Northumbria University
NEWCASTLE

Dissertation

KF7029
MSc Computer Science and Digital Technologies Project

Student Name: Shlok Gola
Student ID: w21026488
Supervisor Name: Dr. Naveed Anwar
Second Marker Name: Phil Anderson

Dissertation Title: Histopathological Cancer Detection Using Convolutional Neural Network

2021/2022

Declaration

I declare the following:

(1) that the material contained in this dissertation is the end result of my own work and that due acknowledgement has been given in the bibliography and references to **ALL** sources be they printed, electronic or personal.

(2) the Word Count of this Dissertation is 10,675.

(3) that unless this dissertation has been confirmed as confidential, I agree to an entire electronic copy or sections of the dissertation to being placed on the eLearning Portal (Blackboard), if deemed appropriate, to allow future students the opportunity to see examples of past dissertations. I understand that if displayed on the eLearning Portal it would be made available for no longer than five years and that students would be able to print off copies or download.

(4) I agree to my dissertation being submitted to a plagiarism detection service, where it will be stored in a database and compared against work submitted from this or any other Department or from other institutions using the service.

In the event of the service detecting a high degree of similarity between content within the service this will be reported back to my supervisor and second marker, who may decide to undertake further investigation that may ultimately lead to disciplinary actions, should instances of plagiarism be detected.

(5) I have read the Northumbria University Policy Statement on Ethics in Research and Consultancy and I confirm that ethical issues have been considered, evaluated and appropriately addressed in this research.

SIGNED: Shlok Gola

DATE: 27/09/2022

Abstract

Cancer is one of the leading diseases on WHOS's list regarding mortality rates across the world. Early cancer detection improves the chances of effective treatment. Histopathological cancer detection is critical because pathological confirmation is used to make the majority of cancer diagnoses. There has recently been a lot of research done on how to effectively implement deep learning into the diagnosis process. Deep learning showed promising results in the field of medical diagnosis. It is used to assist human medical practitioners in various areas. In this study, to classify cancerous tissues and non-cancerous tissues three convolutional neural networks are considered. One CNN model is designed from scratch and two pre-trained models, i.e., ResNet-50 and DenseNet-121 are used to analyze histopathology images for cancer detection. Moreover, different gradient descent optimization algorithms Adam, Adagrad and Nadam are used to analyze their effects on diagnosis accuracy. It turned out from the experiments that Nadam performed well with all the three models and DenseNet-121 remained stable throughout the training with all three optimizers and attain an accuracy of 96% with Nadam.

Table of Contents

1 Introduction

Cancer is one of the leading diseases on WHO's list regarding mortality rates across the world. The claim has followed the fact that close to ten million people died in 2020 alone due to cancer, which makes approximately one cancer death among every six (WHO, 2022). The most common types of cancers are Breast, Lung, Colon, Rectum, and Prostate and the major reasons are excessive usage of tobacco, and alcohol, excessive body mass index, insufficient use of fruits and vegetables, etc. The reasons might look pretty mundane but the statistics state that these simple-looking things are causing more harm than accounted for (WHO, 2022). It is widely accepted, in most world and low to medium profile societies that cancer is not curable, and they deem to believe so because most of the patients around them end up on their deathbeds. However, this is not true. Cancer is curable but early detection is extremely important. Since science is still learning to develop more and more advanced methods of detecting cancer, this is all just the beginning. As mentioned above, the reasons for cancer are so subtle that no one gets an idea until diagnosed. If it is diagnosed early, it can be treated, but in most cases, the delay in diagnosis is followed by fatalities.

High-resolution images of tissues containing cancer are studied by pathologists and decide whether or not to what stage cancer is the patient having. This is an extremely responsible job, sometimes going beyond the human endurable burden as one needs to tell the patient that he has got something that might end up taking his or her life. On the other hand, mental trauma is none another less other factors through which the patient goes once they hear about it. This is when it is extremely relevant to bring in technology so that an accurate and efficient method of early cancer detection could be developed. Over the years, Artificial Intelligence has matured itself well enough to be used in cancer detection and research is still on its way to developing a multitude of methods for cancer detection from the beginning. The development of high-quality digital microscopic scanners and imagers has helped medical science ease the capture of high-quality digital images of tumors and hence digital histopathology is now more accessible than ever. Over the years several systems have been developed which focus on various design features such as fractal, texture, and object features.

The detection of prostate cancer is being studied and proposed by using various AI classifiers such as Bayesian, SVM and KNN (Xu et al., 2012). These classifiers operate on learning methods that the classifier is fed with to classify between the images. A huge amount of literature suggests that image segmentation through supervised learning gives the most efficient results, however, supervised learning involves high-definition delineation and the corresponding data. The data annotation is the most tedious work for pathologists but on the brighter side, provides the most efficient results. Another learning method called the Unsupervised learning method is relatively easy to implement but comes with a cost of result quality deterioration. The unsupervised learning method does not involve manual data annotation hence it is less laborious (Xu et al., 2012). Lastly, middling the extreme ends are weakly supervised learning framework, in which mild annotations are used over iterative functions to help produce finely annotated information. The lymphoid organs that have lymphocyte encircle within a sharp reticular stroma are called lymph nodes. The parts of the structure are capsule, sub-capsular Sinus, Cortex (B-cells, germinal centers), Para-Cortex, medullary sinuses, medullary cords and hilus.

Artificial Intelligence has created a lot of opportunities in the field of early cancer detection. The field of cancer detection using Machine Learning and Deep Learning algorithms is evolving quite rapidly and more and more developments are being introduced. These developments are both in the form of new systems and improvements to the previous ones. However, since the object in question is a human life, hence the research has still a long way to go before it finally gets to the market.

1.1 Motivation

Pathologists review sentinel lymph node histopathology images, they spend a lot of time and effort on the diagnostic process since a broad area of tissue needs to be investigated and tiny metastases are frequently missed.. The suggested convolutional neural network techniques can streamline pathologists' labour and increase detection precision. Additionally, it can help pathologists save time and improve the likelihood of early diagnosis. Regions with benign tissues will be found and areas with a high proportion of tissue will be identified and removed. The main motivation for project is to help pathologists detect cancer at early stage and help them in diagnosing

accurately and efficiently, to reduce the mortality rate and act as helping hand for pathologists

1.2 Aims

The aim of this project is to develop a deep learning algorithm to identify metastatic cancer on histopathology images of lymph nodes to potentially improve the efficiency and accuracy of diagnosis. It also aims to evaluate the effectiveness of automated deep learning algorithms with different optimizers at detecting metastatic in sections of lymph nodes.

1.3 Research Approach

This research is an exploratory research study that is looking to see if deep learning can aid better decision-making and support systems for pathologists for more accurate and efficient metastasis cancer detection. The study adopts an engineering build at its core to build deep learning algorithms that help pathologists in improving their diagnostics. The training subset of the original dataset will be subjected to picture augmentation, and various optimizer combinations will be taken into consideration, in order to develop an effective deep learning method.

1.4 Structure of the Report

This document is structured in 5 sections. Section 1 gives the introduction. Section 2 is about the literature and its review. Section 3 explains the methodology, where all the models used in study is explained in full and all the different evaluation metrics used are explained. Analysis of all the evaluation metrics and Results are discussed in section 4. Finally, section 5 concludes this study, highlights the best performing model and future recommendation is discussed.

2 Literature Review

Researchers and scientists have developed various methods ranging in a wide scale of precision and techniques for early cancer detection. Moreover, the intensity and the patient-specific treatment availability have made it necessary for researchers to develop more and more robust and fine-tuned methods. One of the key elements of cancer detection is to determine the stage of cancer. There are developments in the area of stage determination to help reduce incorrect stage classification. In addition to this awareness needs to be spread and people need to learn to identify the symptoms of early cancer (Ott et al., 2009). Presented a study of medical literature and identified things like machinery and training required for the treatment of cancer in case of early detection because they believe early detection is just the tip of the iceberg, the real challenge lies in treating the patients both mentally and medically in low and middle-income countries. It is also believed by some that although Machine Learning and Artificial Intelligence have achieved much in the field of clinical science and the accuracy and objectivity of the detection process have increased and the workload of pathologists and clinicians has reduced, there are still reasons why so few technologies driven detection methods have hit the actual market despite having so many of them. In the paper, (Ott et al., 2009) took another dimension and identified certain challenges in the usage of Artificial Intelligence so that the side effects could be taken care of before the implementation of AI across the board.

The pattern exhibited by cancer tissues is extremely inconsistent and it needs detailed analysis and annotation before making a decision. Labeling cancerous images is of great clinical importance. (Xu et al., 2012) implemented an algorithm to classify the images of cancer and non-cancer and carried out segmentation at pixel level and reduced the deficiencies in the detection of colon cancers. They used multiple clustered instances learning mechanisms and classified clusters and segments in colon cancer histopathology. The advancements in pharmaceutical sciences have developed personalized medicine. It means that pathologists have now an immense complexity at hand as they have to prescribe patient-specific medicine, keeping in view the other body organisms. (Litjens et al., 2016) in her work presented the application of Machine Learning tools over two examples of Breast Cancer and Prostate Cancer and concluded that engaging the technology in cancer detection can acutely reduce the workload of

pathologists and can increase the objectivity of the process. It also helps to enhance the diagnostic protocols by increasing efficiency and accuracy so that pathologists could determine the exact state of the patient before the prescription.

Every histopathological scan has a great amount of data and information to offer. In addition to this, there are other scans such as CT scans and X-rays which are to be read in conjuncture. Hence, in totality, cancer detection is an extremely tough job and required a great deal of data for precision. (Jia et al., 2019) presented that deep multiple instances learning approaches and deep convolutional machine learning can help in overcoming histopathological cancer detection issues and the classification results are better than other methods. The methods of cancer detection developed using AI are rather new and need more maturity, however, there are some classical methods of cancer detection used by doctors such as Asymmetry, Border, Color and Diameter (ABCD) method, Seven-point Detection, Menzies method, and many more. (Munir et al., 2019) has identified a comparative analysis of these methods and others in his work and presented evaluation criteria based on the ROC curve and AUC. Over a large number of histopathological scans, it is laborious to identify the malign lymph node. In addition to this, the identification of metastatic tissue is a gradual process that adds to the overall complexity of the process. (Jaiswal et al., 2019) presented a convolutional neural network model being implemented to a predefined data set commonly used for Machine Learning fundamentals and improved the results of cancer detection by engaging pseudo labels on the dataset using the semi-supervised learning method.

Japan is facing a shortage of pathologists and hence early detection of cancer and its relevant timely treatment can get delayed. Hence (Sun et al., 2020) presented a CNN model for histopathological cancer detection using Google-developed EfficientNet-B6 architecture and used the presented model with different activation functions and gradient descent optimizers. The EfficientNet-B6 is a small-sized and easy-to-handle model and works very efficiently in classification tasks. The biopsies help in acquiring cancer screenings which can further lead to the determination of cancerous tissue or otherwise but manual testing is an extremely tedious task due to the variations in images. Hence, for simplicity (Ghadekar et al., 2021) developed an algorithm for determining the histopathology of body fluid nodes detection using a deep learning model over scanned tumor images. These models of Neural networks determine

whether the scanned lymph node is malignant or not. Early detections can save lives in all sorts of cancer. Colorectal cancer is made significantly easier to be detected after the implementation of Artificial Neural Networks and Deep Learning networks as histological imaging and its referencing have been made easier to access. (Kavitha et al., 2022) has presented the latest developments in polyp determination and segmentation through imagery and video-based systems.

2.1 Machine Learning

Sahiner et al. (2019), with the help of cutting-edge processing power, a plethora of data, and innovative algorithms, artificial intelligence (AI) technologies is gaining popularity. It is being used in a variety of industries, including industry, medicine, and comfortable lifestyle. There are three main areas for AI. One uses a rule-based search tool to generate responses using a conceptual method. Another is the deep neural network-based connectionism technique (DNNs). Whereas every strategy has advantages and disadvantages, connectionism is currently receiving a great deal of interest as a way to tackle challenging issues. Machine learning is a branch of AI that uses little person input to learn from data in order to categorise objects or forecast unknown future events (Murphy, 2012). ML is classified as non symbolic AI because it is data-driven training and can make predictions based on previously unknown data. segmentation, detection, regression and classification of data and other ML applications are present herein. Data sets for ML typically just include train set, validation, and test sets. From the training data set, it acquires features of data, and from the validation data set, it verifies the learned features. Utilizing test data set, one can ultimately verify the ML's accuracy.

An artificial neural network, neurologically inspired algorithm with layers and connected nodes, which is a part of machine learning. There are hidden layers on the input and output layers. The inputs are on the top layer, while the labelled values are in the bottom layer. During training, the value of each node is determined by setting weights using learning methods like backpropagation. The weights of each node are adjusted with the goal of reducing loss and increasing precision. Iterative backpropagation can be used to generate ideal weights. The training procedure for

artificial neural networks can occasionally result in a local minimum or be optimized only for trained data, which causes problems with generalization.

Recently, experts improved deep learning by adding many hidden layers with interconnected nodes between the input and output layers of artificial neural network to transform it into deep neural network. By combining straightforward judgments between layers, the multilayer may handle more complicated issues. In prediction problems like classification and regression, deep neural network typically outperforms the shallow layered network (He et al., 2016a). To avoid training to settle at a local minimum or to solve overfitting issues, every layer of the deep neural network improved its weights depending on the unsupervised constrained Boltzmann machine (Hinton et al., 2006). Skip connections have recently been discovered to help residual neural networks avoid the vanishing gradient problem (He et al., 2016a). Additionally, the development of big data and graphics processing units may help to accelerate calculation and provide solutions to complex issues.

As a result, the deep learning algorithm is receiving a deal of focus right now for solving a variety of issues in the medical imaging domains. One illustration is the radiology practise of identifying disease or anomalies from X-ray images and categorising them into various disease kinds or severity levels (Qin et al., 2018). The many ML algorithms have been used to carry out tasks of such a nature with adequate optimizations, conceptual, or experimental techniques. Computer-aided detection systems, which have been designed and used in the medical organization since the 1980s, are one instance. Nevertheless, the Computer-aided detection systems system produces more false positives than doctors do, which has increased diagnostic times and resulted in pointless biopsies (Fenton et al., 2007, Lehman et al., 2015). Deep learning methods allowed for the quick and accurate resolution of these issues, freeing up time for humans to work on other useful activities. A convolutional neural network (CNN), a component of deep learning, has lately received attention in machine vision for both supervised and unsupervised learning applications (Krizhevsky et al., 2017). Nevertheless, the advancement of these methods does not ultimately herald the displacement of medical professionals, particularly pathologists. Rather, it aids in a greater accurate diagnosis of individuals by pathologists.

Previously, there were several studies that used convolutional neural network for histopathological cancer detection. Table 1, shows summary of CNN models that were used in previous works.

Models	Activation Function	Optimization Algorithm	Testing Accuracy
VGG19	ReLU	Adam	0.9464
VGG16	ReLU	Adam	0.9794
EfficientNet-B6	Mish	Radam	0.9794
ResNet-50	ReLU	Adam	0.9432

Table 1. Summary of Model Used in Previous Study (Sun et al., 2020)

2.2 DenseNet

The majority of researchers identify medical photos using deep learning. Li et al. (2019b) used the DenseNet-2 model to classify mammography images as benign and malignant malignancy. Inception-Net is used instead of the first convolution layer, which improves model performance. Image augmentation is also utilised to address the over-fitting and data insufficiency issues. DenseNet can be tested on various datasets to determine how well it performs while maintaining its original design. Dense Net can produce accurate findings with fewer parameters, according to recent CNN research (Gottapu and Dagli, 2018). For the segmentation of brain MRI images, DenseNet architecture is employed. This approach is used to identify brain anomalies. Results with and without compression are contrasted. 95.5% accuracy was achieved by this model. To use DenseNet as a benchmark, various datasets (brain segmentation) are examined. This model can even be evaluated as a classifier on images without segmentation for benchmarking purposes (Gottapu and Dagli, 2018).

