RUTGERS

THE STATE UNIVERSITY OF NEW JERSEY

FINAL PROJECT REPORT

Lung Cancer Diagnosis using Convolutional Neural Networks

Team members

Shourya Thatha Ravi (st1037)

Rahul Rajesh Singh (rs2050)

TABLE OF CONTENTS

ABSTRACT

The current standard method of diagnosing lung cancer is a time-consuming and error- prone process that requires doctors to visually examine CT scan images for small nodules and then classify them as benign or malignant. The proposed project aims to develop an AI-based solution that can accurately and efficiently inspect CT images and classify them in a matter of minutes. The primary objective of this project is to reduce the time and effort required for lung cancer diagnosis, while also minimizing the possibility of human errors. We plan to achieve this by utilizing the Convolutional Neural Networks (CNN). We will be using sklearn, tensor flow (keras), matplotlib, pandas and numpy modules in our final project.

INTRODUCTION

Lung cancer remains a major health challenge globally, contributing to a significant number of cancer-related deaths annually. Early and accurate diagnosis plays a critical role in determining appropriate treatment strategies and improving patient outcomes. In recent years, the combination of advanced medical imaging technology and machine learning techniques has opened new possibilities for enhancing the diagnostic process. Convolutional Neural Networks (CNNs), a class of deep learning models, have emerged as a promising tool for assisting in the diagnosis of lung cancer. By harnessing the power of CNNs to analyze medical images, researchers and clinicians can potentially improve the accuracy, efficiency, and accessibility of lung cancer diagnosis, thereby leading to more effective interventions and better patient care.

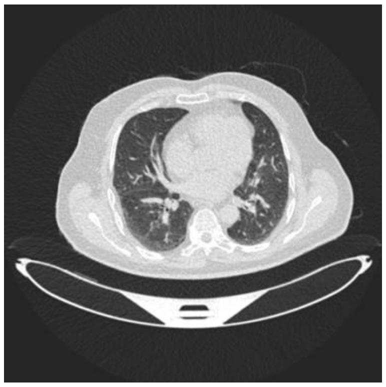
The field of medical imaging has witnessed tremendous advancements with the advent of technologies like computed tomography (CT) scans. CT scans provide detailed cross-sectional images of the lungs, enabling radiologists to identify suspicious lesions or nodules that may indicate the presence of lung cancer. However, the interpretation of these images can be complex and time-consuming, relying heavily on the expertise and experience of radiologists. This is where CNNs come into play.

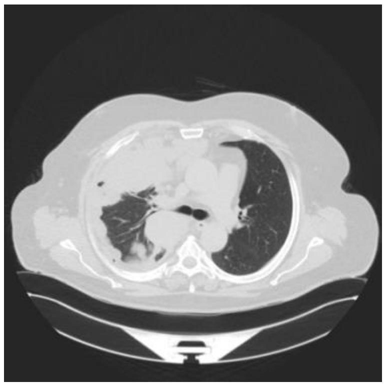
CNNs are specifically designed to process and analyze visual data, making them well-suited for medical image analysis tasks. Unlike traditional machine learning algorithms, CNNs can automatically learn hierarchical features directly from the input data, allowing them to capture intricate patterns and spatial relationships within images. This ability to extract meaningful information from complex medical images makes CNNs particularly advantageous in the context of lung cancer diagnosis.

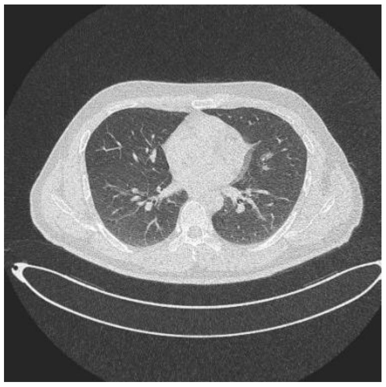
The application of Convolutional Neural Networks in diagnosing lung cancer represents a significant advancement in the field of medical image analysis. By leveraging the power of deep learning, CNNs offer the ability to enhance the accuracy and efficiency of lung cancer diagnosis, empowering healthcare professionals with valuable insights and potentially improving patient care. As research in this field continues to progress, CNNs have the potential to revolutionize lung cancer diagnosis and contribute to the ongoing fight against this devastating disease.

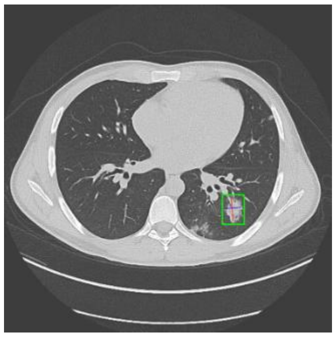
DATASET

- We have a dataset of 1190 CT scan images that represent 110 cases. The cases are classified into three categories, normal, benign, and malignant. Among the cases, 40 are diagnosed as malignant, 15 as benign, and 55 as normal.
- These images will be used to train and test the deep learning model to classify new CT scan images based on the presence or absence of cancerous nodules.
- Each scan contains several slices. The number of these slices range from 80 to 200 slices, each of them represents an image of the human chest with different sides and angles.
- The 110 cases vary in gender, age, educational attainment, area of residence, and living status.
- The CT scans in this study were initially obtained in DICOM format using a Siemens SOMATOM scanner. The CT protocol involved using a voltage of 120 kV and a slice thickness of 1 mm. The window width ranged from 350 to 1200 HU, with a window center from 50 to 600, for optimal image interpretation.
- The scans were acquired during breath-hold at full inspiration to ensure consistent image quality. Before any analysis was conducted, all images were de-identified to safeguard patient privacy.


Benign Case


Malignant case


Normal case


Test case of lung cancer of patients

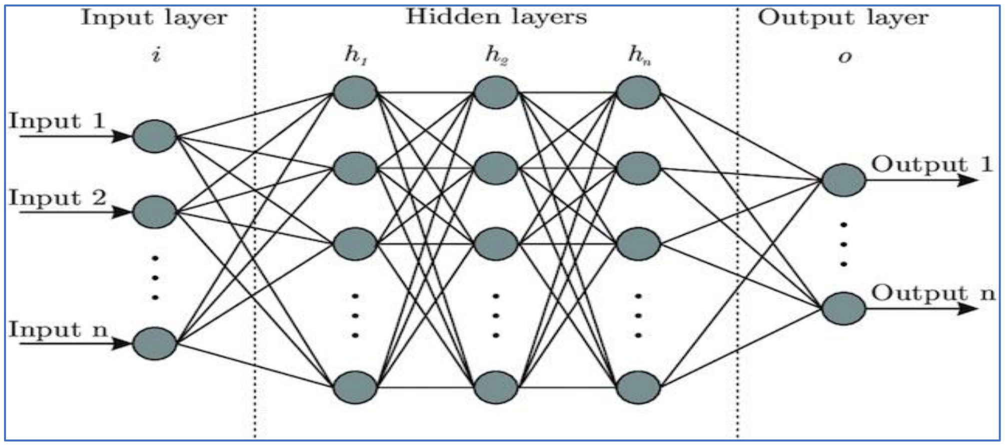
OVERVIEW OF NEURAL NETWORKS

Neural networks, inspired by the complex structure and functionality of the human brain, have emerged as a powerful tool for solving complex problems across various domains. These artificial intelligence models are designed to simulate the behavior of interconnected neurons, enabling them to learn from data, recognize patterns, and make predictions or decisions. Neural networks have revolutionized numerous fields, including image and speech recognition, natural language processing, and predictive analytics. In this project report, we explore the application of neural networks in tackling a specific problem, highlighting their capabilities, advantages, and potential impact.

Neural networks, also known as artificial neural networks or simply NNs, consist of interconnected layers of artificial neurons, also referred to as nodes or units. Each node receives inputs, performs a computation, and generates an output, which is then passed to subsequent layers. The strength of connections, known as weights, determines the influence of inputs on node computations. By iteratively adjusting these weights based on training data, neural networks can learn and generalize patterns from the given input-output relationships.

