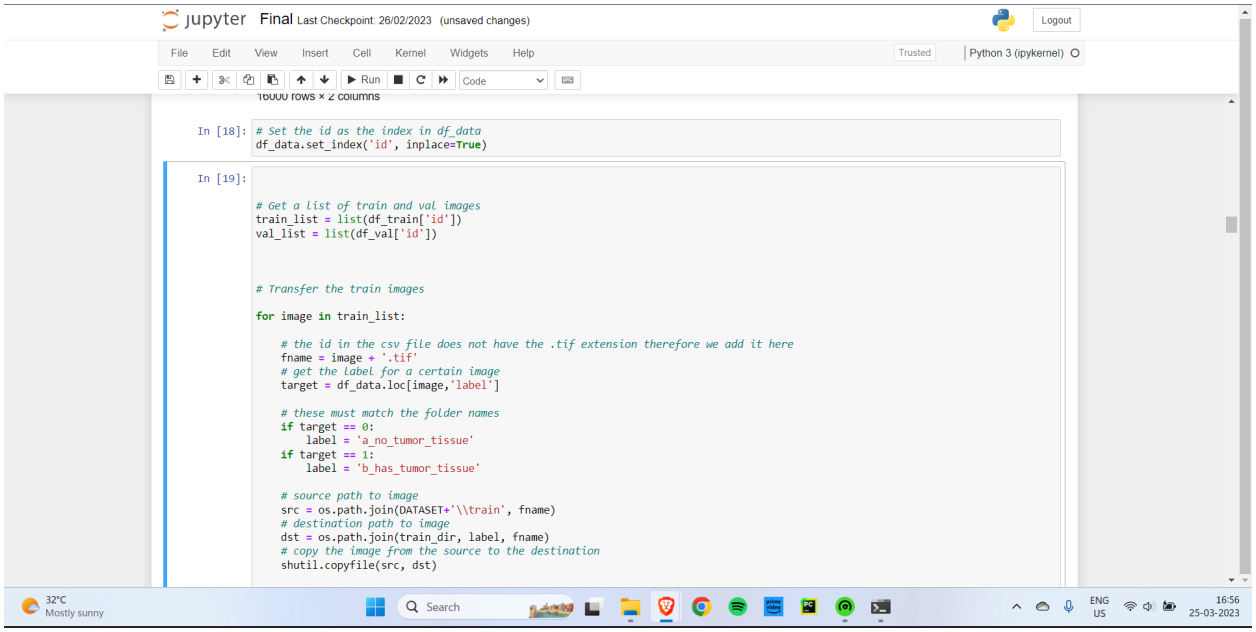


Figure A.1.9 Algorithm to classify tumour and non tumour cells

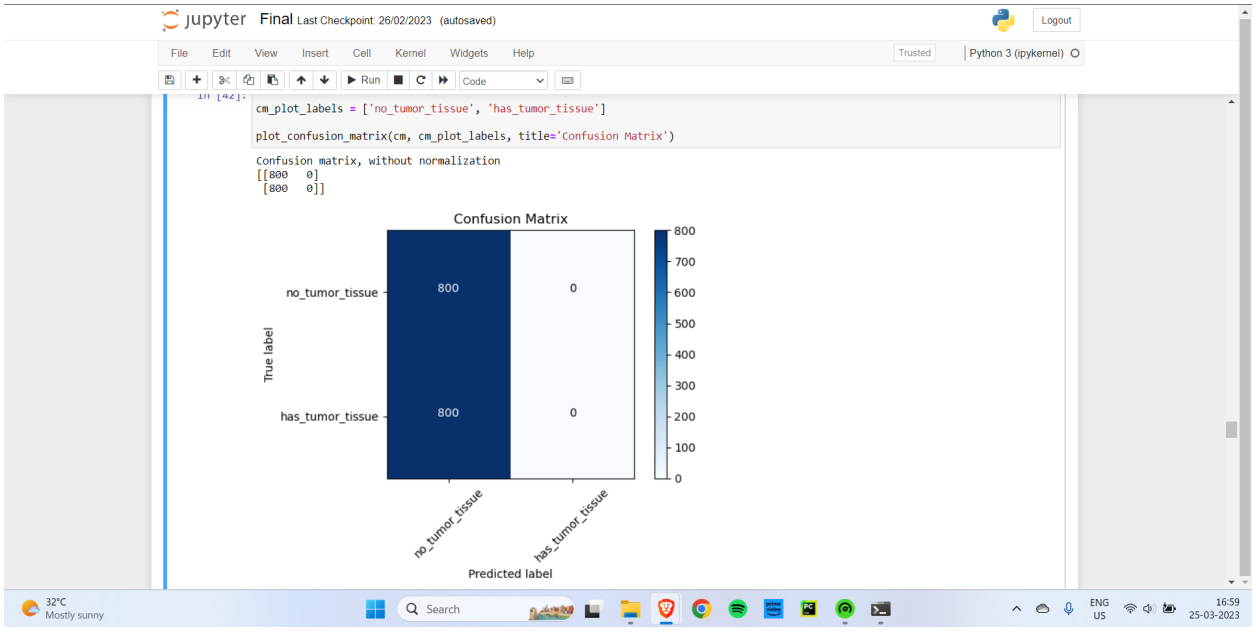


Figure A.1.10 Confusion Matrix

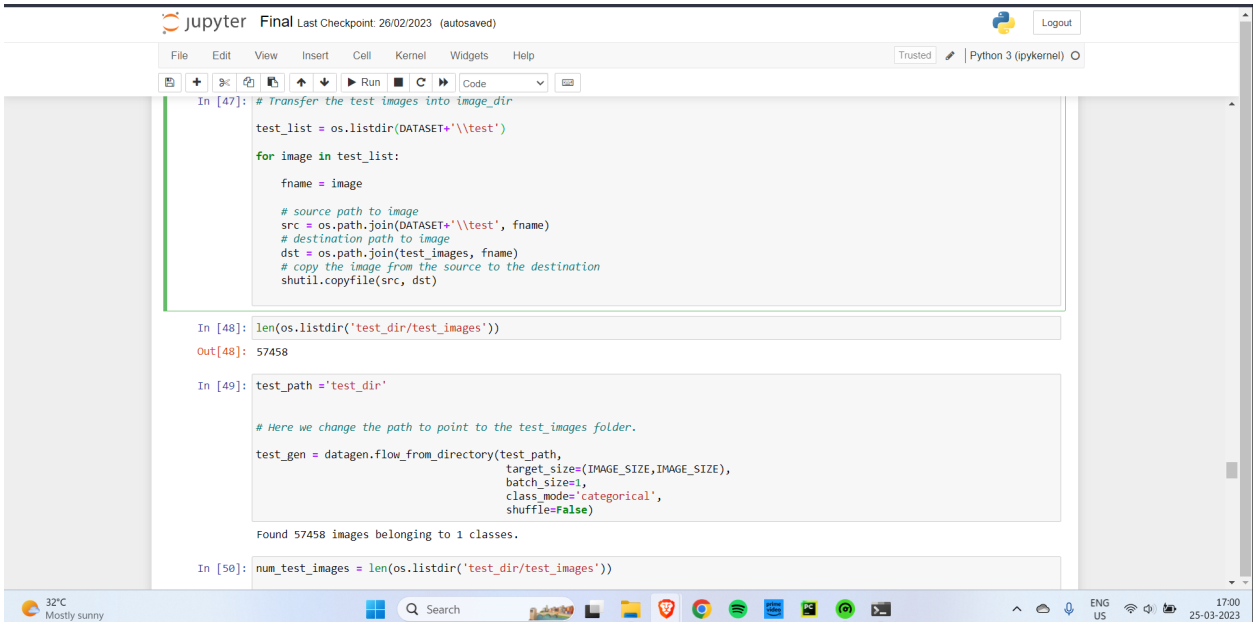


Figure A.1.11 Code for testing the data

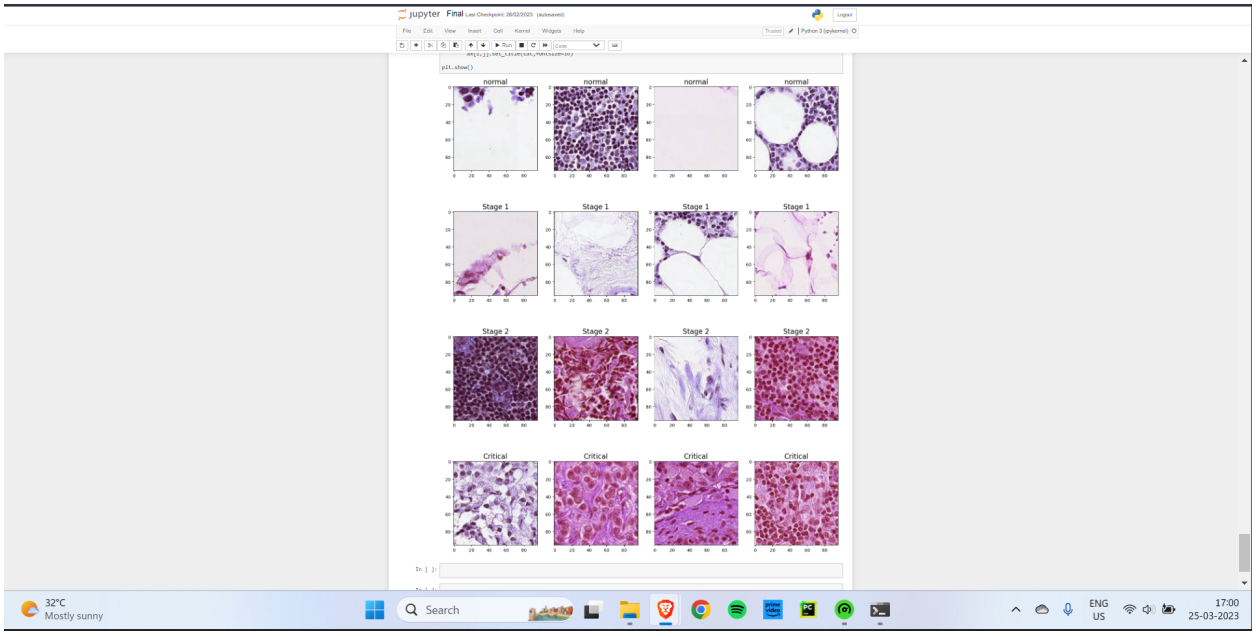


Figure A.1.12 Output

REFERENCES

11. REFERENCES

[1] Nathan Hadjiyski, Kidney Cancer Staging: Deep Learning Neural Network Based Approach (2020)

[2] Guozhen Chen, Chenguang Ding, Yang Li, Xiaojun Hu, Xiao Li, Li Ren, Xiaoming Ding, Puxun Tian, Wujun Xue : Prediction of Chronic Kidney Disease Using Adaptive Hybridized Deep Convolutional Neural Network on the Internet of Medical Things Platform (2020)

