# COMPUTER AIDED DIAGNOSTICS SYSTEM FOR DISEASE PREDICTION USING DEEP LEARNING

Project report submitted in partial fulfilment of the requirement for the degree of Bachelor of Technology

in

**Computer Science and Engineering/Information Technology**

By

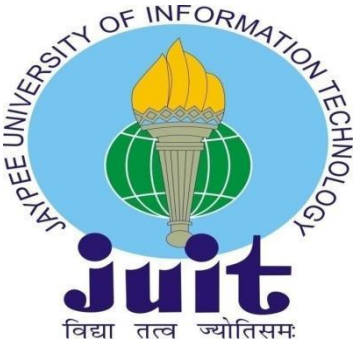
DRON MEHTA (191524)

**UNDER THE SUPERVISION OF**

MR. MUNISH SOOD

DR. HARI SINGH

to




Department of Computer Science & Engineering and Information Technology

**Jaypee University of Information Technology**
**Waknaghat, Solan 173234, Himachal Pradesh,**
**INDIA**

# CANDIDATE'S DECLARATION

I hereby declare that the work presented in this report entitled **"** Computer Aided Diagnostics System for Disease Prediction Using Deep Learning**"** in partial fulfillment of the requirements for the award of the degree of **Bachelor of Technology** in **Computer Science and Engineering/Information Technology** submitted in the department of Computer Science & Engineering and Information Technology**,** Jaypee University of Information Technology Waknaghat is an authentic record of my own work carried out over a period from January 2023 to May 2023 under the supervision of **Mr. Munish Sood** (Assistant Professor (Grade-II), Electronics and Communication Engineering) and **Dr. Hari Singh** (Assistant Professor (SG), Computer Science & Engineering and Information Technology).

I also authenticate that I have carried out the above mentioned project work under the proficiency stream **Artificial Intelligence**.

The matter embodied in the report has not been submitted for the award of any other degree or diploma.

Dron Mehta, 191524.

This is to certify that the above statement made by the candidate is true to the best of my knowledge.

Mr. Munish Sood

Assistant Professor (Grade-II)

Electronics and Communication Engineering

Dr. Hari Singh

Assistant Professor (SG)

Computer Science & Engineering and Information Technology

# PLAGIARISM CERTIFICATE

**JAYPEE UNIVERSITY OF INFORMATION TECHNOLOGY, WAKNAGHAT**

**PLAGIARISM VERIFICATION REPORT**

**Date: .....**

**Type of Document (Tick):** PhD Thesis | M.Tech Dissertation/ Report | B.Tech Project Report | Paper

**Name:** _____ **Department:** _____ **Enrolment No** _____

**Contact No.** _____ **E-mail.** _____

**Name of the Supervisor:** _____

**Title of the Thesis/Dissertation/Project Report/Paper (In Capital letters):** _____

_____

_____

**UNDERTAKING**

I undertake that I am aware of the plagiarism related norms/ regulations, if I found guilty of any plagiarism and copyright violations in the above thesis/report even after award of degree, the University reserves the rights to withdraw/revoke my degree/report. Kindly allow me to avail Plagiarism verification report for the document mentioned above.

**Complete Thesis/Report Pages Detail:**

- Total No. of Pages =
- Total No. of Preliminary pages =
- Total No. of pages accommodate bibliography/references =

**(Signature of Student)**

**FOR DEPARTMENT USE**

We have checked the thesis/report as per norms and found **Similarity Index** at .....(%). Therefore, we are forwarding the complete thesis/report for final plagiarism check. The plagiarism verification report may be handed over to the candidate.

**(Signature of Guide/Supervisor)** **Signature of HOD**

**FOR LRC USE**

The above document was scanned for plagiarism check. The outcome of the same is reported below:

<table>
<tr><th>Copy Received on</th><th>Excluded</th><th>Similarity Index (%)</th><th colspan="2">Generated Plagiarism Report Details (Title, Abstract & Chapters)</th></tr>
<tr><td></td><td rowspan="4">• All Preliminary Pages<br>• Bibliography/Images/Quotes<br>• 14 Words String</td><td rowspan="2"></td><td>Word Counts</td><td></td></tr>
<tr><td>Report Generated on</td><td>Character Counts</td><td></td></tr>
<tr><td></td><td>Submission ID</td><td>Total Pages Scanned</td><td></td></tr>
<tr><td></td><td></td><td>File Size</td><td></td></tr>
</table>

**Checked by**
**Name & Signature** **Librarian**

.....

**Please send your complete thesis/report in (PDF) with Title Page, Abstract and Chapters in (Word File) through the supervisor at plagcheck.juit@gmail.com**

# ACKNOWLEDGEMENTS

Firstly, I express my heartiest thanks and gratefulness to almighty God for His divine blessing making it possible to complete the project work successfully.

I am really grateful and wish my profound indebtedness to Supervisor **Mr. Munish Sood,** Assistant Professor (Grade-II) Department of ECE and **Dr. Hari Singh,** Assistant Professor (SG) Department of CSE/IT, Jaypee University of Information Technology, Waknaghat. Deep knowledge & keen interest of my supervisor in the field of **Artificial Intelligence** to carry out this project. Their endless patience, scholarly guidance, continual encouragement, constant and energetic supervision, constructive criticism, valuable advice, reading many inferior drafts and correcting them at all stages have made it possible to complete this project.

I would also generously welcome each one of those individuals who have helped me straightforwardly or in a roundabout way in making this project a win. In this unique situation, I might want to thank the various staff individuals, both educating and non-instructing, which have developed their convenient help and facilitated my undertaking.

Finally, I must acknowledge with due respect the constant support and patients of my parents.

Group No. 26

**Dron Mehta**

(191524)

# TABLE OF CONTENT

# List of Abbreviations

**ANN** – Artificial Neural Network

**CNN** – Convolutional Neural Network

**ConvNet** – Convolutional Network

**DCNN** – Deep Convolutional Neural Network

**ILSVRC -** ImageNet Large-Scale Visual Recognition Challenge

**MLP** – Multi-Layer Perceptron

**ReLU** – Rectified Linear Unit

**UV** – Ultra-violet

# List of Figures

# List of Graphs

# List of Tables

# ABSTRACT

Due to its negative effects on the central nervous system, brain stroke has recently risen to the top of the list of leading causes of death. Ischemic and hemorrhagic strokes, out of the several forms, mostly harm the central nervous system. However, by recognizing the type of stroke and responding to it quickly through smart health systems, the majority of stroke mortality can be avoided. The World Health Organisation (WHO) estimates that 87% of ischemic stroke cases, 10% of intracerebral haemorrhages, and 3% of subarachnoid haemorrhages impact the global population. Based on a patient's medical records, this study uses deep learning techniques to diagnose, categorise, and predict stroke. The ability of current studies to identify risk variables linked to different forms of stroke is limited.

Using these high features attributes, different models have been trained, they are VGG-16, VGG-19, ResNet-152, Inception V3 and DenseNet-201 for predicting the stroke. The model was trained using a dataset of  patients with images of normal brain and stroke, and its performance was evaluated using various performance metrics. The results showed that the deep learning models ResNet-152 and DenseNet-201 achieved an accuracy of 92%, which is the highest among all models. The findings suggest that deep learning can be an effective tool for stroke prediction and early detection, which can help in preventing or minimizing the impact of stroke.

# Chapter-1
# INTRODUCTION

## 1.1 Introduction

A stroke is a blood clot or bleeding within the brain which could cause everlasting harm that influences mobility, cognition, imaginative and prescient, or conversation. It is a reason for critical neurological harm, headaches and frequently death. Strokes are mostly categorized as ischemic, with two sorts: thrombotic and embolic. In a thrombotic stroke, a blood clot is formed in one of the arteries that supply blood to the brain. An embolic stroke occurs whilst a blood clot forms out of the affected person's brain, normally within the patient's heart, travels through the patient's bloodstream and resorts in narrowed cerebral arteries. A hemorrhage stroke is considered as an extraordinary type of stroke as it occurs while an artery in the brain loses blood or bursts. As a purpose of hemorrhage stroke, the brain cells are damaged through the stress of leaking blood. These stroke types have many similarities, making it difficult to appropriately classify cases using clinical procedures. Furthermore, there are not any clean borders or separation between these kinds.

Stroke symptoms can manifest rapidly or gradually, with some patients experiencing minor symptoms such as amnesia, hearing loss, visual decline, dementia, or abnormal behavior in the early stages. It is crucial to seek prompt medical attention as a stroke can result in irreversible damage if left untreated. Detecting the location of the stroke early enables the doctors to develop appropriate treatment guidelines. Typically, patients undergo CT or MRI before receiving treatment, with the doctor deciding the treatment course based on the results. However, MRI is more expensive and time-consuming than CT, so doctors often recommend the latter. Nonetheless, ischemic stroke is not always apparent in CT images, and the diagnosis relies on the doctor's interpretation of the images. This diagnostic approach heavily relies on the doctor's expertise, and prolonged diagnosis can cause doctor fatigue, leading to a higher chance of diagnostic errors.[1]

The traditional approach to object detection or analysis using Machine Learning (ML) involves several steps, such as image denoising, segmentation, manual feature selection, and classifier training for recognition. These steps are tedious, time-consuming, and labor-intensive, leading to low accuracy and poor diagnostic outcomes. In contrast, Deep Learning (DL) represents a newresearch direction in the field of machine learning, which is more closely aligned with the original purpose of artificial intelligence. Deep Learning can simulate the hierarchical structure of the human brain, learn the inherent principles and presentation levels of data samples, automatically extract features, and exhibit strong learning capabilities. In recent years, the practicality of DL has significantly improved due to the availability of high-performance GPU servers and massive datasets.[14]

A neural network that makes use of convolution is known as a Convolutional Neural Network (ConvNet). Through the use of convolution, two kinds of data can be combined mathematically. In the case of CNN, the data is convolutional after the input is filtered, and the result is a feature map. The filter is also called a kernel, or feature detector, and its dimensions can be, for example, 3x3. To perform convolution, the kernel goes through the input image and performs element-by-element matrix multiplication. The result of each receptive field is written into the object map. Scrolling and indenting can aid in more precise image processing. The convolution layer has a number of filters, each of which produces a filter map. As a result, the layer's output will be a collection of filter maps piled on top of one another. CNN seeks to reduce the size of images without sacrificing elements that are important for precise prediction. Three different types of layers make up the ConvNet architecture: a convolutional layer, a pooling layer, and a fully connected layer.

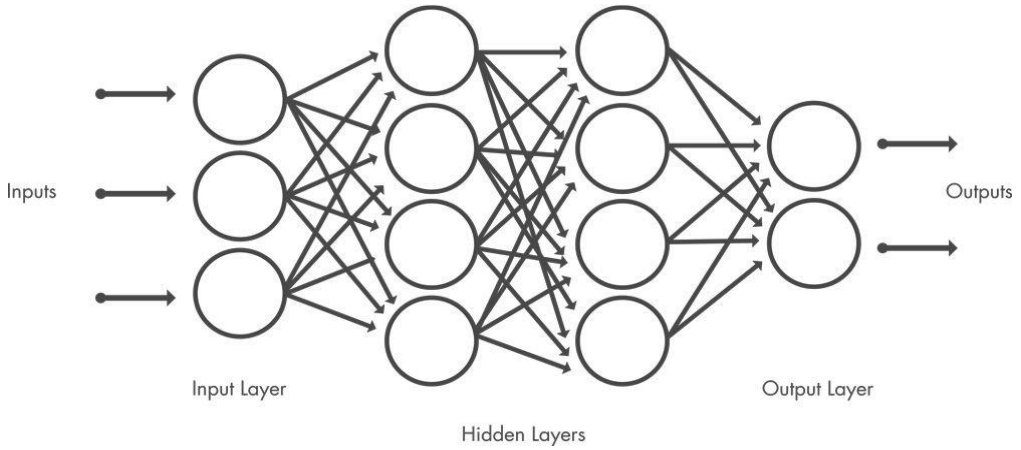



Figure 1: Convolutional Neural Network.

In terms of computing efficiency, CNNs outperform conventional neural networks. CNN uses sharing of parameters and reduction of dimensionality to facilitate quick and simple model deployment. They are adaptable to any technology, including cellphones. ConvNet is not flawless, though. It may appear to be a very sophisticated instrument, but it is still susceptible to attacks from the opposition. For categorization of images, convolutional neural networks are utilized very frequently. CNNs can recognize various things in the images by identifying useful properties. They can be used in medicine, such as magnetic resonance imaging.

## 1.2 Problem Statement

Most research works have a sole contribution to stroke detection, the impact of stroke risk factors and their classification, and are rarely taken into account or considered for research purposes. Also, most studies uses only two to three deep learning algorithms to identify and classify stroke and did not attempt to identify or classify stroke using data collected from case report forms and case summaries.

In this project, deep learning algorithms are planned to overcome the stated drawbacks. The aim of this research work is to get the most significant results for the prediction of stroke using various deep learning techniques. To achieve this aim, we have used techniques like VGG-16, VGG-19, ResNet-152,

Inception V3 and DenseNet-201 on the dataset to get the results. These results will be helpful in future to medical professionals and the patients themselves to spare them from expensive and time taking tests.

## 1.3 Objective

- To build the best classification model using Convolutional Neural Networks such as VGG-16, VGG-19, Inception V3 and other deep learning algorithms which gives the highest accuracy in predicting brain stroke in the images.
- Providing importance to the variation in the dataset to identify stroke with a reasonable accuracy.
- To apply different Data Augmentation techniques to train a model and achieve an optimized model.
- To analyse different feature choosing methods on the dataset and assess the importance of distinct features for predicting the stroke.
- The deep learning models should be able to recognise and learn patterns and relationships in the data to make accurate predictions about the likelihood of stroke. This could involve using techniques like the Convolutional Neural Networks (CNNs) and to process the data, along with feature selection and extraction techniques to identify the most important factors that contribute to stroke risk.

## 1.4 Methodology

In this study, a dataset containing photographs of normal human brain and brain with stroke were used. The dataset used was Brain Stroke CT Image Dataset and was already available in the Kaggle repository. During CNN training, images were pre-processed to increase generalization. Additionally, data sets collected for categorization purposes are used to train CNNs. Finally, test results are analyzed and visualized.