Li et al. (2019a) used the Dense Net feature learning model in the research study to classify pancreatic cancer from CT scan images. For the early diagnosis and segmentation of pancreatic cancer, this method is based on CAD (computer-aided diagnosis). The accuracy of the DenseNet model was 72.8%, which was higher than the diagnostic accuracy. This accuracy is lower than in other studies, so more DenseNet research is needed to improve it. On different image data and imagenet, Huang et al. (2017) compared DenseNet-121, DenseNet-161, DenseNet-201, and DenseNet-BC.

According to study, DenseNet requires fewer parameter changes and less computation time to achieve accuracy comparable to ResNets. DenseNet's performance grows in lockstep with the parameters.

2.3 Summary

In this study of histopathological cancer detection one basic CNN model from scratch will be build and two transfer learning models ResNet-50 and DenseNet-121 with be used. These all three models will be trained with different optimizers Adam, Adagrad and Nadam for comparing the efficiency of the model. DenseNet-121 model and optimizers Adagrad and NAdam will used as novelty in this study.

3 Methodology

This study proposes CNN algorithms on histopathological images to accurately and efficiently classify metastatic cancer. This study also uses transfer learning method. Transfer learning is the process of using a previously trained deep learning models and reusing it for a new but similar task. To begin the training process, a pre-trained model that has been trained on ImageNet, a large image database which is annotated is used. The weights of layers are being fine-tuned in relation to the new problem. This speeds up training and improves the performance of the pre-trained deep learning model when applied to a new problem (Devassy and Antony, 2022). The ResNet-50 pre-trained model is used in this study. The DenseNet-121 model is used as a novelty on the histopathological image dataset to implement the transfer learning method.

3.1 Setup

While training the model, CNN and transfer learning models take longer to process the images. The experiment is run on the Google Collaboratory, which has a 166 GB hard drive, 12.68 GB of RAM, and GPU. All the three models have more layers so they took longer time to execute on large image dataset. The use of GPU during runtime speeds up the execution of CNN, ResNet-50 and DenseNet-121. Python libraries Keras and TensorFlow are used to implement CNN, ResNet-50 and DenseNet121. On Google Collaboratory notebook, Python version 3 is used. Data is directly accessed from Kaggle platform on Google Collaboratory via the Kaggle API using .json file downloaded from Kaggle.

3.2 Dataset

The data set that is used for this project is sourced from the online platform Kaggle. The train data here contains 2,20,025 RGB images with the dimensions of 96*96*3. The data is the only the subset of the original PatchCamelyon (PCam) dataset. The original PCam dataset contains duplicate images due to its probabilistic sampling, the version presented on Kaggle does not contain duplicate images. A brand-new and difficult image classification dataset is PatchCamelyon. It is made up of 327.680 colour images (96*96 pixels each) that were taken from lymph node section histopathologic scans. A new benchmark for machine learning models is provided by PCam, which is

trainable on a single GPU and is larger than CIFAR10 and smaller than imagenet. Each image in the dataset is annotated with a binary label indicating the presence of a tumor. The presented problem is a binary classification problem with the class labels tumour and non-tumor tissues as the associated labels. A binary label '1' denotes tumor tissue in the pixel patch and '0' indicates the non-tumor tissue. The portion of the image exterior to the tumor patch has no effect on the label of a tumor.

3.3 Data Exploration

The data has been directly accessed from Kaggle, an open platform in python notebook on Google Collaboratory. The data is extracted by connecting Google Collaboratory with Kaggle API using .json file downloaded from Kaggle. There are 2,20,025 images in train folder and there labels are stored in .csv in the binary format where '1' indicates cancerous (positive) and '0' indicates non-cancerous (negative) images. Fig.1, shows some sample of positive and negative sample in the dataset.

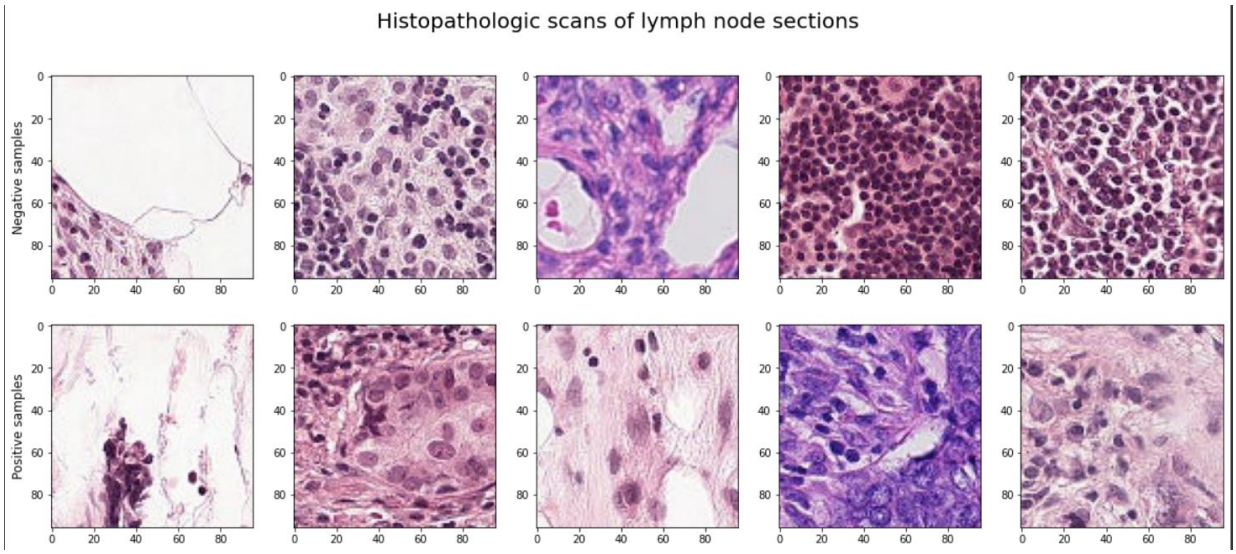

Figure 1. Histopathologic scans of lymph node sections

3.4 Data Pre-Processing

After data exploration is the done, the class imbalance is checked in this study because in binary classification in medical domain, classification of minority class i.e., patient

having cancer is more important than the classification of majority class of cancer free people. Fig. 2, shows the values count of both labels in dataset.

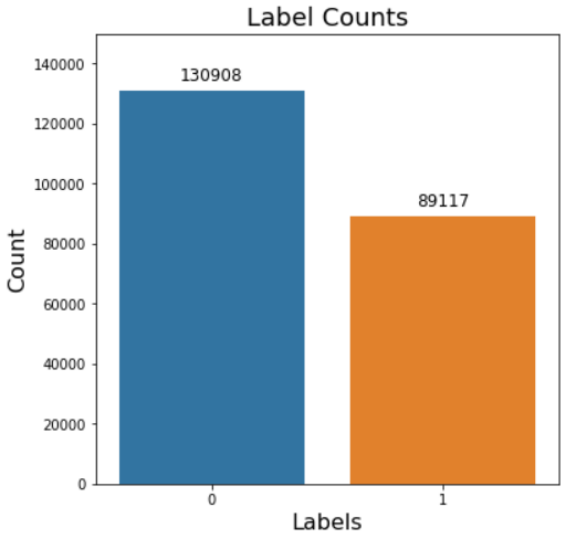

Figure 2. Label Count of Train Images

Fig. 2, clearly shows that there is class imbalance in the training data with significantly more non-cancerous images (label '0') than cancerous images (label '1'). In medical domain, class balancing is very important to avoid the biasedness of model towards one class. To balance both the classes number of label '0' images is reduced by random sampling of that set in order to match the size as the set of images with label '1'.

3.5 Data Preparation

Further, both the set of labels are reduced to 80k by random sampling to reduce the memory usage or potential crash of Google Collaboratory. A new 'base_dir' is formed in which a folder named 'train_dir' under which two sub-folders are created named '0' (non-cancerous) and '1' (cancerous) where new dataset is stored that is that is formed by doing random sampling. Ghadekar et al. (2021), suggests image augmentation in the study of same dataset when there was tremendous change in accuracy. Therefore, images in new dataset will be normalized for readability before being sent to CNN, DenseNet-121, and ResNet-50. Using rotation and zoom parameters, images are then augmented and saved. Images are augmented at run time using various parameters such as rotation, zoom, scaling, horizontal flip, and vertical flip.

Using Keras ImageDataGenerator(), which offers a quick and simple technique to enhance the photos, image normalization and augmentation is performed. It offers a variety of augmentation methods, including standardization, rotation, shifts, flips, brightness changes, and many others. ImageDataGenerator() ensures that every model is given fresh versions of the images. However, it does not include it in the original corpus of images, If this were the case, the model would be overfit because the model would be viewing the original images more than once. The fact that ImageDataGenerator uses less RAM is another benefit. This is because when this class is not used, all the images are loaded at once. However, while using this images are loaded in chunks, which uses much less RAM (Bhandari, 2020).

3.5.1 Image Normalization

Image enhancement is required in medical images to assist pathologists in detecting disease and diagnosing it. For better results, images fed into an automated deep learning model must be improved in quality. Zhuang and Guan (2017) applied histogram equalization approach to improve an image's contrast and brightness without compromising the finer features of the image data. As stated above, the approach is used to increase the contrast and brightness of histopathological image data in this study. Histopathological images must be examined under a microscope, so they may be stained and difficult to read. Image normalization will boost the image's contrast. As a result, it improves the deep learning model's accuracy and performance. Table 2 shows all the normalization parameter used in this study.

Parameters in Normalization	Value Used
Feature wise Std Normalization	True
Sample wise Std Normalization	True
Feature wise Centre	True
Sample wise Centre	True

Table 2. Parameters Used in Image Normalization

3.5.2 Image Augmentation

Images are augmented after normalization before being fed into the CNN model and transfer learning-based DenseNet-121 and ResNet-50. Basic image augmentation is performed in the study (Tellez et al., 2019) using scaling, gaussian blurring, and

gaussian noise. Horizontal and vertical image mirroring, 90-degree and 180-degree image rotations, and image scaling by 0.5 to 1.0 factor are all examples of morphological augmentation. For augmentation rotation of 10 degrees, zoom of 0.6, horizontal flip, vertical flip, and scaling range [0,1] are used in this study. Run time augmentation is performed while providing an input to the classification model by tuning various parameters. Table 2, describes all of the hyperparameters used in this study and their corresponding values for selecting the best suitable parameters used for augmentation.

Parameters in Augmentation	Value Used
Zoom Range	0.6
Rotation Range	10
Horizontal Flip	True
Rescale	1./255
Validation Split	0.2

Table 3. Parameters Used in Image Augmentation

Zoom Range: The zoom augmentation either erratically zooms in or out on the image. The 'zoom_range' argument of ImageDataGenerator() accepts a float value to specify the zoom level.

Rotation Range: One technique for augmentation that is used to enable the model to lose dependence on the orientation of the object is image rotation. The ImageDataGenerator() function enables users to randomly rotate images over any angle between 0 and 360 degrees by providing an integer number as the argument for the 'rotation range' parameter.

Horizontal Flip: The horizontal flip parameter of the ImageDataGenerator() class flips the images along the horizontal axis.

Rescale: ImagesDataGenerator() is used to rescale pixel value from the range 0-255 to the range 0-1. Scaling data to the range of 0-1 is traditionally referred as normalization.

Validation Split: Fraction of images from the dataset is reserved for validation. The percentage of fraction to be kept for validation is given in 'validation_split' parameter. Since the labels for test data were not available, test data is considered as validation data.

3.6 Modelling

In this study, different state of the art architectures are employed with modifications and compared the results. Different optimizers were used with three state of the art models i.e.

1. CNN Model
2. ResNet-50
3. DenseNet-121

The optimizer that were employed were:

1. Adam
2. Adagrad
3. Nadam

In the three state of the art models, the optimizers were replaced while compiling the model were Adam, Adagrad and Nadam. In total 9 experiments were performed.

3.6.1 CNN Model

The convolutional layer's main function is to identify trends, lines, and other features like boundaries. Convolutional layers are used in every hidden layer of CNN to convolve the input array using weight-parameterized convolution kernels. The many kernels provide a variety of feature pictures and enable success in a number of vision-related tasks, including segmentation and classification. Feature maps are gradually and spatially accumulated regionally here between convolutional layers. The length of feature maps is decreased by the pooling layer, which transfers the maximum or average value. This method captures aspects of a picture that are highly responsive to location and form. Max pooling is frequently employed experimentally. These pooling and convolutional layers are alternated regularly in the Convolutional neural network architecture. The fully connected layers are joined at the conclusion of the Convolution neural network and offer a final determination for classification or regression tasks. A loss is calculated throughout training by comparing the labelled output to the expected result. Contrarily, in the segmentation job, pooling layers are followed by convolutional layers and up-sampling layers to recover the size of the input picture. As a result, training loss is assessed by comparing labelled mask picture and Convolution neural network reconstructed output image. Given that Convolution neural network framework is made up of several layers, the amount of training

parameters might exceed millions. That implies that in order for training to achieve acceptable accuracy, a large amount of data is required. The amount of data depends on the nature of the assignment and the images. For example, if one trains the data from beginning, at least 1,000 photos per class are required to achieve a competent result in a classification job (Krizhevsky et al., 2017).

CNN is made up of transition layers, dense blocks, and convolutional layers. CNN is made up of a series of neural networks that are used to extract features. It concatenates all features from earlier layers in order to circulate features at various levels. According to (K. et al., 2018) the CNN architecture consists of five layers. The convolutional layer is used to extract local data from the input. The dot product of the weights of the filter and the input image region is computed. The subsequent activation layer receives the output from the batch normalization layer and regulates it. The gradient process balances the network's weights because normalization is utilized in it. The CNN uses the Rectified Linear Unit (ReLU) as an activation function. ReLU expedites CNN's execution. As CNN gets more in-depth, the features get more sophisticated. The feature map size is reduced using the MaxPooling layer. Each layer's fully linked neurons communicate with the layer below to calculate the number of classes. As a classification function, the output layer consists of a softmax or sigmoid.

Components used in layers for Building CNN Models in this study are as follows:

1. Convo2D layer: The convolution layer of 2 dimensions has the filter that works on the image pixel by pixel depending on the kernel size and filter size.
2. Max pooling: CNN operates by extracting the image's characteristics. Max pooling layer is one of the type of down sampling that considers a matrix of particular size and substitute the maximum value from pixels.
3. Batch Normalization: Batch Normalization is used to normalise output when the output of the previous convolution layer is transferred to the subsequent layer, which minimizes the issue of overfitting.
4. Dense: A dense layer is one that is totally connected to the layer before it, and it takes all of the neurons information from the prior layer and adds bias to the value. The value is then provided to the activation function, which decides this node's value.

5. Dropout: The dropout layer operates by changing the input values of the neurons to 0. It prevents overfitting by replacing the values with 0 and changing the other values with 1/(1-rate) so that the sum of all inputs remains the same.