The training process of neural networks involves presenting a labeled dataset, often referred to as the training set, to the network. The network learns by comparing its predicted outputs with the known labels and adjusting the weights accordingly to minimize the difference between the predicted and desired outputs. This iterative optimization process, known as backpropagation, enables the neural network to gradually improve its performance over time.

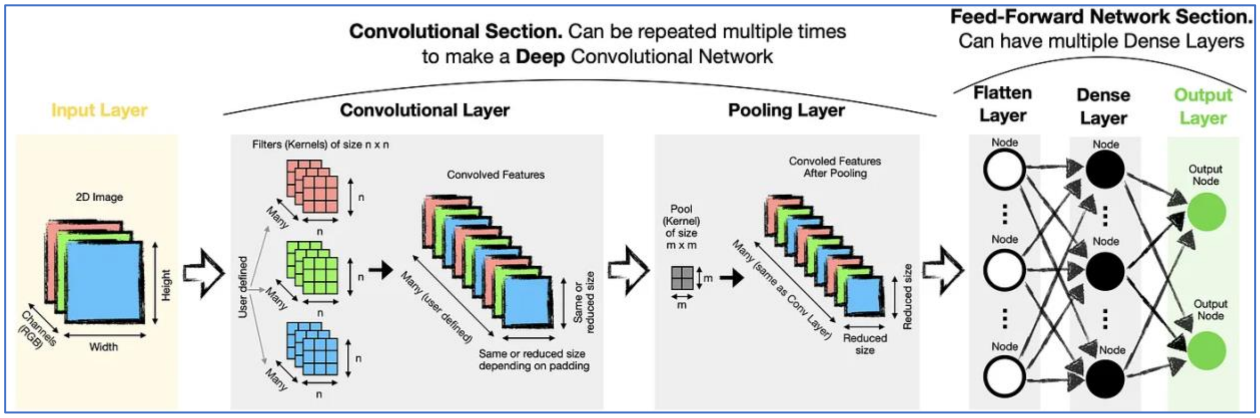
Neural networks excel at solving complex problems that are difficult to capture using traditional rule-based algorithms. They have the ability to automatically extract features from raw data, allowing them to uncover hidden patterns and relationships. This characteristic makes them particularly effective in tasks such as image and speech recognition, where the input data contains intricate patterns and structures.



A neural network with n hidden layers and n inputs

CONVOLUTIONAL NEURAL NETWORKS

Convolutional Neural Networks (CNNs) have become a prominent and effective approach in the field of computer vision and image analysis. With their ability to automatically learn and extract intricate features from visual data, CNNs have revolutionized various applications such as image classification, object detection, and semantic segmentation. In this project report, we explore the application of CNNs to solve a specific problem, outlining the architecture, training process, and evaluation metrics used. By leveraging the power of CNNs, we aim to achieve accurate and robust results in our project.


Convolutional Section. Can be repeated multiple times to make a Deep Convolutional Network
Feed-Forward Network Section. Can have multiple Dense Layers
Input Layer
Convolutional Layer
Pooling Layer
Flatten Layer
Dense Layer
Output Layer
2D Image
Height
Width
Channels (RGB)
Filters (Kernels) of size n x n
User defined
Many
n
Convolved Features
Many (user defined)
Same or reduced size
Same or reduced size depending on padding
Pool (Kernel) of size m x m
m
Convoled Features After Pooling
Many (same as Conv Layer)
Reduced size
Node
Output Node

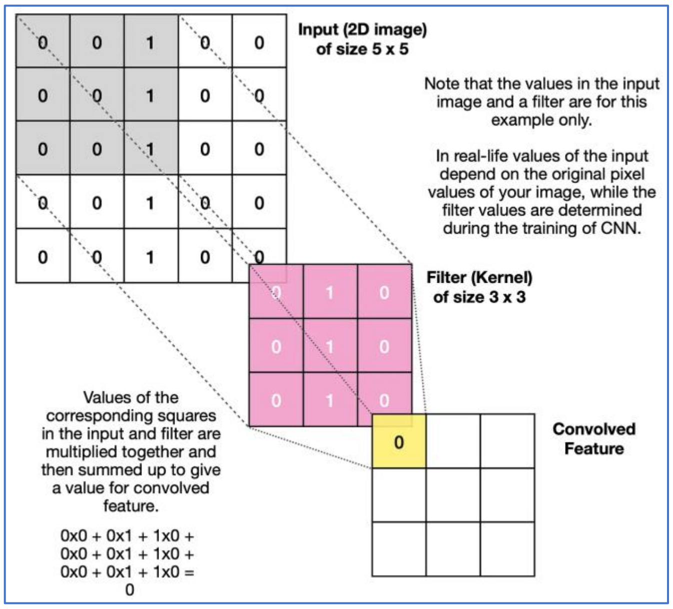
Convolutional Neural Network architecture

CNN Architecture

In a CNN architecture, the main building blocks are convolutional layers, pooling layers, and fully connected layers:

Convolutional Layers

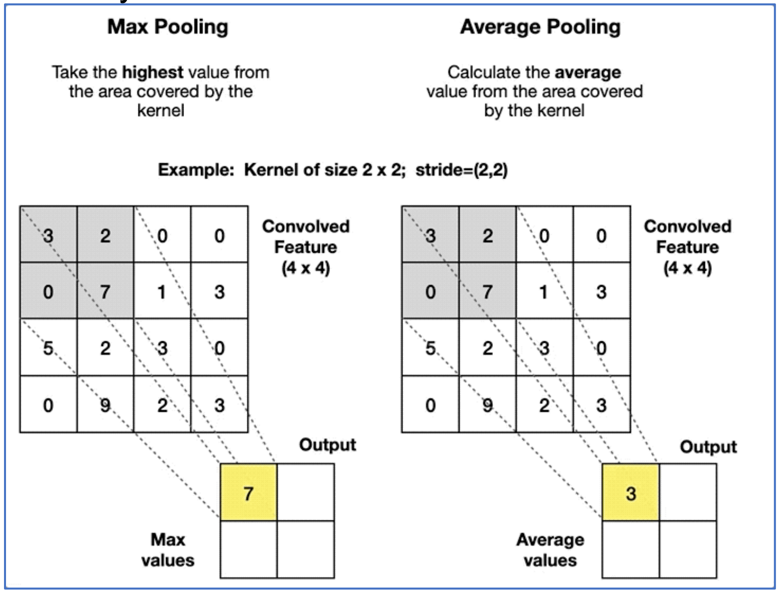
Convolutional layers are responsible for feature extraction from input images. They consist of a set of learnable filters (also known as kernels or feature detectors) that are convolved with the input image to produce feature maps. Each filter detects specific patterns or features in the input, such as edges, corners, or textures. Multiple filters are typically applied in parallel to capture various types of features. Convolutional layers allow the network to learn local and spatially invariant features, capturing both low-level and high-level information.


Input (2D image) of size 5 x 5
Note that the values in the input image and a filter are for this example only.
In real-life values of the input depend on the original pixel values of your image, while the filter values are determined during the training of CNN.
Filter (Kernel) of size 3 x 3
Convolved Feature
Values of the corresponding squares in the input and filter are multiplied together and then summed up to give a value for convolved feature.
0x0 + 0x1 + 1x0 + 0x0 + 0x1 + 1x0 + 0x0 + 0x1 + 1x0 = 0

Convolution layer

Pooling Layers

Pooling layers are used to reduce the spatial dimensions of the feature maps generated by the convolutional layers. The most commonly used pooling operation is max pooling, which selects the maximum value within a small neighborhood. Max pooling helps to down sample the feature maps, reducing their size while retaining the most salient features. By reducing the spatial dimensions, pooling layers enable the network to focus on the most important information and increase computational efficiency.