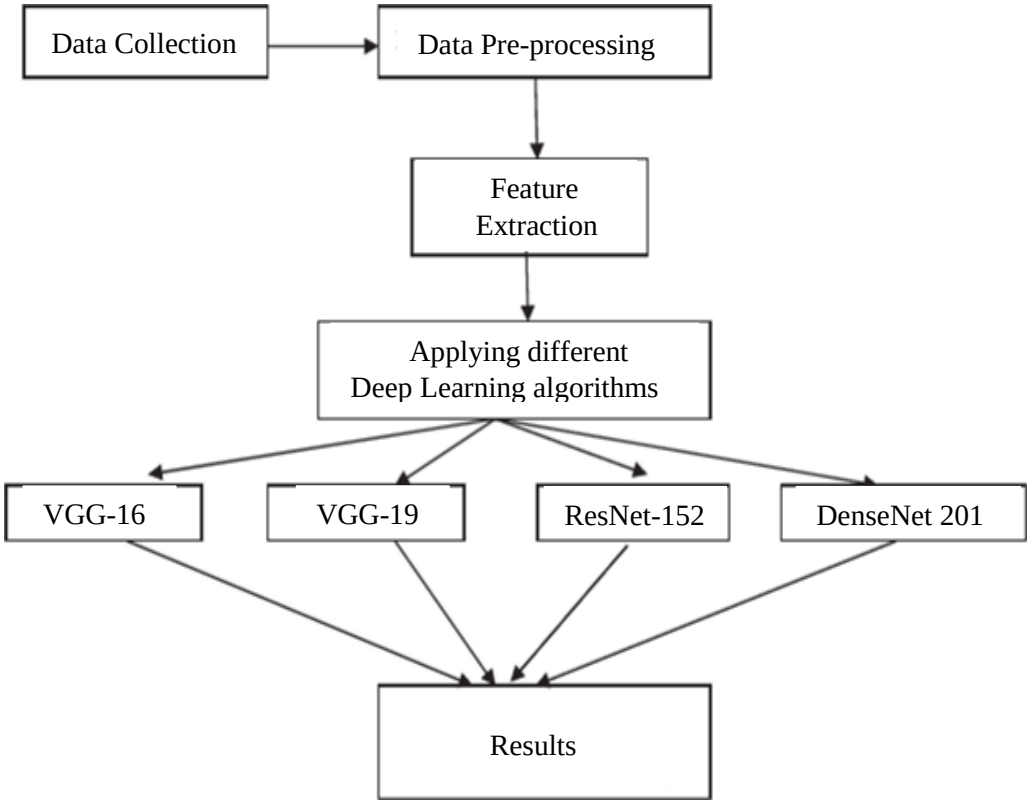



Figure 2: Methodology Used.

1.5 Organization

- Defining the given task and scoping the requirements and determine the feasibility.
- Setting up project codebase and validating data quality.
- Labelling the data and ensuring ground truth to be well-defined.
- Establishing model performance baselines.
- Initializing the project with simple models using start data pipeline.
- Overfitting simple models to the training data.
- Second baseline.
- Performing fixed optimizations model.
- Iteratively repairing the model as the complexity is increased.
- Execution error analysis to examine errors.

# Chapter-2
## LITERATURE SURVEY

We surveyed the existing literature works to obtain the necessary knowledge about various concepts relevant to our current analysis. Here are some of the key conclusions that can be drawn from this:

**[1] Chin, C.L., Lin, B.J., Wu, G.R., Weng, T.C., Yang, C.S., Su, R.C. and Pan, Y.J.** – In recent years, stroke has become one of the top ten leading causes of death in Taiwan. As stroke symptoms require urgent medical attention, early detection is crucial for patient recovery. However, identifying the location of an ischemic stroke in CT images is challenging and relies heavily on medical professionals' assessment. This study uses the Convolutional Neural Network (CNN), a deep learning technique, to develop an automated method for early stroke detection. The system undergoes image pre-processing to remove areas that cannot be the stroke area and selects patch images, which are then increased in amount using data augmentation method. These images with patches are used for training and testing the CNN module. This proposed method was tested with 256 patch images, and the results showed an accuracy of over 90%, demonstrating that the method can effectively assist doctors in diagnosing ischemic stroke.

**[2] Lo, Chung-Ming, Peng-Hsiang Hung, and Daw-Tung Lin. -** Stroke is a major cause of death and disability globally, and quick and accurate diagnosis is essential for the patients. Deep convolutional neural networks (DCNNs) are used in this study to automatically identify acute ischemic stroke from non-contrast computed tomographic (NCCT) images. The image library used to train the DCNN contained 1254 NCCT images from 121 healthy individuals and 96 patients who suffered acute ischemic strokes. The suggested models were developed by transfer learning using ImageNet parameters and machine-generated picture attributes. The models were tested on an independent dataset from another institution and evaluated using tenfold cross-validation. The results showed that AlexNet achieved the highest accuracy which is 97.12%, sensitivity 98.11%, specificity 96.08%, and AUC of 0.9927 without pre-trained

parameters. Using transfer learning, the accuracy of the transferred models are in range 90.49% to 95.49%. AlexNet provided an accuracy of 60.89% when tested on a dataset from a different institution, however the transferred models produced accuracy values between 80.89% and 85.78%. A computer-aided diagnostics system for acute ischemic stroke can be created using this proposed DCNN architecture, and it can produce specialised or generalised models for particular scanners to offer radiologists diagnostic recommendations.

**[3] Xie, Yifeng, Hongnan Yang, Xi Yuan, Qian He, Ruitao Zhang, Qianyun Zhu, Zhenhai Chu, Chengming Yang, Peiwu Qin, and Chenggang Yan. -** The heart is the main source of energy for the brain, an organ that uses a lot of it. An electrocardiogram (ECG) may be able to identify abnormalities in heart function that can be used to diagnose brain disorders like stroke. However, diagnosing brain diseases through ECG requires a significant amount of time and domain expertise. Without the need for prior expert knowledge, deep learning approaches can create nonlinear correlations between ECG and stroke. In this paper, using 12-lead ECG data, we propose the data-driven classifier DenseNet for stroke prediction. After being tweaked, our model had an accuracy for training of 99.99% and an accuracy for prediction of 85.62%. This is the first study to look at the connection between stroke and ECG using deep learning. Our results suggest that ECG is a valuable adjunct technique for stroke diagnosis.

**[4] Oksuz, Ilkay. -** The main aim of this study is to improve diagnostic image quality of brain MRI scans for downstream tasks like segmentation by detecting and correcting motion-related artefacts. In clinical practise, artefacts provide a significant difficulty that might lead to images of poor quality. The proposed method employs a synthetic artefact production mechanism based on MR physics, a residual U-net network for correction, and dense convolutional neural networks for detection. The method achieved 97.8% accuracy for artefact detection in a dataset of 28 brain MRI stroke segmentation cases used for algorithm validation. The suggested correction algorithm also demonstrated enhanced image quality and accurate segmentation. The study highlights the

importance of high-quality images for accurate segmentation and improving automatic image analysis pipelines. The method's effectiveness was assessed using brain MRI stroke segmentation.

**[5] Zhang, Rongzhao, Lei Zhao, Wutao Lou, Jill M. Abrigo, Vincent CT Mok, Winnie CW Chu, Defeng Wang, and Lin Shi.** - A prevalent cerebral vascular disorder that primarily affects the elderly is acute ischemic stroke. In order to increase blood flow to the affected area and lower the risk of disability or death, an accurate diagnosis and prompt treatment are essential. The size and location of infarctions are crucial diagnostic indicators, but manually locating and quantifying stroke lesions can be tiresome and time-consuming. In this article, we introduce a novel deep 3D convolutional neural network-based automated technique for segmenting acute ischemic stroke using diffusion weighted images (DWI). Through an end-to-end, data-driven process, our method efficiently utilises 3D contextual data while automatically learning highly discriminative features. We constructed our network with dense connections so that information and gradients could be transmitted through it without interruption. To address the serious class imbalance issue in the data, we trained our model using the Dice objective function. In order to assess our technique, we gathered a DWI dataset of 242 subjects, including 90 for training, 62 for validation, and 90 for testing. Our model outperformed other state-of-the-art CNN methods notably on a number of parameters, including Dice similarity coefficient (79.13%), lesion-wise precision (92.67%), and lesion-wise F1 score (89.25%). We used the ISLES2015-SSIS dataset to further evaluate the generalizability of the model, and our results were extremely good. Because it is quick and accurate, our suggested method might be used in clinical settings.

**[6] Gaidhani, Bhagyashree Rajendra, R. R. Rajamenakshi, and Samadhan Sonavane.** - Machine learning has been used more and more in the previous couple of decades to analyse medical datasets. Deep learning technology has recently found broad success in a number of fields, particularly in the field of radiology. These fields include computer vision, image recognition, and natural language processing. This study focuses on the use of deep learning models and

convolutional neural networks (CNNs) to identify brain stroke using MRI data. The suggested strategy entails dividing brain stroke MRI images into normal and pathological groups and then utilising semantic segmentation to pinpoint problematic areas. LeNet and SegNet, two different kinds of CNNs, are employed in this process. Pre-processed stroke MRI images are used for classification, and all layers of the LeNet network are trained to distinguish between normal and abnormal patients. The SegNet autoencoder-decoder model, which is used to segment the aberrant patient data, is then fed the two-dimensional array of abnormal patient data. All layers of SegNet—aside from the fully connected layer—are trained before being used. According to the experimental findings, the segmentation model's accuracy ranges from 85-87%, while the classification model's accuracy ranges from 96-97%. These findings show that deep learning models are useful for diagnosing medical images, particularly for identifying brain strokes.

**[7] Choi, Yoon-A., Se-Jin Park, Jong-Arm Jun, Cheol-Sig Pyo, Kang-Hee Cho, Han-Sung Lee, and Jae-Hak Yu.** - The aging population and the COVID-19 pandemic have increased the need for smart healthcare services that can be used in daily life and provide non-face-to-face health services. The majority of research have relied on costly and time-consuming imaging modalities since stroke is a disease that necessitates ongoing medical observation and monitoring, particularly in the older population. The use of non-invasive EEGs to predict stroke has gained more attention in recent studies, although the processing and prediction algorithms take time. The authors suggest a novel methodology for the direct deployment of deep learning models on unprocessed EEG data without relying on the frequency features of EEG in order to overcome this problem. With the use of various specialised deep learning models for time series data categorization and prediction, including LSTM, Bidirectional LSTM, CNN-LSTM, and CNN-Bidirectional LSTM, the proposed deep learning-based stroke illness prediction model was created and trained using real-time EEG sensor data. The experimental results demonstrate that the CNN-bidirectional LSTM model can predict stroke with 94.0% accuracy, low FPR (6.0%) and FNR (5.7%) utilising the raw EEG data,

highlighting the potential of non-invasive techniques for real-time stroke diagnosis in everyday life. These discoveries are expected to significantly enhance early stroke diagnosis with less expense and suffering than current measuring methodologies.

**[8] Rao, B. Nageswara, Sudhansu Mohanty, Kamal Sen, U. Rajendra Acharya, Kang Hao Cheong, and Sukanta Sabut.** - Intracerebral haemorrhage (ICH), which happens when a blood vessel in the brain tissue bursts, is the most typical type of hemorrhagic stroke. The identification of this ailment is a major medical emergency that requires immediate attention. Radiologists manually analyse a substantial number of non-contrast computed tomography (NCCT) brain images as part of the existing procedure for diagnosing ICH. In this research, we describe a deep learning automatic transfer approach that combines ResNet-50 and dense layer to precisely detect cerebral bleeding on NCCT brain pictures. We used a total of 1164 NCCT brain scans from 62 patients at the Kalinga Institute of Medical Science in Bhubaneswar who had hemorrhagic strokes to assess the model. Based on the input data it receives, the proposed approach classifies each individual CT image as either hemorrhagic or normal. Our deep transfer learning approach fared better than ResNet-50 alone, with accuracy, specificity, and sensitivity rates of 99.6%, 99.7%, and 99.4%, respectively. These findings show the deep transfer learning model's potential as a clinical decision support tool to aid radiologists in the automatic hemorrhagic stroke diagnosis.

**[9] Ashrafuzzaman, Md, Suman Saha, and Kamruddin Nur.** - This research work focuses on stroke, a medical illness that develops when the blood supply to the brain is interrupted or diminished, leading to the death or impairment of brain cells. For stroke victims, early diagnosis and treatment are essential, and machine learning has the potential to make these procedures better. The authors propose a Convolutional Neural Network (CNN) model to forecast the probability of stroke in patients at an early stage. The model is an improved version of a multi-layer perceptron, consisting of input and output layers, as well as several hidden layers. To train and test the model, the authors use a healthcare

dataset with eleven features and one target class. The most crucial features for categorization are extracted using feature selection techniques. The suggested model outperforms other machine learning models tested with an accuracy of 95.5%. The scientists contend that by using their methodology, stroke diagnoses might be made more precise and effective, which would eventually improve patient outcomes.

**[10] Gautam, Anjali, and Balasubramanian Raman.** - The classification of computed tomography (CT) scan images into normal, hemorrhagic stroke, and ischemic stroke is intended to change medical treatment for brain strokes, a leading cause of mortality worldwide. It uses a recently developed CNN (convolutional neural network) model that combines CNN and image fusion techniques. The quality of the CT images is preprocessed using multi-focus image fusion, and they are then fed into a 13-layer CNN architecture for the purpose of classifying strokes. With the use of two separate datasets and two different experiments, the robustness of the CNN method is assessed. In the first trial, dataset 1's stroke classification accuracy was 98.33%, and in the second, it was 98.77%. For dataset 2, the first and second experiments' accuracy results are 92.22% and 93.33%, respectively. The Himalayan Institute of Medical Sciences (HIMS), Dehradun, India, provided the datasets for the experiments. The results demonstrate that the suggested approach outperforms the CNN architectures AlexNet and ResNet50.

**[11] Kaur, Mandeep, Sachin R. Sakhare, Kirti Wanjale, and Farzana Akter. -** The advancement of modern technology has led to the development of non-invasive techniques for aiding healthcare systems. Stroke is one of the most life-threatening cardiovascular diseases, and detecting it in its early stages can be crucial to saving a patient's life. Prior research has shown that patients often experience ministrokes or transient ischemic attacks (TIA) before a full-blown stroke occurs. However, using expensive methods such as MRI and CT scans for diagnosis can be prohibitive, particularly in countries such as India where the incidence of stroke is increasing. This paper seeks to address this issue by

proposing a non-invasive approach to early stroke diagnosis. The study uses time series-based algorithms, including LSTM, biLSTM, GRU, and FFNN, to predict strokes based on processed EEG data. The results show that all algorithms perform well, but GRU produces the most accurate results with 95.6% accuracy. The study shows the possibility of using brain wave measurements to forecast the development of strokes, which can help doctors spot stroke victims early and possibly save their lives.

**[12] Liu, Yan, Bo Yin, and Yanping Cong.** - With ischemic strokes making up around 85% of all cerebral strokes, cerebral stroke has grown to be a significant health concern. The incidence of the disease has been proven to decrease with early detection and prevention, although forecasting ischemic stroke is difficult due to the multi-modal nature of associated data. This problem has been solved by using a novel multi-model fusion convolutional neural network structure for forecasting the likelihood of suffering a stroke. This method improves the accuracy of ischemic stroke prediction by interpreting multi-modal data using several end-to-end neural networks. The suggested method begins by extracting variables like age, gender, history of hypertension, and other demographics from both structured and streamed data using a convolutional neural network. The features from structured and streaming data are then combined using a neural network feature fusion model. Finally, training is used to create a predictive model for stroke probability. Experimental results show that the accuracy of ischaemic stroke prediction using this method is 98.53%. This high accuracy can be beneficial in preventing the occurrence of stroke.

**[13] Lai, Yu-Liang, Yu-Dan Wu, Huan-Jui Yeh, Ya-Ting Wu, Hsin-Yu Tsai, and Jung-Chih Chen.** - Currently, stroke specialists predict the functional outcomes of their patients based on extensive data and clinical experience. The goal of this effort is to construct a model to accurately identify imaging signals that may predict these outcomes. The researchers used magnetic resonance imaging (MRI) of both ischemic and hemorrhagic stroke patients to train a convolutional neural network (CNN) dubbed VGG-16 to predict the

functional outcomes of patients following a 28-day hospital stay. 44 patients (24 men and 20 women) from Taoyuan General Hospital and China Medical University Hsinchu Hospital were enrolled for this study. The patients were evaluated using the “modified Rankin Scale (mRS)” and “National Institutes of Health Stroke Scale (NIHSS)” assessments, and the CNN was trained separately for men, women, and mixed groups. The findings demonstrate that the VGG-16 CNN was highly accurate in predicting stroke patients' functional outcomes. Doctors may predict the functional outcomes of stroke patients in clinical settings with the help of this cutting-edge deep-learning technique, and provide an automated decision support system for tailored suggestions and therapies.

