**ABSTRACT**

Medical imaging tools are essential in early-stage lung cancer diagnostics and the monitoring of lung cancer during treatment. Various medical imaging modalities, such as chest X-ray, magnetic resonance imaging, positron emission tomography, computed tomography, and molecular imaging techniques, have been extensively studied for lung cancer detection.

These techniques have some limitations, including not classifying cancer images automatically, which is unsuitable for patients with other pathologies. It is urgently necessary to develop a sensitive and accurate approach to the early diagnosis of lung cancer. Deep learning is one of the fastest-growing topics in medical imaging, with rapidly emerging applications spanning medical image-based and textural data modalities. With the help of deep learning-based medical imaging tools, clinicians can detect and classify lung nodules more accurately and quickly.

Therefore, this work implements the advanced modifications in CNN model for the detection of lung cancer from chest scan images. The proposed CNN model is able to classify the being and malignant i.e., normal, and cancerous with higher accuracy as compared to state-of-the-art machine learning approach called support vector machine (SVM) classifier. In addition, the obtained quality metrics discloses the superiority of proposed deep CNN model for assisting the expertise in an enhanced diagnosis.

**CHAPTER 1**

**INTRODUCTION**

Lung cancer is the primary cause of cancer death worldwide, with 2.09 million new cases and 1.76 million people dying from lung cancer in 2018 [1]. Four case-controlled studies from Japan reported in the early 2000s that the combined use of chest radiographs and sputum cytology in screening was effective for reducing lung cancer mortality. In contrast, two randomized controlled trials conducted from 1980 to 1990 concluded that screening with chest radiographs was not effective in reducing mortality in lung cancer [2, 3]. Although the efficacy of chest radiographs in lung cancer screening remains controversial, chest radiographs are more cost-effective, easier to access, and deliver lower radiation dose compared with low dose computed tomography (CT). A further disadvantage of chest CT is excessive false positive (FP) results. It has been reported that 96% of nodules detected by low-dose CT screening are FPs, which commonly leads to unnecessary follow-up and invasive examinations. Chest radiography is inferior to chest CT in terms of sensitivity but superior in terms of specificity. Taking these characteristics into consideration, the development of a computer-aided diagnosis (CAD) model for chest radiograph would have value by improving sensitivity while maintaining low FP results [4].

Many computer-aided detection (CAD) systems have been extensively studied for lung cancer detection and classification [5, 6]. Compared to trained radiologists, CAD systems provide better lung nodules and cancer detection performance in medical images. Generally, the CAD-based lung cancer detection system includes four steps: image processing, extraction of the region of interest (ROI), feature selection, and classification. Among these steps, feature selection and classification play the most critical roles in improving the accuracy and sensitivity of the CAD system, which relies on image processing to capture reliable features. However, benign, and malignant nodule classification is a challenge. Therefore, a rapid, cost-effective, and highly sensitive deep learning-based CAD system for lung cancer prediction is urgently needed.

The recent application of convolutional neural networks (CNN), a field of deep learning (DL), has led to dramatic, state-of-the-art improvements in radiology. DL-based models have also shown promise for nodule/mass detection on chest radiographs, which have reported sensitivities in the range of 0.51–0.84 and mean number of FP indications per image (mFPI) of 0.02–0.34. In addition, radiologist performance for detecting nodules was better with these CAD models than without them. In clinical practice, it is often challenging for radiologists to detect nodules and to differentiate between benign and malignant nodules. Normal anatomical structures often appear as if they are nodules, which is why radiologists must pay careful attention to the shape and marginal properties of nodules. As these problems are caused by the conditions rather than the ability of the radiologist, even skilful radiologists can misdiagnose. Therefore, the main purpose of this work was to train and validate a DL-based model capable of detecting lung cancer on chest radiographs, and to evaluate the characteristics of this DL-based model to improve sensitivity while maintaining low FP results.

**CHAPTER 2**

**LITERATURE SURVEY**

The development of malignant cells in the lungs is known as lung cancer. Overall men and women's mortality rates have increased as a result of growing cancer incidence. Lung cancer is a disease wherein the cells in the lungs quickly multiply. Lung cancer cannot be eradicated, but it can be reduced [7]. The number of people affected with lung cancer is precisely equal to the number of people who smoke continuously. Lung cancer treatment was evaluated using classification approaches such as Naive Bayes, SVM, Decision Tree, and Logistic Regression. Pradhan et al. [8] conduct an empirical evaluation of multiple machine learning methods that can be used to identify lung cancer using IoT devices. A survey of roughly 65 papers employing machine learning techniques to forecast various diseases was conducted in this study. The study focuses on a variety of machine learning methods for detecting a variety of diseases in order to identify a gap in prospective lung cancer detection in medical IoT. Deep residual learning is used by Bhatia et al. [9] to identify lung cancer from CT scans. With the UNet and ResNet algorithms, we propose a series of pre-processing approaches for emphasising cancer-prone lung regions and retrieving characteristics. The extracted features are fed through several classifiers, namely Adaboost and Random Forest, and the individual predictions are ensembled to calculate the likelihood of a CT scan becoming cancerous.

Without learning inadequate human information, Shin et al. [10, 11] use deep learning to investigate the characteristics of cell exosomes and determine the similarities in human plasma extracellular vesicles. The deep learning classifier was tested with exosome SERS data from regular and lung cancer cell lines and was able to categorise them with 95% efficiency. The deep learning algorithm projected that 90.7% of patients' plasma exosomes were more similar to lung cancer cell extracellular vesicles than the mean of healthy controls in 43 patients, encompassing stage I and II cancer patients. In the ability to forecast lung ADC subtypes, researchers looked at four clinical factors: age, sex, tumour size, and smoking status, as well as 40 radiomic markers. The LIFEx software was used to extract radiomic characteristics from PET scans of segmented cancers. The clinical and radio mic variables were ranked, and a subset of meaningful features was chosen based on Gini coefficient scores for histopathological class relationships [12]. In the estimation of survival, a deep learning network with a tumour cell and metastatic staging system was used to examine the dependability of individual therapy suggestions supplied by the deep learning preservation neural network. The C statistics were employed to evaluate the performance of the model. The computational intelligence survival neural network model's longevity forecasts and treatment strategies were made easier with the use of a customer interface [13].

A lung cancer detection model that utilizes image analysis and machine intelligence to identify the occurrence of lung cancer in CT scans and blood tests has been developed. Despite the fact that CT scan findings are more efficient than mammograms, patient CT scan pictures are divided into normal and abnormal categories [14, 15]. Even in the same tumour stage, non-small-cell cancer patients have a wide range of clinical performance and results. This research investigates deep learning applications such as medical imaging, which could help with patient stratification by automating the measurement of radiographic properties.

**CHAPTER 3**

**EXISTING TECHNIQUE**

**3.1 Support Vector Machine**

SVM is a supervised machine learning algorithm that can be used for both binary and multiclass classification tasks. It aims to find a hyperplane that best separates different classes of data. In the context of lung cancer classification, SVM can be trained on features extracted from CT scan images to distinguish between cancerous and non-cancerous regions.SVM typically relies on handcrafted features extracted from the images, such as texture, intensity, and shape descriptors. These features need to be carefully designed and selected, which can be a challenging and time-consuming process.It can utilize the kernel trick to map the data into a higher-dimensional space, making it possible to capture complex relationships between features. This can be advantageous when dealing with non-linear patterns in CT scan images.Top of Form

**3.1.1 Working**

Here's an overview of how SVM works for this specific application:

1. Data Collection and Preprocessing:

* First, a dataset of CT scan images is collected. This dataset should include labeled examples where each image is associated with a class label (cancerous or non-cancerous).
* The images may be preprocessed to enhance their quality, normalize pixel values, and resize them to a consistent resolution. Preprocessing can also involve noise reduction and contrast adjustment.