Max Pooling
Take the highest value from the area covered by the kernel
Average Pooling
Calculate the average value from the area covered by the kernel
Example: Kernel of size 2 x 2; stride=(2,2)
Convolved Feature (4 x 4)
Output
Max values
Average values

Max Pooling and Average Pooling

Fully Connected Layers

After the convolutional and pooling layers, the fully connected layers are introduced. These layers are similar to those in traditional artificial neural networks. Each node in a fully connected layer is connected to every node in the previous layer. The fully connected layers receive the flattened feature maps as input and perform classification or regression tasks based on the learned representations. These layers help in capturing global dependencies and making high-level predictions based on the extracted features.

Activation Functions

Activation functions are applied after each layer to introduce non-linearity into the network. Common activation functions used in CNNs include Rectified Linear Unit (ReLU), which sets negative values to zero and keeps positive values unchanged, and variants such as Leaky ReLU or Exponential Linear Units (ELUs). Activation functions are crucial for introducing non-linearities, enabling the network to learn complex relationships between features.

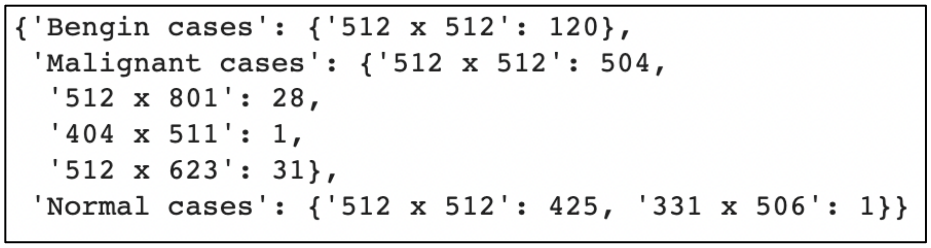
The overall architecture of a CNN is typically designed with alternating convolutional and pooling layers, followed by one or more fully connected layers for classification or regression tasks. The number of layers, filter sizes, and the depth of the network depend on the complexity of the problem and the available computational resources. Architectural choices are often made based on empirical studies and domain-specific requirements.

CNN architectures leverage convolutional layers for local feature extraction, pooling layers for downsampling and feature selection, and fully connected layers for global processing and decision-making. The interplay between these layers enables CNNs to learn hierarchical representations from raw input data, making them highly effective for image analysis and computer vision tasks.

DATA PREPARATION

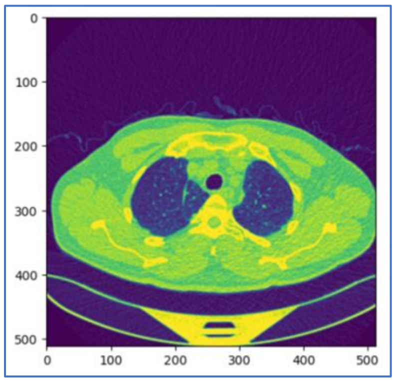
SMOTE

We first checked the image size variation for benign, malignant and normal cases respectively. There are about 120 benign, 564 malignant and 426 normal images in our dataset.

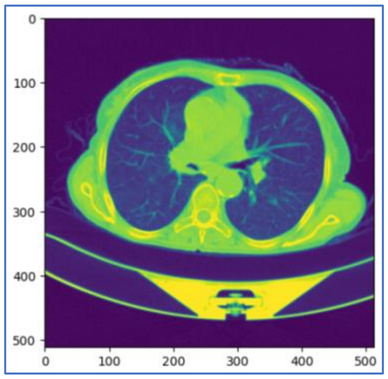


```
{'Bengin cases': {'512 x 512': 120},
 'Malignant cases': {'512 x 512': 504,
  '512 x 801': 28,
  '404 x 511': 1,
  '512 x 623': 31},
 'Normal cases': {'512 x 512': 425, '331 x 506': 1}}
```

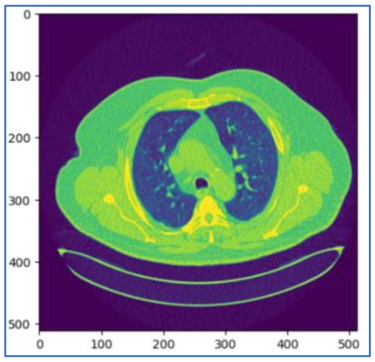
Number of images and their dimensions



Benign case



Malignant case



Normal case

To tackle imbalanced datasets, one method is to address the minority class through oversampling. The straightforward approach involves duplicating instances from the minority class, which doesn't introduce new information to the model. Alternatively, a data augmentation technique called Synthetic Minority Oversampling Technique (SMOTE) can be employed to generate synthetic examples based on existing ones from the minority class.

SMOTE

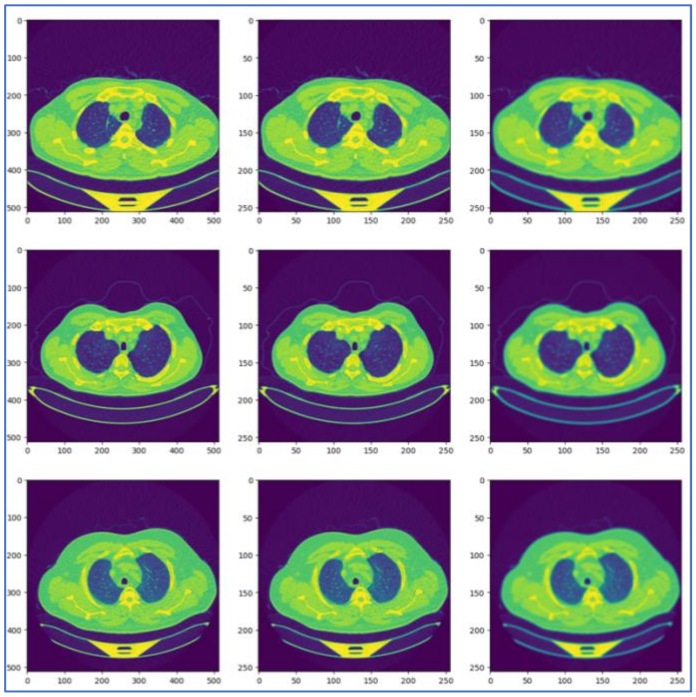
SMOTE operates by identifying closely located examples in the feature space, establishing a line connecting these examples, and generating a new sample along that line. The process begins by randomly selecting an instance from the minority class. Subsequently, a set of k nearest neighbors is determined for that particular instance (usually k=5). From this group, one neighbor is randomly chosen, and a synthetic example is formed at a randomly selected position along the line connecting the initial instance and the selected neighbor in the feature space. A module ‘imblearn’ on Python is used to implement SMOTE.

Reshaping

As seen previously in the Dimensions Figure, we noticed that the images had varying sizes, resulting in another imbalance. Some functions used ahead require images to be of the same dimension. In order to deal with it, we used the “resize” function to adjust the image dimensions to a consistent size of 256 x 256 pixels.

This reduction in size was done to make the computations less cumbersome and balance the images prepared.

Gaussian Blur
Gaussian Blur is a method that smooths images by getting rid of noise or high frequency components (usually pixels where there's unusual change in values, like that of an edge). Applying a Gaussian Blur, smoothens the image and gets rid of potential irregularities, so that the algorithm can focus on details which matter. This is achieved by applying a Gaussian filter to each image, which calculates a weighted average of nearby pixels based on a kernel size. The weights assigned to each pixel are based on a Gaussian curve, which gives more weight to pixels that are closer and less weight to those that are further away. This results in a low-pass filter that helps to smooth out the image and eliminate any noise that might be present.


Original Images followed by Resized and Smoothed Versions

CNN IMPLEMENTATION

Before building a CNN model, it's important to understand the variables and hyperparameters that can be adjusted to affect the model's performance in terms of accuracy and training time. Some of the key variables include:

- **Number of Layers**: A CNN model typically consists of multiple layers, and the complexity of the dataset can determine the number of layers needed.