6. Filter Size: As the filters pass across a picture, the filter size extracts the features and delivers the value by multiplying an n-dimensional matrix with only the values along the diagonal, which are 1. It extracts those characteristics and returns the value. The n-th filter size is determined by the training data and type of features.

7. Input Shape: The input shape is determined by the pixel size of the images.

8. Activation Functions:

A. Sigmoid: The sigmoid function returns values ranging from 0 to 1. It employs a function that considers the 1/1+exp equation (-x). As a result, it returns a number close to zero if x is little and close to 1 if x is large.

B. ReLu: Rectified Linear unit is a function that returns the greatest of two values, the first of which is 0 and the second of which is x. (the input vector). This gives all the positive numbers.

9. Optimizers:

A. Adam: Adam is a stochastic gradient descent method that is based on adaptive estimating. It is by far the most popular optimizer. It has a 0.001 learning rate. Because Adam is so powerful, the model tends to overfit, resulting in low validation accuracy.

3.6.2 DenseNet-121

Huang et al. (2017), assume a convolutional network being applied to a single input image, x_0. The network has L layers, where l indexes the layer and H_l (.) stands for a non-linear transformation that each layer in the network applies. Batch Normalization (BN) (Ioffe and Szegedy, 2015), rectified linear units (ReLU) (Glorot et al., 2011), pooling (He et al., 2016b), or convolution are examples of procedures that can be used to form H_l (.) (Conv). The output of the l^{th} layer is designated as xl. In residual networks the result of the l^{th} layer is connected through a conventional convolutional feed-forward network as input to the $(l + 1)^{th}$ layer (Krizhevsky et al., 2017), resulting in the layer transition

$$x_l = H_l(x_{l-1}) \ldots\ldots\ldots\ldots\ldots(1)$$

By passing the non-linear transformations with an identity function, ResNets (He et al., 2016a) include a skip-connection:

$$x_l = H_l(x_{l-1}) + x_{l-1} \ldots\ldots\ldots\ldots\ldots(2)$$

ResNets has the benefit of using the identity function to allow the gradient to move directly between more recent layers and previous layers. The network's ability to transmit information may be hampered by the summing that combines the identity function with the output of Hl. Authors suggested dense connectivity as an alternative connectivity structure: they provided direct linkages from whatever layer to all following layers in order to significantly optimise the information transit between layers.

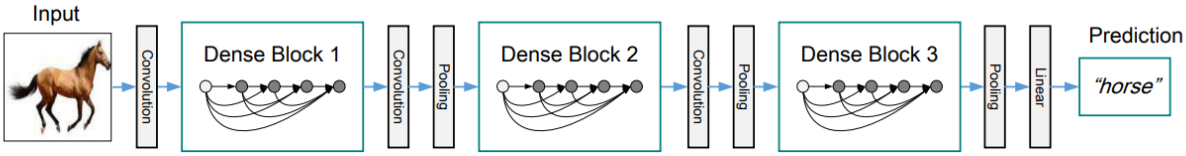

Figure 3. Deep DenseNet

As a result, the feed for the l^{th} layer consists of feature-maps of all previous layers, x_0,..., x_{l-1}:

$$x_l = H_l([x_0, x_1, \ldots, x_{l-1}]) \ldots\ldots\ldots\ldots\ldots(3)$$

where $[x_0, x_1, \ldots, x_{l-1}]$ is the feature-maps' concatenation. created in layers 0,..., l_{l-1}. This network design is referred as a "Dense Convolutional Network" because to its dense connectedness (DenseNet). To make execution simpler, authors combined H_l inputs from equation into a single tensor. composite performance. It is described $H_l(.)$ as a composite function of three subsequent operations, batch normalisation (BN**)**, rectified linear unit (ReLU) and a 3x3 convolution, as inspired by (He *et al.*, 2016b). When the scale of feature-maps varies, the concatenated procedure utilised in Equation is no longer feasible. Down-sampling layers, which alter the scale of feature-maps, are frequently used in convolutional networks. The architecture splits the

network into numerous densely linked blocks to enable down sampling; layers in between blocks are referred as as transitional layers because they perform pooling and convolution. The transition layers in tests consist of a batch normalisation layer, a 1*1 convolutional layer, and a 2*2 average pooling layer. If each function in Hl returns k feature maps, where k_0 is the number of channels in the input layer, then the lth layer has k_0 + k (l-1) input feature-maps.

As opposed to current network topologies, DenseNet may have very thin layers, such as k = 12. This is a significant distinction between the two. The network development rate is referred to as the hyperparameter k. The datasets that were evaluated, a comparatively modest development rate is adequate to provide state-of-the-art outcomes. This may be explained by the fact that every layer has accessibility to all the feature-maps that came before it in its block and, consequently, to the "accumulated understanding" of the network. The feature-maps may be thought of as the network's overall state. This state receives k feature-map additions from each tier. The pace of development controls how much fresh data each layer adds to the overall state.

In contrast to conventional network topologies, it is not requirement to repeat the global state from one layer to another because, when written, it may be retrieved from anywhere within the network. layered bottlenecks. Every layer contains numerous more inputs even if it only creates k output feature-maps. A l_1 convolution can be added as a bottleneck layer before each 33 convolution- in order to decrease the amount of given input and hence increase computational speed. The network used in this study, uses the BN-ReLU-Conv(1*1)-BN-ReLU-Conv(3*3) version of Hl, has a blockage layer that is believed to be very successful for DenseNet. The network is called DenseNet-B.

In the tests authors conducted, authors allowed each 1 to 1 convolution to generate 4k feature-maps. Compression. By fewer feature-maps at transition layers, further model simplicity can be increased. The model is referred to as DenseNet-BC if a dense block comprises m feature-maps than the subsequent transition layer to produce m output feature-maps, where 0 1 are employed.

Layers	Output Size	DenseNet-121	DenseNet-169	DenseNet-201	DenseNet-264
Convolution	112 × 112	7 × 7 conv, stride 2			
Pooling	56 × 56	3 × 3 max pool, stride 2			
Dense Block (1)	56 × 56	[1 × 1 conv, 3 × 3 conv] × 6	[1 × 1 conv, 3 × 3 conv] × 6	[1 × 1 conv, 3 × 3 conv] × 6	[1 × 1 conv, 3 × 3 conv] × 6
Transition Layer (1)	56 × 56	1 × 1 conv			
	28 × 28	2 × 2 average pool, stride 2			
Dense Block (2)	28 × 28	[1 × 1 conv, 3 × 3 conv] × 12	[1 × 1 conv, 3 × 3 conv] × 12	[1 × 1 conv, 3 × 3 conv] × 12	[1 × 1 conv, 3 × 3 conv] × 12
Transition Layer (2)	28 × 28	1 × 1 conv			
	14 × 14	2 × 2 average pool, stride 2			
Dense Block (3)	14 × 14	[1 × 1 conv, 3 × 3 conv] × 24	[1 × 1 conv, 3 × 3 conv] × 32	[1 × 1 conv, 3 × 3 conv] × 48	[1 × 1 conv, 3 × 3 conv] × 64
Transition Layer (3)	14 × 14	1 × 1 conv			
	7 × 7	2 × 2 average pool, stride 2			
Dense Block (4)	7 × 7	[1 × 1 conv, 3 × 3 conv] × 16	[1 × 1 conv, 3 × 3 conv] × 32	[1 × 1 conv, 3 × 3 conv] × 32	[1 × 1 conv, 3 × 3 conv] × 48
Classification Layer	1 × 1	7 × 7 global average pool			
		1000D fully-connected, softmax			

Figure 4. Architectures of DenseNets

Layers	Output Size	DenseNet-121
Convolution	112*112	7*7 conv, stride 2
Pooling	56*56	3*3 max pool, stride 2
Dense Block 1	56*56	[1*1 conv, 3*3 conv] *6
Transition Layer 1	56*56	1*1 conv
	28*28	2*2 average pool, stride 2
Dense Block 2	28*28	[1*1 conv, 3*3 conv] *12
Transition Layer 2	28*28	1*1 conv
	14*14	2*2 average pool, stride 2
Dense Block 3	14*14	[1*1 conv, 3*3 conv] *24
Transition Layer 3	14*14	1*1 conv
	7*7	2*2 average pool, stride 2
Dense Block 4	7*7	[1*1 conv, 3*3 conv] *16
Classification Layer	1*1	7*7 global average pool
	14D fully connected, sigmoid	

Table 4. Architecture of DenseNet-121 (Allaouzi et al., 2019)

3.6.3 ResNet-50

A convolutional neural networks (CNN) with 50 layers is the ResNet-50 model. ResNet (stands for Deep Residual Network) is an Artificial Neural Networks (ANN) that, generates a network by stacking leftover blocks on top of one another. (Boesch, 2022). He et al. (2015) introduces a deep residual learning framework, which avoids hoping and instead directly fits a few stacks of layers into a desired underlying mapping, these layers explicitly fit a residual mapping.

Residual Learning

He et al. (2016a), assume H(x) is a mapping that x serves as the input to the first layer and can be fitted by a few stacked layers instead of the full network.. It is equivalent to assuming that multiple nonlinear layers can monotonically estimate the residual functions, or H(x)- x, if it is assumed that multiple nonlinear layers can monotonically estimate complex functions. Since this is the case, these layers are specifically permit to estimate a residual function:

$$\mathcal{F}(x) \coloneqq \mathcal{H}(x) - x \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots (4)$$

Thus, the initial function becomes $\mathcal{F}(x) + x$. The efficiency of learning may vary between the two forms, while both should be capable to monotonically approximating the target functions (as was anticipated). The paradoxical behavior regarding the degradation issue serves as the inspiration for this reformulation. As it was mentioned before, a deeper model ought to have training error that is no larger than that of its shallow equivalent if the additional layers can be built as identity mappings. According to the degrading challenge, solutions may struggle to approximate identity mappings using several nonlinear layers. If identity mappings are the best option, the solutions might only use the residual learning restructuring to pull the weights of the various nonlinear layers in the direction of zero to achieve a mapping that is almost identical.

Identity mappings are probably not the best option in practical situations, but our redefinition could assist to condition the issue. It should be simpler for the solver to recognise the disturbances with relation to an identity mapping than it would be to obtain the function as a unique one if the optimal function is close to an identical mapping than to a zero mapping. By conducting tests, it turned out that learnt residual functions typically have tiny reactions, indicating that identity mappings offer adequate setpoints.

Identity Mappings

To a few layered layers, we apply residual learning. In the following figure, a residual block is plotted

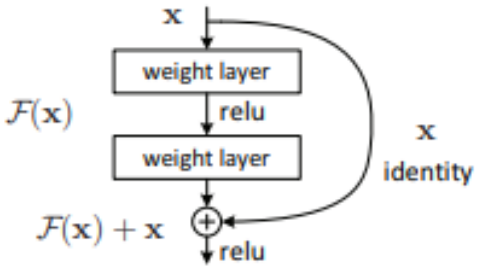

Figure 5: Residual Block

Technically, authors examined a construction block in the work denoted as:

$$y = \mathcal{F}(x, \{W_i\}) + x \ldots\ldots\ldots\ldots\ldots\ldots (5)$$

In this case, the output and input vectors of the layers under consideration are x and y. The residual mapping to be learnt is represented by the function $\mathcal{F}(x, \{W_i\})$. For the illustration, the fig. 5 has two layers:

$$\mathcal{F} = W_2\sigma(W_1 x) \ldots\ldots\ldots\ldots\ldots\ldots (6)$$

Here, the term σ denotes ReLU (Nair and Hinton, 2010) and the biases are removed for the simplification of notations. By use of a shortcut connection and elementally addendum, the operation F + x is carried out.

Authors adopted the second nonlinearity after the addition (i.e., σ(y)). The equation's shortcut connections don't add any new parameters or make the computation more complex. This is significant in comparisons between plain and residual networks and is not only appealing in practise. The plain/residual networks that have the same number of parameters, depth, width, and computing cost can be objectively compared (apart from the insignificant element-wise addition). In the equation, x and F's dimensions must be equal. If this is not the case (for example, when altering the input/output channels), the dimensions can be matched by executing a linear projection Ws by the shortcut connections:

$$y = \mathcal{F}(x, \{W_i\}) + W_s x \ldots\ldots\ldots\ldots\ldots\ldots (7)$$

A square matrix W_s can also be used. However, authors shown through studies that the identity mapping is sufficient and cost-effective for handling the degradation problem, therefore W_s is only utilised when matching dimensions. The residual function F can take on several shapes. The function F used in the experiments in this study has two or three layers, while more layers are feasible. F, however, resembles a linear layer if it only contains one layer, as shown by the formula: $y = W_{1x} + x$, for which no advantages have been found. Also to be noted is the fact that convolutional layers can benefit from the aforementioned notations, which, for the sake of simplicity, refer to fully-connected layers. Multiple convolutional layers can be represented by the function $F(x, W_i)$. Two feature maps are added element by element, channel by channel.

Architectures:

The authors investigated different plain/residual nets and found recurring trends. The plain network and residual network is compared. Both are trained on ImageNet model. The plain baselines are mainly inspired by the theory of VGG nets (Simonyan and Zisserman, 2014). Most convolutional layers have 3*3 filters and adhere to two

straightforward design principles: (i) the layers have the same quantity of filters for the same output feature map size.; and (ii) to maintain the time complexity per layer when the size of the feature map is cut in half, the number of filters is doubled. Direct convolutional layers with a stride of 2 were used to conduct down sampling. The network's final layer consists of a 1000-way fully-connected layer with softmax and a layer for global average pooling. There are 34 weighted layers in all. It is important to note that the model is simpler than VGG networks and has fewer filters. (Simonyan and Zisserman, 2014) . Only 18% of VGG-19's 3.6 billion FLOPs (multiply-adds) are found in the 34-layer baseline (19.6 billion FLOPs).