2. Feature Extraction:

* Before applying SVM, relevant features are extracted from the CT scan images. Feature extraction is a crucial step because it transforms raw image data into a format that can be used by the SVM.
* Features can include texture descriptors, intensity histograms, shape characteristics, or more advanced features obtained from image processing techniques.

3. Data Splitting: The dataset is typically divided into three subsets: training, validation, and testing sets. The training set is used to train the SVM model, the validation set helps fine-tune hyperparameters, and the testing set evaluates the model's performance.

4. SVM Training:

* The SVM model aims to find a hyperplane that best separates the cancerous and non-cancerous classes in the feature space.
* The SVM training process involves finding the optimal hyperplane by maximizing the margin (distance) between the nearest data points of each class. The data points that lie closest to the hyperplane are called support vectors.

5. Hyperparameter Tuning: SVM has hyperparameters, such as the regularization parameter (C) and the kernel parameters, that affect the model's performance. These hyperparameters are tuned using the validation set to optimize the model's accuracy and generalization.

6. Classification: Once the SVM model is trained and optimized, it can be used to classify new CT scan images. The extracted features from a new image are passed through the SVM model, which assigns a class label (cancerous or non-cancerous) based on the location of the feature vector relative to the decision boundary (hyperplane).

7. Evaluation

* The performance of the SVM model is evaluated using the testing set. Common evaluation metrics include accuracy, precision, recall, F1-score, and the confusion matrix.
* The model's ability to correctly classify lung cancer in CT scan images is assessed, and its performance is compared to other models or approaches.

In summary, SVM works for lung cancer classification from CT scan images by learning a decision boundary that best separates cancerous and non-cancerous regions in a high-dimensional feature space. It is a powerful algorithm for binary classification tasks when the appropriate features are extracted and when data separation is feasible. However, for more complex image classification tasks, deep learning techniques like convolutional neural networks (CNNs) may offer superior performance due to their ability to automatically learn relevant features from raw image data.

**3.2 Limitations**

* Limited Feature Extraction: SVM's performance heavily depends on the quality and relevance of handcrafted features. In the case of lung cancer classification from CT scans, designing effective features that capture subtle patterns indicative of cancer can be difficult.
* Data Size and Imbalance: SVM may not perform well when dealing with large datasets, and it can be sensitive to class imbalance. In medical imaging, collecting a balanced dataset with a sufficient number of cancerous and non-cancerous cases can be challenging.
* Scalability: SVMs can become computationally expensive when dealing with high-dimensional feature spaces, as is often the case with image data. Training on large datasets or using complex kernels can lead to longer training times.
* Lack of Spatial Information: SVM does not inherently capture spatial relationships within the images, which can be crucial for identifying tumors in CT scans. CNNs, on the other hand, are designed to automatically learn hierarchical features, including spatial features, from raw image data.

**CHAPTER 4**

**PROPOSED METHODOLOGY**

**4.1 Overview**

A deep CNN model for lung cancer classification from CT scan images is a powerful approach that leverages the capabilities of deep learning to automatically learn and extract relevant features from raw image data. Here is an overview of how a deep CNN model can be used for classifying CT scan images into normal and malignant categories:

1. Data Collection and Preprocessing:

* A dataset of CT scan images is collected, where each image is labeled as normal (non-cancerous) or malignant (cancerous).
* Preprocessing steps are applied to the images, including resizing them to a consistent resolution, normalizing pixel values, and enhancing image quality.

2. Dataset Splitting: The dataset is divided into two subsets: training, and testing sets. This division allows for training the model, tuning hyperparameters, and evaluating its performance independently.

3. Architecture Design:

* The deep CNN architecture is designed to learn hierarchical features from the CT scan images. A typical architecture consists of multiple convolutional layers followed by pooling layers and fully connected layers.
* Convolutional layers use learnable filters to detect features such as edges, textures, and shapes at different spatial scales.
* Pooling layers reduce the spatial dimensions of the feature maps while retaining the most important information.
* Fully connected layers combine the extracted features and make the final classification decision.

4. Model Training:

* The CNN model is trained using the training dataset. During training, the model learns to optimize its internal parameters (weights and biases) to minimize a specified loss function (e.g., binary cross-entropy).
* Backpropagation and gradient descent techniques are used to update the model's parameters iteratively.

5. Hyperparameter Tuning: Hyperparameters, such as learning rate, batch size, and the number of layers and neurons, are tuned using the validation set to optimize the model's performance.

7. Regularization Techniques: Regularization methods like dropout and batch normalization may be incorporated to prevent overfitting and enhance model robustness.

8. Evaluation:

* The model's performance is evaluated using the testing dataset. Common evaluation metrics include accuracy, precision, recall, F1-score, and the confusion matrix.
* The model's ability to correctly classify lung cancer CT scan images into normal and malignant categories is assessed.

The deep CNN models have demonstrated remarkable success in medical image classification tasks, including lung cancer classification. Their ability to automatically learn relevant features from raw image data makes them a valuable tool for improving diagnostic accuracy and efficiency in healthcare.

Figure 4.1: Overall design of proposed CNN model for lung cancer classification from CT scan images.

**4.2 Data preprocessing**

Data pre-processing is a process of preparing the raw data and making it suitable for a machine learning model. It is the first and crucial step while creating a machine learning model.When creating a machine learning project, it is not always a case that we come across the clean and formatted data. And while doing any operation with data, it is mandatory to clean it and put in a formatted way. So, for this, we use data pre-processing task.A real-world data generally contains noises, missing values, and maybe in an unusable format which cannot be directly used for machine learning models. Data pre-processing is required tasks for cleaning the data and making it suitable for a machine learning model which also increases the accuracy and efficiency of a machine learning model.

* Getting the dataset
* Importing libraries
* Importing datasets

**Importing Libraries:** To perform data preprocessing using Python, we need to import some predefined Python libraries. These libraries are used to perform some specific jobs. There are three specific libraries that we will use for data preprocessing, which are:

Numpy: Numpy Python library is used for including any type of mathematical operation in the code. It is the fundamental package for scientific calculation in Python. It also supports to add large, multidimensional arrays and matrices. So, in Python, we can import it as:

import numpy as nm

Here we have used nm, which is a short name for Numpy, and it will be used in the whole program.

Matplotlib: The second library is matplotlib, which is a Python 2D plotting library, and with this library, we need to import a sub-library pyplot. This library is used to plot any type of charts in Python for the code. It will be imported as below:

import matplotlib.pyplot as mpt

Here we have used mpt as a short name for this library.

Pandas: The last library is the Pandas library, which is one of the most famous Python libraries and used for importing and managing the datasets. It is an open-source data manipulation and analysis library. Here, we have used pd as a short name for this library. Consider the below image:

Text

Description automatically generated

If we compute any two values from age and salary, then salary values will dominate the age values, and it will produce an incorrect result. So, to remove this issue, we need to perform feature scaling for machine learning.

**4.3 Splitting the Dataset**

In machine learning data preprocessing, we divide our dataset into a training set and test set. This is one of the crucial steps of data preprocessing as by doing this, we can enhance the performance of our machine learning model.Suppose if we have given training to our machine learning model by a dataset and we test it by a completely different dataset. Then, it will create difficulties for our model to understand the correlations between the models.If we train our model very well and its training accuracy is also very high, but we provide a new dataset to it, then it will decrease the performance. So we always try to make a machine learning model which performs well with the training set and also with the test dataset. Here, we can define these datasets as:

A picture containing shape

Description automatically generated

Figure 4.2: Splitting the dataset.

**Training** **Set**: A subset of dataset to train the machine learning model, and we already know the output.

**Test** **set**: A subset of dataset to test the machine learning model, and by using the test set, model predicts the output.