Each Convolution Layer is followed by a Pooling Layer, which reduces the dimensionality and retains important features. These layers can be added repeatedly until the desired output is achieved, based on the desired level of accuracy or complexity.

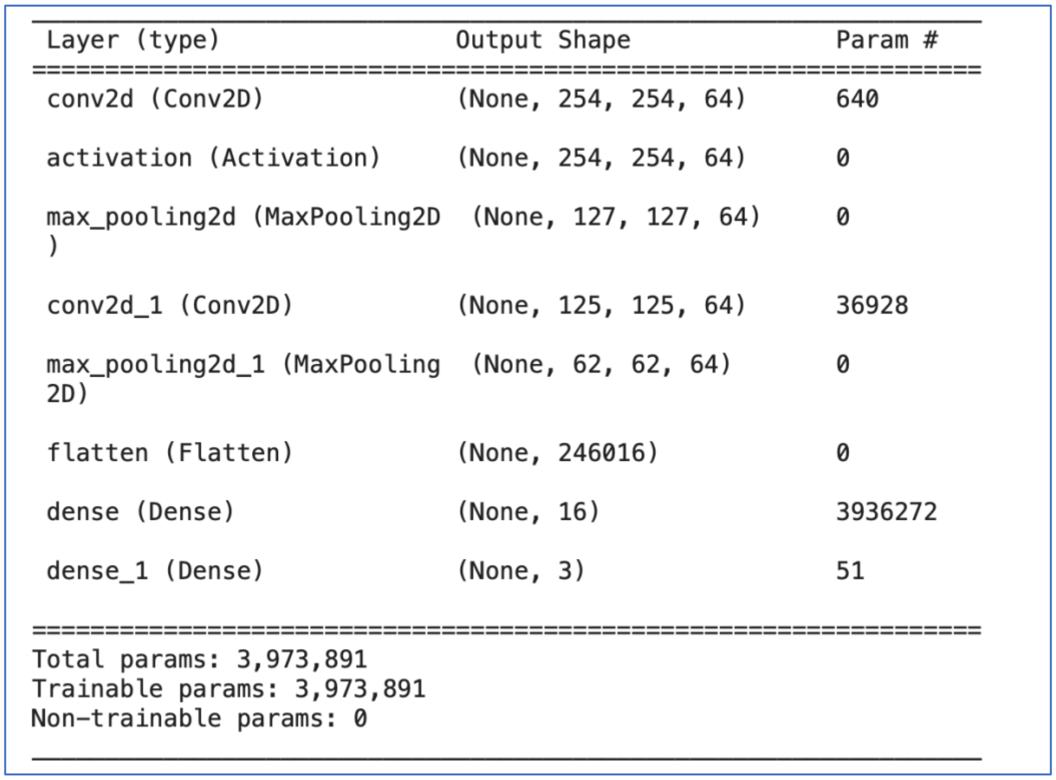
- **Batch Size**: The dataset is usually divided into batches to make the computation feasible. The batch size determines how many inputs are considered during each iteration. A larger batch size can be computationally intensive, but more accurate, and vice versa.
- **Epochs**: The number of epochs determines how many times the model iterates through the dataset during training. A higher number of epochs can lead to overfitting, so it's important to find a balance between training time and model performance.
- **Learning Rate**: The learning rate determines the step size at which the model adjusts its weights during training. A higher learning rate can lead to faster convergence but also the possibility of overshooting the optimal weights, while a lower learning rate may take longer to converge but results in more accurate weights. We used ADAM, a popular optimizer that automatically adjusts the learning rate based on the past gradients, making it more efficient than manually tuning the learning rate.
- **Loss Function**: The loss function we used in the model is a variant of categorical cross-entropy, known as sparse categorical cross-entropy, which is suitable for multi-class classification problems where the labels are integers instead of one-hot encoded vectors.

By varying these parameters, we can set up and compare multiple models to find the best one.

CNN MODELS

Model 1: Imbalanced Data

We started by splitting the dataset in training and testing sets, followed by training the model on the imbalanced data as a baseline model. The model was initialized with two convolution layers, a batch size of 8 and a total of 5 epochs. The model summary is shown below:

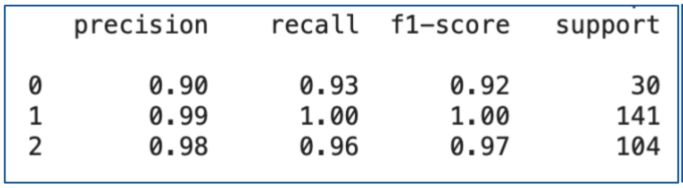


```
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 254, 254, 64)      640

 activation (Activation)     (None, 254, 254, 64)      0

 max_pooling2d (MaxPooling2D  (None, 127, 127, 64)     0
 )

 conv2d_1 (Conv2D)           (None, 125, 125, 64)      36928

 max_pooling2d_1 (MaxPooling  (None, 62, 62, 64)       0
 2D)

 flatten (Flatten)           (None, 246016)            0

 dense (Dense)               (None, 16)                3936272

 dense_1 (Dense)             (None, 3)                 51

=================================================================
Total params: 3,973,891
Trainable params: 3,973,891
Non-trainable params: 0
_________________________________________________________________
```

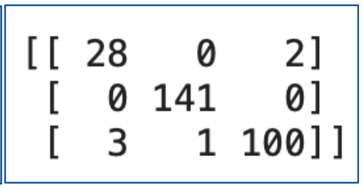
Summary of CNN Model fitted on Imbalanced Data

In the image shown above, we note that the convolution layer deploys a total of 64 filters (manually chosen by us). As previously explained, the pooling layers reduce the sizes of the images as is seen in reduced output shape, which changes the previous image size of 254x254 to 127x127 size.
Additionally, we notice the obvious reduction of dimensions in the flatten layer.
Finally, the ultimate dense layer has 3 outputs, each corresponding to the possibility of an image belonging to each of the classes. These will be observed in all the models depending on the hyperparameters chosen.

The model's test results are shown below:



```
     precision    recall  f1-score   support

0         0.90      0.93      0.92        30
1         0.99      1.00      1.00       141
2         0.98      0.96      0.97       104
```

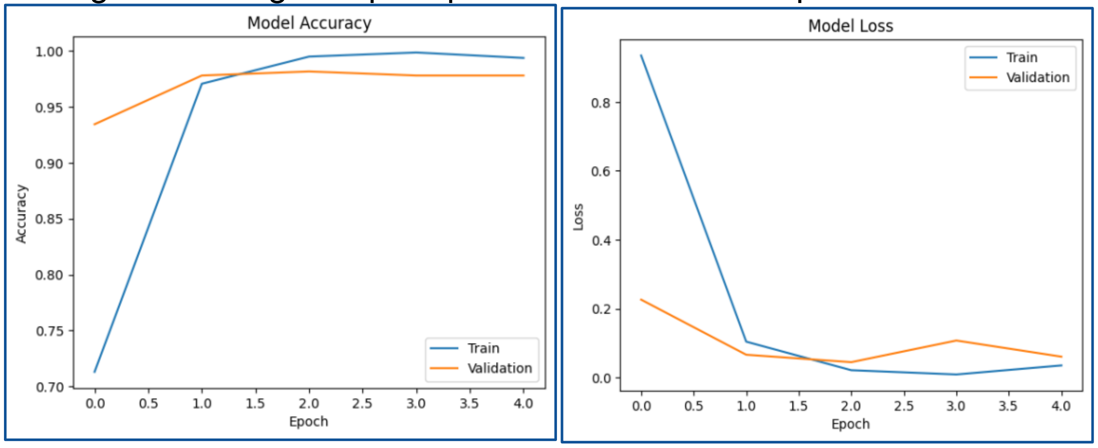


```
[[ 28   0   2]
 [  0 141   0]
 [  3   1 100]]
```

Results per Class & Confusion Matrix of Model 1

The classes in the confusion matrix are arranged in an alphabetical order of Benign, Malignant and Normal Case, with the rows representing the true classes and columns the predicted classes.