# Chapter-3
## SYSTEM DESIGN AND DEVELOPMENT

### 3.1 Analytical

Date Set Used in the Project:

Brain Stroke CT Image Dataset from Kaggle is used in this project work. The dataset consisted of 2501 total images, with 1551 normal brain images and 950 images of brain with stroke. Figure 3 and 4 show images from normal brain and images of brain with stroke datasets respectively.

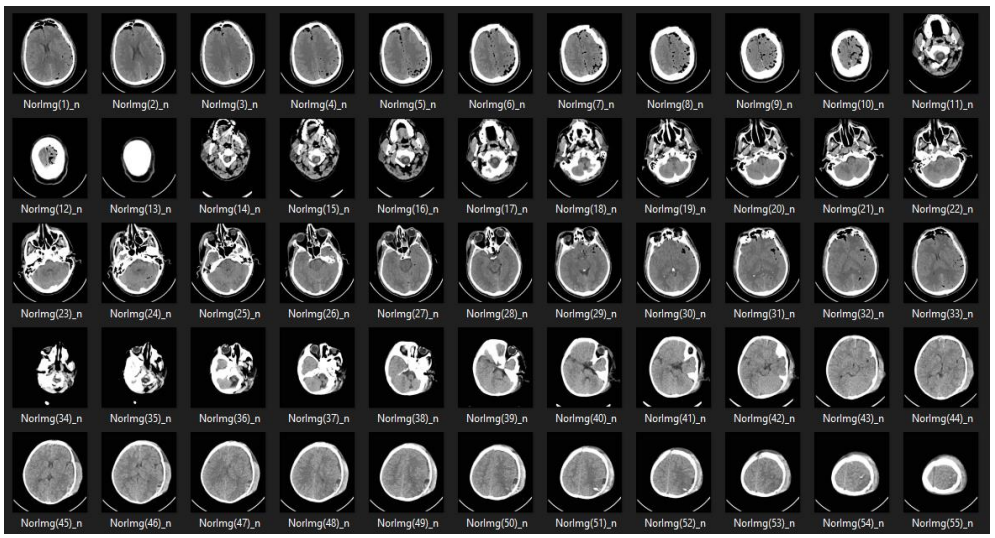



Figure 3: Normal Brain images.

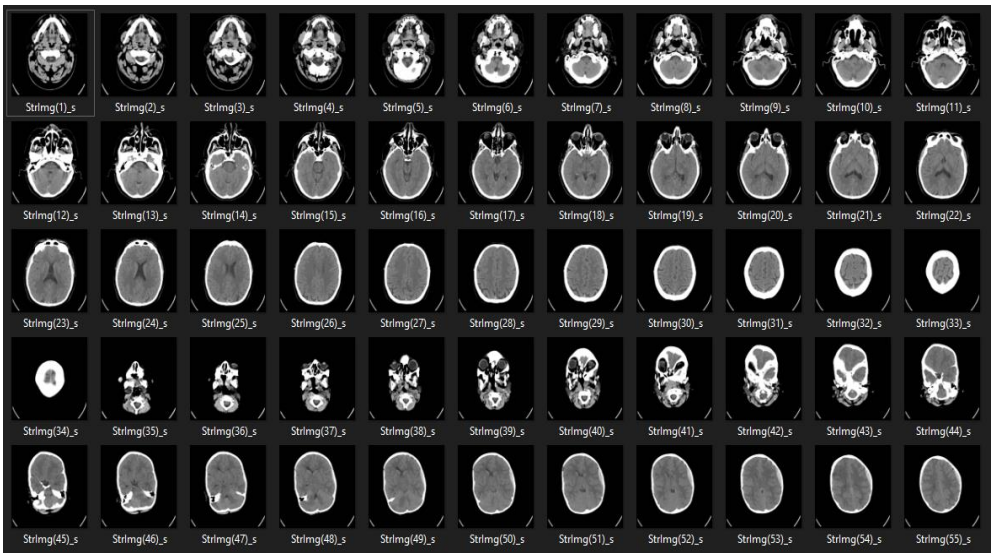



Figure 4: Images of Brain with Stroke.

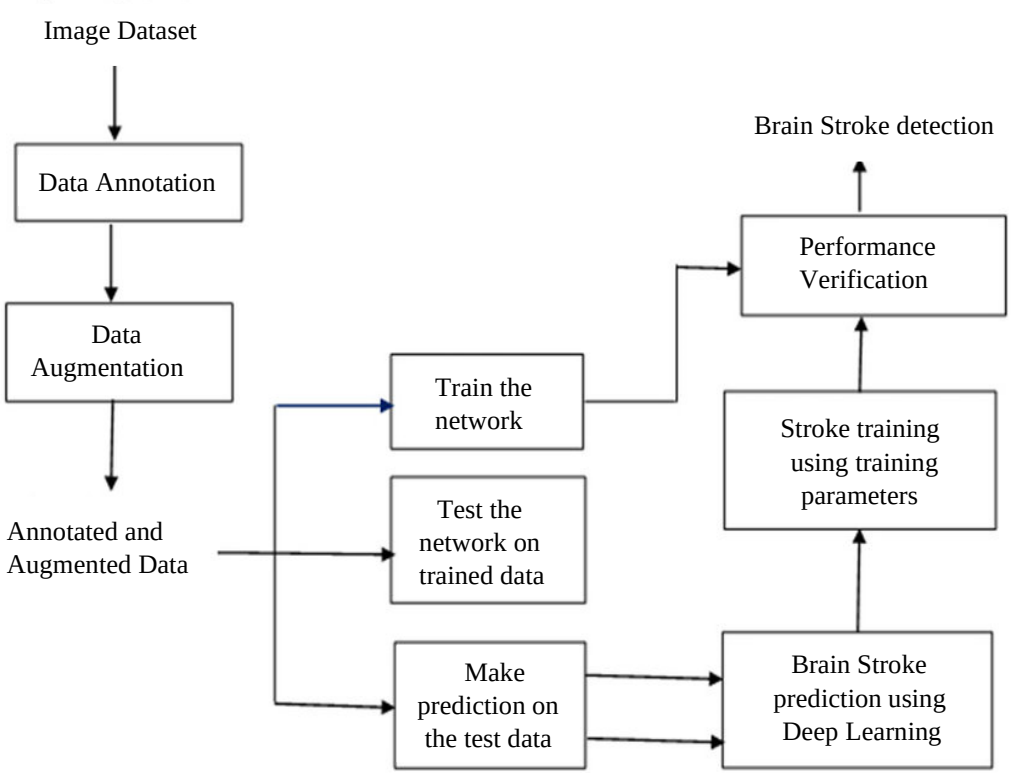



Figure 5: Data Flow Diagram.

**Convolution Neural Networks**

Deep learning is a novel method for pattern identification that uses the Convolutional Neural Network (CNN) including images. Automating picture classification may be aided by this method. An input layer, an output layer, and a number of hidden layers make up a CNN. In the 1980s, CNNs were initially created and put to use. The best CNN could do at that time was decrypt scribbled numbers, used to read pin codes, zip codes, and other data primarily in the postal industry.

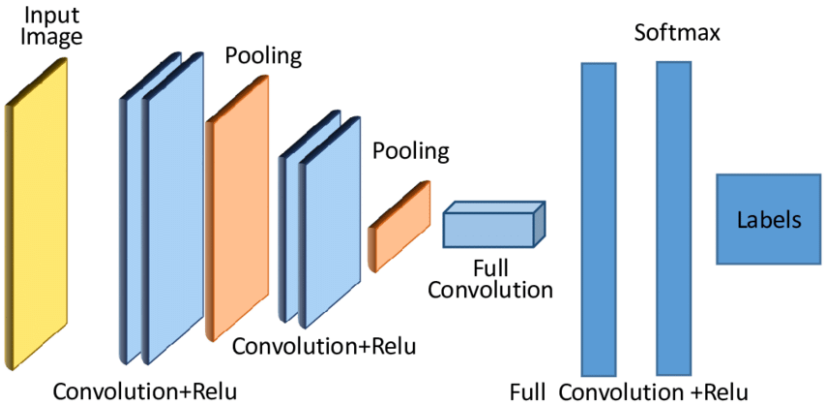



Figure 6: Convolution Neural Networks Architecture.

Since it is effective at solving issues in computer vision domains, such as systems for segmentation, monitoring devices, classification techniques, and other machine learning and video analysis, the convolutional neural network (CNN) is somewhat a well-known technique. The size and number of kernels, the number of hidden layers, and the kind of activation function are some of the choices that can directly affect the CNN design. In most cases, the first layer captures important properties like horizontal and diagonal edges. This result is forwarding to the next stage of object detection, which finds more intricate objects like corners or linked edges. More sophisticated features, such as objects, faces, and other attributes, can be recognized as we investigate further into the network. Finding patterns in images for item, face, and scene identification using CNNs is very useful. Additionally, it excels at categorising non-image data, including audio, time series, and signal data. Neurons in CNNs contain biases and weights. With each new training example, the model constantly updates these values that it has gained throughout training. The CNN architecture then moves on to classification after learning the features at various levels.

CNN model is primarily composed of layers such as input layer, convolution layer, pooling layer and output layer.

1. Input Layer: The training data and input shape are provided to the model in the input layer, which is necessary for the model to function successfully in the next phases.

2. Convolution Layer: The foundation of a convolution neural network is the convolution layer. The model's convolution layer is crucial for adding nonlinearity. Convolution layer uses filters which brings more parameters to the computation which helps the model to train efficiently on the given dataset. A convolutional layer takes an image $A^{(m-1)}$ (with $K^m$ channels) as input and computes a new image $A^{(m)}$ (consisting of Om channels) as output. The output at each channel is called the functional map and is calculated as:

$$\mathbf{A_o^{(m)} = g_m \left(\Sigma_k \, W_{ok}^{(m)} * A_k^{(m-1)} + b_o^{(m)}\right)}$$

(Where * denotes the (2D) convolution operation)

$$W_{ok} * A_k[s,t] = \Sigma_{p,q} A_k[s+p,t+q] W_{ok}[P-1-p,Q-1-q]$$

where $W^{(m)}_{ok}$ is a matrix of the form $P_m \times Q_m$ and $b_o^{(m)}$. The matrix $W^{(m)}_{ok}$ defines the spatial filters that the layer can use to detect or enhance features in the input image. During the network training process, the specific actions of this filter are automatically learned from the data.

2.1 Filters: When applied to the input, filters act as a single template or pattern that locates correspondences between the stored template and various locations/regions in the input image. The filter will carefully scan the image and pull out its most important information. There are different types of filters like Gaussian Blur, Prewitt Filter and many more. Filters are generally matrix of smaller dimensions like 3x3 or 5x5, the smaller the dimensions the better the details in the input image is found.

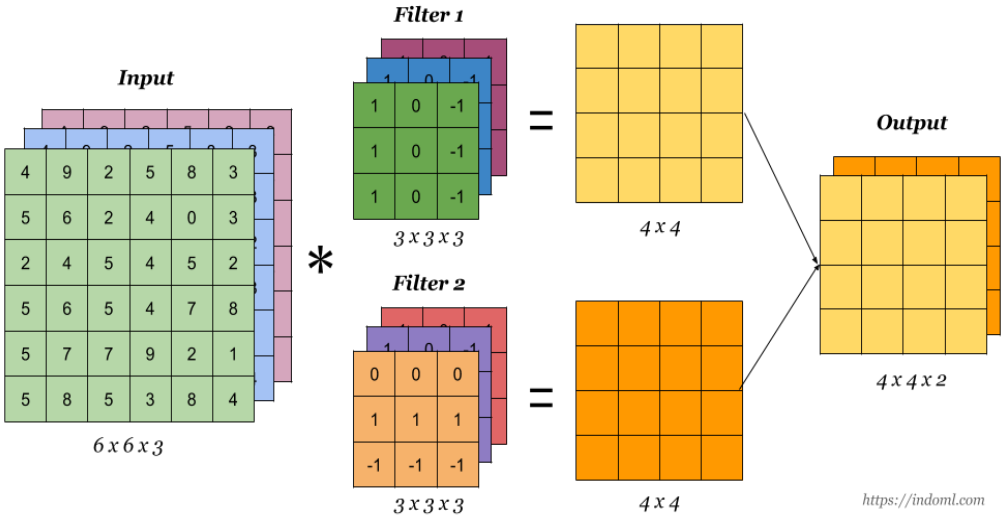



Figure 7: Filters in CNN.

Various filters can be used over the same input which can be merged to generate better output. Different datasets react differently to different filters, therefore it is necessary to try different types of filters to find out which one suite the given dataset (generally stack of images).

3. Pooling Layer: In order to pool the features inside the filter's coverage region, a two-dimensional filter must be applied to each channel of the feature map.

Layers are stacked to reduce the dimensionality of the feature maps. As a result, the network has to do fewer computations and learn fewer parameters. A pooling layer condenses the features that are present in the region of the convolutional layer's feature map. Further operations are performed on the combined features rather than the neatly arranged features produced by the convolutional layer. The model is therefore better able to manage changes in feature position in the input image.

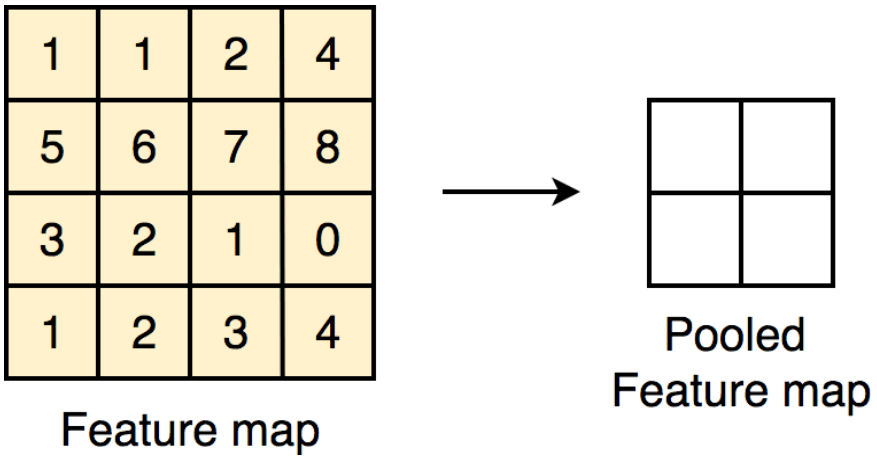



Figure 8: Pooling Layer.

Pooling Layer Types:

Max Pooling: This feature map area is covered by the filter, and a pooling procedure is used to choose the largest elements. The output after the max pooling layer, therefore, will be a feature map that includes the standout features from the prior feature map.

The average of the components discovered in the feature map area that the filter has chosen is evaluated using average pooling. Accordingly, max pooling creates the most noticeable features in a particular patch of the feature map, whereas average pooling produces the average of the features present in the patches.

Global Pooling: Global pooling reduces the channels of the feature map to a single value. As a result, the nh x nw x nc feature map becomes a 1 x 1 x nc feature map. This is equivalent to using a filter with nh x nw feature map dimensions. There are further choices for global maximum or global average pooling.