For splitting the dataset, we will use the below lines of code:

from sklearn.model\_selection import train\_test\_split

x\_train, x\_test, y\_train, y\_test= train\_test\_split(x, y, test\_size= 0.2, random\_state=0)

**Explanation**

* In the above code, the first line is used for splitting arrays of the dataset into random train and test subsets.
* In the second line, we have used four variables for our output that are
* x\_train: features for the training data
* x\_test: features for testing data
* y\_train: Dependent variables for training data
* y\_test: Independent variable for testing data
* In train\_test\_split() function, we have passed four parameters in which first two are for arrays of data, and test\_size is for specifying the size of the test set. The test\_size maybe .5, .3, or .2, which tells the dividing ratio of training and testing sets.
* The last parameter random\_state is used to set a seed for a random generator so that you always get the same result, and the most used value for this is 42.

**4.3 CNN Basics**

According to the facts, training and testing of proposed model involves in allowing every source image via a succession of convolution layers by a kernel or filter, rectified linear unit (ReLU), max pooling, fully connected layer and utilize SoftMax layer with classification layer to categorize the objects with probabilistic values ranging from . Convolution layer as is the primary layer to extract the features from a source image and maintains the relationship between pixels by learning the features of image by employing tiny blocks of source data. It’s a mathematical function which considers two inputs like source image where and denotes the spatial coordinates i.e., number of rows and columns. is denoted as dimension of an image (here , since the source image is RGB) and a filter or kernel with similar size of input image and can be denoted as .

Shape

Description automatically generated

Fig. 4.3: Representation of convolution layer process.

The output obtained from convolution process of input image and filter has a size of , which is referred as feature map. Let us assume an input image with a size of and the filter having the size of . The feature map of input image is obtained by multiplying the input image values with the filter values.

A grid with numbers and a star

Description automatically generated

(a)

A picture containing diagram

Description automatically generated

(b)

Fig. 4.4: Example of convolution layer process (a) an image with size is convolving with kernel (b) Convolved feature map

**4.3.1 ReLU layer**

Networks those utilizes the rectifier operation for the hidden layers are cited as rectified linear unit (ReLU). This ReLU function is a simple computation that returns the value given as input directly if the value of input is greater than zero else returns zero. This can be represented as mathematically using the function over the set of and the input as follows:

**4.3.2 Max pooing layer**

This layer mitigates the number of parameters when there are larger size images. This can be called as subsampling or down sampling that mitigates the dimensionality of every feature map by preserving the important information. Max pooling considers the maximum element form the rectified feature map.

**4.3.3 Softmax classifier**

Generally, as seen in the above picture softmax function is added at the end of the output since it is the place where the nodes are meet finally and thus, they can be classified. Here, X is the input of all the models and the layers between X and Y are the hidden layers and the data is passed from X to all the layers and Received by Y. Suppose, we have 10 classes, and we predict for which class the given input belongs to. So, for this what we do is allot each class with a particular predicted output. Which means that we have 10 outputs corresponding to 10 different class and predict the class by the highest probability it has.

Diagram

Description automatically generated

Fig. 4.5: Vehicle prediction using SoftMax classifier.

A diagram of a network

Description automatically generated

Fig. 4.6: Example of SoftMax classifier.

In Figure 8, and we must predict what is the object that is present in the picture. In the normal case, we predict whether the crop is A. But in this case, we must predict what is the object that is present in the picture. This is the place where softmax comes in handy. As the model is already trained on some data. So, as soon as the picture is given, the model processes the pictures, send it to the hidden layers and then finally send to softmax for classifying the picture. The softmax uses a One-Hot encoding Technique to calculate the cross-entropy loss and get the max. One-Hot Encoding is the technique that is used to categorize the data. In the previous example, if softmax predicts that the object is class A then the One-Hot Encoding for:

Class A will be [1 0 0]

Class B will be [0 1 0]

Class C will be [0 0 1]

From the diagram, we see that the predictions are occurred. But generally, we don’t know the predictions. But the machine must choose the correct predicted object. So, for machine to identify an object correctly, it uses a function called cross-entropy function.

So, we choose more similar value by using the below cross-entropy formula.

A diagram of a computer network

Description automatically generated

Fig. 4.7: Example of SoftMax classifier with test data.

In the above example we see that 0.462 is the loss of the function for class specific classifier. In the same way, we find loss for remaining classifiers. The lowest the loss function, the better the prediction is. The mathematical representation for loss function can be represented as: -

**CHAPTER 5**

**UML DIAGRAMS**

UML stands for Unified Modeling Language. UML is a standardized general-purpose modeling language in the field of object-oriented software engineering. The standard is managed, and was created by, the Object Management Group. The goal is for UML to become a common language for creating models of object-oriented computer software. In its current form UML is comprised of two major components: a Meta-model and a notation. In the future, some form of method or process may also be added to; or associated with, UML.

The Unified Modeling Language Is a standard language for specifying, Visualization, Constructing and documenting the artifacts of software system, as well as for business modeling and other non-software systems. The UML represents a collection of best engineering practices that have proven successful in the modeling of large and complex systems. The UML is a very important part of developing objects-oriented software and the software development process. The UML uses mostly graphical notations to express the design of software projects.

**GOALS:** The Primary goals in the design of the UML are as follows:

* Provide users a ready-to-use, expressive visual modeling Language so that they can develop and exchange meaningful models.
* Provide extendibility and specialization mechanisms to extend the core concepts.
* Be independent of particular programming languages and development process.
* Provide a formal basis for understanding the modeling language.
* Encourage the growth of OO tools market.
* Support higher level development concepts such as collaborations, frameworks, patterns and components.
* Integrate best practices.

**Class Diagram**

The class diagram is used to refine the use case diagram and define a detailed design of the system. The class diagram classifies the actors defined in the use case diagram into a set of interrelated classes. The relationship or association between the classes can be either an “is-a” or “has-a” relationship. Each class in the class diagram may be capable of providing certain functionalities. These functionalities provided by the class are termed “methods” of the class. Apart from this, each class may have certain “attributes” that uniquely identify the class.

A screenshot of a computer program

Description automatically generated

**Data flow diagram**

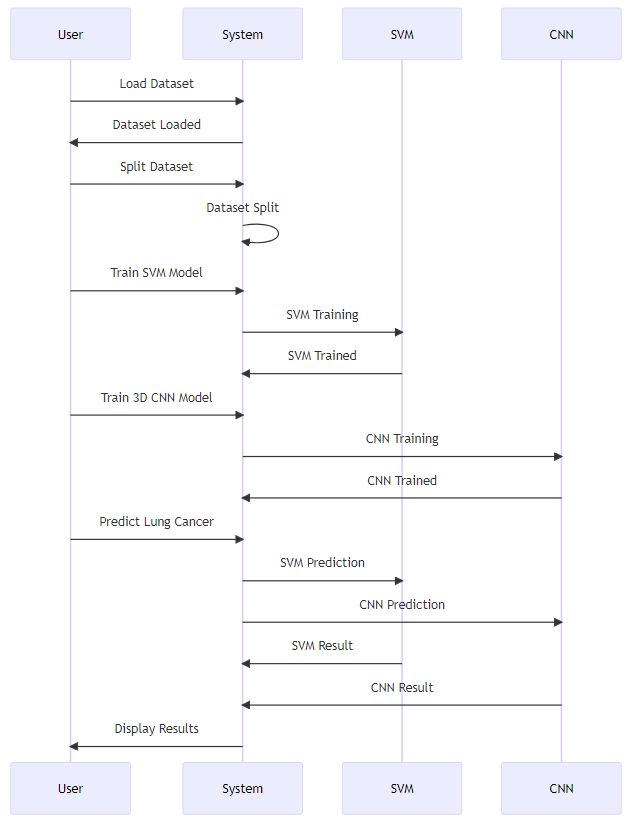
A Data Flow Diagram (DFD) is a visual representation of the flow of data within a system or process. It is a structured technique that focuses on how data moves through different processes and data stores within an organization or a system. DFDs are commonly used in system analysis and design to understand, document, and communicate data flow and processing.