Here, we notice that the Model was able to classify the images with good enough power but misclassified 4 instances of Class 2 or “Normal” and 2 instances of Class 1 or Benign.
The training and testing loss per epoch are shown in the plots below:

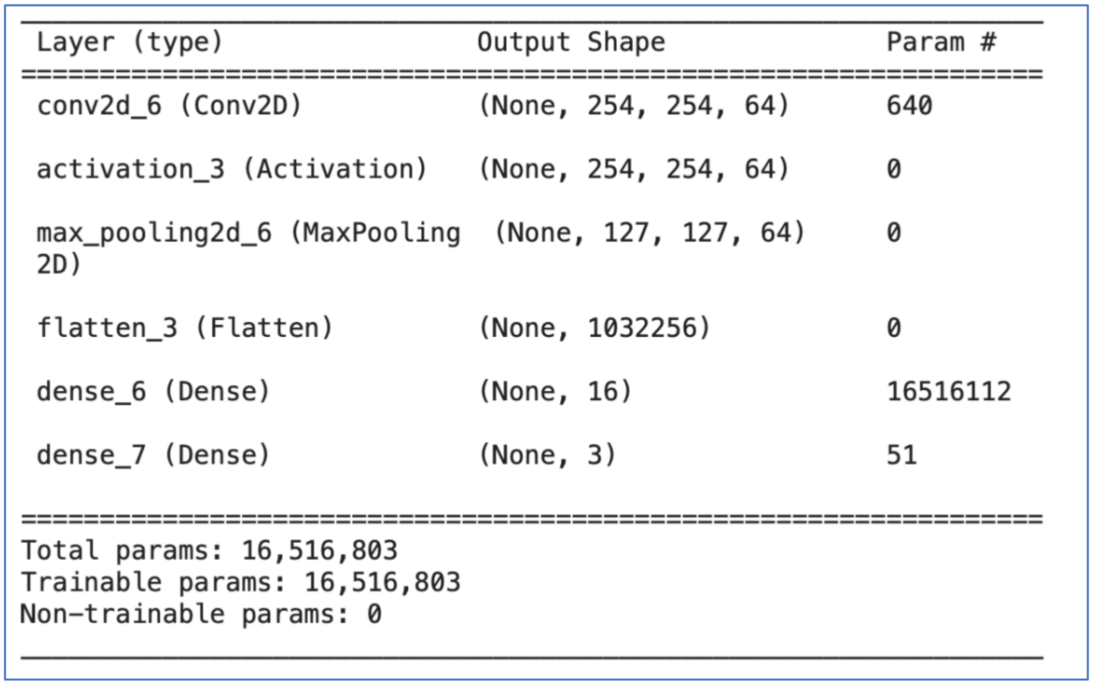


Accuracy and Loss Plots of Model 1

From the above plots, we concur that the desired loss and accuracy are achieved gradually, with the data validation loss still being significant at a value of 0.1 even in the final epoch.

Model 2: SMOTE Based Data: 1 Layer, Batch Size 8, Epochs 10

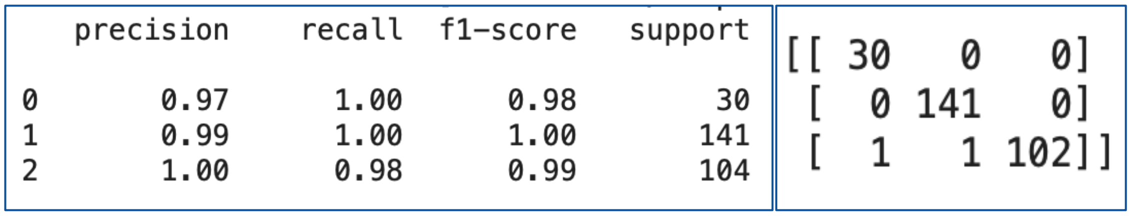
As an improvement, we used the SMOTE synthesized data as the training dataset for our CNN Model. We started off with a CNN Model using a single Convolution Layer, a batch size of 8 and a total of 10 epochs. The model summary is shown below:



```
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_6 (Conv2D)           (None, 254, 254, 64)      640

 activation_3 (Activation)   (None, 254, 254, 64)      0

 max_pooling2d_6 (MaxPooling  (None, 127, 127, 64)     0
 2D)

 flatten_3 (Flatten)         (None, 1032256)           0

 dense_6 (Dense)             (None, 16)                16516112

 dense_7 (Dense)             (None, 3)                 51

=================================================================
Total params: 16,516,803
Trainable params: 16,516,803
Non-trainable params: 0
_________________________________________________________________
```

Summary of CNN Model fitted on SMOTE Balanced Data

In the above summary, we can clearly see the absence of a second Convolution Layer (hence another Pooling Layer), leading to a larger Flattened output.
The results of the Testing data are shown below:

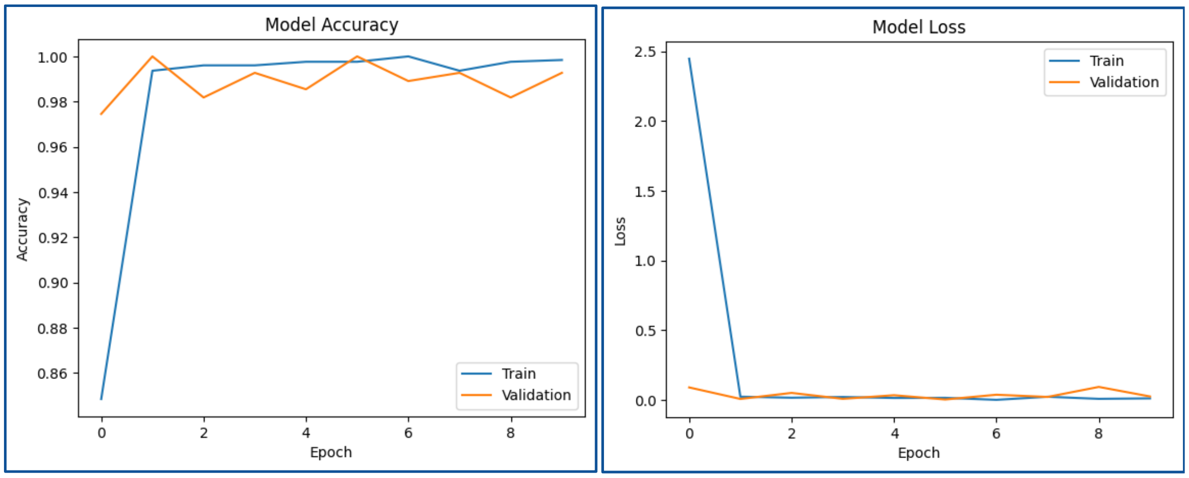


```
     precision    recall  f1-score   support

0         0.97      1.00      0.98        30
1         0.99      1.00      1.00       141
2         1.00      0.98      0.99       104
```

```
[[ 30   0   0]
 [  0 141   0]
 [  1   1 102]]
```