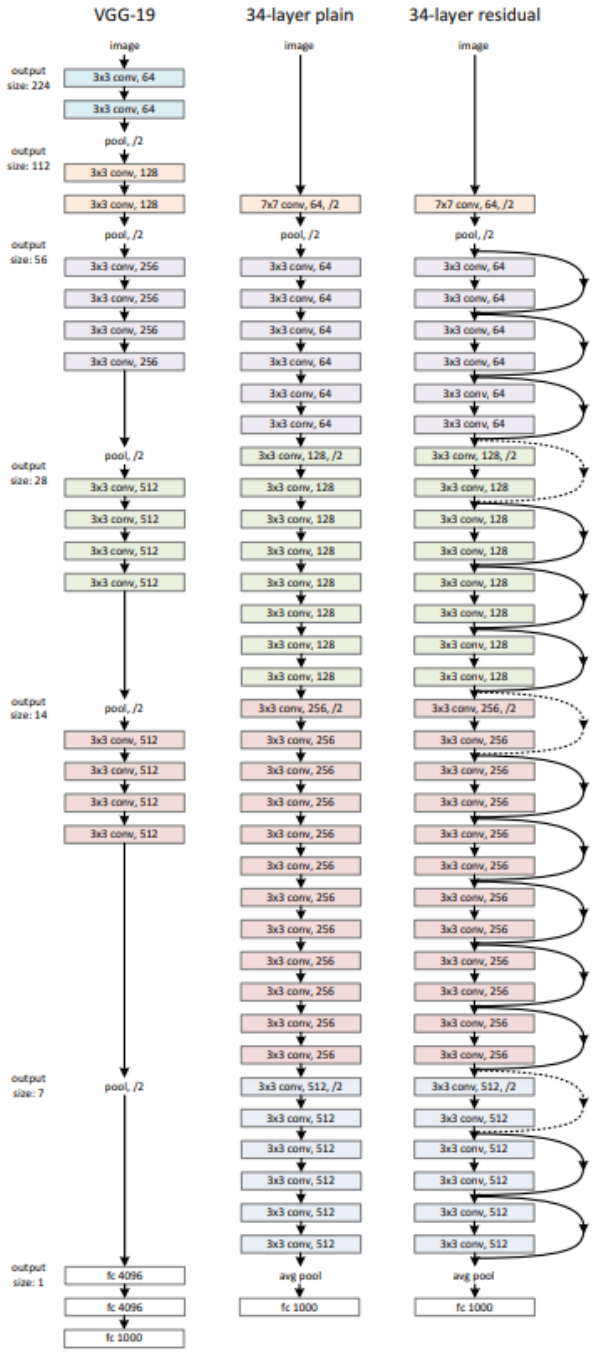

Figure 6. ResNet Architecture

Residual Network

On the basis of above plain ResNet network, shortcut connections were inserted which transforms the network into its residual counterpart. When the input and output have the same dimensions, the identity shortcuts can be utilised immediately (solid line shortcuts in above figure). Two options were taken into consideration as the dimensions grew (dotted line in above figure): (A) Even with additional zero entries

padding out the shortcut for growing dimensions, identity mapping is still performed. This choice doesn't add a new parameter; (B) To match dimensions (which is accomplished by 1*1 convolutions), the projection shortcut is employed. When the shortcuts navigate feature maps of varying sizes for either option, they are carried out with a stride of 2.

3.6.4 Adagrad

Adagrad (Duchi et al., 2011) is a gradient-based optimization technique that does this by adjusting the learning rate (lr) to the parameters and making tiny changes. Larger updates (i.e., high lr) are used for parameters associated with uncommon features, whereas smaller updates (i.e., low lr) are used for attributes linked with often occurring features. This makes it a good choice for handling sparse data. Adagrad significantly increased the robustness of SGD, according to (Dean et al., 2012), who employed it to train massive neural networks at Google that, among many other things, learnt to detect cats in online videos. Additionally (Pennington et al., 2014), trained GloVe word embeddings using Adagrad because uncommon words demand significantly bigger updates than common class. In the past, authors updated each parameter θ_i individually because they all utilised the identical learning rate η. Initially, Adagrad's per-parameter updated before vectorizing it since each parameter θ_i at each time step t has a distinct learning rate. For simplicity, it is abbreviate the gradient at time step t as g_t. The objective function's partial derivative with respect to the attribute at time step t is then represented as $g_{t,i}$

$$g_{t,i} = \nabla_\theta J(\theta_{t,i}) \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots(8)$$

The SGD update for every parameter θ_i at each time step t then becomes:

$$\theta_{t+1,i} = \theta_{t,i} - \eta . g_{t,i} \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots(9)$$

Relying on the previously calculated gradients for each parameter Adagrad updates the overall learning rate η in its update rule for every time step t for every parameter θ_i

$$\theta_{t+1,i} = \theta_{t,i} - \frac{\eta}{\sqrt{G_{t,ii} + \epsilon}} . g_{t,i} \ldots\ldots\ldots\ldots(10)$$

$G_t \epsilon \mathbb{R}^{dxd}$ here is a diagonal matrix where each diagonal Component i,i I is the total of the gradient squared with respect to θ_i up to time step t, and ϵ is a smoothness factor that prevents division by zero (often on the order of 1e-8). It's noteworthy to see that the method performs substantially poorer lacking the square root function.

The approache now can be vectorize by carrying out a matrix-vector product since G_t holds the sum of the squares of the previous gradients with respect to any and all factors along its diagonally.

Product between G_t and g_t:

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{G_t + \epsilon}} . g_t \ldots\ldots\ldots\ldots\ldots\ldots(11)$$

The fact that Adagrad does away with the necessity to manually adjust the learning rate is one of its key advantages. The majority of solutions utilise 0.01 as the standard setting and leaving it at that. Adagrad's primary flaw is the buildup of squared gradients in the denominator, which continues expanding over time throughout training because each term inserted is positive. In return, this leads the learning rate to decrease until it finally approaches zero, at which time the method is no more capable of learning new information. The methods that follow attempt to fix this problem.

3.6.5 Nadam

Adam may be considered as the product of RMSprop and momentum, where RMSprop adds the exponentially decaying average of previous squared gradients v_t and momentum the exponentially decaying mean of prior gradients m_t. Additionally, it has been demonstrated that NAG is stronger to conventional momentum.

Thus, Adam and NAG are combined in Nadam (Nesterov-accelerated Adaptive Moment Estimation) (Dozat, 2016). We ought to change Adam's momentum word m_t in to include NAG.

Let's first review the momentum updating criterion utilizing the syntax that is already use.

$$g_t = \nabla_{\theta_t} J(\theta_t)$$

$$m_t = \gamma m_{t-1} + \eta g_t$$

$$\theta_{t+1} = \theta_t - m_t \ldots\ldots\ldots\ldots\ldots(12)$$

where J is the objective function, η is the step size, and γ is the momentum decay term. The expansion of equation (12) results in:

$$\theta_{t+1} = \theta_t - (\gamma m_{t-1} + \eta g_t) \ldots\ldots\ldots\ldots\ldots\ldots(13)$$

This proves once more that momentum entails moving in the plane of the present gradient and the orientation of the prior momentum vector.

Furthermore, by upgrading the variables with the momentum phase prior to calculating the gradient, NAG enables to conduct a more precise step in the gradient orientation. Thus, to get to NAG, the gradient g_t merely need to change:

$$g_t = \nabla_{\theta_t} J(\theta_t - \gamma m_{t-1})$$

$$m_t = \gamma m_{t-1} + \eta g_t$$

$$\theta_{t+1} = \theta_t - m_t \ldots\ldots\ldots\ldots\ldots\ldots(14)$$

Dozat (2016) suggested changing NAG in the preceeding manner: Now look-ahead momentum vectors is used immediately to update the existing parameters instead of using the momentum step again, first for upgrading the gradient g_t and again for upgrading the parameters θ_{t+1}:

$$g_t = \nabla_{\theta_t} J(\theta_t)$$

$$m_t = \gamma m_{t-1} + \eta g_t$$

$$\theta_{t+1} = \theta_t - \gamma m_t + \eta g_t \ldots\ldots\ldots\ldots\ldots\ldots(15)$$

As it can be seen, the current momentum vector m_{t-1} is now utilized to gaze forward instead of the prior momentum vector m_t as in the equation of the extended momentum updating rule above. Thus, likewise swap out the prior momentum vector for the present momentum vector to add Nesterov momentum to Adam. The Adam updating rule is as follows (notice that v_t does not have to be modified).

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t$$

$$\widehat{m_t} = \frac{m_t}{1 - \beta_1^t} \ldots\ldots\ldots\ldots\ldots\ldots(16)$$

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{\hat{v}} + \epsilon} \widehat{m_t} \ldots\ldots\ldots\ldots\ldots\ldots(17)$$

If the equation (17) is expanded according to the definitions of $\widehat{m_t}$ and m_t, the resultant equation becomes

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{\widehat{v_t}+\epsilon}} \left(\frac{\beta_1 m_{t-1}}{1-\beta_1^t} + \frac{(1-\beta_1)g_t}{1-\beta_1^t}\right) \ldots\ldots\ldots\ldots\ldots\ldots(18)$$

Keep in mind that $\frac{\beta_1 m_{t-1}}{1-\beta_1^t}$ is nothing more than an estimate of the momentum vector from the preceding time step in which there is no bias. Thus, it may be substituted it with $\widehat{m_{t-1}}$

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{\widehat{v_t}+\epsilon}} \left(\beta_1 \widehat{m}_{t-1} + \frac{(1-\beta_1)g_t}{1-\beta_1^t}\right) \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots(19)$$

Noting that it disregard the denominator $1-\beta_1^t$ for the sake of simplification is and not $1-\beta_1^{t-1}$, as the denominator will be changed in the following phase. This formula resembles our preceding enlarged momentum update algorithm quite a little. By merely substituting the bias-corrected estimate of the momentum vector of the preceding time step, $\widehat{m_{t-1}}$, with the bias-corrected estimation of the existing momentum vector, $\widehat{m_t}$, the Nesterov momentum can be added in the same manner as before, giving the Nadam updated rule:

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{\widehat{v_t}+\epsilon}} \left(\beta_1 \widehat{m}_t + \frac{(1-\beta_1)g_t}{1-\beta_1^t} \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots(20)\right.$$

3.7 Evaluation

The most important aspect of model construction is evaluating its performance. There are various metrics for task classification. Some metrics, such as precision-recall, are useful for a variety of tasks. Before rolling out the model for unseen data, the model's overall predictive power can be improved by using different metrics for performance evaluation. Without performing a proper model evaluation using different evaluation metrics and relying solely on accuracy, there may be a concern when the corresponding model is deployed on unseen data, resulting in poor predictions (Agrawal, 2021). For evaluating the performance of models different methods were adopted in this study. The adopted method are as follows:

- Model Accuracy and Loss Graph

- Confusion Matrix
- Classification Report
- ROC Curve

3.7.1 Model Accuracy and Loss Graph

For evaluation, every model's loss and accuracy are calculated for each epoch (Sun et al., 2020). For training and validation set, an accuracy and loss graph is displayed. The test accuracy is determined using the test data and prediction.

Training & Test Accuracy: The degree of training is typically, that is obtained when the model is applied to the training data, but the accuracy of the test is determined by the accuracy of the test data.

Training Loss: Training loss is the value of the objective function that need to be minimized. Based on the specific objective function of the training data, the value can be positive or negative. For all three models, training losses are calculated across the entire training dataset.

Validation Loss: Validation loss is calculated and interpreted during training and verification based on the performance of the models. This is the sum of the errors created in each example of test set.

3.7.2 Confusion Matrix

Confusion matrix is generated that focuses on the strength of classifier and analyse the predictive performance of the model. The values that are predicted by the model are compared to the actual values of training data, are shown in confusion matrix. Four possible combinations are possible (Hashtings, 2021a):

- True Positive (TP): If both values are 1 (yes).
- True Negative (TN): If both values are 0 (no).
- False Positive (FP): If predicted value is 1, but actual value is 0.
- False Negative (FN): If predicted value is 0, but actual value is 1.

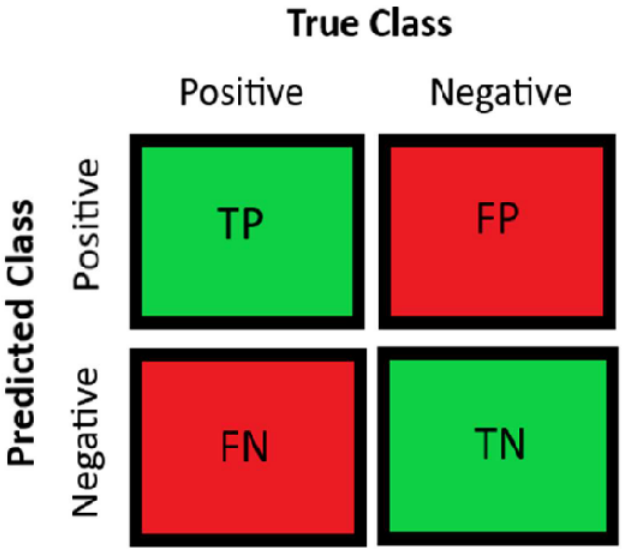

Figure 7. Confusion Matrix

3.7.3 Classification Report

According to, Agrawal (2021) the classification report of each model shows the precision, recall and F1-score of target class. The precision, recall and F1 score all three are calculated from the values that are obtained from confusion matrix.

Precision: Precision explains the instances that were correctly predicted came true. When False Positives are more problematic than False Negatives, precision comes in handy. Precision is calculated as number of true positives divided by the number of predicted positives.

$$Precision = \frac{TP}{TP + FP}$$

Recall: Recall measures the proportion of real positive cases that the model correctly predicted. It is a useful indicator when False Negative is more significant than False Positive. It is crucial in medical situations because even if a false alarm is raised, but the real positive shouldn't go unnoticed. Recall is calculated as numbers of true positives divided by the total number of actual positives.

$$Recall = \frac{TP}{TP + FN}$$

F1 Score: A summary of the Precision and Recall measurements is given. When Precision and Recall are equal, it performs best. The F1 Score is the harmonic mean of recall and precision.

$$F1 = 2.\frac{Precision * Recall}{Precision + Recall}$$

3.7.4 ROC Curve

The receiver operating characteristic (ROC) curve is assessed to determine how well the model performs in binary classification. The genuine positive rate and false positive rate are plotted on this graph. A classifier's performance is better when its curve is closer to the top-left corner. The model is less accurate on the test set when the curve is closer to the ROC space's 45-degree diagonal.

4 Analysis and Results

4.1 Model Accuracy and Loss Graph

Figure 8, shows the model accuracy and model loss graph of CNN model with all the three optimizers. The graph shows the performance of model at every epoch when the model is trained with different optimizers for 20 epoch each.

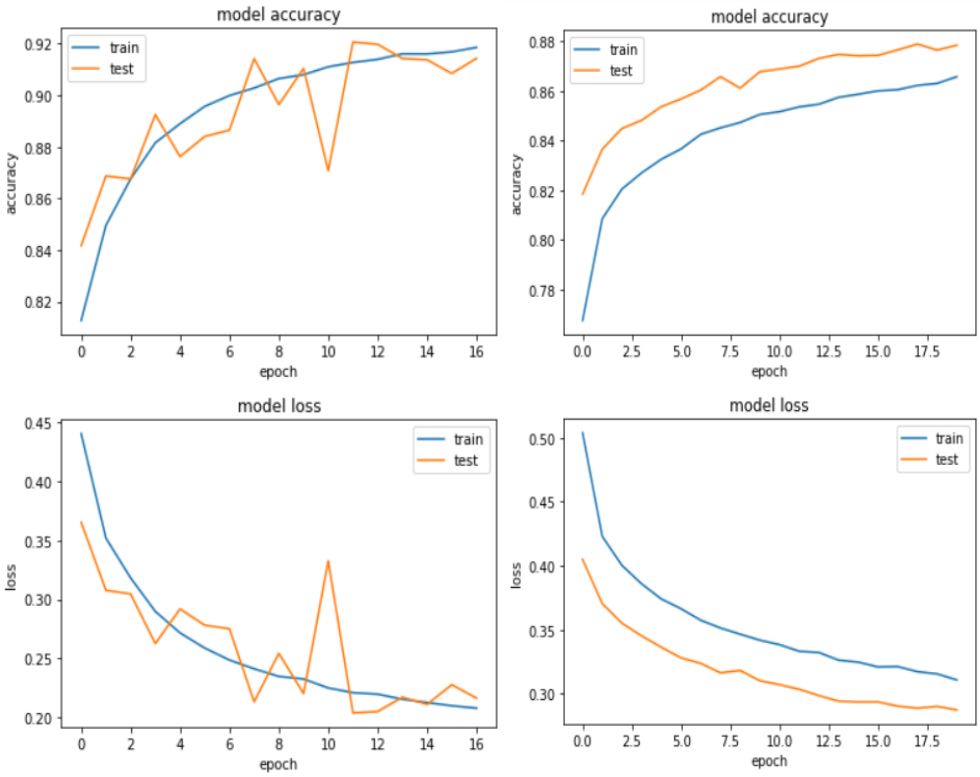

Figure 8. CNN model with optimizers (a) Adam and (b) Adagrad

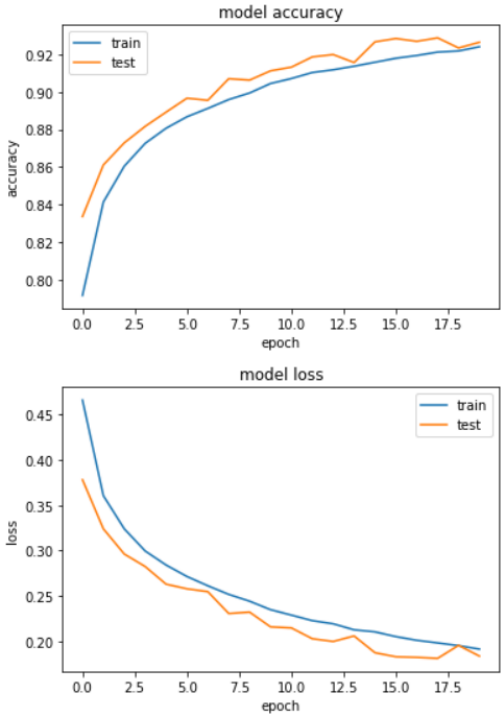

Figure 8(b) CNN model with optimizer Nadam

Test accuracy can be found varying in Adam, whereas Adagrad has quite stable train and test accuracy and loss variation but in Adagrad it can be observed the distance between train and test accuracies. Nadam is the optimum model here as it can be observed the stability in train and test accuracy and loss gap is very less.