A diagram of a computer

Description automatically generated

**Sequence Diagram**

A sequence diagram in Unified Modeling Language (UML) is a kind of interaction diagram that shows how processes operate with one another and in what order. It is a construct of a Message Sequence Chart. A sequence diagram shows, as parallel vertical lines (“lifelines”), different processes or objects that live simultaneously, and as horizontal arrows, the messages exchanged between them, in the order in which they occur. This allows the specification of simple runtime scenarios in a graphical manner.



**Activity diagram**

Activity diagram is another important diagram in UML to describe the dynamic aspects of the system.

A diagram of a model

Description automatically generated

**CHAPTER 6**

**SOFTWARE ENVIRONMENT**

**What is Python?**

Below are some facts about Python.

* Python is currently the most widely used multi-purpose, high-level programming language.
* Python allows programming in Object-Oriented and Procedural paradigms. Python programs generally are smaller than other programming languages like Java.
* Programmers have to type relatively less and indentation requirement of the language, makes them readable all the time.
* Python language is being used by almost all tech-giant companies like – Google, Amazon, Facebook, Instagram, Dropbox, Uber… etc.

The biggest strength of Python is huge collection of standard library which can be used for the following –

* Machine Learning
* GUI Applications (like Kivy, Tkinter, PyQt etc. )
* Web frameworks like Django (used by YouTube, Instagram, Dropbox)
* Image processing (like Opencv, Pillow)
* Web scraping (like Scrapy, BeautifulSoup, Selenium)
* Test frameworks
* Multimedia

**Advantages of Python**

Let’s see how Python dominates over other languages.

1. **Extensive Libraries**

Python downloads with an extensive library and it contain code for various purposes like regular expressions, documentation-generation, unit-testing, web browsers, threading, databases, CGI, email, image manipulation, and more. So, we don’t have to write the complete code for that manually.

**2. Extensible**

As we have seen earlier, Python can be extended to other languages. You can write some of your code in languages like C++ or C. This comes in handy, especially in projects.

**3. Embeddable**

Complimentary to extensibility, Python is embeddable as well. You can put your Python code in your source code of a different language, like C++. This lets us add scripting capabilities to our code in the other language.

**4. Improved Productivity**

The language’s simplicity and extensive libraries render programmers more productive than languages like Java and C++ do. Also, the fact that you need to write less and get more things done.

**5. IOT Opportunities**

Since Python forms the basis of new platforms like Raspberry Pi, it finds the future bright for the Internet Of Things. This is a way to connect the language with the real world.

**6. Simple and Easy**

When working with Java, you may have to create a class to print ‘Hello World’. But in Python, just a print statement will do. It is also quite easy to learn, understand, and code. This is why when people pick up Python, they have a hard time adjusting to other more verbose languages like Java.

**7. Readable**

Because it is not such a verbose language, reading Python is much like reading English. This is the reason why it is so easy to learn, understand, and code. It also does not need curly braces to define blocks, and indentation is mandatory. This further aids the readability of the code.

**8. Object-Oriented**

This language supports both the procedural and object-oriented programming paradigms. While functions help us with code reusability, classes and objects let us model the real world. A class allows the encapsulation of data and functions into one.

**9. Free and Open-Source**

Like we said earlier, Python is freely available. But not only can you download Python for free, but you can also download its source code, make changes to it, and even distribute it. It downloads with an extensive collection of libraries to help you with your tasks.

**10. Portable**

When you code your project in a language like C++, you may need to make some changes to it if you want to run it on another platform. But it isn’t the same with Python. Here, you need to code only once, and you can run it anywhere. This is called Write Once Run Anywhere (WORA). However, you need to be careful enough not to include any system-dependent features.

1. **Interpreted**

Lastly, we will say that it is an interpreted language. Since statements are executed one by one, debugging is easier than in compiled languages.

Any doubts till now in the advantages of Python? Mention in the comment section.

**Advantages of Python Over Other Languages**

1. **Less Coding**

Almost all of the tasks done in Python requires less coding when the same task is done in other languages. Python also has an awesome standard library support, so you don’t have to search for any third-party libraries to get your job done. This is the reason that many people suggest learning Python to beginners.

**2. Affordable**

Python is free therefore individuals, small companies or big organizations can leverage the free available resources to build applications. Python is popular and widely used so it gives you better community support.

The 2019 Github annual survey showed us that Python has overtaken Java in the most popular programming language category.

**3. Python is for Everyone**

Python code can run on any machine whether it is Linux, Mac or Windows. Programmers need to learn different languages for different jobs but with Python, you can professionally build web apps, perform data analysis and machine learning, automate things, do web scraping and also build games and powerful visualizations. It is an all-rounder programming language.

**Disadvantages of Python**

So far, we’ve seen why Python is a great choice for your project. But if you choose it, you should be aware of its consequences as well. Let’s now see the downsides of choosing Python over another language.

1. **Speed Limitations**

We have seen that Python code is executed line by line. But since Python is interpreted, it often results in slow execution. This, however, isn’t a problem unless speed is a focal point for the project. In other words, unless high speed is a requirement, the benefits offered by Python are enough to distract us from its speed limitations.

**2. Weak in Mobile Computing and Browsers**

While it serves as an excellent server-side language, Python is much rarely seen on the client-side. Besides that, it is rarely ever used to implement smartphone-based applications. One such application is called Carbonnelle.

The reason it is not so famous despite the existence of Brython is that it isn’t that secure.

**3. Design Restrictions**

As you know, Python is dynamically typed. This means that you don’t need to declare the type of variable while writing the code. It uses duck-typing. But wait, what’s that? Well, it just means that if it looks like a duck, it must be a duck. While this is easy on the programmers during coding, it can raise run-time errors.

**4. Underdeveloped Database Access Layers**

Compared to more widely used technologies like JDBC (Java DataBase Connectivity) and ODBC (Open DataBase Connectivity), Python’s database access layers are a bit underdeveloped. Consequently, it is less often applied in huge enterprises.

**5. Simple**

No, we’re not kidding. Python’s simplicity can indeed be a problem. Take my example. I don’t do Java, I’m more of a Python person. To me, its syntax is so simple that the verbosity of Java code seems unnecessary.

This was all about the Advantages and Disadvantages of Python Programming Language.

**Modules Used in Project**

**NumPy**

NumPy is a general-purpose array-processing package. It provides a high-performance multidimensional array object, and tools for working with these arrays.

It is the fundamental package for scientific computing with Python. It contains various features including these important ones:

* A powerful N-dimensional array object
* Sophisticated (broadcasting) functions
* Tools for integrating C/C++ and Fortran code
* Useful linear algebra, Fourier transform, and random number capabilities

Besides its obvious scientific uses, NumPy can also be used as an efficient multi-dimensional container of generic data. Arbitrary datatypes can be defined using NumPy which allows NumPy to seamlessly and speedily integrate with a wide variety of databases.

**Pandas**

Pandas is an open-source Python Library providing high-performance data manipulation and analysis tool using its powerful data structures. Python was majorly used for data munging and preparation. It had very little contribution towards data analysis. Pandas solved this problem. Using Pandas, we can accomplish five typical steps in the processing and analysis of data, regardless of the origin of data load, prepare, manipulate, model, and analyze. Python with Pandas is used in a wide range of fields including academic and commercial domains including finance, economics, Statistics, analytics, etc.

**Matplotlib**

Matplotlib is a Python 2D plotting library which produces publication quality figures in a variety of hardcopy formats and interactive environments across platforms. Matplotlib can be used in Python scripts, the Python and Ipython shells, the Jupyter Notebook, web application servers, and four graphical user interface toolkits. Matplotlib tries to make easy things easy and hard things possible. You can generate plots, histograms, power spectra, bar charts, error charts, scatter plots, etc., with just a few lines of code. For examples, see the sample plots and thumbnail gallery.