4. Dense Layer in CNN: A dense layer in any neural network is one that has a strong connection to the layer below it. To put it another way, each neuron in the layer above is connected to each neuron in the layer below. This layer is the one that artificial neural networks use most frequently. Each neuron in the layer below is output by a dense layer neuron in the model. Vectors and matrices are multiplied there by thick layer neurons. The row vector of the output from the previous layer is comparable to the column vector of the dense layer when using the matrix-vector multiplication technique. Every neuron in the dense layer receives the output from every neuron in the layer above. In this case, we assert that the output has passed through the dense layer if the preceding layer generates a (M x N) matrix after aggregating the output from each neuron. There should be N neurons in the thick layer. It is workable in Keras. The output format depends on how many neurons or units are set in the dense layer. Dense layer functions in Keras implement the following operations –

**Output = Activation (Points (Input, Kernel) + Bias)**

In the equation above, the activation function is utilised to carry out per-element activation, with the kernel being the weight matrix and bias vector produced by the layer. The Keras dense layer in the output layer performs the inner product of the input tensor and the weight kernel matrix. The output values are elementwise activated when a bias vector is introduced.

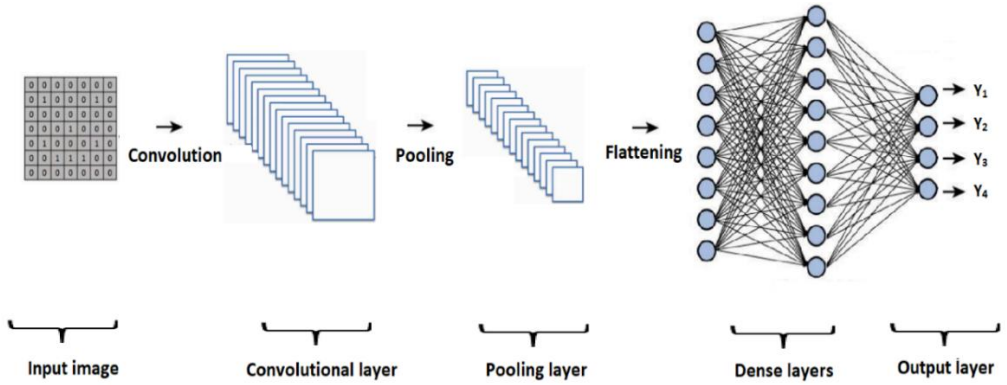



Figure 9: Dense Layer in CNN.

With transfer learning, we leverage an existing model to address several, connected issues. In essence, our goal is to use the knowledge gained from one activity to help in generalization in a new different one. To address the issue,

we employ the model's pre-trained weights or model architecture. On the provided dataset, we have used the pre-trained weights of the VGG16, VGG19, and Inception V3 models, changed the output layer, and solved a classification problem.

Various Deep Learning Algorithms are discussed out of which some are used to perform various classifications on the dataset.

A. Visual Geometry Group VGG-16 [15]

The 2014 ILSVR competition was won using the convolution neural network (CNN) architecture VGG16. It is observed as one of the most innovative vision model designs ever created. The furthermost significant feature of VGG16 is that it prioritised preserving convolution layers of 3x3 filtering systems with a stride 1 rather than moving between multiple padding and max pool layers. It also constantly followed the same padding & max pooling layer of 2x2 filters with a stride 2 and the max pooling and convolution layers are evenly spaced throughout the whole design. It's outputs are ultimately employed as two totally connected layers (FC) and a softmax. When classifying 1000 images into 1000 different groups, the object recognition and categorization algorithm VGG16 achieves an overall better accuracy. It is a popular method for categorising photographs and is straightforward to use to communicate knowledge. The 16 weighted layers are represented by the 16th digit of VGG16. The weighted layers, sometimes referred to as trainable parameter layers, make up only 16 of the total 21 layers in VGG16. There are 5 max pooling layers, 3 thick layers, and 13 convolutional layers.

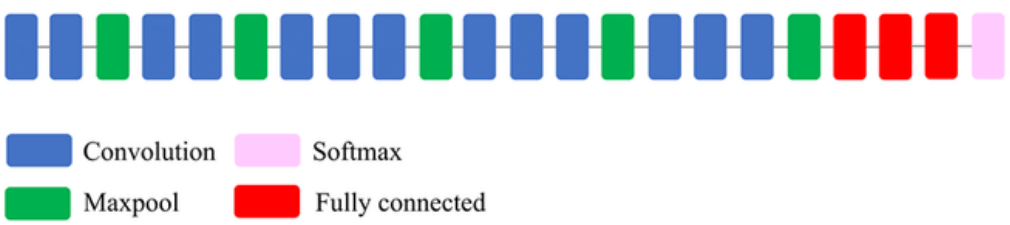



Figure 10: VGG-16 Schematic Diagram.

VGG16 was demonstrated to be the model with the top performance among all the settings on the ImageNet dataset. Any network configuration is thought to have a fixed input with R, G, and B channels that is 224 x 224 pixels in size. The only pre-processing carried out is to normalise each pixel's RGB values. The average score is subtracted from each pixel to achieve this. After ReLU activations, the image is routed via the initial stacking of two convolution layers, each of which has an incredibly small receptive size of 3 3. These two layers each include 64 filters. The activations then proceed via a second stack that is identical to the first but has 128 filters as opposed to the first stack's 64 filters. The size is therefore 56 x 56 x 128 after the second layer. A max pooling layer and 3 convolutional layers make up the final stack, which is subsequently appended. The convolution layers stacks are followed by three fully linked layers and a flattening layer. There is also a pre-trained VGG16 model in the keras applications package. The ImageNet weights are present in the pre-trained model. To leverage the pre-trained model and train on our personalized photographs, we may apply transfer learning concepts.

Although this concept is incredibly straightforward, attractive, and user-friendly, there are some difficulties with it. This model has more than 138M total parameters, and it is more than 500MB in size. The model's application is severely constrained as a result, especially in edge computing where a longer inference time is needed. Additionally, there is no unique solution to the issue of vanishing or exploding gradients. Both GoogleNet and ResNet used skip connections and inception modules to solve this issue.

B. VGG-19 [16]

Model used in the project is VGG-19 which, unlike other models, is 19 layers deep, where first layer is used for taking input and the last layer is for giving the predicted values. VGG-19 was proved to be very useful for this model where we have used activation functions, and the output layer is using sigmoid function for giving us the final value (‘softmax’ is also a potent function which can be used). The Visual Geometry Group at the University of Oxford School of

Engineering has developed a pre-trained convolutional neural network called VGG-19. From the ImageNet database, we can load a pre-trained version of the network that was built using more than 1 million images. Images of objects like keyboards, pencils, and numerous animals can be categorized into 1000 object categories by a pretrained network. The network subsequently learned detailed representations of a variety of images. The input network image measures 224 by 224 pixels.

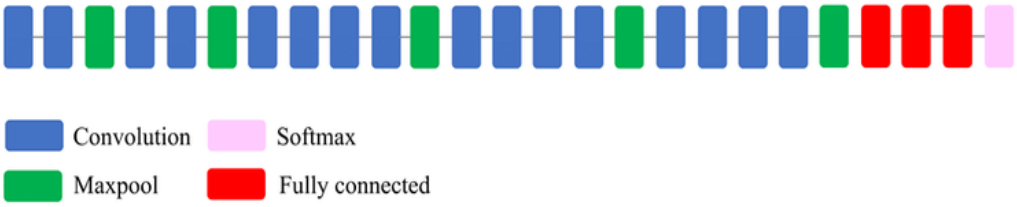



Figure 11: VGG-19 Schematic Diagram.

C. Inception V3 [17]

Network with convolutions Object recognition and image analysis are made easier with the use of Inception V3, a GoogleNet plugin that was created. The ImageNet Recognition Challenge first saw the Google Inception Convolutional Neural Network, that is now in its third generation. Inception V3 features less than 25 million parameters as opposed to AlexNet's 60 million, which was designed with the intention of allowing deeper networks without letting the number of variables to become unmanageably huge.

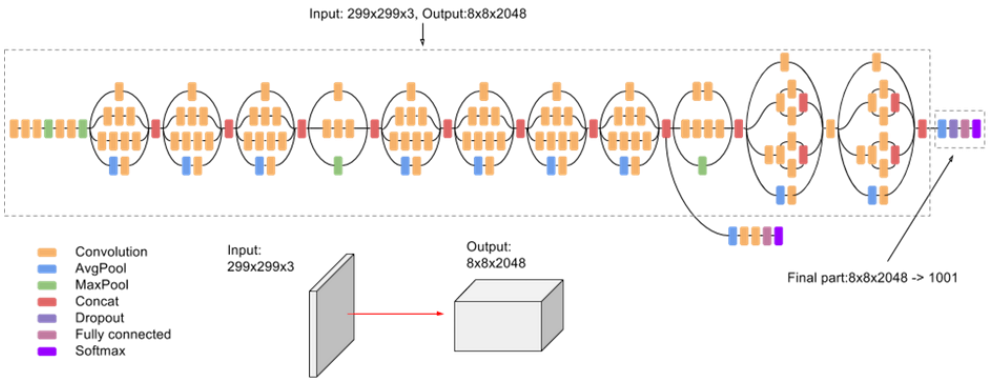



Figure 12: Inception V3 Schematic Diagram.

Similar to how ImageNet may be viewed as a database of categorised visual objects, Inception helps in the computer vision field of categorization.

Numerous applications have made use of the Inception V3 architecture, frequently in conjunction with ImageNet's pre-trained models. Inception V3 is a 42-layer deep convolutional neural network. The ImageNet database contains a pre-trained deep version of the system that has been trained on even more than a million photos. The pretrained network can classify photos into 1000 different item categories, including various animals, a keyboard, a mouse, and a pencil. The network now offers rich visual elements for a variety of photographs as a result. The network's input image has a size of 299 by 299 pixels. The Inception V3 model has 42 total layers, which is a few more than the Inception V1 and V2 models. However, this method's effectiveness is truly amazing.

D. DenseNet-201 [18]

The research team of Facebook AI Research created the deep convolutional neural network (CNN) architecture known as DenseNet 201. It is a development of the DenseNet architecture, which aims to solve the vanishing gradient issue in extremely deep neural networks.

The number of layers in the network is indicated by the "201" in DenseNet 201. It consists of dense blocks and transition layers throughout its 201 total layers. A dense block is a collection of convolutional layers where each layer feeds the feature maps of all levels before it while also passing on its own feature maps to all layers after it. Due to enhanced feature reuse and gradient flow made possible by the layers' dense connection, training time was cut in half and accuracy increased.

DenseNet 201's transition layers serve to prevent overfitting and lower the computational cost of the network by reducing the number of feature mappings and their spatial resolution. A fully connected layer with softmax activation comes after a global average pooling layer in the network's final layer, which generates the output for classification.

On the ImageNet dataset and other image classification tasks, DenseNet 201 has achieved cutting-edge performance. It has also been applied to medical imaging tasks including classifying mammograms and segmenting brain tumours.

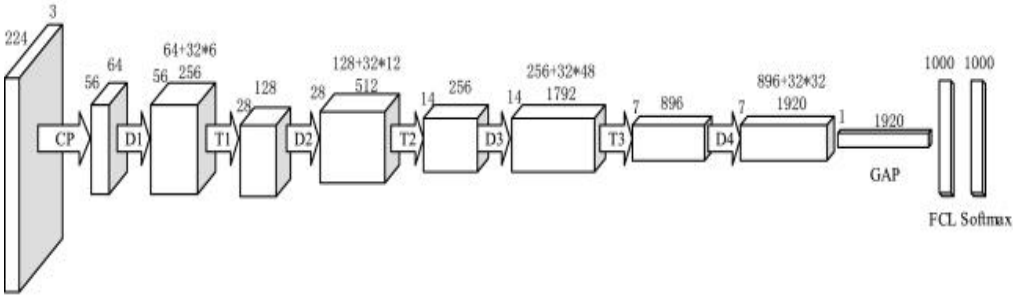



Figure 13: DenseNet 201 Schematic Diagram.

E. ResNet152 [19]

ResNet-152 is a deep convolutional neural network (CNN) architecture developed by Microsoft Research. It is an extension of the original ResNet architecture, which was designed to address the problem of vanishing gradients in very deep neural networks.

The "152" in ResNet-152 refers to the number of layers in the network. It has a total of 152 layers, with a combination of convolutional layers, batch normalization layers, and pooling layers. The network also includes skip connections, which allow information to bypass certain layers and flow directly to deeper layers in the network.

These skip connections help in solving the problem of vanishing gradients and allow the training of very deep neural networks.

ResNet-152 has achieved state-of-the-art performance on various image classification tasks, including the ImageNet dataset. It has also been used in applications such as object detection, semantic segmentation, and speech recognition.

In medical imaging, ResNet-152 has been used for tasks such as diagnosing diabetic retinopathy and detecting lung cancer. Its ability to learn complex features from large amounts of data makes it a powerful tool for analyzing medical images and assisting in clinical decision-making.

F. MobileNet [20]

MobileNet is a lightweight deep convolutional neural network (CNN) architecture designed for efficient mobile and embedded vision applications. It was developed by Google researchers, and its main goal is to provide a good trade-off between model size, latency, and accuracy.

MobileNet uses depthwise separable convolutions, which split the standard convolutional operation into two separate operations: a depthwise convolution, which applies a single filter to each input channel, and a pointwise convolution, which applies a 1x1 filter to combine the outputs of the depthwise convolution. This reduces the number of parameters and computations required by the network, making it more efficient and faster to run on mobile devices.

MobileNet has several versions with different sizes and complexities, ranging from MobileNetV1 to MobileNetV3. MobileNetV1 has achieved state-of-the-art performance on various image classification tasks, including the ImageNet dataset, while MobileNetV2 and V3 have introduced several improvements to the architecture, such as linear bottlenecks, inverted residuals, and squeeze-and-excitation modules, which have further increased the accuracy and efficiency of the network.

MobileNet has been used in various applications, such as object detection, face recognition, and gesture recognition, and it is also a popular choice for mobile and embedded deep learning frameworks such as TensorFlow Lite and Core ML.

G. AlexNet [21]

AlexNet is a deep convolutional neural network created specifically for image classification tasks. The accuracy of picture classification has greatly improved since its introduction in 2012 by Alex Krizhevsky, Ilya Sutskever, and Geoffrey Hinton. That same year, it also won the ImageNet Large Scale Visual Recognition Challenge (ILSVRC).

AlexNet has a total of eight layers consisting three completely linked layers and five convolutional layers. It makes use of dropout regularisation and the Rectified Linear Unit (ReLU) activation function to prevent overfitting. It also

uses data augmentation methods like random cropping and horizontal flinging to increase the training dataset.

Utilising graphics processing units (GPUs) to speed up the computation of the convolutional layers is one of AlexNet's key innovations. This made it possible to train the network despite its numerous characteristics in a manageable length of time.

On the ILSVRC dataset, AlexNet demonstrated state-of-the-art performance, and many other deep learning models have used this architecture as a starting point.

The design and development of a brain stroke prediction system using deep learning typically involves several steps:

1. Data Collection: The first step in developing a stroke prediction system is to collect data from a variety of sources, including medical images such as MRI or CT scans of the brain, time-series data such as EEG or blood pressure readings, and patient data such as demographic information, medical history, and lab test results.