Results per Class & Confusion Matrix of Model 2

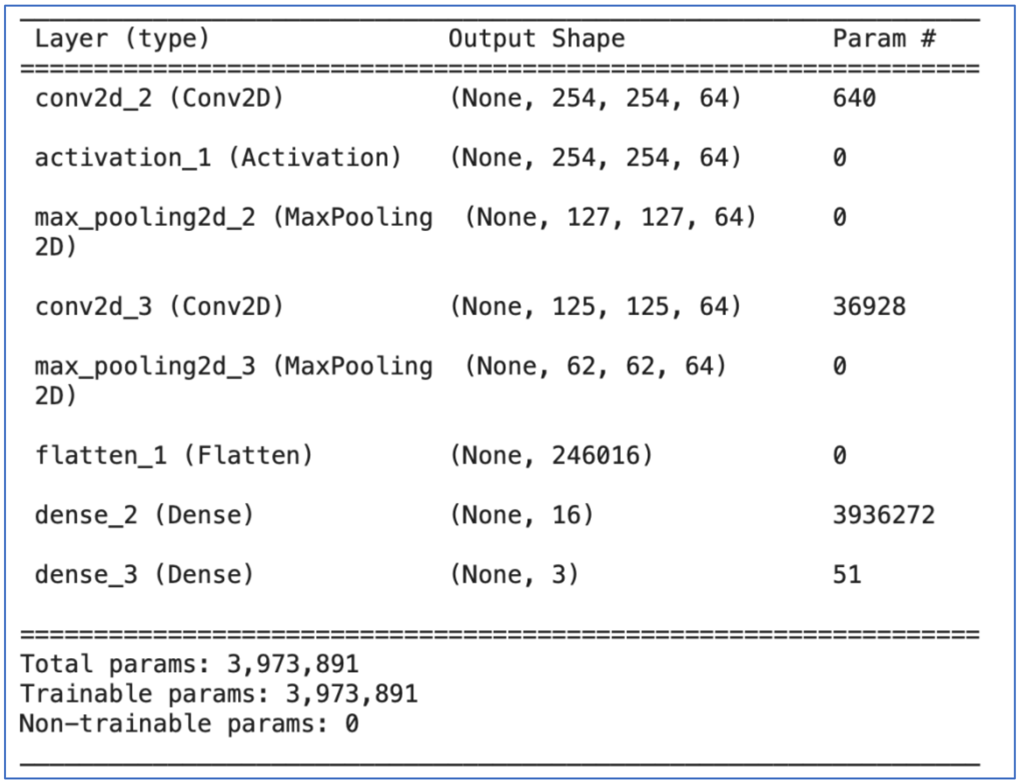
From the confusion matrix, we notice that we get 2 misclassifications for the Normal Case. This is still an improvement over the previous model.



Accuracy and Loss Plots of Model 2

From the above plots we can see that the validation loss was of a similar magnitude in all of the epochs. Thus, using a SMOTE synthesized data helped improve the accuracy.

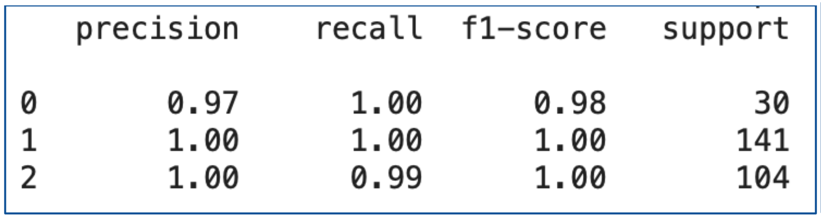
Model 3: SMOTE Based Data: 2 Layers, Batch Size 4, Epochs 10
Next, we added another Convolution Layer to see how that affected the model and its predictions. The model was initialized with two convolution layers, a batch size of 4 and a total of 10 epochs. The model summary shown below confirms the same:



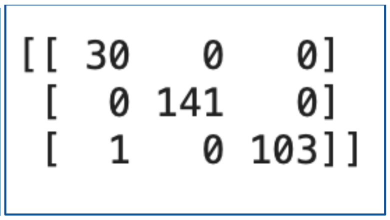
```
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_2 (Conv2D)           (None, 254, 254, 64)      640

 activation_1 (Activation)   (None, 254, 254, 64)      0

 max_pooling2d_2 (MaxPooling  (None, 127, 127, 64)     0
 2D)

 conv2d_3 (Conv2D)           (None, 125, 125, 64)      36928

 max_pooling2d_3 (MaxPooling  (None, 62, 62, 64)       0
 2D)

 flatten_1 (Flatten)         (None, 246016)            0

 dense_2 (Dense)             (None, 16)                3936272

 dense_3 (Dense)             (None, 3)                 51

=================================================================
Total params: 3,973,891
Trainable params: 3,973,891
Non-trainable params: 0
_________________________________________________________________
```

Summary of CNN Model fitted on Model 3

The model summary above displays the 2 Convolution Layers as we had previously discussed. The images below depict the prediction results for the testing dataset.



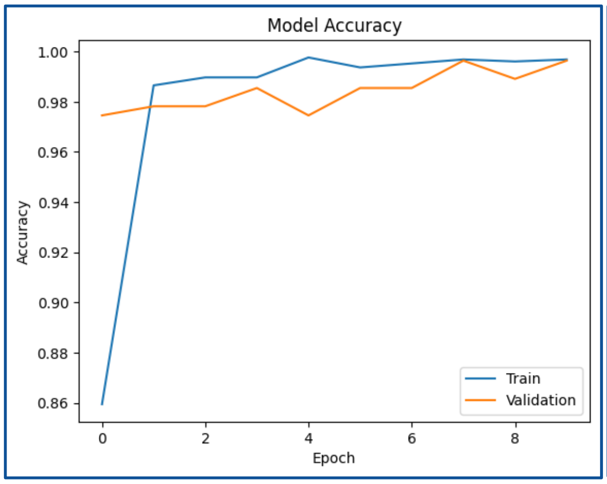
```
   precision    recall  f1-score   support

0       0.97      1.00      0.98        30
1       1.00      1.00      1.00       141
2       1.00      0.99      1.00       104
```

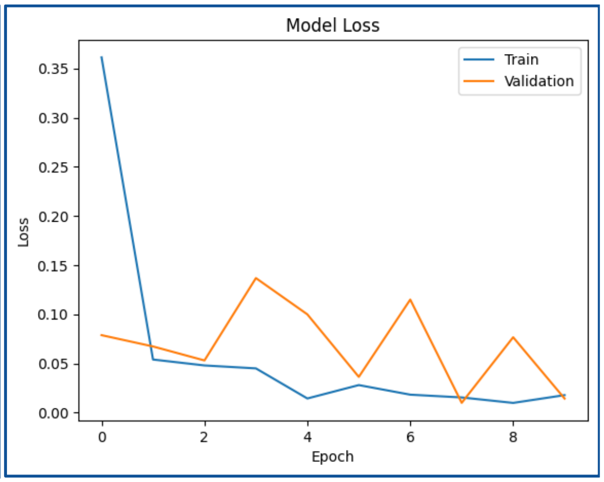


```
[[ 30   0   0]
 [  0 141   0]
 [  1   0 103]]
```

Results per Class & Confusion Matrix of Model 3

Considerable improvement over the previous model, as there's only a single instance of misclassification

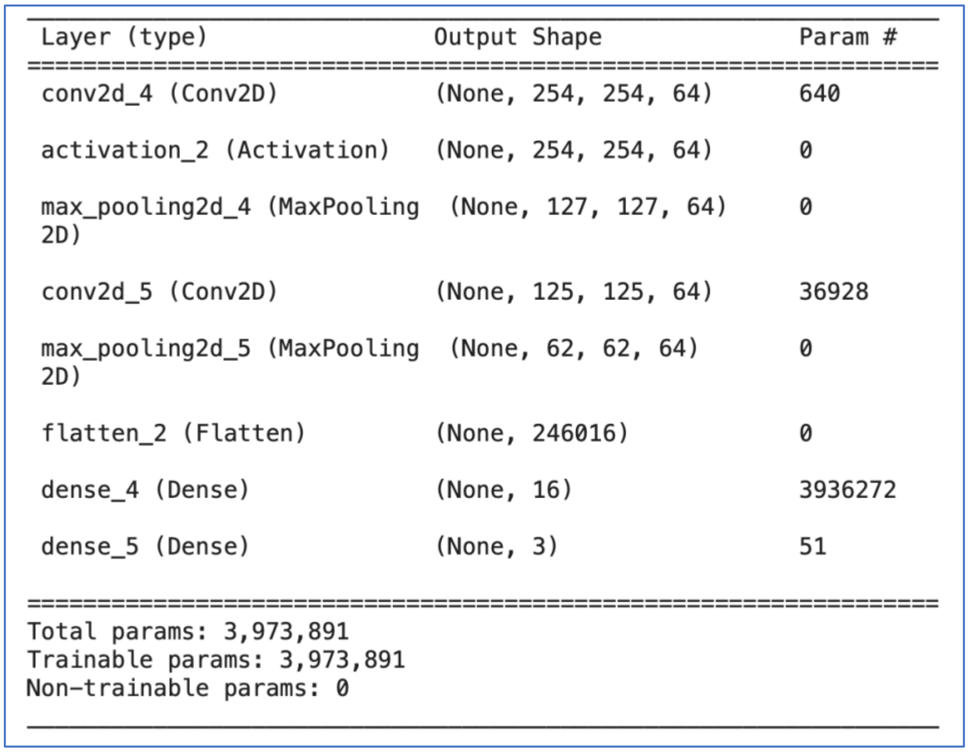




Accuracy and Loss Plots of Model 3

Validation Accuracy and Loss improve as the number of epochs increase, justifying the choice of 10 epochs.