For simple plotting the pyplot module provides a MATLAB-like interface, particularly when combined with Ipython. For the power user, you have full control of line styles, font properties, axes properties, etc, via an object oriented interface or via a set of functions familiar to MATLAB users.

**Scikit – learn**

Scikit-learn provides a range of supervised and unsupervised learning algorithms via a consistent interface in Python. It is licensed under a permissive simplified BSD license and is distributed under many Linux distributions, encouraging academic and commercial use. Python

**Install Python Step-by-Step in Windows and Mac**

Python a versatile programming language doesn’t come pre-installed on your computer devices. Python was first released in the year 1991 and until today it is a very popular high-level programming language. Its style philosophy emphasizes code readability with its notable use of great whitespace.

The object-oriented approach and language construct provided by Python enables programmers to write both clear and logical code for projects. This software does not come pre-packaged with Windows.

**How to Install Python on Windows and Mac**

There have been several updates in the Python version over the years. The question is how to install Python? It might be confusing for the beginner who is willing to start learning Python but this tutorial will solve your query. The latest or the newest version of Python is version 3.7.4 or in other words, it is Python 3.

Note: The python version 3.7.4 cannot be used on Windows XP or earlier devices.

Before you start with the installation process of Python. First, you need to know about your System Requirements. Based on your system type i.e. operating system and based processor, you must download the python version. My system type is a Windows 64-bit operating system. So the steps below are to install python version 3.7.4 on Windows 7 device or to install Python 3. Download the Python Cheatsheet here.The steps on how to install Python on Windows 10, 8 and 7 are divided into 4 parts to help understand better.

**Download the Correct version into the system**

Step 1: Go to the official site to download and install python using Google Chrome or any other web browser. OR Click on the following link: <https://www>.python.org

A screenshot of a computer

Description automatically generated with medium confidence

Now, check for the latest and the correct version for your operating system.

Step 2: Click on the Download Tab.

Graphical user interface, application

Description automatically generated

Step 3: You can either select the Download Python for windows 3.7.4 button in Yellow Color or you can scroll further down and click on download with respective to their version. Here, we are downloading the most recent python version for windows 3.7.4

Graphical user interface, application

Description automatically generated

Step 4: Scroll down the page until you find the Files option.

Step 5: Here you see a different version of python along with the operating system.

Graphical user interface, text

Description automatically generated

* To download Windows 32-bit python, you can select any one from the three options: Windows x86 embeddable zip file, Windows x86 executable installer or Windows x86 web-based installer.
* To download Windows 64-bit python, you can select any one from the three options: Windows x86-64 embeddable zip file, Windows x86-64 executable installer or Windows x86-64 web-based installer.

Here we will install Windows x86-64 web-based installer. Here your first part regarding which version of python is to be downloaded is completed. Now we move ahead with the second part in installing python i.e. Installation

Note: To know the changes or updates that are made in the version you can click on the Release Note Option.

**Installation of Python**

Step 1: Go to Download and Open the downloaded python version to carry out the installation process.

Graphical user interface, text, application

Description automatically generated

Step 2: Before you click on Install Now, Make sure to put a tick on Add Python 3.7 to PATH.

Graphical user interface, text, application, chat or text message

Description automatically generated

Step 3: Click on Install NOW After the installation is successful. Click on Close.

Graphical user interface, text, application, chat or text message

Description automatically generated

With these above three steps on python installation, you have successfully and correctly installed Python. Now is the time to verify the installation.

Note: The installation process might take a couple of minutes.

**Verify the Python Installation**

Step 1: Click on Start

Step 2: In the Windows Run Command, type “cmd”.

Graphical user interface, application

Description automatically generated

Step 3: Open the Command prompt option.

Step 4: Let us test whether the python is correctly installed. Type python –V and press Enter.

A screenshot of a computer

Description automatically generated with medium confidence

Step 5: You will get the answer as 3.7.4

Note: If you have any of the earlier versions of Python already installed. You must first uninstall the earlier version and then install the new one.

**Check how the Python IDLE works**

Step 1: Click on Start

Step 2: In the Windows Run command, type “python idle”.

Application

Description automatically generated with low confidence

Step 3: Click on IDLE (Python 3.7 64-bit) and launch the program

Step 4: To go ahead with working in IDLE you must first save the file. Click on File > Click on Save

Graphical user interface, text, application, email

Description automatically generated

Step 5: Name the file and save as type should be Python files. Click on SAVE. Here I have named the files as Hey World.

Step 6: Now for e.g. enter print (“Hey World”) and Press Enter.

Graphical user interface, text, application, email

Description automatically generated

You will see that the command given is launched. With this, we end our tutorial on how to install Python. You have learned how to download python for windows into your respective operating system.

Note: Unlike Java, Python does not need semicolons at the end of the statements otherwise it won’t work.

**CHAPTER 7**

**SYSTEM REQUIREMENTS**

**Software Requirements**

The functional requirements or the overall description documents include the product perspective and features, operating system and operating environment, graphics requirements, design constraints and user documentation.

The appropriation of requirements and implementation constraints gives the general overview of the project in regard to what the areas of strength and deficit are and how to tackle them.

* Python IDLE 3.7 version (or)
* Anaconda 3.7 (or)
* Jupiter (or)
* Google colab

**Hardware Requirements**

Minimum hardware requirements are very dependent on the particular software being developed by a given Enthought Python / Canopy / VS Code user. Applications that need to store large arrays/objects in memory will require more RAM, whereas applications that need to perform numerous calculations or tasks more quickly will require a faster processor.

* Operating system : Windows, Linux
* Processor : minimum intel i3
* Ram : minimum 4 GB
* Hard disk : minimum 250GB

**CHAPTER 8**

**FUNCTIONAL REQUIREMENTS**

**Output Design**

Outputs from computer systems are required primarily to communicate the results of processing to users. They are also used to provides a permanent copy of the results for later consultation. The various types of outputs in general are:

* External Outputs, whose destination is outside the organization
* Internal Outputs whose destination is within organization and they are the
* User’s main interface with the computer.
* Operational outputs whose use is purely within the computer department.
* Interface outputs, which involve the user in communicating directly.

**Output Definition**

The outputs should be defined in terms of the following points:

* Type of the output
* Content of the output
* Format of the output
* Location of the output
* Frequency of the output
* Volume of the output
* Sequence of the output

It is not always desirable to print or display data as it is held on a computer. It should be decided as which form of the output is the most suitable.

**Input Design**

Input design is a part of overall system design. The main objective during the input design is as given below:

* To produce a cost-effective method of input.
* To achieve the highest possible level of accuracy.
* To ensure that the input is acceptable and understood by the user.

**Input Stages**

The main input stages can be listed as below:

* Data recording
* Data transcription
* Data conversion
* Data verification
* Data control
* Data transmission
* Data validation
* Data correction

**Input Types**

It is necessary to determine the various types of inputs. Inputs can be categorized as follows:

* External inputs, which are prime inputs for the system.
* Internal inputs, which are user communications with the system.
* Operational, which are computer department’s communications to the system?
* Interactive, which are inputs entered during a dialogue.

**Input Media**

At this stage choice has to be made about the input media. To conclude about the input media consideration has to be given to;

* Type of input
* Flexibility of format
* Speed
* Accuracy
* Verification methods
* Rejection rates
* Ease of correction
* Storage and handling requirements
* Security
* Easy to use
* Portability

Keeping in view the above description of the input types and input media, it can be said that most of the inputs are of the form of internal and interactive. As

Input data is to be the directly keyed in by the user, the keyboard can be considered to be the most suitable input device.

**Error Avoidance**

At this stage care is to be taken to ensure that input data remains accurate form the stage at which it is recorded up to the stage in which the data is accepted by the system. This can be achieved only by means of careful control each time the data is handled.