2. Data Pre-processing: After the data is collected, it is required to be pre-processed to prepare it for use by the deep learning algorithms. This may involve tasks such as image segmentation, noise reduction, or feature extraction.

3. Model Selection: Selecting an appropriate deep learning model that can learn from the pre-processed data and make accurate predictions is the next step. Common deep learning models used in stroke prediction include Convolutional Neural Networks (CNNs) for image data.

4. Model Training: Once the model has been selected, it needs to be trained on the pre-processed data. This involves feeding the data into the model and adjusting its parameters so that it can learn to make accurate predictions.

5. Model Validation: After training the model, it needs to be validated on independent datasets to ensure that it can generalize to new data and make accurate predictions.

6. Deployment: Once the model has been validated, it can be deployed in a clinical setting to aid in stroke prediction. This may involve integrating the model with medical devices or software systems used by healthcare professionals.
7. Monitoring and Maintenance: The stroke prediction system should be continually monitored and maintained to ensure that it remains accurate and up-to-date with the latest medical research and best practices.

Overall, the design and development of a brain stroke prediction system using deep learning is a complex process that requires expertise in both machine and deep learning and healthcare. The system must be validated on independent datasets and continually monitored and maintained to ensure its reliability and accuracy.

# Chapter-4
## EXPERIMENTS AND RESULTS ANALYSIS

Various stages of the Project:

- Libraries Used

The following libraries have been used for the implementation of the model and for training the data.

```
import pandas as pd
import numpy as np
import os
from glob import glob
import random
import matplotlib.pylab as plt
import keras.backend as K
from sklearn.model_selection import train_test_split
import tensorflow as tf
import keras
from keras.utils.np_utils import to_categorical
from keras.models import Sequential
from tensorflow.keras.optimizers import SGD, RMSprop, Adam, Adagrad, Adadelta
from keras.layers import Dense, Dropout, Activation, Flatten,
BatchNormalization, Conv2D, MaxPool2D, MaxPooling2D, GlobalAveragePooling2D
from tensorflow.keras.callbacks import ReduceLROnPlateau, ModelCheckpoint, EarlyStopping
from tensorflow.keras import Model
from keras.preprocessing.image import ImageDataGenerator
%matplotlib inline
```

Figure 14: Libraries Used.

In Python, a file is regarded as a module. The module must be imported using the import keyword before being used. By importing the module, the function or variables existing in the file can be utilised in another file. Instead of importing the entire module, we can merely import the necessary functions and variable names from the module. The "from" keyword can be used to import only certain items when we only want those items imported. By using import and the module name, i.e., the filename or the library to be used, we can import the entire module.

A Python library is a term that is widely used to describe a group of linked modules. It contains code bundles that can be used with many different software.

For programmers, it makes python programming simpler and more practical, due to the fact that we wouldn't have to write the same code for different apps.

➢ Dataset

- Loading the Dataset.

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

```
imagePatches = glob('/content/drive/MyDrive/Brain_Stroke_Data/**/*.jpg', recursive=True)
for filename in imagePatches[0:20]:
    print(filename)
```

```
/content/drive/MyDrive/Brain_Stroke_Data/Stroke/StrImg(1)_s.jpg
/content/drive/MyDrive/Brain_Stroke_Data/Stroke/StrImg(10)_s.jpg
/content/drive/MyDrive/Brain_Stroke_Data/Stroke/StrImg(12)_s.jpg
/content/drive/MyDrive/Brain_Stroke_Data/Stroke/StrImg(11)_s.jpg
/content/drive/MyDrive/Brain_Stroke_Data/Stroke/StrImg(13)_s.jpg
/content/drive/MyDrive/Brain_Stroke_Data/Stroke/StrImg(14)_s.jpg
/content/drive/MyDrive/Brain_Stroke_Data/Stroke/StrImg(16)_s.jpg
/content/drive/MyDrive/Brain_Stroke_Data/Stroke/StrImg(15)_s.jpg
/content/drive/MyDrive/Brain_Stroke_Data/Stroke/StrImg(17)_s.jpg
/content/drive/MyDrive/Brain_Stroke_Data/Stroke/StrImg(2)_s.jpg
/content/drive/MyDrive/Brain_Stroke_Data/Stroke/StrImg(18)_s.jpg
/content/drive/MyDrive/Brain_Stroke_Data/Stroke/StrImg(19)_s.jpg
/content/drive/MyDrive/Brain_Stroke_Data/Stroke/StrImg(20)_s.jpg
/content/drive/MyDrive/Brain_Stroke_Data/Stroke/StrImg(21)_s.jpg
/content/drive/MyDrive/Brain_Stroke_Data/Stroke/StrImg(22)_s.jpg
/content/drive/MyDrive/Brain_Stroke_Data/Stroke/StrImg(23)_s.jpg
/content/drive/MyDrive/Brain_Stroke_Data/Stroke/StrImg(24)_s.jpg
/content/drive/MyDrive/Brain_Stroke_Data/Stroke/StrImg(25)_s.jpg
```

Figure 15: Loading the dataset.

Uploading the Brain Stroke dataset images of which were taken from the Kaggle repository. The pre-existing dataset consist of 2501 images which were distributed among two classes – Normal and Stroke.

- Dividing the dataset into two classes

```
class0=[] #0=Normal
class1=[] #1=Stroke
for filename in imagePatches:
  if filename.endswith('n.jpg'):
    class0.append(filename)
  else:
    class1.append(filename)
```

Figure 16: Datatypes in dataset.

- Data and Image Augmentation

Algorithms can utilise machine learning to classify and differentiate between different objects for image recognition. This emerging method employs data augmentation to build models with higher performance. Machine learning models need to be capable of identifying objects in all lighting conditions, including rotation, close-up, and fuzzy images. A synthetic approach of combining training data with precise changes was sought by researchers. By creating new data points from the current data, a variety of techniques referred to as data augmentation may be employed to artificially enhance the amount of data. In order to do this, deep learning models are used to either add new data points or make a few little adjustments to the data that already exists. By employing data augmentation to develop machine learning models more rapidly and accurately, businesses can reduce their dependency on the processing and collecting of training data. By including more distinctive instances to training datasets, data augmentation enhances the functionality and output of machine learning models. Deep learning models perform more accurately when given enough and sizable datasets. The collection and classification of data for deep learning models may require a lot of time and money. The initial image can be altered in a number of ways with the help of data augmentation, which may also be useful in giving adequate data for larger models. Making straightforward modifications to visual data is common in data augmentation. The following are typical image processing tasks for data augmentation: padding, random rotation, rescaling, translation, cropping, zooming, vertical and horizontal flipping, etc.

```
#function to resize image size
from matplotlib.image import imread
import cv2
def get_image_arrays(data, label):
    img_arrays = []
    for i in data:
        if i.endswith('.jpg'):
            img = cv2.imread(i ,cv2.IMREAD_COLOR)
            img_sized = cv2.resize(img, (75, 75), interpolation=cv2.INTER_LINEAR)
            img_arrays.append([img_sized, label])
    return img_arrays
```

Figure 17: Resizing the images.

A one-dimensional feature vector is used as an input to a deep learning model. Convolutional neural networks, two- and three-dimensional feature tensors, and other more modern learning models are also available. As training advances, the machine adjusts its internal settings to point each feature tensor in the right direction. After training, the computer can predict the target for feature tensors that were not known before. In other words, each sample must contain the same amount of attributes. The fact that words and images usually have a wide range of sizes and, as a result, a diversity of features, makes it difficult to process them. A three-dimensional tensor is used as the input to a convolutional neural network (CNN) used to categorise images, with each channel's value standing in for a feature. The three-dimensional feature tensor must have the same dimensions for every image. The size of images and the feature tensors that accompany them, however, is often different. Photographs may be resized to the same size, but it might be challenging to do so without destroying any potential patterns.

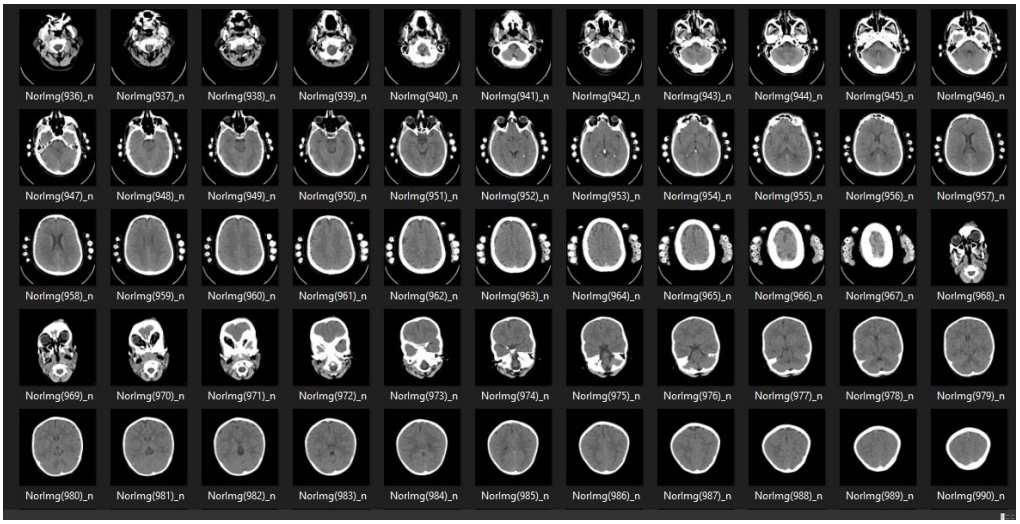



Figure 18: Images after Augmentation.

➢ Combining the dataset

```
combined_data = np.concatenate((class0_array, class1_array),dtype='object')
random.seed(42)
random.shuffle(combined_data)
```

```
<__array_function__ internals>:180: VisibleDeprecationWarning: Creating an ndarray from ragged
```

```
X = []
y = []
for features,label in combined_data:
    X.append(features)
    y.append(label)
```

```
X = np.array(X).reshape(-1,75,75,3)
X.shape
```

```
(1900, 75, 75, 3)
```

Figure 19: Combining the images of both datasets.

- Dividing the dataset in training and testing

```
[ ] X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20, random_state=42)
```

```
[ ] y_train=to_categorical(y_train)
```

```
[ ] y_test=to_categorical(y_test)
```

```
[ ] print(X_train.shape, X_test.shape, y_train.shape, y_test.shape)
    (1520, 75, 75, 3) (380, 75, 75, 3) (1520, 2) (380, 2)
```

Figure 20: Splitting the dataset in training and testing.

The model might overfit the training dataset, which is a risk throughout the training process. In other words, the model can develop a function that is highly specialized and effective on our training data but ineffective on photos it has never seen before. The train-test split is a method for assessing how well a machine learning system is working. It can be applied to solve problems requiring classification, regression, or any other supervised learning method. The dataset must first be divided into two categories. The initial subset for parameter estimation is the training dataset. Instead of using the second subset, the model is trained using the input component of the dataset, and after that, estimates are generated and compared to the predicted value. The second sample under consideration is the test dataset. When machine learning algorithms are

used to make predictions based on data that was not used to train the system, the train-test split technique is used to evaluate how well they work. To evaluate prediction performance objectively, we must divide the dataset.

In a neural network, the activation function is responsible for activating the node or output for each input by transforming the node's cumulative weighted input. ReLU, or rectified linear function, will output 0 if the input is negative and its full value if it is positive. Since a model using it is simpler to train and frequently outperforms others, it has become the norm for many different types of neural networks. In required to practice deep neural networks using gradient descent with training algorithm of errors, an activation function that looks and acts like a linear model but is actually a nonlinear function is needed. The function must also be more adaptive to the input of the activation sum and prevent simple exhaustion. The adoption of ReLU and associated technologies, such as those that currently enable the creation of very deep neural networks even on a regular basis, may be viewed as one of the few turning points in the deep learning revolution. When creating most types of neural networks, the rectified linear activation function has quickly taken over as the standard activation function.

Advantages of ReLU are –

- Computational Simplicity.
- Representational Sparsity.
- Linear Behavior.
- Train Deep Networks.

- **Applying VGG-16 model**

VGG16 is a convolutional neural network (CNN) architecture which was proposed by the Visual Geometry Group at the University of Oxford. It has 16 layers, including 13 convolutional layers and 3 fully connected layers, and was designed to improve the accuracy of image classification tasks. The VGG16 architecture is characterized by the use of very small 3x3 convolution filters,

which were found to be more effective at capturing image features than larger filters. The network also includes max-pooling layers that help to reduce the spatial dimensions of the input image.

```
from tensorflow.keras.applications import VGG16
```

```
base_model = VGG16(weights='imagenet', include_top=False, input_shape=(75,75,3))
base_model.trainable = False

x = base_model.output
x = GlobalAveragePooling2D()(x)
x = Dropout(0.2)(x)
x = Dense(4096,activation="relu")(x)
x = Dense(4096,activation="relu")(x)
x = Dropout(0.2)(x)
x = Dense(2096,activation="relu")(x)
predictions = Dense(2, activation='sigmoid')(x)
model = Model(inputs=base_model.input, outputs=predictions)

# confirm unfrozen layers
for layer in model.layers:
    if layer.trainable==True:
        print(layer)
```

Figure 21: Importing VGG-16 model.

- Finding Epoch Values of VGG-16

```
%time
history=model.fit(X_train, y_train,validation_data=(X_test, y_test),verbose = 1,epochs = 15,callbacks=callbacks)
```

```
CPU times: user 4 µs, sys: 1 µs, total: 5 µs
Wall time: 11.9 µs
Epoch 1/15
48/48 [==============================] - 172s 4s/step - loss: 4.7917 - accuracy: 0.8020 - val_loss: 0.4636 - val_accuracy: 0.8289
Epoch 2/15
48/48 [==============================] - 152s 3s/step - loss: 0.4196 - accuracy: 0.8546 - val_loss: 0.4178 - val_accuracy: 0.8342
Epoch 3/15
48/48 [==============================] - 154s 3s/step - loss: 0.3768 - accuracy: 0.8572 - val_loss: 0.4140 - val_accuracy: 0.8342
Epoch 4/15
48/48 [==============================] - 167s 4s/step - loss: 0.3490 - accuracy: 0.8645 - val_loss: 0.3768 - val_accuracy: 0.8474
Epoch 5/15
48/48 [==============================] - 174s 4s/step - loss: 0.3294 - accuracy: 0.8750 - val_loss: 0.3772 - val_accuracy: 0.8447
Epoch 6/15
48/48 [==============================] - 170s 4s/step - loss: 0.3046 - accuracy: 0.8868 - val_loss: 0.3486 - val_accuracy: 0.8526
Epoch 7/15
48/48 [==============================] - 170s 4s/step - loss: 0.2999 - accuracy: 0.8783 - val_loss: 0.3721 - val_accuracy: 0.8553
Epoch 8/15
48/48 [==============================] - 169s 4s/step - loss: 0.2698 - accuracy: 0.9000 - val_loss: 0.3549 - val_accuracy: 0.8605
Epoch 9/15
48/48 [==============================] - 177s 4s/step - loss: 0.2644 - accuracy: 0.9046 - val_loss: 0.3665 - val_accuracy: 0.8789
Epoch 10/15
48/48 [==============================] - 169s 4s/step - loss: 0.2560 - accuracy: 0.8987 - val_loss: 0.3171 - val_accuracy: 0.8763
Epoch 11/15
48/48 [==============================] - 172s 4s/step - loss: 0.2299 - accuracy: 0.9138 - val_loss: 0.3203 - val_accuracy: 0.8737
```

Figure 22: VGG-16 epoch results.