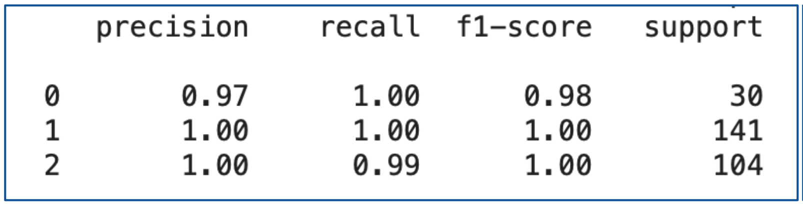
Model 4: SMOTE Based Data: 2 Layers, Batch Size 8, Epochs 10
In an attempt to see whether the accuracy can be further improved, we increased the batch size to 8 and checked the performance of the model. The model was initialized with two convolution layers, a batch size of 8 and a total of 10 epochs. The Model Summary is shown below:



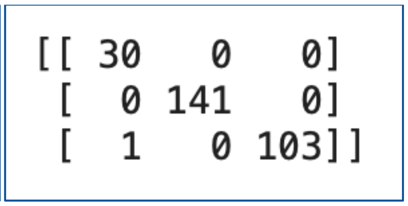
```
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_4 (Conv2D)           (None, 254, 254, 64)      640

 activation_2 (Activation)   (None, 254, 254, 64)      0

 max_pooling2d_4 (MaxPooling  (None, 127, 127, 64)     0
 2D)

 conv2d_5 (Conv2D)           (None, 125, 125, 64)      36928

 max_pooling2d_5 (MaxPooling  (None, 62, 62, 64)       0
 2D)

 flatten_2 (Flatten)         (None, 246016)            0

 dense_4 (Dense)             (None, 16)                3936272

 dense_5 (Dense)             (None, 3)                 51

=================================================================
Total params: 3,973,891
Trainable params: 3,973,891
Non-trainable params: 0
_________________________________________________________________
```

Summary of CNN Model fitted on Model 4

The results for predictions of the testing set are shown below.



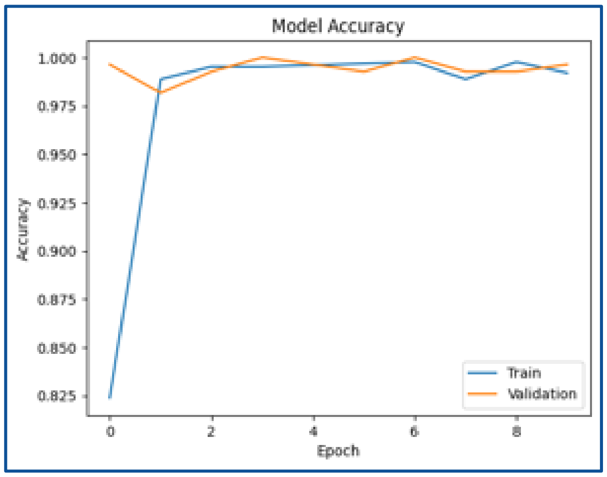
```
     precision    recall  f1-score   support

0         0.97      1.00      0.98        30
1         1.00      1.00      1.00       141
2         1.00      0.99      1.00       104
```

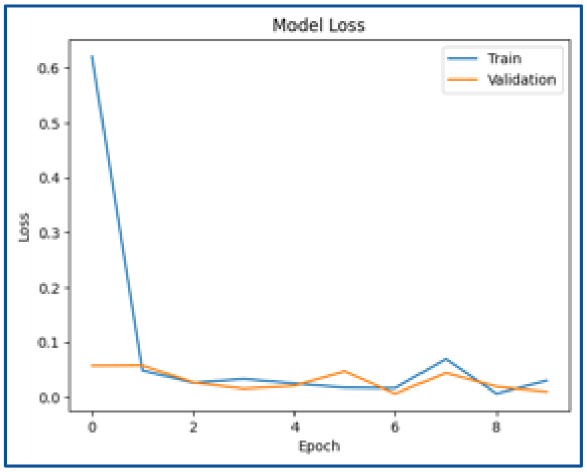


```
[[ 30   0   0]
 [  0 141   0]
 [  1   0 103]]
```

Results per Class & Confusion Matrix of Model 4

From the confusion matrix, we notice that we get one misclassification for a Normal Case. These results are the same as the ones we had obtained on the previous model, but at the cost of longer computation time. The Accuracy and Loss Plots are shown below:





Accuracy and Loss Plots of Model 4

The accuracy and loss plots show gradual convergence as the epochs increase.

Model 5: Class Weighted Approach

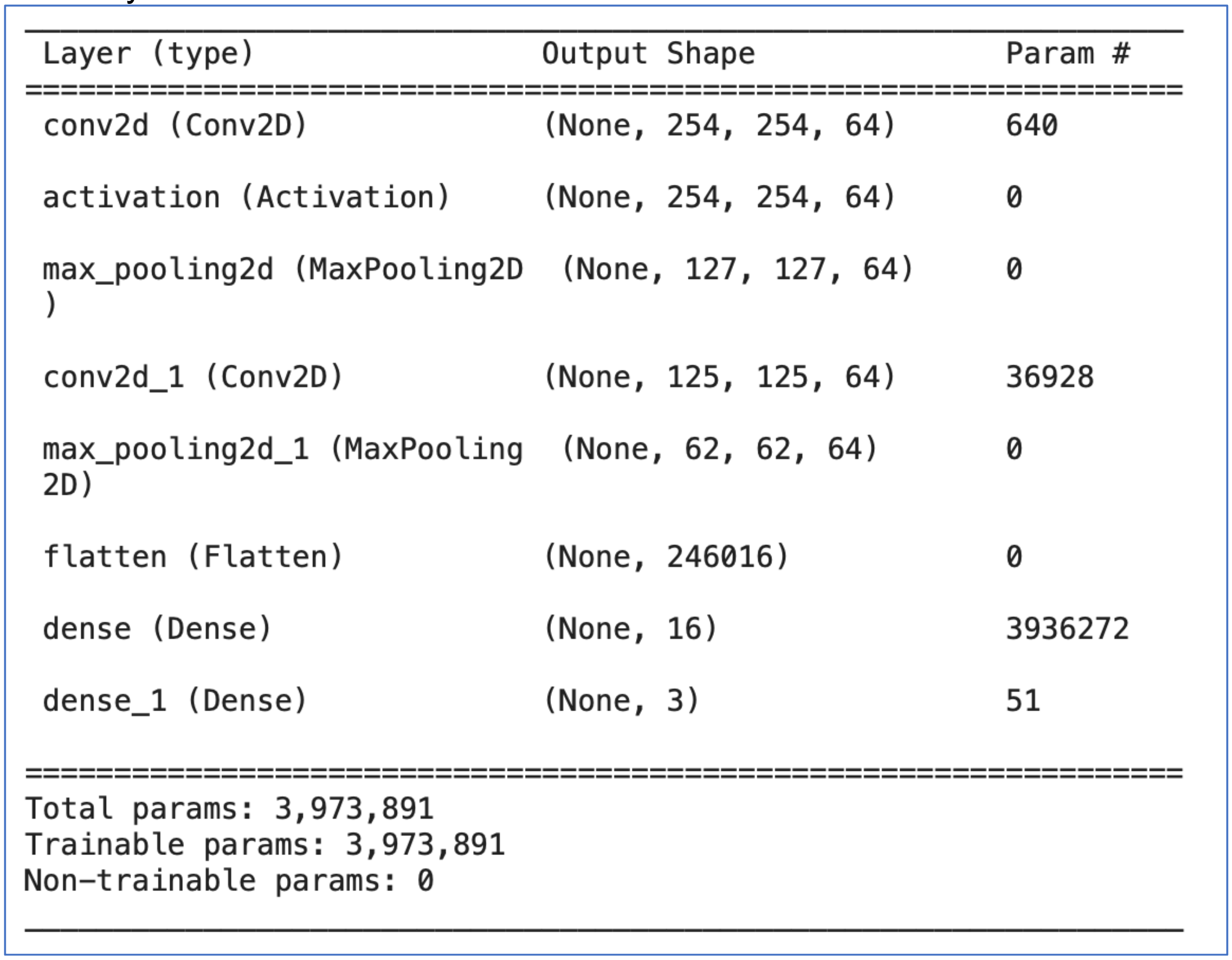
Finally, we use a Class Weighted Approach to handle the class imbalance. This technique relies on assigning weights to each class inversely proportional to the the class size, which is later incorporated in the loss function. Essentially, we are guiding the model to train better for instances of the smaller class by punishing with a heavier loss, only to be accounted for during back propagation.