**Error Detection**

Even though every effort is make to avoid the occurrence of errors, still a small proportion of errors is always likely to occur, these types of errors can be discovered by using validations to check the input data.

**Data Validation**

Procedures are designed to detect errors in data at a lower level of detail. Data validations have been included in the system in almost every area where there is a possibility for the user to commit errors. The system will not accept invalid data. Whenever an invalid data is keyed in, the system immediately prompts the user and the user has to again key in the data and the system will accept the data only if the data is correct. Validations have been included where necessary.

The system is designed to be a user friendly one. In other words the system has been designed to communicate effectively with the user. The system has been designed with popup menus.

**User Interface Design**

It is essential to consult the system users and discuss their needs while designing the user interface:

**User Interface Systems Can Be Broadly Clasified As:**

* User initiated interface the user is in charge, controlling the progress of the user/computer dialogue. In the computer-initiated interface, the computer selects the next stage in the interaction.
* Computer initiated interfaces

In the computer-initiated interfaces the computer guides the progress of the user/computer dialogue. Information is displayed and the user response of the computer takes action or displays further information.

**User Initiated Interfaces**

User initiated interfaces fall into two approximate classes:

* Command driven interfaces: In this type of interface the user inputs commands or queries which are interpreted by the computer.
* Forms oriented interface: The user calls up an image of the form to his/her screen and fills in the form. The forms-oriented interface is chosen because it is the best choice.

**Computer-Initiated Interfaces**

The following computer – initiated interfaces were used:

* The menu system for the user is presented with a list of alternatives and the user chooses one; of alternatives.
* Questions – answer type dialog system where the computer asks question and takes action based on the basis of the users reply.

Right from the start the system is going to be menu driven, the opening menu displays the available options. Choosing one option gives another popup menu with more options. In this way every option leads the users to data entry form where the user can key in the data.

**Error Message Design**

The design of error messages is an important part of the user interface design. As user is bound to commit some errors or other while designing a system the system should be designed to be helpful by providing the user with information regarding the error he/she has committed.

This application must be able to produce output at different modules for different inputs.

**Performance Requirements**

Performance is measured in terms of the output provided by the application. Requirement specification plays an important part in the analysis of a system. Only when the requirement specifications are properly given, it is possible to design a system, which will fit into required environment. It rests largely in the part of the users of the existing system to give the requirement specifications because they are the people who finally use the system. This is because the requirements have to be known during the initial stages so that the system can be designed according to those requirements. It is very difficult to change the system once it has been designed and on the other hand designing a system, which does not cater to the requirements of the user, is of no use.

The requirement specification for any system can be broadly stated as given below:

* The system should be able to interface with the existing system
* The system should be accurate
* The system should be better than the existing system
* The existing system is completely dependent on the user to perform all the duties.

**CHAPTER 9**

**SOURCE CODE**

from tkinter import messagebox

from tkinter import \*

from tkinter import simpledialog

import tkinter

from tkinter import filedialog

import matplotlib.pyplot as plt

import numpy as np

from tkinter.filedialog import askopenfilename

import pandas as pd

import os

import cv2

import numpy as np

from sklearn import svm

from sklearn.metrics import accuracy\_score

from sklearn.model\_selection import train\_test\_split

from sklearn.decomposition import PCA

from keras.utils.np\_utils import to\_categorical

from keras.layers import MaxPooling2D

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

from keras.layers import Convolution2D

from keras.models import Sequential

main = tkinter.Tk()

main.title("Detecting Lung Cancer from CT Images using SVM Classification and Comparing Survival Rates with 3D CNN")

main.geometry("1300x1200")

global filename

global classifier

global svm\_acc, cnn\_acc

global X, Y

global X\_train, X\_test, y\_train, y\_test

global pca

def uploadDataset():

global filename

filename = filedialog.askdirectory(initialdir=".")

text.delete('1.0', END)

text.insert(END,filename+" loaded\n");

def splitDataset():

global X, Y

global X\_train, X\_test, y\_train, y\_test

global pca

text.delete('1.0', END)

X = np.load('features/X.txt.npy')

Y = np.load('features/Y.txt.npy')

X = np.reshape(X, (X.shape[0],(X.shape[1]\*X.shape[2]\*X.shape[3])))

pca = PCA(n\_components = 100)

X = pca.fit\_transform(X)

print(X.shape)

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, Y, test\_size=0.2)

text.insert(END,"Total CT Scan Images Found in dataset : "+str(len(X))+"\n")

text.insert(END,"Train split dataset to 80% : "+str(len(X\_train))+"\n")

text.insert(END,"Test split dataset to 20% : "+str(len(X\_test))+"\n")

def executeSVM():

global classifier

global svm\_acc

text.delete('1.0', END)

cls = svm.SVC()

cls.fit(X\_train, y\_train)

predict = cls.predict(X\_test)

svm\_acc = accuracy\_score(y\_test,predict) \* 100

classifier = cls

text.insert(END,"SVM Accuracy : "+str(svm\_acc)+"\n")

def executeCNN():

global cnn\_acc

X = np.load('features/X.txt.npy')

Y = np.load('features/Y.txt.npy')

Y = to\_categorical(Y)

classifier = Sequential()

classifier.add(Convolution2D(32, 3, 3, input\_shape = (64, 64, 3), activation = 'relu'))

classifier.add(MaxPooling2D(pool\_size = (2, 2)))

classifier.add(Convolution2D(32, 3, 3, activation = 'relu'))

classifier.add(MaxPooling2D(pool\_size = (2, 2)))

classifier.add(Flatten())

classifier.add(Dense(output\_dim = 256, activation = 'relu'))

classifier.add(Dense(output\_dim = 2, activation = 'softmax'))

print(classifier.summary())

classifier.compile(optimizer = 'adam', loss = 'categorical\_crossentropy', metrics = ['accuracy'])

hist = classifier.fit(X, Y, batch\_size=16, epochs=12, shuffle=True, verbose=2)

hist = hist.history

acc = hist['accuracy']

cnn\_acc = acc[9] \* 100

text.insert(END,"CNN Accuracy : "+str(cnn\_acc)+"\n")

def predictCancer():

filename = filedialog.askopenfilename(initialdir="testSamples")

img = cv2.imread(filename)

img = cv2.resize(img, (64,64))

im2arr = np.array(img)

im2arr = im2arr.reshape(64,64,3)

im2arr = im2arr.astype('float32')

im2arr = im2arr/255

test = []

test.append(im2arr)

test = np.asarray(test)

test = np.reshape(test, (test.shape[0],(test.shape[1]\*test.shape[2]\*test.shape[3])))

test = pca.transform(test)

predict = classifier.predict(test)[0]

msg = ''

if predict == 0:

msg = "Uploaded CT Scan is Normal"

if predict == 1:

msg = "Uploaded CT Scan is Abnormal"

img = cv2.imread(filename)

img = cv2.resize(img, (400,400))

cv2.putText(img, msg, (10, 25), cv2.FONT\_HERSHEY\_SIMPLEX,0.7, (0, 255, 255), 2)

cv2.imshow(msg, img)

cv2.waitKey(0)

def graph():

height = [svm\_acc, cnn\_acc]

bars = ('SVM Accuracy','CNN Accuracy')

y\_pos = np.arange(len(bars))

plt.bar(y\_pos, height)

plt.xticks(y\_pos, bars)

plt.show()

font = ('times', 14, 'bold')

title = Label(main, text='Detecting Lung Cancer from CT Images using SVM Classification and Comparing Survival Rates with 3D CNN')

title.config(bg='deep sky blue', fg='white')

title.config(font=font)

title.config(height=3, width=120)

title.place(x=0,y=5)

font1 = ('times', 12, 'bold')

text=Text(main,height=20,width=150)

scroll=Scrollbar(text)

text.configure(yscrollcommand=scroll.set)

text.place(x=50,y=120)

text.config(font=font1)

font1 = ('times', 13, 'bold')

uploadButton = Button(main, text="Upload Lung Cancer Dataset", command=uploadDataset)

uploadButton.place(x=50,y=525)

uploadButton.config(font=font1)

readButton = Button(main, text="Read & Split Dataset", command=splitDataset)

readButton.place(x=325,y=525)

readButton.config(font=font1)

svmButton = Button(main, text="SVM Classifier", command=executeSVM)

svmButton.place(x=550,y=525)

svmButton.config(font=font1)

kmeansButton = Button(main, text="3D CNN Model", command=executeCNN)

kmeansButton.place(x=750,y=525)

kmeansButton.config(font=font1)

predictButton = Button(main, text="Predict Lung Cancer", command=predictCancer)

predictButton.place(x=950,y=525)

predictButton.config(font=font1)

graphButton = Button(main, text="Accuracy Graph", command=graph)

graphButton.place(x=50,y=575)

graphButton.config(font=font1)

main.config(bg='LightSteelBlue3')

main.mainloop()