[3] Ahmed Elmahy, Sherin Aly, Fayek Elkhwsky : Cancer Stage Prediction From Gene Expression Data Using Weighted Graph Convolution Network (2021)

[4] Dong Liu; Jinkai Shao; Honggang Liu; Wei Cheng : Design on Early Warning System for Renal Cancer Recurrence Based on CNN-Based Internet of Things (2021)

[5] Christopher Ricketts, Aguirre Andres De Cubas : The Cancer Genome Atlas Comprehensive Molecular Characterization of Renal Cell Carcinoma (2018)

[6] Nicolas Coudray, Paolo Santiago Ocampo, Theodore Sakellaropoulos, Navneet

Narula, Matija Snuderl, David Fenyö, Andre L Moreira, Narges Razavian, Aristotelis Tsirigos : Classification and mutation prediction from non-small cell lung cancer histopathology images using deep learning (2018)

[7] Jun Cheng, Jie Zhang : Classification and mutation prediction from non-small cell lung cancer histopathology images using deep learning (2017)

[8] Murari Lal Dhanetwal, Sonia Badwal, Gaurav Pratap Singh Gahlot, Kavita Sahai, A K Shukla : Immunohistological Diagnosis of Primary and Metastatic Renal Cell Carcinoma Using Panel of Immunohistochemical Markers (2019)

[9] Yun Liu, Krishna Gadepalli, Mohammad Norouzi, George E. Dahl, Timo Kohlberger, Aleksey Boyko, Subhashini Venugopalan, Aleksei Timofeev, Philip Q. Nelson, Greg S. Corrado, Jason D. Hipp, Lily Peng, Martin C. Stumpe
Detecting Cancer Metastases on Gigapixel Pathology Images (2017)

[10] Pegah Khosravi, Ehsan Kazemi, Marcin Imielinski, Olivier Elemento, Iman Hajirasouliha : Deep Convolutional Neural Networks Enable Discrimination of Heterogeneous Digital Pathology Images (2018)