Overfitting is a major issue when neural networks are trained using sample data. A neural network model will learn very precise patterns in the sample data if more training epochs are employed than are necessary. As a result, the model won't fit the new data set properly. On the training data set (sample data), this model achieves great accuracy; however, on the test data set, it does not. In other words, by overfitting the training set, the model loses its capacity to generalize.

- Confusion matrix of VGG-16 model.

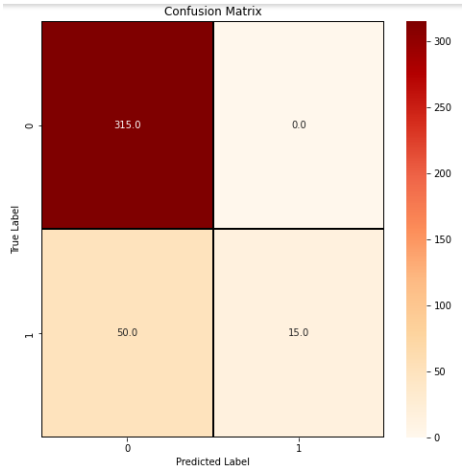



Figure 23: Confusion Matrix for VGG-16.

- Results obtained from VGG-16 model.

```
from sklearn.metrics import classification_report
print(classification_report(Y_true, Y_pred_classes))
```

| | precision | recall | f1-score | support |
|---|---|---|---|---|
| 0 | 0.86 | 1.00 | 0.93 | 315 |
| 1 | 1.00 | 0.23 | 0.38 | 65 |
| accuracy | | | 0.87 | 380 |
| macro avg | 0.93 | 0.62 | 0.65 | 380 |
| weighted avg | 0.89 | 0.87 | 0.83 | 380 |

Figure 24: Results obtained from VGG-16.

The Accuracy achieved by the VGG-16 model is 87%. When the result might potentially comprises of two or more classes, it is a technique to evaluate how

effectively a deep learning categorization system works. The table contains four possible combinations of predicted and actual values. This program can be very effectively used to evaluate AUC-ROC curves, recall, precision, specificity, accuracy, and—most crucially—all of the aforementioned.

- **Applying VGG-19 model.**

  VGG19 is a convolutional neural network (CNN) architecture that is an extension of the VGG16 architecture. It was proposed by the University of Oxford's Visual Geometry Group and was intended to increase the accuracy of picture classification tasks, much as VGG16. With 19 layers total, comprising 16 convolutional layers and 3 fully linked layers, VGG19 is distinguished by its depth. In order to shrink the spatial dimensions of the input image, it incorporates max-pooling layers in addition to the same tiny 3x3 convolution filters as VGG16. In order to fine-tune the network for a particular job, a smaller dataset is used after the network has been pre-trained on a larger dataset (such as ImageNet) using transfer learning. This allows the network to learn more quickly and achieve higher accuracy on new datasets. Compared to VGG16, VGG19 is deeper and therefore more complex, which can lead to better performance on some tasks, but may also require more computational resources and training time. Several computer vision tasks, including segmentation, object detection, and image recognition, have been performed using VGG19. It has also been applied as a feature extractor in situations involving transfer learning.

```
from tensorflow.keras.applications import VGG19
```

```
base_model = VGG19(weights='imagenet', include_top=False, input_shape=(75,75,3))
base_model.trainable = False

x = base_model.output
x = GlobalAveragePooling2D()(x)
x = Dropout(0.2)(x)
x = Dense(4096,activation="relu")(x)
x = Dense(4096,activation="relu")(x)
x = Dropout(0.2)(x)
x = Dense(2096,activation="relu")(x)
predictions = Dense(2, activation='sigmoid')(x)
model = Model(inputs=base_model.input, outputs=predictions)

# confirm unfrozen layers
for layer in model.layers:
    if layer.trainable==True:
        print(layer)
```

Figure 25: Importing VGG-19 model.

- Finding Epoch Values of VGG-19.

```
%time
history=model.fit(X_train, y_train,validation_data=(X_test, y_test),verbose = 1,epochs = 15,callbacks=callbacks)

CPU times: user 3 µs, sys: 0 ns, total: 3 µs
Wall time: 7.15 µs
Epoch 1/15
48/48 [==============================] - 15s 90ms/step - loss: 3.8580 - accuracy: 0.8132 - val_loss: 0.4345 - val_accuracy: 0.8289
Epoch 2/15
48/48 [==============================] - 3s 70ms/step - loss: 0.3854 - accuracy: 0.8586 - val_loss: 0.4011 - val_accuracy: 0.8395
Epoch 3/15
48/48 [==============================] - 2s 39ms/step - loss: 0.3596 - accuracy: 0.8612 - val_loss: 0.4611 - val_accuracy: 0.8342
Epoch 4/15
48/48 [==============================] - 2s 38ms/step - loss: 0.3389 - accuracy: 0.8645 - val_loss: 0.4130 - val_accuracy: 0.8316
Epoch 5/15
48/48 [==============================] - 3s 66ms/step - loss: 0.3168 - accuracy: 0.8737 - val_loss: 0.3369 - val_accuracy: 0.8658
Epoch 6/15
48/48 [==============================] - 3s 69ms/step - loss: 0.2843 - accuracy: 0.8947 - val_loss: 0.3347 - val_accuracy: 0.8632
Epoch 7/15
48/48 [==============================] - 2s 47ms/step - loss: 0.2892 - accuracy: 0.8803 - val_loss: 0.3818 - val_accuracy: 0.8553
Epoch 8/15
48/48 [==============================] - 4s 75ms/step - loss: 0.2779 - accuracy: 0.8855 - val_loss: 0.3338 - val_accuracy: 0.8711
Epoch 9/15
48/48 [==============================] - 3s 70ms/step - loss: 0.2718 - accuracy: 0.8941 - val_loss: 0.3296 - val_accuracy: 0.8868
Epoch 10/15
48/48 [==============================] - 2s 39ms/step - loss: 0.2458 - accuracy: 0.9026 - val_loss: 0.3643 - val_accuracy: 0.8632
Epoch 11/15
48/48 [==============================] - 2s 46ms/step - loss: 0.2425 - accuracy: 0.9020 - val_loss: 0.4986 - val_accuracy: 0.8474
Epoch 12/15
```

Figure 26: VGG-19 epoch results.

To reduce overfitting and improve the generalization ability of neural networks, models should be trained with an optimal number of epochs. Validate the model using a portion of the training data and check the performance of the model after each training epoch. The training and validation sets are monitored for loss and accuracy, and the number of epochs before the model starts to overfit is checked.

- Train-test accuracy and loss graph.

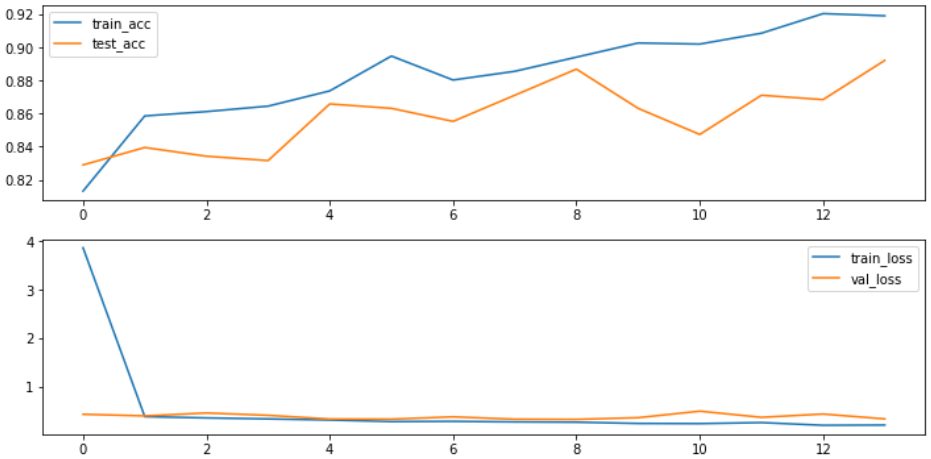



Figure 27: Accuracy and loss graphs for VGG-19.

- Model summary of VGG-19.

```
model.summary()
```

```
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_1 (InputLayer)        [(None, 75, 75, 3)]       0

 block1_conv1 (Conv2D)       (None, 75, 75, 64)        1792

 block1_conv2 (Conv2D)       (None, 75, 75, 64)        36928

 block1_pool (MaxPooling2D)  (None, 37, 37, 64)        0

 block2_conv1 (Conv2D)       (None, 37, 37, 128)       73856

 block2_conv2 (Conv2D)       (None, 37, 37, 128)       147584

 block2_pool (MaxPooling2D)  (None, 18, 18, 128)       0

 block3_conv1 (Conv2D)       (None, 18, 18, 256)       295168

 block3_conv2 (Conv2D)       (None, 18, 18, 256)       590080

 block3_conv3 (Conv2D)       (None, 18, 18, 256)       590080

 block3_conv4 (Conv2D)       (None, 18, 18, 256)       590080
```

Figure 28: VGG-19 model summary.

- Confusion matrix of VGG-19 model.

```
from sklearn.metrics import confusion_matrix
import seaborn as sns

Y_pred = model.predict(X_test)
Y_pred_classes = np.argmax(Y_pred,axis = 1)
Y_true = np.argmax(y_test,axis = 1)
confusion_mtx = confusion_matrix(Y_true, Y_pred_classes)
f,ax = plt.subplots(figsize=(8, 8))
sns.heatmap(confusion_mtx, annot=True, linewidths=0.01,cmap="OrRd",linecolor="black", fmt= '.1f',ax=ax)
plt.xlabel("Predicted Label")
plt.ylabel("True Label")
plt.title("Confusion Matrix")
plt.show()
```

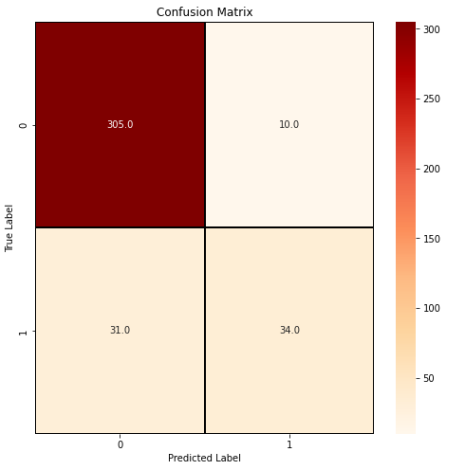



Figure 29: Confusion Matrix for VGG-19.

It might be difficult to compare two models with great recall but low accuracy. To compare them, we thus use F1-Score. The F1-score assists in concurrently assessing recall and accuracy. It uses the harmonic mean instead of the arithmetic mean by punishing the outliers more severely.

- Results obtained from VGG-19 model.

```
from sklearn.metrics import classification_report
print(classification_report(Y_true, Y_pred_classes))
```

| | precision | recall | f1-score | support |
|---|---|---|---|---|
| 0 | 0.91 | 0.97 | 0.94 | 315 |
| 1 | 0.77 | 0.52 | 0.62 | 65 |
| accuracy | | | 0.89 | 380 |
| macro avg | 0.84 | 0.75 | 0.78 | 380 |
| weighted avg | 0.88 | 0.89 | 0.88 | 380 |

Figure 30: Results obtained from VGG-19.

The Accuracy achieved by the VGG-19 model is 89%.

For convolutional neural networks (CNNs), the confusion matrix shows where the model gets confused; which classes are predicted correctly and which classes are predicted incorrectly.

The confusion matrix is a technique for describing the performance of a classification algorithm. Classification accuracy alone can be misleading when there are more than two classes in the data set or when there aren't an equal number of observations in each class. By computing the confusion matrix, we acquire a better understanding of the advantages and disadvantages of the classification strategy.

- **Applying ResNet-152 model.**

  One of the most complex neural networks ever created, it is a variation on the ResNet architecture with 152 layers. The use of residual connections, which enable the network to learn a residual mapping of the input rather than attempting to learn the complete mapping from scratch, is the main novelty of the ResNet architecture. This makes deep neural networks less susceptible to the vanishing gradient problem and enables ResNet to perform at the cutting edge across a variety of computer vision applications.

  ResNet-152 has more layers than its predecessors, ResNet-50 and ResNet-101, which enables it to capture more complex features and patterns in images. However, this also makes it more computationally expensive to train and deploy. ResNet-152 has been shown to achieve state-of-the-art performance on several benchmarks in the field of computer vision, including the ImageNet classification task, which involves classifying images into one of 1,000 different categories.

```
from tensorflow.keras.applications.resnet import ResNet152
```

```
base_model = ResNet152(weights='imagenet', include_top=False, input_shape=(75,75,3))
base_model.trainable = False

x = base_model.output
x = GlobalAveragePooling2D()(x)
x = Dropout(0.2)(x)
x = Dense(4096,activation="relu")(x)
x = Dense(4096,activation="relu")(x)
x = Dropout(0.2)(x)
x = Dense(2096,activation="relu")(x)
predictions = Dense(2, activation='sigmoid')(x)
model = Model(inputs=base_model.input, outputs=predictions)

# confirm unfrozen layers
for layer in model.layers:
    if layer.trainable==True:
        print(layer)
```

Figure 31: Importing ResNet 152 model.

- Finding Epoch Values of ResNet152.

```
%time
history=model.fit(X_train, y_train,validation_data=(X_test, y_test),verbose = 1,epochs = 15,callbacks=callbacks)
```

```
CPU times: user 3 µs, sys: 1e+03 ns, total: 4 µs
Wall time: 9.3 µs
Epoch 1/15
48/48 [==============================] - 188s 4s/step - loss: 1.6043 - accuracy: 0.8243 - val_loss: 0.4219 - val_accuracy: 0.8289
Epoch 2/15
48/48 [==============================] - 175s 4s/step - loss: 0.3970 - accuracy: 0.8579 - val_loss: 0.4113 - val_accuracy: 0.8421
Epoch 3/15
48/48 [==============================] - 157s 3s/step - loss: 0.3553 - accuracy: 0.8586 - val_loss: 0.4057 - val_accuracy: 0.8289
Epoch 4/15
48/48 [==============================] - 171s 4s/step - loss: 0.3131 - accuracy: 0.8638 - val_loss: 0.4505 - val_accuracy: 0.8316
Epoch 5/15
48/48 [==============================] - 154s 3s/step - loss: 0.2670 - accuracy: 0.8849 - val_loss: 0.3842 - val_accuracy: 0.8579
Epoch 6/15
48/48 [==============================] - 163s 3s/step - loss: 0.2580 - accuracy: 0.8928 - val_loss: 0.3840 - val_accuracy: 0.8658
Epoch 7/15
48/48 [==============================] - 172s 4s/step - loss: 0.2104 - accuracy: 0.9145 - val_loss: 0.3396 - val_accuracy: 0.8684
Epoch 8/15
48/48 [==============================] - 171s 4s/step - loss: 0.1872 - accuracy: 0.9204 - val_loss: 0.3399 - val_accuracy: 0.8763
Epoch 9/15
48/48 [==============================] - 176s 4s/step - loss: 0.1558 - accuracy: 0.9421 - val_loss: 0.3129 - val_accuracy: 0.9000
Epoch 10/15
48/48 [==============================] - 173s 4s/step - loss: 0.1548 - accuracy: 0.9349 - val_loss: 0.2634 - val_accuracy: 0.9105
Epoch 11/15
48/48 [==============================] - 171s 4s/step - loss: 0.1310 - accuracy: 0.9500 - val_loss: 0.3956 - val_accuracy: 0.8684
```

Figure 32: ResNet152 epoch results.