**CHAPTER 10**

**RESULTS AND DISCUSSION**

**Implementation description**

This project implements a graphical user interface (GUI) application using the Tkinter library for detecting lung cancer from Computed Tomography (CT) images. The application uses two different classification techniques: Support Vector Machine (SVM) and Convolutional Neural Network (CNN) to classify CT scan images as either normal or abnormal (indicative of lung cancer). The main functionality of this application involves loading a dataset of CT scan images, splitting it into training and testing sets, training either an SVM or a CNN model, and then using the trained model to predict the status of new CT scan images. The accuracy of the models is also displayed, and the user can generate an accuracy graph. Here's the step-by-step implementation:

1. Importing necessary libraries:

* tkinter is used for creating the GUI.
* messagebox is used for displaying message boxes.
* matplotlib.pyplot is used for creating plots.
* numpy is used for numerical operations.
* sklearn is used for machine learning tasks like SVM classification, accuracy calculation, and data splitting.
* pandas is used for handling data.
* os is used for file and directory operations.
* cv2 (OpenCV) is used for image processing.
* keras is used for building the CNN model.

2. Creating the main GUI window:

* The Tkinter Tk() function is used to create the main GUI window.
* The title and geometry of the window are set.

3. Defining global variables: Several global variables are defined to store data and model information.

4. Functions:

* uploadDataset(): Opens a file dialog to select a directory containing the dataset.
* splitDataset(): Loads and splits the dataset into training and testing sets using PCA for dimensionality reduction.
* executeSVM(): Trains an SVM classifier on the training data and calculates its accuracy on the test data.
* executeCNN(): Defines and trains a CNN model on the dataset and calculates its accuracy.
* predictCancer(): Allows the user to select an image, preprocess it, and use the trained model (either SVM or CNN) to predict whether the image is normal or abnormal.
* graph(): Plots a bar graph comparing the accuracy of the SVM and CNN models.

5. GUI components:

* Labels, buttons, and text boxes are created and positioned on the GUI window to interact with the user.
* Buttons have associated functions to perform actions when clicked.

6. Main event loop: The mainloop() function starts the main event loop, which handles user interactions and GUI updates.

**Results description**

This project uses a total of 138 CT scan images out of which 80% i.e., 110 images are used for training and 20% i.e., 28 images are used for testing the SVM, and CNN models for detecting the normal, and cancer from test CT scan images. It uses the Tkinter library for creating a graphical user interface (GUI) to perform lung cancer detection and classification tasks using Support Vector Machine (SVM) and Convolutional Neural Network (CNN) models. Figure 1 shows an illustration of the GUI of the lung cancer detection system. It includes buttons, labels, and other GUI elements designed for user interaction. Figure 2 displays the user interface after the dataset has been loaded and split into training and testing sets. It shows the dataset statistics related to data splitting. Figure 3 presents the accuracy achieved by the SVM classifier when used for lung cancer detection.

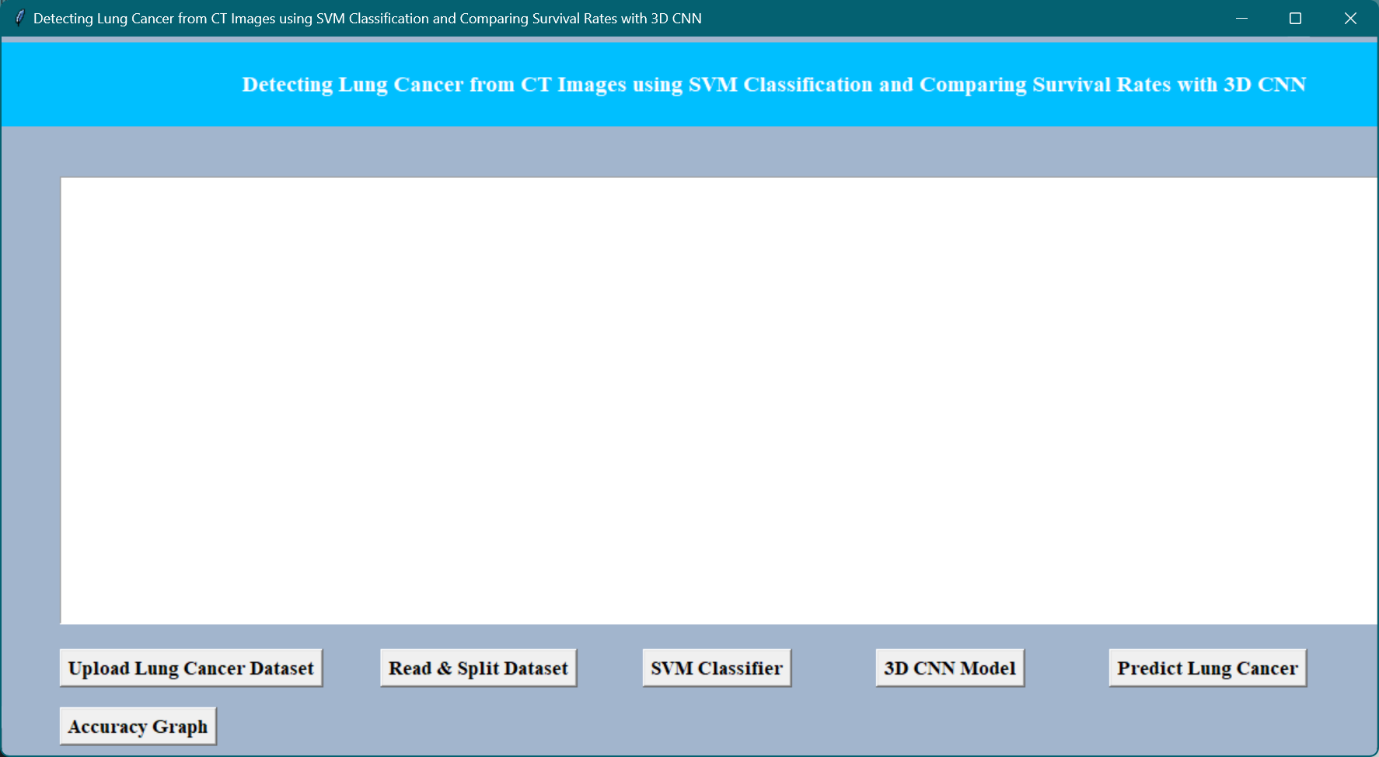


Figure 1: Illustration of UI application developed for proposed lung cancer detection system.

A screenshot of a computer

Description automatically generated

Figure 2: Demonstration of UI application after applying data splitting.

Figure 4 provides a summary of the architecture of the proposed 3D CNN model. It shows the layers, and structure of the CNN used for lung cancer detection. Figure 5 displays plots depicting how the loss and accuracy of the 3D CNN model change with increasing epochs during training. It helps to visualize the model's training progress.