The class weights generated are shown below

```
{Benign: 3.0814814814814815,
Malignant: 0.6556343577620173,
Normal: 0.8693834900731452}
```

Class Weights

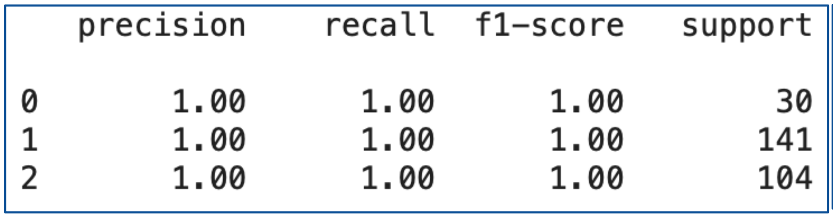
As seen above, the Benign Class, which had the lowest number of samples, has the largest weight out of the three. These weights were passed as a parameter, alongside 2 Convolution Layers, a batch size of 8 and 10 epochs. The model summary is shown below:



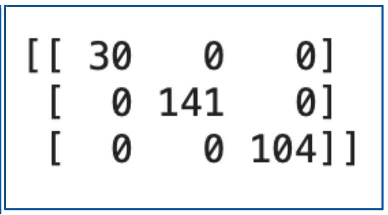
```
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 254, 254, 64)      640

 activation (Activation)     (None, 254, 254, 64)      0

 max_pooling2d (MaxPooling2D  (None, 127, 127, 64)     0
 )

 conv2d_1 (Conv2D)           (None, 125, 125, 64)      36928

 max_pooling2d_1 (MaxPooling  (None, 62, 62, 64)       0
 2D)

 flatten (Flatten)           (None, 246016)            0

 dense (Dense)               (None, 16)                3936272

 dense_1 (Dense)             (None, 3)                 51

=================================================================
Total params: 3,973,891
Trainable params: 3,973,891
Non-trainable params: 0
_________________________________________________________________
```

Summary of CNN Model fitted with Class Weights

The results after testing on the Test Set are shown below:



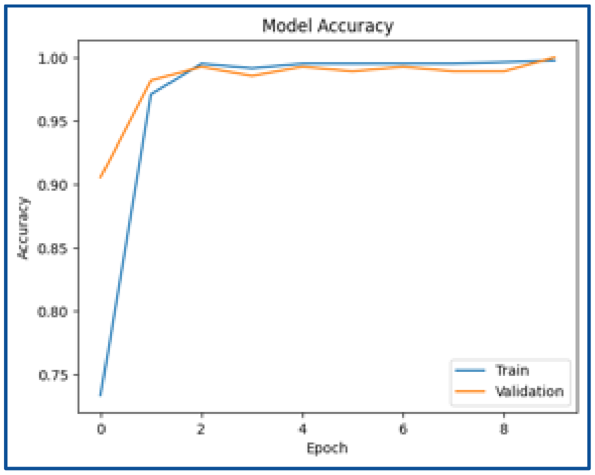
```
    precision    recall  f1-score   support

0       1.00      1.00      1.00        30
1       1.00      1.00      1.00       141
2       1.00      1.00      1.00       104
```

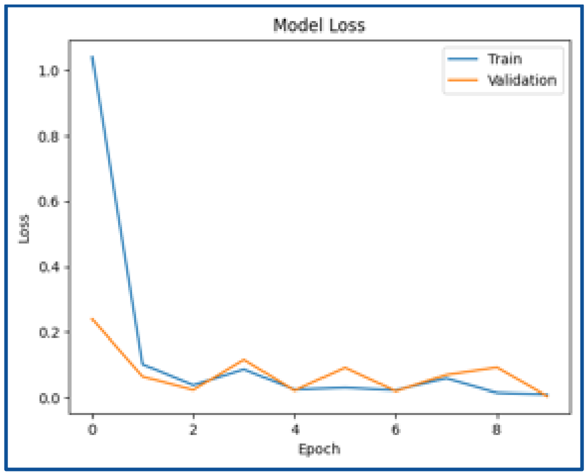


```
[[ 30   0   0]
 [  0 141   0]
 [  0   0 104]]
```

Results per Class & Confusion Matrix of Model 5

Surprisingly, this method gave a perfect accuracy, and was also faster than the rest of the methods.





Accuracy and Loss Plots of Model 5

The accuracy and loss plots show gradual convergence as the epochs increase. This pattern was similar in all of the above models used.

RESULTS

With the results derived above, we can conclude that lung cancer can be successfully diagnosed using a CNN architecture. The results from the above models can be summarized in the following table as follows:

Model	Avg. Precision	Avg. Recall	Avg. F1 Score	Avg. Accuracy
Imbalanced	95.67%	96.33%	96.33%	97.82%
SMOTE- 1 Conv	98.67%	99.33%	99%	99.27%
SMOTE- 2 Conv, BS 4	99%	99.67%	99.33%	99.64%
SMOTE- 2 Conv, BS 8	99%	99.67%	99.33%	99.64%
Class Weighted Approach	100%	100%	100%	100%

Table comparing results of all Models

CONCLUSION

Based on the results obtained, we conclude that Class Weighted approach is suitable for dealing with datasets having class imbalance, specifically for our case of lung cancer diagnosis. While this approach had a perfect classification for all its test cases, the same can be ascribed to various other extraneous factors such as the train-test split, hyperparameters selected and evolution of weights. It is also prone to the bias generated by manually assigning the weight of each class based on the size alone.
However, it is irrefutable that the Class Weighted approach didn't require synthesizing new data and performed on par with other models in terms of precision, outperforming them in time complexity.
Thus, a more reliable model would be a CNN Model with no weights, using extrapolated data from SMOTE and trained with two convolution layers, a batch size of 8.
With the results derived above, we can conclude that lung cancer can be successfully diagnosed using a CNN architecture.

FUTURE SCOPE

- Different angles to have a better picture of the entire lungs.
- Determining stage and the condition of tumor.
- Prediction/ Prescription of Treatment based on the tumor identified.

LITERATURE CITED

- Classification of Imbalanced Data: https://www.tensorflow.org/tutorials/structured_data/imbalanced_data#class_weights
- SMOTE: https://machinelearningmastery.com/smote-oversampling-for-imbalanced-classification/