Figure 9, shows the model accuracy and model loss graph of ResNet-50 model with all the three optimizers. ResNet-50 is also trained for 20 epoch with each optimizer.

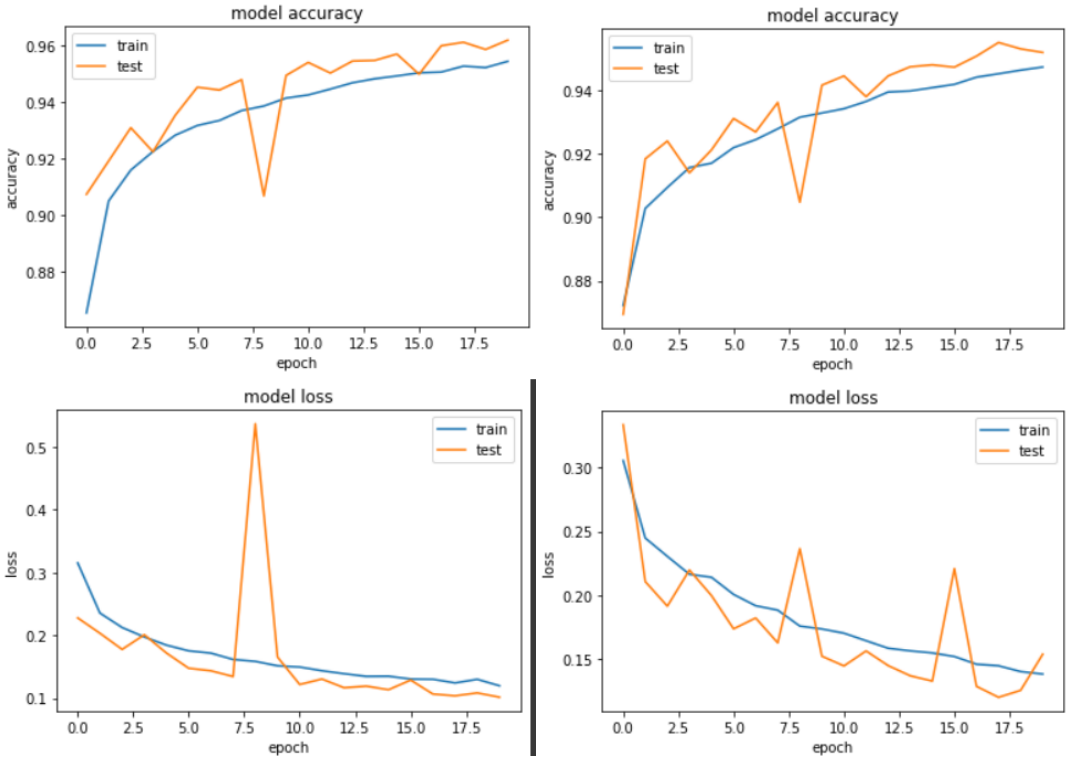

Figure 9. ResNet-50 Model with Optimizers (a) Adam and (b) Adagrad

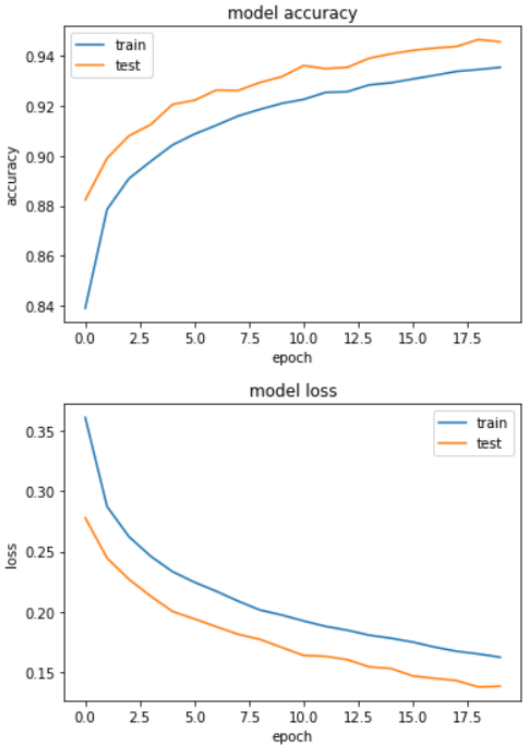

Figure 9(c). ResNet-50 Model with Optimizers Nadam

Test accuracy can be found varying in both Adam and Adagrad while Nadam is the optimum optimizer as it can be observed that there is stable train and test accuracy and loss variation and the train and test accuracy and loss gap is also less.

Figure 10, shows the model accuracy and model loss graph of DenseNet-121 model with all the three optimizers. Similarly, DenseNet-121 is also trained for 20 epochs with each optimizer just like CNN model and ResNet-50.

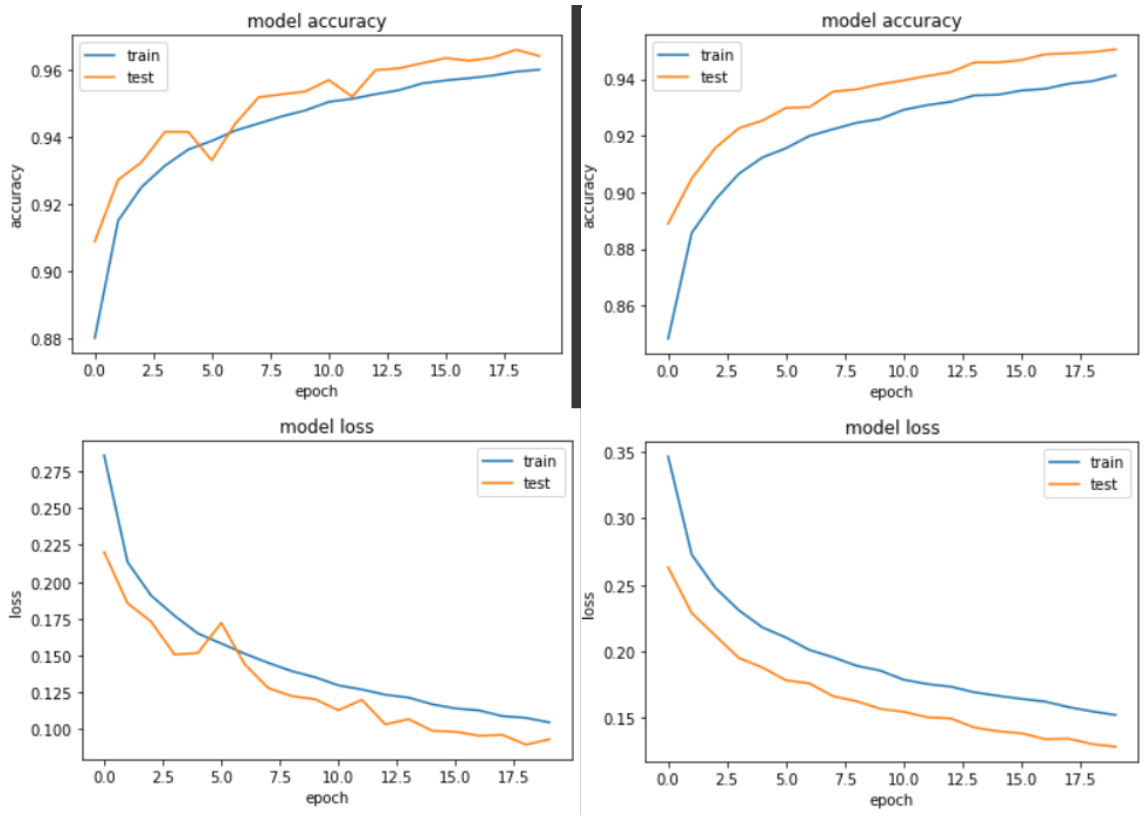

Figure 10. DenseNet-121 Model with Optimizers (a) Adam and (b) Adagrad

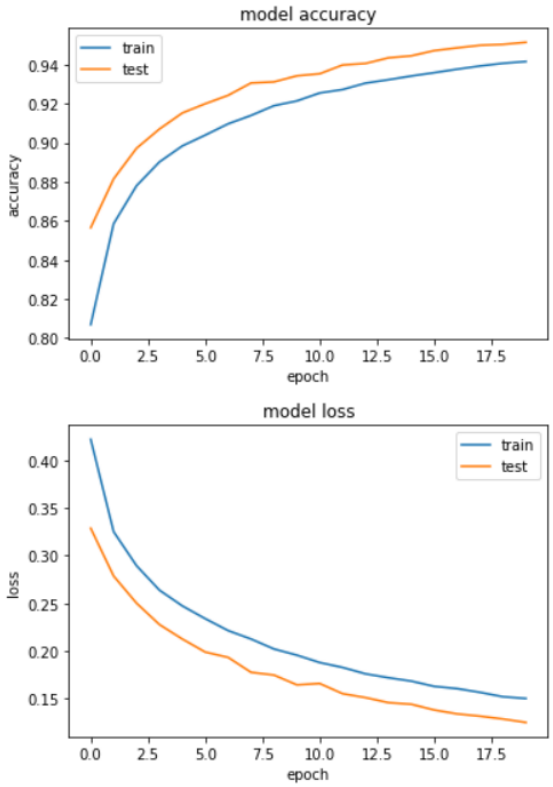

Figure 10(c). DenseNet-121 Model with Optimizers Nadam

Test accuracy can be found varying in Adam, whereas Adagrad has quite stable train and test accuracy and loss variation but in Adagrad it can be observed the distance between train and test accuracies. Nadam is the optimum model here as it can be observed the train and test accuracy and loss gap is very less and train and test accuracy and loss variation is also quite stable.

4.2 Confusion Matrix

Figure 11, shows the confusion matrix of CNN model with all three optimizers.

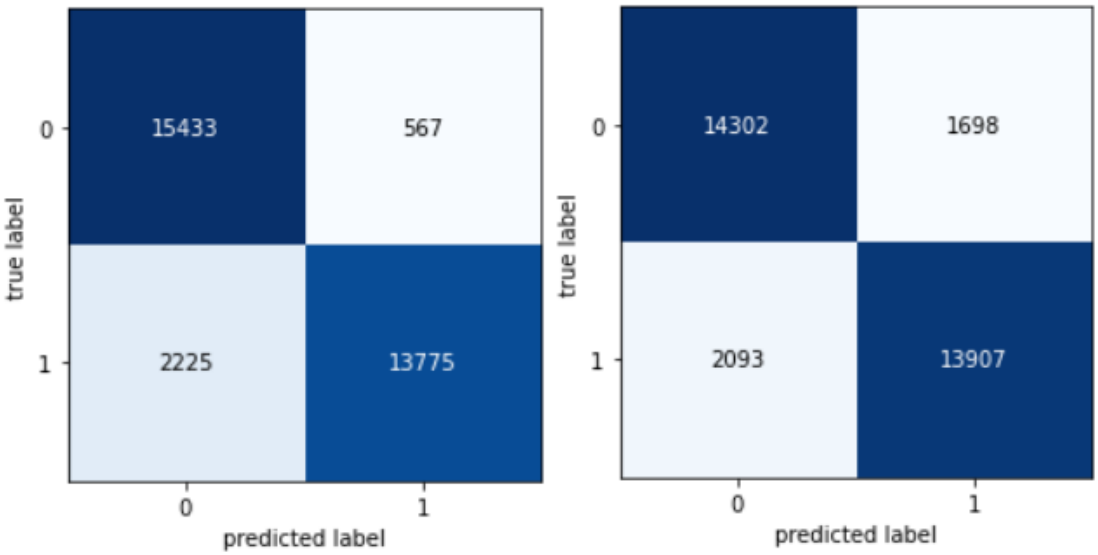

Figure 11. CNN Model with Optimizers (a) Adam and (b) Adagrad

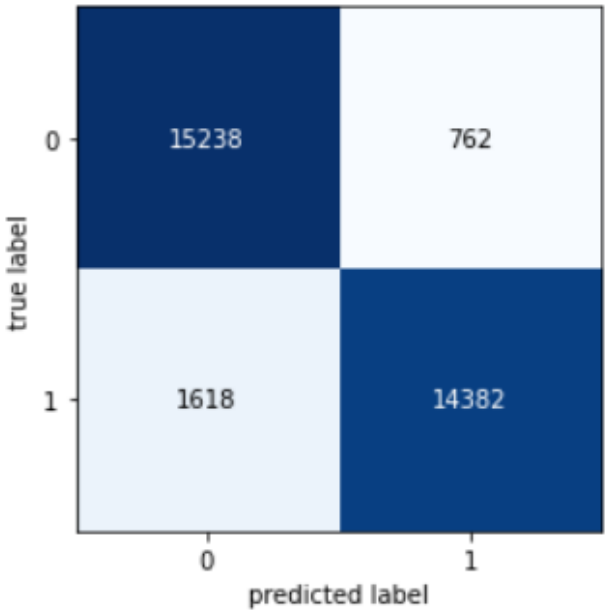

Figure 11(c). CNN Model with Optimizers Nadam

As discussed above in section 3.7.3 false negative is more important than false positive in medical domain and it can be seen in fig. 11, that CNN model with optimizer Nadam is giving less false negative predictions i.e., 1618 as compared to Adam giving 2225and Adagrad giving 2093.

Figure 12, shows the confusion matrix of ResNet-50 with all three optimizers.