- Train-test accuracy and loss graph.

```
#plot the accuracy graph
plt.figure(figsize = (12,6))
plt.subplot(2,1,1)
plt.plot(history.history['accuracy'], label="train_acc")
plt.plot(history.history['val_accuracy'], label = "test_acc")
plt.legend()
plt.subplot(2,1,2)
plt.plot(history.history['loss'], label = "train_loss")
plt.plot(history.history['val_loss'], label = "val_loss")
plt.legend()
```

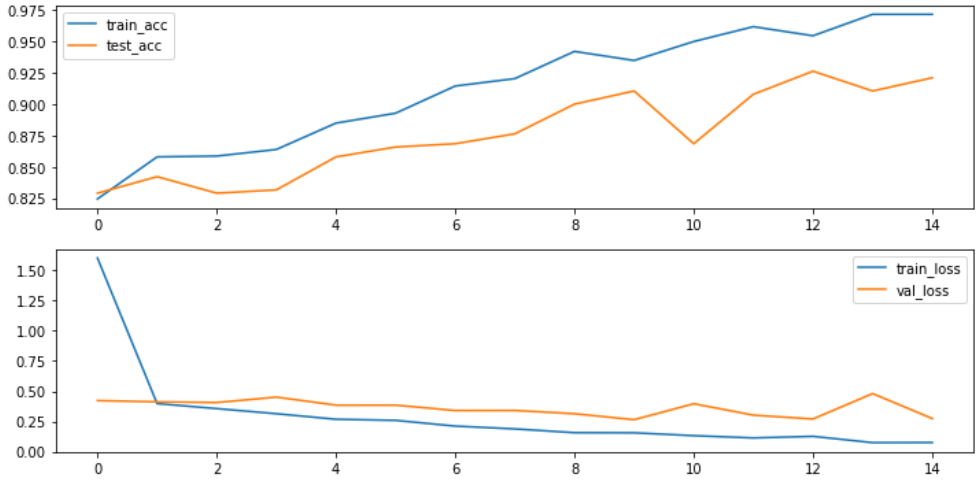



Figure 33: Accuracy and loss graphs for ResNet152.

- Confusion matrix of ResNet152 model.

```
from sklearn.metrics import confusion_matrix

import seaborn as sns
y_pred = model.predict(X_test)
y_pred_classes = np.argmax(y_pred,axis = 1)
y_true = np.argmax(y_test,axis = 1)
confusion_mtx = confusion_matrix(y_true, y_pred_classes)
f,ax = plt.subplots(figsize=(8, 8))
sns.heatmap(confusion_mtx, annot=True, linewidths=0.01,cmap="ORRd",linecolor="black", fmt= '.1f',ax=ax)
plt.xlabel("True Label")
plt.ylabel("Predicted Label")
plt.title("Confusion Matrix")
plt.show()
```

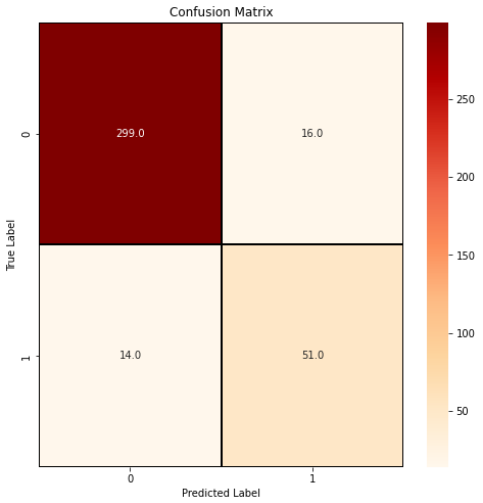



Figure 34: Confusion Matrix for ResNet152.

- Results obtained from ResNet152 model.

```
from sklearn.metrics import classification_report
print(classification_report(Y_true, Y_pred_classes))
```

| | precision | recall | f1-score | support |
|---|---|---|---|---|
| 0 | 0.96 | 0.95 | 0.95 | 315 |
| 1 | 0.76 | 0.78 | 0.77 | 65 |
| accuracy | | | 0.92 | 380 |
| macro avg | 0.86 | 0.87 | 0.86 | 380 |
| weighted avg | 0.92 | 0.92 | 0.92 | 380 |

Figure 35: Results obtained from VGG-19.

The Accuracy achieved by the ResNet152 model is 92%.

- **Applying Inception V3 model.**

  It is an evolution of the original Inception architecture, which was designed to improve the computational efficiency of deep neural networks. The key innovation of the Inception v3 architecture is the use of “factorized” convolutional layers, which split the standard 3x3 convolution into two smaller convolutions: a 1x3 convolution followed by a 3x1 convolution. This approach

reduces the number of parameters in the network, making it more computationally efficient while still maintaining high accuracy.

Inception v3 also includes other design features such as aggressive use of batch normalization, factorized reduction, and the incorporation of auxiliary classifiers at intermediate layers, which help to improve the gradient flow and regularize the network.

On a number of benchmarks in the field of computer vision, including the ImageNet classification task, which involves classifying images into one of 1,000 different categories, Inception v3 has been demonstrated to achieve state-of-the-art performance. It is widely utilised in numerous academic and commercial applications, and has also been employed for other tasks like object detection and image segmentation.

```
from tensorflow.keras.applications.inception_v3 import InceptionV3
```

```
base_model = InceptionV3(weights='imagenet', include_top=False, input_shape=(75,75,3))
base_model.trainable = False

x = base_model.output
x = GlobalAveragePooling2D()(x)
x = Dropout(0.2)(x)
x = Dense(4096,activation="relu")(x)
x = Dense(4096,activation="relu")(x)
x = Dropout(0.2)(x)
x = Dense(2096,activation="relu")(x)
predictions = Dense(2, activation='sigmoid')(x)
model = Model(inputs=base_model.input, outputs=predictions)

# confirm unfrozen layers
for layer in model.layers:
    if layer.trainable==True:
        print(layer)
```

Figure 36: Importing Inception V3 model.

- Finding Epoch Values of InceptionV3.

```
callbacks = [EarlyStopping(monitor='val_loss', patience=5, verbose=1),ModelCheckpoint('inception_v3_model.hdf5',save_best_only=True)]
```

```
opt = Adam(learning_rate=0.001)
model.compile(loss='categorical_crossentropy',optimizer=opt,metrics=['accuracy'])
```

```
%time
history=model.fit(X_train, y_train,validation_data=(X_test, y_test),verbose = 1,epochs = 15,callbacks=callbacks)
```

```
CPU times: user 4 µs, sys: 1 µs, total: 5 µs
Wall time: 9.78 µs
Epoch 1/15
48/48 [==============================] - 64s 1s/step - loss: 21.4082 - accuracy: 0.7829 - val_loss: 0.5149 - val_accuracy: 0.7921
Epoch 2/15
48/48 [==============================] - 56s 1s/step - loss: 0.4256 - accuracy: 0.8572 - val_loss: 0.4626 - val_accuracy: 0.8316
Epoch 3/15
48/48 [==============================] - 55s 1s/step - loss: 0.4013 - accuracy: 0.8559 - val_loss: 0.4114 - val_accuracy: 0.8316
Epoch 4/15
48/48 [==============================] - 58s 1s/step - loss: 0.3707 - accuracy: 0.8664 - val_loss: 0.4927 - val_accuracy: 0.8316
Epoch 5/15
48/48 [==============================] - 54s 1s/step - loss: 0.3570 - accuracy: 0.8684 - val_loss: 0.4022 - val_accuracy: 0.8395
Epoch 6/15
48/48 [==============================] - 57s 1s/step - loss: 0.3467 - accuracy: 0.8737 - val_loss: 0.4012 - val_accuracy: 0.8368
Epoch 7/15
48/48 [==============================] - 55s 1s/step - loss: 0.3002 - accuracy: 0.8862 - val_loss: 0.4100 - val_accuracy: 0.8316
Epoch 8/15
48/48 [==============================] - 64s 1s/step - loss: 0.2972 - accuracy: 0.8908 - val_loss: 0.3870 - val_accuracy: 0.8447
```

Figure 37: Inception V3 epoch results.

The Inception V3 model stopped after 14 epochs due to early stopping.

- Train-test accuracy and loss graph.

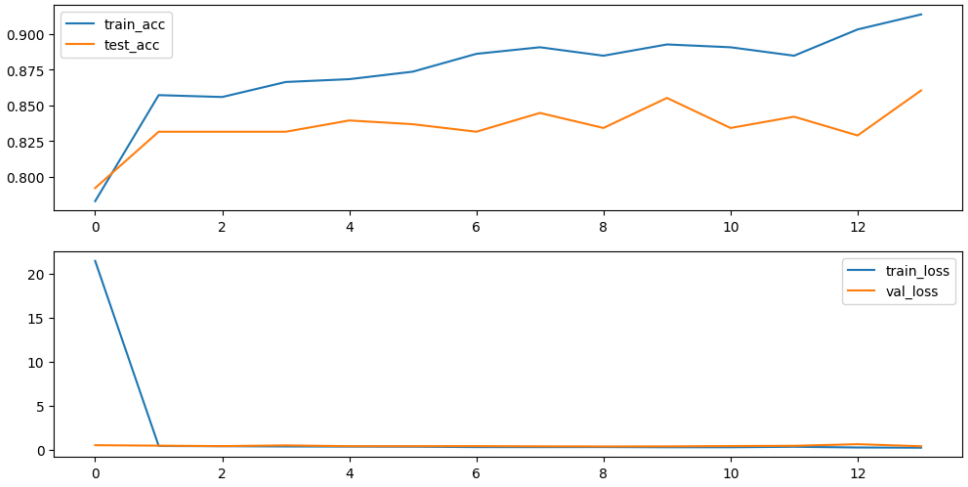



Figure 38: Accuracy and loss graphs for Inception V3.

- Confusion matrix of Inception V3 model.

```
from sklearn.metrics import confusion_matrix
import seaborn as sns

Y_pred = model.predict(X_test)
Y_pred_classes = np.argmax(Y_pred,axis = 1)
Y_true = np.argmax(y_test,axis = 1)
confusion_mtx = confusion_matrix(Y_true, Y_pred_classes)
f,ax = plt.subplots(figsize=(8, 8))
sns.heatmap(confusion_mtx, annot=True, linewidths=0.01,cmap="OrRd",linecolor="black", fmt= '.1f',ax=ax)
plt.xlabel("Predicted Label")
plt.ylabel("True Label")
plt.title("Confusion Matrix")
plt.show()
```

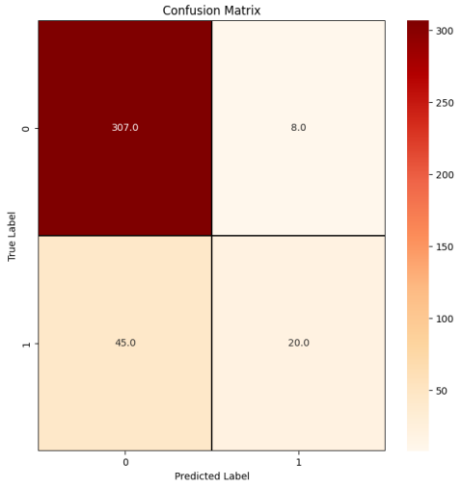



Figure 39: Confusion Matrix for Inception V3.

- Results obtained from Inception V3 model.

```
from sklearn.metrics import classification_report
print(classification_report(Y_true, Y_pred_classes))
```

| | precision | recall | f1-score | support |
|---|---|---|---|---|
| 0 | 0.87 | 0.97 | 0.92 | 315 |
| 1 | 0.71 | 0.31 | 0.43 | 65 |
| accuracy | | | 0.86 | 380 |
| macro avg | 0.79 | 0.64 | 0.68 | 380 |
| weighted avg | 0.85 | 0.86 | 0.84 | 380 |

Figure 40: Results obtained from Inception V3.

The Accuracy achieved by the Inception V3 model is 86%.

- **Applying DenseNet-201 model.**

  The usage of dense blocks, which connect each layer to every other layer in a feed-forward manner, is the main innovation of the DenseNet design. As a result, the network is better able to execute with fewer parameters while reusing features and gradients more effectively. Furthermore, the vanishing gradient issue that can arise in very deep neural networks is lessened by the dense connections.

  DenseNet also incorporates other design features such as batch normalization, global average pooling, and dropout regularization, which help to improve the gradient flow and prevent overfitting.

  On a number of benchmarks in the field of computer vision, including the ImageNet classification task, which involves classifying images into one of 1,000 different categories, DenseNet has been demonstrated to achieve state-of-the-art performance. It is widely employed in numerous academic and commercial applications, and has also been applied for other tasks like object detection and semantic segmentation.

```
from tensorflow.keras.applications import DenseNet201
```

```
base_model = DenseNet201(weights='imagenet', include_top=False, input_shape=(75,75,3))
base_model.trainable = False

x = base_model.output
x = GlobalAveragePooling2D()(x)
x = Dropout(0.2)(x)
x = Dense(4096,activation="relu")(x)
x = Dense(4096,activation="relu")(x)
x = Dropout(0.2)(x)
x = Dense(2096,activation="relu")(x)
predictions = Dense(2, activation='sigmoid')(x)
model = Model(inputs=base_model.input, outputs=predictions)

# confirm unfrozen layers
for layer in model.layers:
    if layer.trainable==True:
        print(layer)
```

Figure 41: Importing DenseNet 201 model.

- Finding Epoch Values of DenseNet 201.

```
%time
history=model.fit(X_train, y_train,validation_data=(X_test, y_test),verbose = 1,epochs = 15,callbacks=callbacks)
```

```
CPU times: user 4 µs, sys: 0 ns, total: 4 µs
Wall time: 10.5 µs
Epoch 1/15
48/48 [==============================] - 117s 2s/step - loss: 8.2674 - accuracy: 0.8020 - val_loss: 0.4766 - val_accuracy: 0.8289
Epoch 2/15
48/48 [==============================] - 99s 2s/step - loss: 0.4207 - accuracy: 0.8507 - val_loss: 0.4393 - val_accuracy: 0.8289
Epoch 3/15
48/48 [==============================] - 89s 2s/step - loss: 0.4066 - accuracy: 0.8507 - val_loss: 0.4238 - val_accuracy: 0.8316
Epoch 4/15
48/48 [==============================] - 82s 2s/step - loss: 0.3872 - accuracy: 0.8513 - val_loss: 0.4200 - val_accuracy: 0.8316
Epoch 5/15
48/48 [==============================] - 93s 2s/step - loss: 0.3963 - accuracy: 0.8572 - val_loss: 0.4040 - val_accuracy: 0.8342
Epoch 6/15
48/48 [==============================] - 82s 2s/step - loss: 0.3817 - accuracy: 0.8579 - val_loss: 0.5160 - val_accuracy: 0.8395
Epoch 7/15
48/48 [==============================] - 97s 2s/step - loss: 0.3754 - accuracy: 0.8632 - val_loss: 0.4039 - val_accuracy: 0.8474
Epoch 8/15
48/48 [==============================] - 100s 2s/step - loss: 0.3597 - accuracy: 0.8638 - val_loss: 0.4707 - val_accuracy: 0.8342
Epoch 9/15
48/48 [==============================] - 89s 2s/step - loss: 0.3840 - accuracy: 0.8658 - val_loss: 0.3860 - val_accuracy: 0.8395
Epoch 10/15
48/48 [==============================] - 93s 2s/step - loss: 0.3753 - accuracy: 0.8539 - val_loss: 0.3938 - val_accuracy: 0.8500
Epoch 11/15
48/48 [==============================] - 86s 2s/step - loss: 0.3558 - accuracy: 0.8671 - val_loss: 0.4571 - val_accuracy: 0.8395
```

Figure 42: DenseNet 201 epoch results.