A screenshot of a computer

Description automatically generated

Figure 3: Obtained accuracy of SVM classifier for lung cancer detection.

A screenshot of a computer program

Description automatically generated

Figure 4: Summary of proposed 3D CNN model.

A screenshot of a computer

Description automatically generated

Figure 5: Obtained loss, and accuracy values with increasing in epochs for proposed 3D CNN model.

A screenshot of a computer

Description automatically generated

Figure 6: Illustration of UI application with the resulted accuracy of SVM, and CNN models for lung cancer detection from CT scan images.

Figure 6 shows the user interface after the SVM and CNN models have been trained, displaying their respective accuracy results. It provides a summary of the system's performance. Figure 7 presents an example of a prediction outcome generated by the 3D CNN model. It includes an image indicating whether a CT scan is normal or abnormal based on the model's prediction. Figure 8 shows a comparative analysis of the accuracy achieved by the SVM and 3D CNN models for lung cancer detection, where the proposed 3D CNN achieved enhanced accuracy i.e., 97.82% while the SVM obtained 78.57% of accuracy.

A screenshot of a cellphone

Description automatically generated A screenshot of a cell phone

Description automatically generated

A screenshot of a medical scan

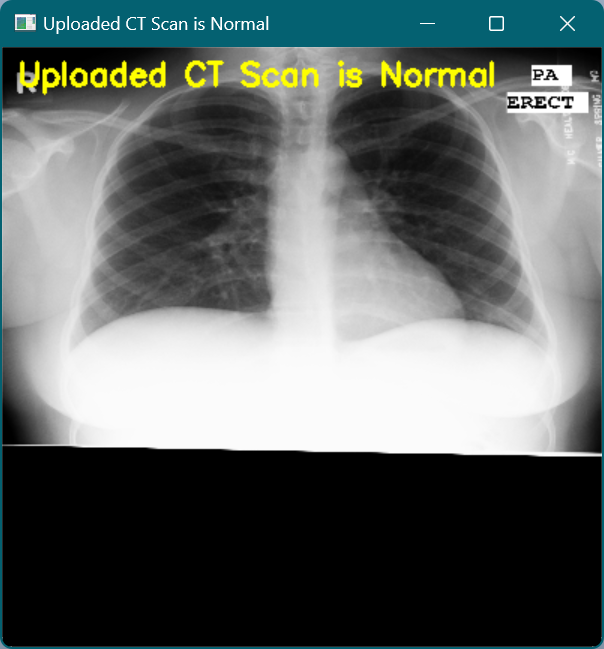
Description automatically generated 

Figure 7: Sample prediction outcome obtained using proposed 3D CNN model.

A screenshot of a computer

Description automatically generated

Figure 8: Performance comparison of accuracy for lung cancer detection system using SVM, and proposed 3D CNN model.

**CHAPTER 11**

**CONCLUSIONS AND FUTURE SCOPE**

This work implements an advanced CNN modifications for the detection of lung cancer from CT scan images. The higher accuracy of the 3D CNN can be attributed to its ability to automatically learn relevant features from the images through multiple convolutional layers, which is especially advantageous for complex and high-dimensional data like CT scans. On the other hand, SVM relies on handcrafted features and may struggle with capturing complex patterns in the data. Based on the obtained results, it is clear that the proposed 3D CNN model significantly outperformed the SVM classifier in the task of detecting lung cancer from CT images. The 3D CNN achieved an accuracy of 97.82%, whereas the SVM achieved an accuracy of 78.57%. This indicates that the CNN model is much more effective at classifying CT scan images as either normal or abnormal, which is crucial for the early detection of lung cancer.

In conclusion, the promising results obtained by the 3D CNN model in lung cancer detection highlight the potential of deep learning in medical image analysis. Future endeavors should focus on refining model performance, scalability, and seamless integration into clinical workflows, with the ultimate goal of positively impacting patient care and outcomes across various medical domains.

**REFERENCES**

1. Y. Xu, A. Hosny, R. Zeleznik, C. Parmar, T. Coroller, I. Franco, H.J. Aerts, Deep learning predicts lung cancer treatment response from serial medical imaging, Clin. Cancer Res. 25 (11) (2019) 3266–3275.
2. M.I. Faisal, S. Bashir, Z.S. Khan, F.H. Khan, An evaluation of machine learning classifiers and ensembles for early stage prediction of lung cancer, December, in: 2018 3rd International Conference on Emerging Trends in Engineering, Sciences and Technology (ICEEST), IEEE, 2018, pp. 1–4.
3. N. Coudray, P.S. Ocampo, T. Sakellaropoulos, N. Narula, M. Snuderl, D. Feny¨o, A. Tsirigos, Classification and mutation prediction from non–small cell lung cancer histopathology images using deep learning, Nat. Med. 24 (10) (2018) 1559–1567.
4. D.M. Ibrahim, N.M. Elshennawy, A.M. Sarhan, Deep-chest: multi-classification deep learning model for diagnosing COVID-19, pneumonia, and lung cancer chest diseases, Comput. Biol. Med. 132 (2021), 104348.
5. Avanzo, J. Stancanello, G. Pirrone, G. Sartor, Radiomics and deep learning in lung cancer, Strahlenther. Onkol. 196 (10) (2020) 879–887.
6. S.H. Hyun, M.S. Ahn, Y.W. Koh, S.J. Lee, A machine-learning approach using PET- based radiomics to predict the histological subtypes of lung cancer, Clin. Nucl. Med. 44 (12) (2019) 956–960.
7. P. R. Radhika, R.A. Nair, G. Veena, A comparative study of lung cancer detection using machine learning algorithms, 2019, February, in: IEEE International Conference on Electrical, Computer and Communication Technologies (ICECCT), IEEE, 2019, pp. 1–4.
8. K. Pradhan, P. Chawla, Medical Internet of things using machine learning algorithms for lung cancer detection, J. Manag. Anal. 7 (4) (2020) 591–623.
9. S. Bhatia, Y. Sinha, L. Goel, Lung cancer detection: a deep learning approach, in: Soft Computing for Problem Solving, Springer, Singapore, 2019, pp. 699–705.
10. H. Shin, S. Oh, S. Hong, M. Kang, D. Kang, Y.G. Ji, Y. Choi, Early-stage lung cancer diagnosis by deep learning-based spectroscopic analysis of circulating exosomes, ACS Nano 14 (5) (2020) 5435–5444.
11. V. Rajasekar, B. Predi´c, M. Saracevic, M. Elhoseny, D. Karabasevic, D. Stanujkic, P. Jayapaul, Enhanced multimodal biometric recognition approach for smart cities based on an optimized fuzzy genetic algorithm, Sci. Rep. 12 (1) (2022) 1–11.
12. S.H. Hyun, M.S. Ahn, Y.W. Koh, S.J. Lee, A machine-learning approach using PET- based radiomics to predict the histological subtypes of lung cancer, Clin. Nucl. Med. 44 (12) (2019) 956–960.
13. Y. She, Z. Jin, J. Wu, J. Deng, L. Zhang, H. Su, C. Chen, Development and validation of a deep learning model for non–small cell lung cancer survival, JAMA Netw. Open 3 (6) (2020) e205842, e205842.
14. W. Rahane, H. Dalvi, Y. Magar, A. Kalane, S. Jondhale, Lung cancer detection using image processing and machine learning healthcare, 2018, March, in: International Conference on Current Trends towards Converging Technologies (ICCTCT), IEEE, 2018, pp. 1–5.
15. V. Rajasekar, S. Krishnamoorthi, M. Saraˇcevi´c, D. Pepic, M. Zajmovic, H. Zogic, Ensemble machine learning methods to predict the balancing of ayurvedic constituents in the human body: ensemble machine learning methods to predict, Comput. Sci. 23 (1) (2022), https://doi.org/10.7494/csci.2022.23.1.4315.