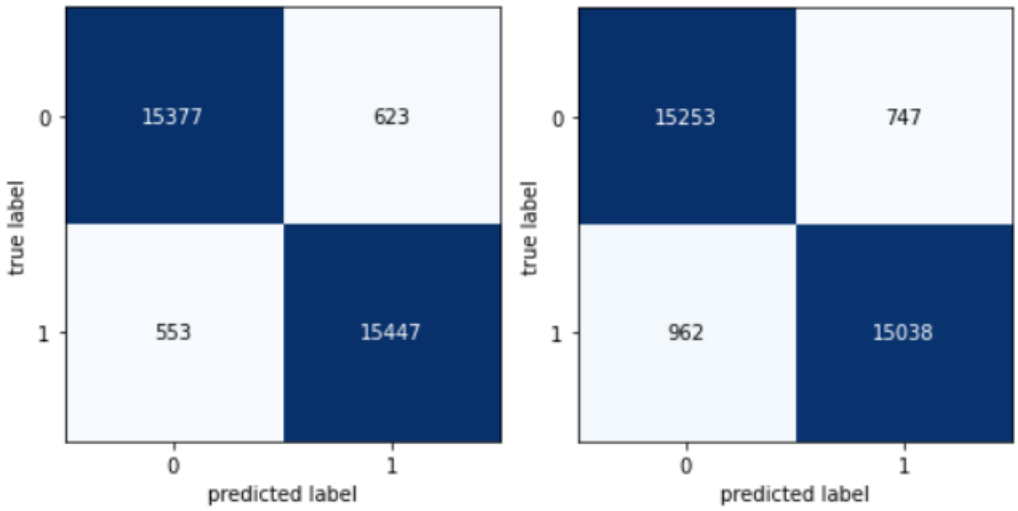

Figure 12. ResNet-50 Model with Optimizers (a) Adam and (b) Adagrad

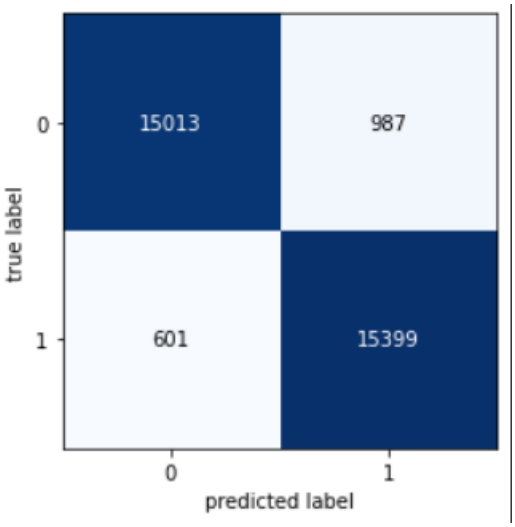

Figure 12(c). ResNet-50 Model with Optimizers Nadam

In fig. 12, ResNet-50 model with optimizer Adam gives less false negative predictions i.e., 563 as compared to Adagrad giving 962 and Nadam giving 601.

Figure 13, shows the confusion matrix of DenseNet-121 with all three optimizers.

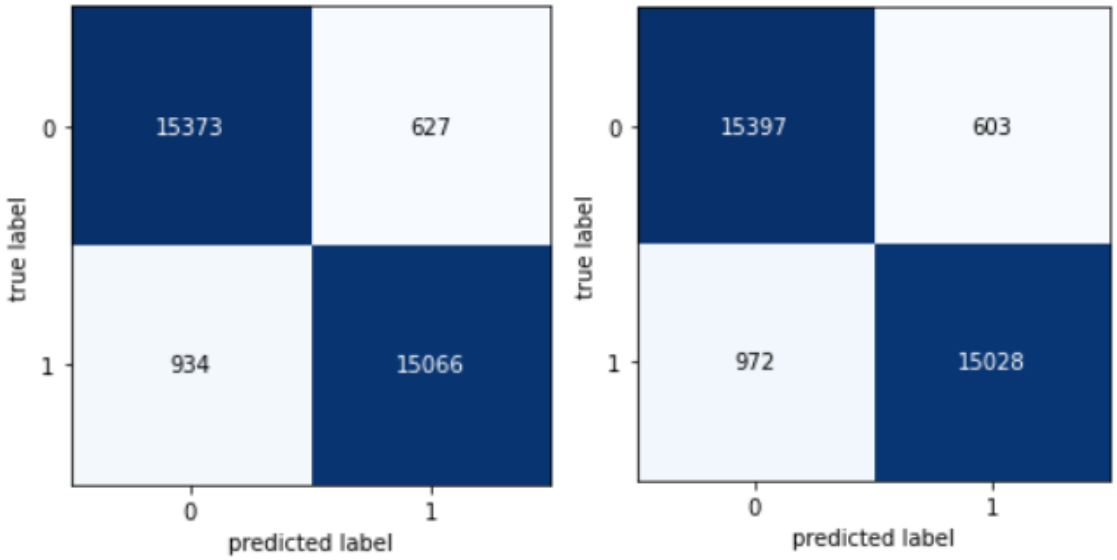

Figure 13. DenseNet-121 Model with Optimizers (a) Adam and (b) Adagrad

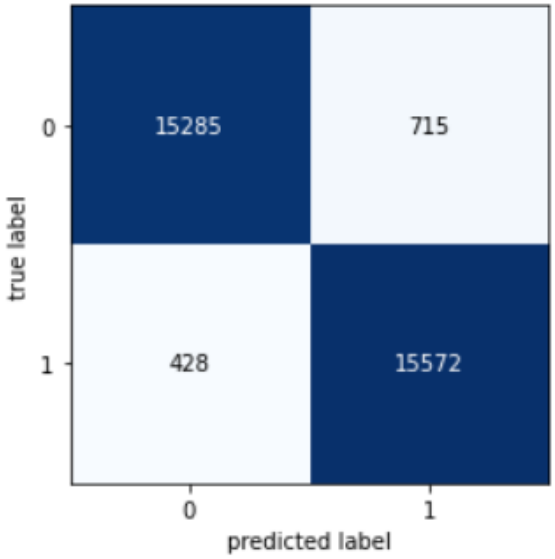

Figure 13(c). DenseNet-121 Model with Optimizers NAdam

In fig. 14, the DenseNet-121 model with optimizer Nadam gives the lease false negative predictions i.e., 428 as compared to Adam and Adagrad with 934 and 972flase negative predictions.

4.3 Classification Report

Figure 14, show the classification report of CNN Model with different optimizers.

	precision	recall	f1-score	support
no_tumor	0.87	0.96	0.92	16000
has_tumor	0.96	0.86	0.91	16000
accuracy			0.91	32000
macro avg	0.92	0.91	0.91	32000
weighted avg	0.92	0.91	0.91	32000

Figure 14(a). CNN Model with Optimizers Adam

	precision	recall	f1-score	support
no_tumor	0.87	0.89	0.88	16000
has_tumor	0.89	0.87	0.88	16000
accuracy			0.88	32000
macro avg	0.88	0.88	0.88	32000
weighted avg	0.88	0.88	0.88	32000

Figure 14(b). CNN Model with Optimizers Adagrad

	precision	recall	f1-score	support
no_tumor	0.90	0.95	0.93	16000
has_tumor	0.95	0.90	0.92	16000
accuracy			0.93	32000
macro avg	0.93	0.93	0.93	32000
weighted avg	0.93	0.93	0.93	32000

Figure 14(c). CNN Model with Optimizers Nadam

In fig. 14, the classification report of CNN model shows that recall and F1 score is greater for Nadam optimizer than Adan and Adagrad. As discussed above it is very important in medical situation that true tumor is detected even if false alarm is raised.

Figure 15, show the classification report of ResNet-50 with different optimizers.

	precision	recall	f1-score	support
no_tumor	0.90	0.95	0.93	16000
has_tumor	0.95	0.90	0.92	16000
accuracy			0.93	32000
macro avg	0.93	0.93	0.93	32000
weighted avg	0.93	0.93	0.93	32000

Figure 15(a). ResNet-50 Model with Optimizers Adam

	precision	recall	f1-score	support
no_tumor	0.94	0.95	0.95	16000
has_tumor	0.95	0.94	0.95	16000
accuracy			0.95	32000
macro avg	0.95	0.95	0.95	32000
weighted avg	0.95	0.95	0.95	32000

Figure 15(b). ResNet-50 Model with Optimizers Adagrad

	precision	recall	f1-score	support
no_tumor	0.96	0.94	0.95	16000
has_tumor	0.94	0.96	0.95	16000
accuracy			0.95	32000
macro avg	0.95	0.95	0.95	32000
weighted avg	0.95	0.95	0.95	32000

Figure 15(c). ResNet-50 Model with Optimizers Nadam

In fig. 15, the recall and F1 score is almost similar for optimizers Adagrad and Nadam.

Figure 16, show the classification report of DenseNet-121 Model with different optimizers.

	precision	recall	f1-score	support
no_tumor	0.94	0.96	0.95	16000
has_tumor	0.96	0.94	0.95	16000
accuracy			0.95	32000
macro avg	0.95	0.95	0.95	32000
weighted avg	0.95	0.95	0.95	32000

Figure 16(a). DenseNet-121 Model with Optimizers Adam

	precision	recall	f1-score	support
no_tumor	0.94	0.96	0.95	16000
has_tumor	0.96	0.94	0.95	16000
accuracy			0.95	32000
macro avg	0.95	0.95	0.95	32000
weighted avg	0.95	0.95	0.95	32000

Figure 16(b). DenseNet-121 Model with Optimizers Adagrad

	precision	recall	f1-score	support
no_tumor	0.97	0.96	0.96	16000
has_tumor	0.96	0.97	0.96	16000
accuracy			0.96	32000
macro avg	0.96	0.96	0.96	32000
weighted avg	0.96	0.96	0.96	32000

Figure 46(c). DenseNet-121 Model with Optimizers Nadam

In fig. 16, the classification report of Densenet-121 shows that recall and F1 score for Nadam optimizer is higher than both Adam and Adagrad.

4.4 ROC Score

Figure 17, shows the ROC graph of CNN models with different optimizers.

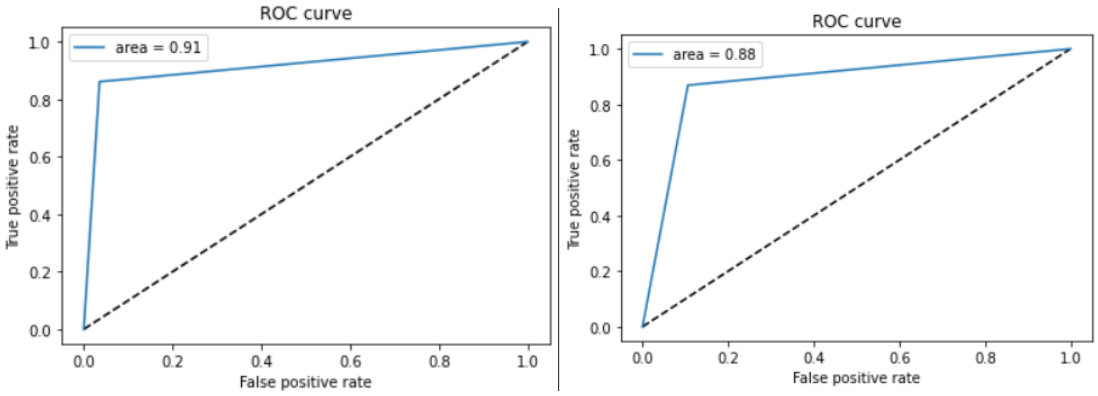

Figure 17. CNN Model with Optimizers (a) Adam and (b) Adagrad

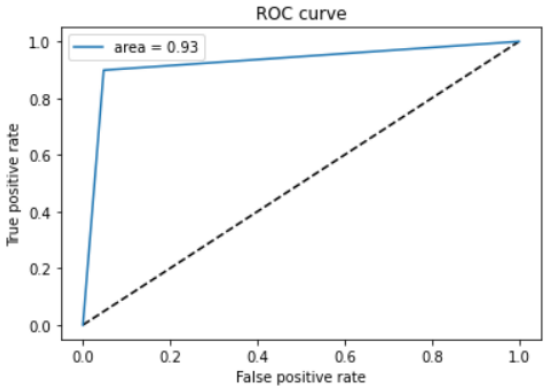

Figure 17 (c). CNN Model with Optimizers Nadam

In fig.17, the ROC score of CNN with model decreased to 0.88 from 0.91 when optimizers changed to Adagrad from Adam and when the optimizer is changed to Nadam it given the highest ROC score for CNN i.e., 0.93.

Figure 18, shows the ROC graph of ResNet-50 with different optimizers.

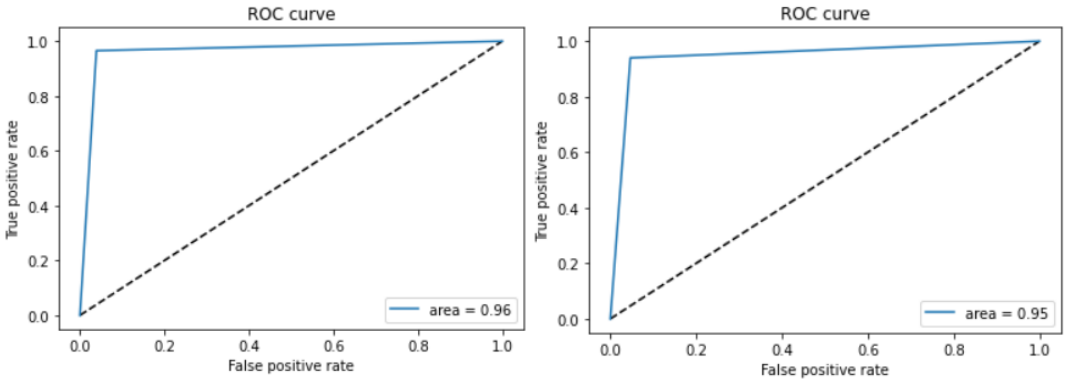

Figure 18. ResNet-50 Model with Optimizers (a) Adam and (b) Adagrad

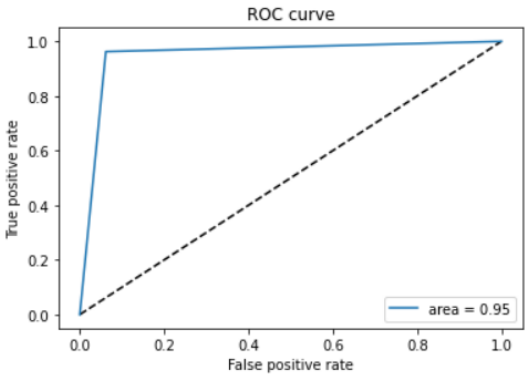

Figure 18(c). ResNet-50 Model with Optimizers Nadam

In fig. 18, the ROC score for ResNet-50 decreases to 0.95 when the optimizers are changed to Adagrad and Nadam.

Figure 19, shows the ROC graph of DenseNet-121 with different optimizers.

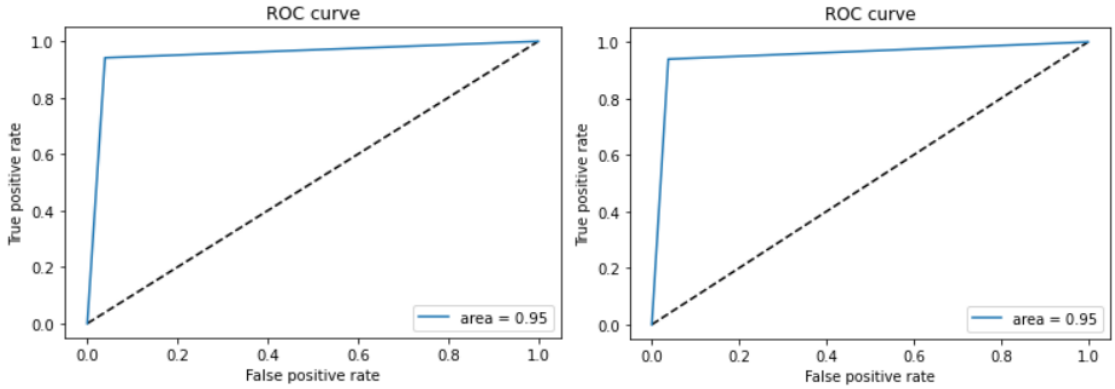

Figure 19. DenseNet-121 Model with Optimizers (a) Adam and (b) Adagrad

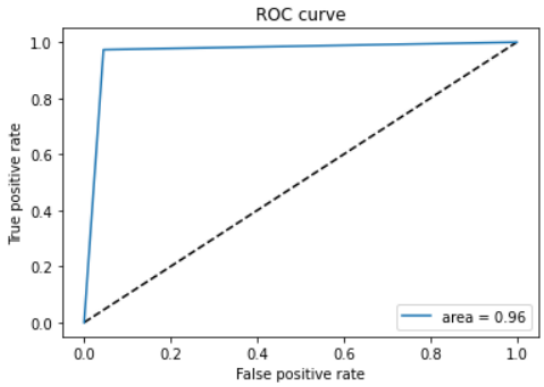

Figure 19(c). DenseNet-121 Model with Optimizers NAdam

In fig. 19, the ROC score for DenseNet-121 remains same for optimizers Adam and Adagrad but increases to 0.96 when optimizer is changed to Nadam.

4.5 Results

After evaluating all the three models above, models behave differently with different optimizers. In case of CNN, the accuracy decreased when optimizer is changed to Adagrad from Adam but it increases when changed to Nadam. Both the pre-trained models performed almost similar, but in case of ResNet-50 the accuracy decreased when the optimizers are changed to Adagrad and Nadam whereas in DenseNet-121 the accuracy remained same for Adagrad but in increases when optimizers is changed

to Nadam. Table 3, shows the accuracy of all the three models with different optimizers.

Model / Optimizers	Adam	Adagrad	Nadam
CNN Model	0.91	0.88	0.93
ResNet-50	0.96	0.95	0.95
DenseNet-121	**0.95**	**0.95**	**0.96**

Table 5. Accuracy of Models

In Fig. 10, it can be seen that train and test accuracy of DenseNet-121 model is quite stable as compared to CNN in Fig. 8 and RestNet-50 model in Fig. 9 also train and test loss gap is very less when compared to other two models. Moreover, while evaluating confusion matrix it is seen that DenseNet-121 with Nadam optimizer (Fig. 13(c)) has given less false negative predictions i.e., 458, then CNN predicting 1618 (Fig. 11(c)) and ResNet-50 predicting 601 (Fig. 12(c)) false negative. Also the recall the F1-score of DenseNet-121 with Nadam optimizer is highest i.e., 0.97 and 0.96 (Fig. 16(c)).

5 Conclusion and Future Scope

This research implemented a CNN model and two pre-trained model i.e., ResNet-50 and DenseNet-121 to detect cancerous cell from histopathology scans of lymph node. Both pre-trained model performed better then the CNN model. But, DenseNet-121 has performed best as compared to ResNet-50 and gives the best accuracy with Nadam optimizers of 96%. The DenseNet-121 model has remained stable throughout the training at every epoch with all three optimizers. The recall and F1 sore of DenseNet-121 turns out to be highest i.e., 0.97 and 0.96 with the minimum false negative predictions of 458. Also the ROC score of DenseNet-121 is highest i.e., 0.96 with optimizer Nadam in comparison with CNN model and ResNet-50 model. The DenseNet-121 model achieved the classification accuracy close to the one trained in VGG16 and EfficientNet-B6, Hence, it can be concluded that DenseNet-121 with Nadam optimzer is also accurate and efficient in detecting cancer from histopathology scans of lymph node.

As there were limited resources available and the models have been trained on Google Colab with limited GPU access and memory constraint therefore, there was limited amount of changes that could be applied to the model and pre trained models. The suggestion for higher dimensions of image is also prevalent as if more resources are available. Further, using the DenseNet-121 model a CAD (computer aided diagnosis) can be created that can act as second opinion for pathologists to speed up the diagnosis for better result and efficiency. It can also help pathologists where diagnosis is taking time due to complexity of lymph node and where there is chance of missing very small tumor regions in histopathology scans of lymph node. Such, CAD can become a helping hand for pathologists and proper treatment can be provided to patient if the diagnosis result came timely. Fast diagnosis system can also reduce the mortality rate.

References

AGRAWAL, S. K. 2021. Metrics to Evaluate your Classification Model to take the right decisions. *Analytics Vidhya* [Online]. Available from: https://www.analyticsvidhya.com/blog/2021/07/metrics-to-evaluate-your-classification-model-to-take-the-right-decisions/.

ALLAOUZI, I., AHMED, M. B. & BENAMROU, B. 2019. An Encoder-Decoder model for visual question answering in the medical domain.

ALTMAN, N. S. 1992. An Introduction to Kernel and Nearest-Neighbor Nonparametric Regression. The American Statistician, 46, 175-185.

BHANDARI, A. 2020. Image Augmentation on the fly using Keras ImageDataGenerator! Analytics Vidhya [Online].

BOESCH, G. 2022. Deep Residual Networks (ResNet, ResNet50). viso.ai [Online]. Available from: https://viso.ai/deep-learning/resnet-residual-neural-network/.

DEAN, J., CORRADO, G. S., MONGA, R., CHEN, K., DEVIN, M., LE, Q. V., MAO, M. Z., RANZATO, M. A., SENIOR, A., TUCKER, P., YANG, K. & NG, A. Y. 2012. Large Scale Distributed Deep Networks.

DEVASSY, B. R. & ANTONY, J. K. 2022. Histopathological Image Classification Using Ensemble Transfer Learning. Machine Learning and Big Data Analytics (Proceedings of International Conference on Machine Learning and Big Data Analytics (ICMLBDA) 2021).

DOZAT, T. 2016. INCORPORATING NESTEROV MOMENTUM INTO ADAM.

DUCHI, J., HAZAN, E. & SINGER, Y. 2011. Adaptive Subgradient Methods for Online Learning and Stochastic Gradient. Machine Learning Research, 12.

FENTON, J. J., TAPLIN, S. H., CARNEY, P. A., ABRAHAM, L., SICKLES, E. A., D'ORSI, C., BERNS, E. A., CUTTER, G., HENDRICK, R. E. & BARLOW, W. E. 2007. Influence of computer-aided detection on performance of screening mammography. New England Journal of Medicine, 356, 1399-1409.

GHADEKAR, P., KHANDELWAL, A., ROY, P., GAWAS, A. & JOSHI, C. 2021. Histopathological Cancer Detection using Deep Learning. 2021 International Conference on Artificial Intelligence and Machine Vision (AIMV).

GLOROT, X., BORDES, A. & BENGIO, Y. 2011. Deep Sparse Rectifier Neural Networks. In: GEOFFREY, G., DAVID, D. & MIROSLAV, D. (eds.) Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics. Proceedings of Machine Learning Research: PMLR.

GOTTAPU, R. D. & DAGLI, C. H. 2018. DenseNet for anatomical brain segmentation. Procedia Computer Science, 140, 179-185.

HASHTINGS, D. 2021a. Classification & Clustering Workshop. Northumbria University.

HE, K., ZHANG, X., REN, S. & SUN, J. 2015. Deep Residual Learning for Image Recognition.

HE, K., ZHANG, X., REN, S. & SUN, J. 2016a. Deep Residual Learning for Image Recognition. IEEE Conference on Computer Vision and Pattern Recognition (CVPR).

HE, K., ZHANG, X., REN, S. & SUN, J. 2016b. Identity Mappings in Deep Residual Networks. Computer Vision – ECCV 2016.

HINTON, G. E., OSINDERO, S. & TEH, Y.-W. 2006. A Fast Learning Algorithm for Deep Belief Nets. Neural Computation, 18**,** 1527-1554.

HUANG, G., LIU, Z., MAATEN, L. V. D. & WEINBERGER, K. Q. 2017. Densely Connected Convolutional Networks. IEEE Conference on Computer Vision and Pattern Recognition (CVPR).

IOFFE, S. & SZEGEDY, C. 2015. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. In: FRANCIS, B. & DAVID, B. (eds.) Proceedings of the

32nd International Conference on Machine Learning. Proceedings of Machine Learning Research: PMLR.
JAISWAL, A. K., PANSHIN, I., SHULKIN, D., ANEJA, N. & ABRAMOV, S. 2019. Semi-Supervised Learning for Cancer Detection of Lymph Node Metastases.
JIA, M., YAN, X. & FU, S. 2019. Histopathologic Cancer Detection Based on Deep Multiple Instance Learning. 2019 15th International Conference on Mobile Ad-Hoc and Sensor Networks (MSN).
K., S. B., NAIR, M. S. & G.R., B. 2018. Automatic mitosis detection in breast histopathology images using Convolutional Neural Network based deep transfer learning. Elsevier.
KAVITHA, M. S., GANGADARAN, P., JACKSON, A., VENMATHI MARAN, B. A., KURITA, T. & AHN, B. C. 2022. Deep Neural Network Models for Colon Cancer Screening. Cancers (Basel), 14.
KRIZHEVSKY, A., SUTSKEVER, I. & HINTON, G. E. 2017. ImageNet classification with deep convolutional neural networks. Communications of the ACM, 60, 84-90.
LEHMAN, C. D., WELLMAN, R. D., BUIST, D. S., KERLIKOWSKE, K., TOSTESON, A. N., MIGLIORETTI, D. L. & BREAST CANCER SURVEILLANCE, C. 2015. Diagnostic Accuracy of Digital Screening Mammography With and Without Computer-Aided Detection. JAMA Intern Med, 175, 1828-37.
LITJENS, G., SANCHEZ, C. I., TIMOFEEVA, N., HERMSEN, M., NAGTEGAAL, I., KOVACS, I., HULSBERGEN-VAN DE KAA, C., BULT, P., VAN GINNEKEN, B. & VAN DER LAAK, J. 2016. Deep learning as a tool for increased accuracy and efficiency of histopathological diagnosis. Sci Rep, 6**,** 26286.
LI, H., REICHERT, M., LIN, K., TSELOUSOV, N., BRAREN, R., FU, D., SCHMID, R., LI, J., MENZE, B. & SHI, K. Differential diagnosis for pancreatic cysts in CT scans using densely-connected convolutional networks. 2019 41st Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC), 2019a. IEEE, 2095-2098.
LI, H., ZHUANG, S., LI, D.-A., ZHAO, J. & MA, Y. 2019b. Benign and malignant classification of mammogram images based on deep learning. *Biomedical Signal Processing and Control,* 51**,** 347-354.
MUNIR, K., ELAHI, H., AYUB, A., FREZZA, F. & RIZZI, A. 2019. Cancer Diagnosis Using Deep Learning: A Bibliographic Review. Cancers (Basel), 11.
MURPHY, K. P. 2012. Machine learning: a probabilistic perspective, MIT press.
NAIR, V. & HINTON, G. E. Rectified linear units improve restricted boltzmann machines. Icml, 2010.
OTT, J. J., ULLRICH, A. & MILLER, A. B. 2009. The importance of early symptom recognition in the context of early detection and cancer survival. Eur J Cancer, 45**,** 2743-8.
PARK, B., CHO, Y., LEE, G., LEE, S. M., CHO, Y.-H., LEE, E. S., LEE, K. H., SEO, J. B. & KIM, N. 2019. A curriculum learning strategy to enhance the accuracy of classification of various lesions in chest-PA X-ray screening for pulmonary abnormalities. Scientific reports, 9, 1-9.
PENNINGTON, J., SOCHER, R. & MANNING, C. D. Glove: Global vectors for word representation. Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP), 2014. 1532-1543.
QIN, C., YAO, D., SHI, Y. & SONG, Z. 2018. Computer-aided detection in chest radiography based on artificial intelligence: a survey. Biomed Eng Online, 17, 113.
SAHINER, B., PEZESHK, A., HADJIISKI, L. M., WANG, X., DRUKKER, K., CHA, K. H., SUMMERS, R. M. & GIGER, M. L. 2019. Deep learning in medical imaging and radiation therapy. Medical physics, 46, e1-e36.
SCHNEIDER, A., HOMMEL, G. & BLETTNER, M. 2010. Linear regression analysis: part 14 of a series on evaluation of scientific publications. Deutsches Ärzteblatt International, 107, 776.
SIMONYAN, K. & ZISSERMAN, A. 2014. Very deep convolutional networks for large-scale image recognition. arXiv preprint arXiv:1409.1556.

SUN, Y., HAMZAH, F. A. B. & MOCHIZUKI, B. 2020. Optimized Light-Weight Convolutional Neural Networks for Histopathologic Cancer Detection. IEEE 2nd Global Conference on Life Sciences and Technologies.

WHO. 2022. Cancer. Available from: https://www.who.int/news-room/fact-sheets/detail/cancer#:~:text=Key%20facts,and%20rectum%20and%20prostate%20cancers.%20[Accessed].

XU, Y., ZHU, J.-Y., CHANG, E. & TU, Z. 2012. Multiple Clustered Instance Learning for Cancer image classification. IEEE.

ZHUANG, L. & GUAN, Y. 2017. Image Enhancement via Subimage Histogram Equalization Based on Mean and Variance. Comput Intell Neurosci, 2017**,** 6029892.

Appendix A. Research Proposal

The research proposal can be view with link provided below:

https://livenorthumbriaac-my.sharepoint.com/:f:/g/personal/w21026488_northumbria_ac_uk/Er1hQNadfNBOrA4O7YRoxYoBYnFOh-zDE1tWg23yvbPhUA?e=krHvVF

Appendix B. Code

All the libraries imported for this study.

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
import cv2
import seaborn as sns
from sklearn.utils import shuffle
from sklearn.metrics import confusion_matrix
from sklearn.model_selection import train_test_split
import itertools
import shutil
from PIL import Image
import os
import tensorflow as tf
import keras
from tensorflow.keras.preprocessing.image import ImageDataGenerator
import random
from random import randint
from skimage.io import imread
from keras.models import Sequential
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from tensorflow.keras.layers import Conv2D, Dense, Dropout, MaxPooling2D, Flatten
from keras.layers import LeakyReLU
from keras.layers import BatchNormalization
from tensorflow.keras.callbacks import EarlyStopping
from skimage.transform import resize
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report
from sklearn.metrics import roc_auc_score, roc_curve, auc
from mlxtend.plotting import plot_confusion_matrix
```

Connecting with Kaggle and Downloading Dataset

```
[ ] !pip install kaggle

Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Requirement already satisfied: kaggle in /usr/local/lib/python3.7/dist-packages (1.5.12)
Requirement already satisfied: certifi in /usr/local/lib/python3.7/dist-packages (from kaggle) (2022.6.15)
Requirement already satisfied: tqdm in /usr/local/lib/python3.7/dist-packages (from kaggle) (4.64.0)
Requirement already satisfied: python-dateutil in /usr/local/lib/python3.7/dist-packages (from kaggle) (2.8.2)
Requirement already satisfied: requests in /usr/local/lib/python3.7/dist-packages (from kaggle) (2.23.0)
Requirement already satisfied: six>=1.10 in /usr/local/lib/python3.7/dist-packages (from kaggle) (1.15.0)
Requirement already satisfied: python-slugify in /usr/local/lib/python3.7/dist-packages (from kaggle) (6.1.2)
Requirement already satisfied: urllib3 in /usr/local/lib/python3.7/dist-packages (from kaggle) (1.24.3)
Requirement already satisfied: text-unidecode>=1.3 in /usr/local/lib/python3.7/dist-packages (from python-slugify->kaggle) (1.3)
Requirement already satisfied: chardet<4,>=3.0.2 in /usr/local/lib/python3.7/dist-packages (from requests->kaggle) (3.0.4)
Requirement already satisfied: idna<3,>=2.5 in /usr/local/lib/python3.7/dist-packages (from requests->kaggle) (2.10)

[ ] ! mkdir ~/.kaggle
    ! cp kaggle.json ~/.kaggle/
    ! chmod 600 ~/.kaggle/kaggle.json

    mkdir: cannot create directory '/root/.kaggle': File exists

Downloading the Dataset

[ ] ! kaggle competitions download histopathologic-cancer-detection

    Downloading histopathologic-cancer-detection.zip to /content
    100% 6.30G/6.31G [00:28<00:00, 236MB/s]
    100% 6.31G/6.31G [00:28<00:00, 240MB/s]
```

Printing Random Images for both Positive & Negative Samples.

Printing Random Images for both Positive (Cancerous) & Negative (Non-Cancerous) Samples

```
[ ] train_dir = '/content/train'
    print("Train Size: {}".format(len(os.listdir(train_dir))))
```

```
Train Size: 220025
```

```
[ ] fig, ax = plt.subplots(2,5, figsize=(20,8))
    fig.suptitle('Histopathologic scans of lymph node sections',fontsize=20)


    # Negatives
    for i, idx in enumerate(df_train[df_train['label'] == 0]['id'][:5]):
        path = os.path.join(train_dir, idx)
        img = imread(path +'.tif')
        ax[0,i].imshow(img)
    ax[0,0].set_ylabel('Negative samples', size='large')


    # # Positives
    for i, idx in enumerate(df_train[df_train['label'] == 1]['id'][:5]):
        path = os.path.join(train_dir, idx)
        img = imread(path+'.tif')
        ax[1,i].imshow(img)
    ax[1,0].set_ylabel('Positive samples', size='large');
```

Counting Number of Samples in Each label.

Counting Number of Samples in Each Label

```
[ ] df_train['label'].value_counts()
```

```
0    130908
1     89117
Name: label, dtype: int64
```

```
[ ] fig = plt.figure(figsize = (6,6))
    ax = sns.countplot(df_train.label).set_title('Label Counts', fontsize = 18)
    plt.annotate(df_train.label.value_counts()[0],
                 xy = (0,df_train.label.value_counts()[0] + 2000),
                 va = 'bottom',
                 ha = 'center',
                 fontsize = 12)
    plt.annotate(df_train.label.value_counts()[1],
                 xy = (1,df_train.label.value_counts()[1] + 2000),
                 va = 'bottom',
                 ha = 'center',
                 fontsize = 12)
    plt.ylim(0,150000)
    plt.ylabel('Count', fontsize = 16)
    plt.xlabel('Labels', fontsize = 16)
    plt.show()
```

Balancing the labels.

Balancing the labels using Random Sampling and reduce the memory usage or potential crash.

Taking 80K images from both labels.

```
SAMPLE_SIZE = 80000
# taking a random sample of class 0
df_0 = df_train[df_train['label'] == 0].sample(SAMPLE_SIZE, random_state = 32)
# taking a random sample of class 1
df_1 = df_train[df_train['label'] == 1].sample(SAMPLE_SIZE, random_state = 32)

# concat the dataframes
df_train = pd.concat([df_0, df_1], axis = 0).reset_index(drop = True)
# shuffle
df_train = shuffle(df_train)

df_train['label'].value_counts()
```

```
1    80000
0    80000
Name: label, dtype: int64
```

Image Normalisation and Augmentation

Applying Image Normalization & Augmentation

```
train_datagen = ImageDataGenerator(
    featurewise_std_normalization=True,
    samplewise_std_normalization=True,
    featurewise_center=True,
    samplewise_center=True,
    #shear_range = 0.6,
    zoom_range = 0.6,
    rotation_range=10,
    horizontal_flip=True,
    # vertical_flip=True,
    rescale = 1./255,
    validation_split=0.2) #Splitting Validation Set for Testing Model
train_generator = train_datagen.flow_from_directory(train_dir,
                                    target_size = (96,96),
                                    batch_size = 32,
                                    class_mode = 'binary')
validation_generator = train_datagen.flow_from_directory(train_dir,
                                                target_size = (96,96),
                                                batch_size = 32,
                                                class_mode = 'binary',
                                                subset = 'validation',
                                                shuffle = False)
```

```
Found 160000 images belonging to 2 classes.
Found 32000 images belonging to 2 classes.
```

CNN Model with Adam Optimizer

```
[ ]  model=Sequential()
     model.add(Conv2D(32,(3,3),strides=1,padding='Same',activation='relu',input_shape=(96, 96, 3)))
     model.add(MaxPooling2D(2,2))
     model.add(BatchNormalization())
     model.add(Conv2D(64,(3,3), strides=1,padding= 'Same', activation='relu'))
     model.add(MaxPooling2D(2,2))
     model.add(BatchNormalization())
     model.add(Conv2D(128,(3,3), strides=1,padding= 'Same', activation='relu'))
     model.add(MaxPooling2D(2,2))
     model.add(BatchNormalization())
     model.add(Conv2D(256,(3,3), strides=1,padding= 'Same', activation='relu'))
     model.add(MaxPooling2D(2,2))
     model.add(BatchNormalization())
     model.add(Flatten())
     model.add(Dropout(0.2))
     model.add(Dense(512, activation = "relu"))
     model.add(Dropout(0.2))
     model.add(Dense(1, activation = "sigmoid"))

     model.summary()

     model.compile(optimizer = tf.keras.optimizers.Adam(learning_rate=0.001,beta_1=0.9,beta_2=0.999) , loss = "binary_crossentropy", metrics=["accuracy"])


     callback = EarlyStopping(monitor='val_accuracy', patience=5)

     seed = 1234
     np.random.seed(seed)
     history = model.fit(train_generator, validation_data=(validation_generator), epochs=20, batch_size=64, callbacks=[callback])
```

CNN Model with Adagrad Optimizer

```
[ ]  model7=Sequential()
     model7.add(Conv2D(32,(3,3),strides=1,padding='Same',activation='relu',input_shape=(96, 96, 3)))
     model7.add(MaxPooling2D(2,2))
     model7.add(BatchNormalization())
     model7.add(Conv2D(64,(3,3), strides=1,padding= 'Same', activation='relu'))
     model7.add(MaxPooling2D(2,2))
     model7.add(BatchNormalization())
     model7.add(Conv2D(128,(3,3), strides=1,padding= 'Same', activation='relu'))
     model7.add(MaxPooling2D(2,2))
     model7.add(BatchNormalization())
     model7.add(Conv2D(256,(3,3), strides=1,padding= 'Same', activation='relu'))
     model7.add(MaxPooling2D(2,2))
     model7.add(BatchNormalization())
     model7.add(Flatten())
     model7.add(Dropout(0.2))
     model7.add(Dense(512, activation = "relu"))
     model7.add(Dropout(0.2))
     model7.add(Dense(1, activation = "sigmoid"))

     model7.summary()

     model7.compile(optimizer = tf.keras.optimizers.Adagrad(0.001) , loss = "binary_crossentropy", metrics=["accuracy"])


     callback = EarlyStopping(monitor='val_accuracy', patience=5)

     seed = 1234
     np.random.seed(seed)
     history = model7.fit(train_generator, validation_data=(validation_generator), epochs=20, batch_size=64, callbacks=[callback])
```

CNN Model with Nadam Optimizer

```
[ ]  model16=Sequential()
     model16.add(Conv2D(32,(3,3),strides=1,padding='Same',activation='relu',input_shape=(96, 96, 3)))
     model16.add(MaxPooling2D(2,2))
     model16.add(BatchNormalization())
     model16.add(Conv2D(64,(3,3), strides=1,padding= 'Same', activation='relu'))
     model16.add(MaxPooling2D(2,2))
     model16.add(BatchNormalization())
     model16.add(Conv2D(128,(3,3), strides=1,padding= 'Same', activation='relu'))
     model16.add(MaxPooling2D(2,2))
     model16.add(BatchNormalization())
     model16.add(Conv2D(256,(3,3), strides=1,padding= 'Same', activation='relu'))
     model16.add(MaxPooling2D(2,2))
     model16.add(BatchNormalization())
     model16.add(Flatten())
     model16.add(Dropout(0.2))
     model16.add(Dense(512, activation = "relu"))
     model16.add(Dropout(0.2))
     model16.add(Dense(1, activation = "sigmoid"))

     model16.summary()

     model16.compile(optimizer = tf.keras.optimizers.Nadam(0.0001) , loss = "binary_crossentropy", metrics=["accuracy"])


     callback = EarlyStopping(monitor='val_accuracy', patience=5)

     seed = 1234
     np.random.seed(seed)
     history = model16.fit(train_generator, validation_data=(validation_generator), epochs=20, batch_size=64, callbacks=[callback])
```

ResNet-50 Model with Adam Optimizer

1. ResNet-50

```
[ ]  ResNet50 = tf.keras.applications.resnet.ResNet50(input_shape=(96,96,3),
                                                      include_top=False,
                                                      weights='imagenet')
     mod1=ResNet50.output
     mod1=tf.keras.layers.Flatten()(mod1)
     mod1=tf.keras.layers.Dense(units=128, activation=tf.nn.relu)(mod1)
     output1=tf.keras.layers.Dense(units=1, activation=tf.nn.sigmoid)(mod1)
     model1= tf.keras.models.Model(inputs=ResNet50.inputs,outputs=output1)

     model1.compile(optimizer=tf.keras.optimizers.Adam(0.00005),
                    loss=tf.keras.losses.BinaryCrossentropy(from_logits= False),
                    metrics=['accuracy'])
     model1.summary()
     hist1=model1.fit(train_generator,epochs=20,validation_data=(validation_generator), verbose=1)
     print("Model Training Complete...")
```

ResNet-50 Model with Adagrad Optimizer

1. ResNet-50 (Adagrad)

```
RestNet50 = tf.keras.applications.resnet.ResNet50(input_shape=(96,96,3),
                                                  include_top=False,
                                                  weights='imagenet')
mod1=RestNet50.output
mod1=tf.keras.layers.Flatten()(mod1)
mod1=tf.keras.layers.Dense(units=128, activation=tf.nn.relu)(mod1)
output1=tf.keras.layers.Dense(units=1, activation=tf.nn.sigmoid)(mod1)
model1= tf.keras.models.Model(inputs=RestNet50.inputs,outputs=output1)

model1.compile(optimizer=tf.keras.optimizers.Adagrad(0.001),
               loss=tf.keras.losses.BinaryCrossentropy(from_logits= False),
               metrics=['accuracy'])
model1.summary()
hist1=model1.fit(train_generator,epochs=20,validation_data=(validation_generator), verbose=1)
print("Model Training Complete...")
```

ResNet-50 Model with Nadam Optimizer

1. ResNet-50 (Nadam)

```
RestNet50 = tf.keras.applications.resnet.ResNet50(input_shape=(96,96,3),
                                                  include_top=False,
                                                  weights='imagenet')
mod1=RestNet50.output
mod1=tf.keras.layers.Flatten()(mod1)
mod1=tf.keras.layers.Dense(units=128, activation=tf.nn.relu)(mod1)
output1=tf.keras.layers.Dense(units=1, activation=tf.nn.sigmoid)(mod1)
model1= tf.keras.models.Model(inputs=RestNet50.inputs,outputs=output1)

model1.compile(optimizer=tf.keras.optimizers.Nadam(0.0001),
               loss=tf.keras.losses.BinaryCrossentropy(from_logits= False),
               metrics=['accuracy'])
model1.summary()
hist1=model1.fit(train_generator,epochs=20,validation_data=(validation_generator), verbose=1)
print("Model Training Complete...")
```

DenseNet-121 Model with Adam Optimizer

```
2. DenseNet-121

[ ]  DenseNet121 = tf.keras.applications.DenseNet121(input_shape = (96,96,3),
                                                     include_top=False,
                                                     weights='imagenet')
     mod1=DenseNet121.output
     mod1=tf.keras.layers.Flatten()(mod1)
     mod1=tf.keras.layers.Dense(units=128, activation=tf.nn.relu)(mod1)
     output1=tf.keras.layers.Dense(units=1, activation=tf.nn.sigmoid)(mod1)
     model77=tf.keras.models.Model(inputs=DenseNet121.inputs,outputs=output1)

     model77.compile(optimizer=tf.keras.optimizers.Adam(0.000005),
                     loss=tf.keras.losses.BinaryCrossentropy(from_logits=False),
                     metrics=['accuracy'])
     model77.summary()

     callback = EarlyStopping(monitor='val_accuracy', patience=5)

     seed = 1234
     np.random.seed(seed)
     hist77 = model77.fit(train_generator, validation_data=(validation_generator), epochs=20, batch_size=32, callbacks=[callback])
```

DenseNet-121 Model with Adagrad Optimizer

```
2. DenseNet-121

[ ]  DenseNet121 = tf.keras.applications.DenseNet121(input_shape = (96,96,3),
                                                     include_top=False,
                                                     weights='imagenet')
     mod1=DenseNet121.output
     mod1=tf.keras.layers.Flatten()(mod1)
     mod1=tf.keras.layers.Dense(units=128, activation=tf.nn.relu)(mod1)
     output1=tf.keras.layers.Dense(units=1, activation=tf.nn.sigmoid)(mod1)
     model77=tf.keras.models.Model(inputs=DenseNet121.inputs,outputs=output1)

     model77.compile(optimizer=tf.keras.optimizers.Adagrad(0.001),
                     loss=tf.keras.losses.BinaryCrossentropy(from_logits=False),
                     metrics=['accuracy'])
     model77.summary()

     callback = EarlyStopping(monitor='val_accuracy', patience=5)

     seed = 1234
     np.random.seed(seed)
     hist77 = model77.fit(train_generator, validation_data=(validation_generator), epochs=20, batch_size=32, callbacks=[callback])
```

DenseNet-121 Model with Nadam Optimizer

2. DenseNet-121 (Nadam)

```
[ ]  DenseNet121 = tf.keras.applications.DenseNet121(input_shape = (96,96,3),
                                                     include_top=False,
                                                     weights='imagenet')
     mod1=DenseNet121.output
     mod1=tf.keras.layers.Flatten()(mod1)
     mod1=tf.keras.layers.Dense(units=128, activation=tf.nn.relu)(mod1)
     output1=tf.keras.layers.Dense(units=1, activation=tf.nn.sigmoid)(mod1)
     model77=tf.keras.models.Model(inputs=DenseNet121.inputs,outputs=output1)

     model77.compile(optimizer=tf.keras.optimizers.Nadam(0.0001),
                     loss=tf.keras.losses.BinaryCrossentropy(from_logits=False),
                     metrics=['accuracy'])
     model77.summary()

     callback = EarlyStopping(monitor='val_accuracy', patience=5)

     seed = 1234
     np.random.seed(seed)
     hist77 = model77.fit(train_generator, validation_data=(validation_generator), epochs=20, batch_size=32, callbacks=[callback])
```