- Train-test accuracy and loss graph.

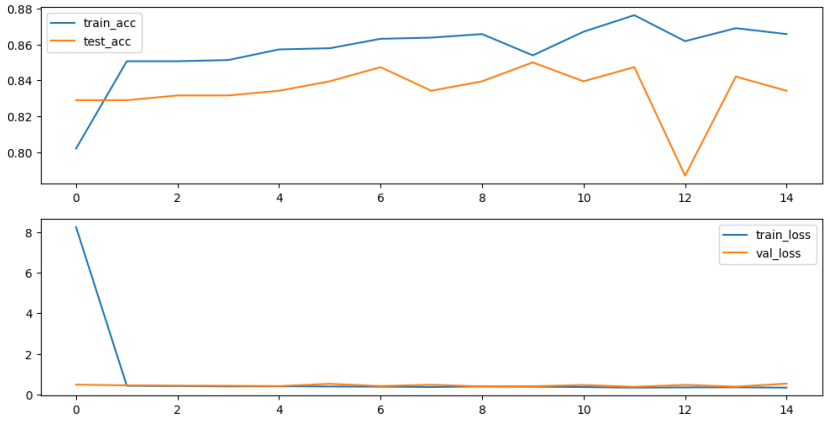



Figure 43: Accuracy and loss graphs for DenseNet 201.

- Confusion matrix of DenseNet 201 model.

```
from sklearn.metrics import confusion_matrix
import seaborn as sns

Y_pred = model.predict(X_test)
Y_pred_classes = np.argmax(Y_pred,axis = 1)
Y_true = np.argmax(y_test,axis = 1)
confusion_mtx = confusion_matrix(Y_true, Y_pred_classes)
f,ax = plt.subplots(figsize=(8, 8))
sns.heatmap(confusion_mtx, annot=True, linewidths=0.01,cmap="OrRd",linecolor="black", fmt= '.1f',ax=ax)
plt.xlabel("Predicted Label")
plt.ylabel("True Label")
plt.title("Confusion Matrix")
plt.show()
```

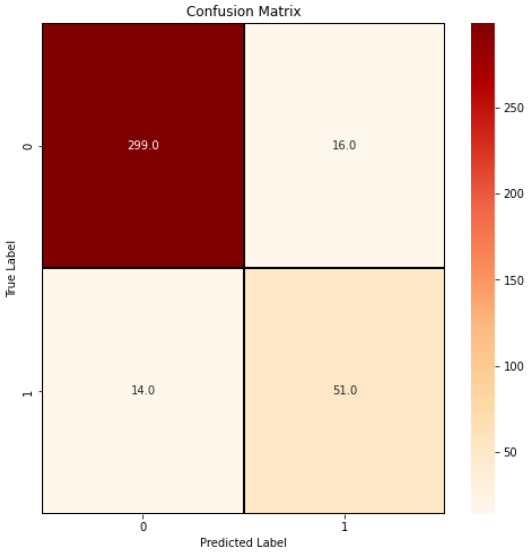



Figure 44: Confusion Matrix for DenseNet 201.

- Results obtained from Inception V3 model.

```
from sklearn.metrics import classification_report
print(classification_report(Y_true, Y_pred_classes))
```

| | precision | recall | f1-score | support |
|---|---|---|---|---|
| 0 | 0.96 | 0.95 | 0.95 | 315 |
| 1 | 0.76 | 0.78 | 0.77 | 65 |
| accuracy | | | 0.92 | 380 |
| macro avg | 0.86 | 0.87 | 0.86 | 380 |
| weighted avg | 0.92 | 0.92 | 0.92 | 380 |

Figure 45: Results obtained from DenseNet 201.

The Accuracy achieved by the DenseNet 201 model is 92%.

**Results**:

The proposed CNN-based brain stroke detection is evaluated in two phases. Loss values for the training and validation phases are calculated. Accuracy is calculated using these ratios of actual and predicted signs. Actual predicted values are computed from the base truth and predicted values. Several statistical measures are used to base evaluations made during the testing phase. False positives (Fp) are strokes that were mistakenly recognised as positives in the ground truth data while true positives (Tp) are strokes that were successfully identified as strokes. True negatives (Tn) are ground truth negative named strokes that are recognised as being negatives, whereas false negatives (Fn) are recognised as being false, where the equations are used to get the sensitivity (St), specificity (Sf), true positive rate (TPR), false positive rate (FPR), error rate (ERR), and accuracy (A). (1-8).

STASTICAL ANALYSIS –

$$S_t = T_p / (T_p + F_n) \quad \text{- (1)}$$

$$S_f = T_n / (T_n + F_p) \quad \text{- (2)}$$

$$TPR = T_p / (T_p + F_n) \quad \text{- (3)}$$

$$FPR = F_p / (F_p + T_n) \quad \text{-(4)}$$

$$A = (T_p + T_n) / (T_p + T_n + F_p + F_n) \quad \text{-(5)}$$

$$Precision(P) = T_p / (T_p + F_n) \quad \text{-(6)}$$

$$Recall(R) = T_p / (T_p + F_p) \quad \text{-(7)}$$

$$ERR = (F_p + F_n) / (T_p + T_n + F_p + F_n) \quad \text{-(8)}$$

| Models | Accuracy |
|---|---|
| VGG-16 | 87% |
| VGG-19 | 89% |
| ResNet152 | 92% |
| Inception V3 | 86% |
| DenseNet201 | 92% |

Table 1. Results obtained from different models.

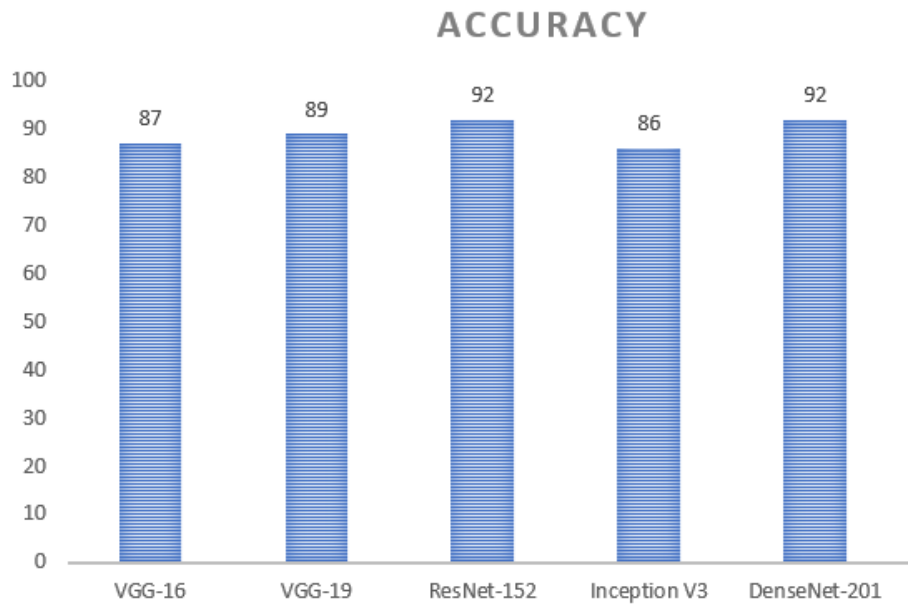



Figure 46: Accuracies of different models.

Figure 46 represents a bar plot according to the resultant accuracies shown in Table 1. Highest accuracy is 92% achieved by ResNet-152 and DenseNet-201.

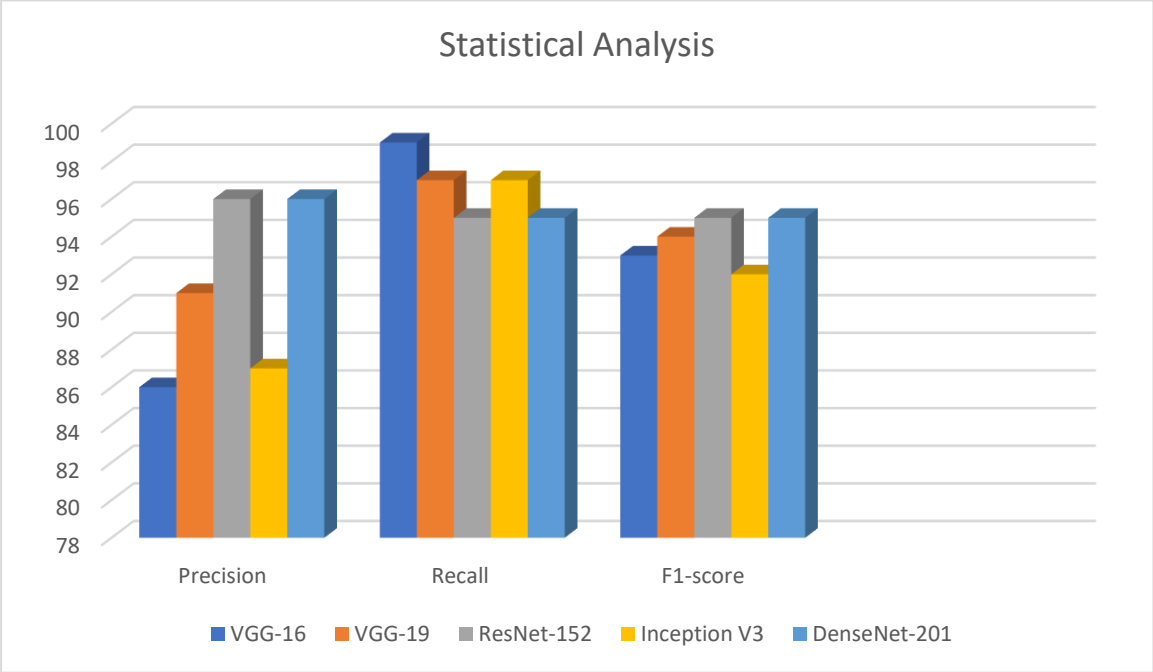



Figure 47: Graph of Precision, Recall and F1-score for different models.

Figure 47 represents the Precision, Recall and F1-score obtained by using different models and their results.

# Chapter-5
# CONCLUSIONS

## 5.1 Conclusions

According to the study, convolutional neural networks are fantastic tools for computer vision and related fields since they can identify features in unprocessed data. They are capable of identifying the relationships between various pixels in the training set and utilizing this knowledge to create their own features, ranging from low-level (edges, circles) to high-level (faces, hands, eyes). The issue is that certain features may be difficult for humans to understand. A crazy pixel in a photograph might occasionally produce unexpected new outcomes, too. Deep learning applications are becoming more and more common in healthcare. Using clinical laboratory data to build predictive models that can predict stroke risk can save many lives. This study was proposed to minimize data loss and increase the accuracy of the stroke detection process performing alternating epochs. In this study, we used different deep learning algorithms to build a predictive model out of which ResNet-152 and DenseNet-201 achieved an accuracy rate of 92%. This model can be integrated with electronic medical records to provide real-time stroke prediction based on laboratory tests.

## 5.2 Future Scope

In future we are expecting to compare and hybridise on other deep learning algorithms to obtain the most optimum result with the same dataset. Optimum result defines the results where the accuracy and precision of the model is significant than the obtained results till now. To get the optimum accuracy and precision of the model, we will train the dataset on genetic algorithms or hybridise algorithms. Future research aims to use data that provide information about different types of stroke to create predictive models for each type of stroke.

## 5.3 Applications

The application of this project is very beneficial to medical professionals because it can be very helpful in identifying a patient's risk for brain disease at a very early stage. It can also be very beneficial to patients because it can spare them from having to undergo expensive tests and procedures. Although this model will require a proper production and development, also to say an extensive level of testing before using it in real life, improvement can be done to make that possible. By identifying potential risk factors and enhancing our understanding of the condition, deep learning can be used to forecast brain strokes and enhance stroke research. Researchers can use this data to create fresh treatments and interventions for stroke sufferers.

# REFERENCES

[1] C.-L. Chin, B.-J. Lin, G.-R. Wu, T.-C. Weng, C.-S. Yang, R.-C. Su and Y.-J. Pan, "An automated early ischemic stroke detection system using CNN deep learning algorithm," 2017 IEEE 8th International Conference on Awareness Science and Technology (iCAST), Nov. 2017, doi: https://doi.org/10.1109/icawst.2017.8256481.

[2] C.-M. Lo, P.-H. Hung, and D.-T. Lin, "Rapid Assessment of Acute Ischemic Stroke by Computed Tomography Using Deep Convolutional Neural Networks," Journal of Digital Imaging, May 2021, doi: https://doi.org/10.1007/s10278-021-00457-y.

[3] Y. Xie, H. Yang, X. Yuan, Q. He, R. Zhang, Q. Zhu, Z. Chu, C. Yang, P. Qin and C. Yan, "Stroke prediction from electrocardiograms by deep neural network," Multimedia Tools and Applications, vol. 80, no. 11, pp. 17291–17297, Oct. 2020, doi: https://doi.org/10.1007/s11042-020-10043-z.

[4] I. Oksuz, "Brain MRI artefact detection and correction using convolutional neural networks," Computer Methods and Programs in Biomedicine, vol. 199, p. 105909, Feb. 2021, doi: https://doi.org/10.1016/j.cmpb.2020.105909.

[5] R. Zhang, L. Zhao, W. Lou, J. M. Abrigo, V. C. T. Mok, W. C. W. Chu, D. Wang and L. Shi, "Automatic Segmentation of Acute Ischemic Stroke From DWI Using 3-D Fully Convolutional DenseNets," IEEE Transactions on Medical Imaging, vol. 37, no. 9, pp. 2149–2160, Sep. 2018, doi: https://doi.org/10.1109/tmi.2018.2821244.

[6] B. R. Gaidhani, R. R.Rajamenakshi, and S. Sonavane, "Brain Stroke Detection Using Convolutional Neural Network and Deep Learning Models," 2019 2nd International Conference on Intelligent Communication and Computational Techniques (ICCT), Sep. 2019, doi: https://doi.org/10.1109/icct46177.2019.8969052.

[7] Y.-A. Choi, S.-J. Park, J.-A. Jun, C.-S. Pyo, K.-H. Cho, H.-S. Lee and J.-H. Yu, "Deep Learning-Based Stroke Disease Prediction System Using Real-Time Bio Signals," *Sensors*, vol. 21, no. 13, p. 4269, Jun. 2021, doi: https://doi.org/10.3390/s21134269.

[8] B. N. Rao, S. Mohanty, K. Sen, U. R. Acharya, K. H. Cheong, and S. Sabut, "Deep Transfer Learning for Automatic Prediction of Hemorrhagic Stroke on CT Images," *Computational and Mathematical Methods in Medicine*, vol. 2022, pp. 1–10, Apr. 2022, doi: https://doi.org/10.1155/2022/3560507.

[9] Md. Ashrafuzzaman, S. Saha, and K. Nur, “Prediction of Stroke Disease Using Deep CNN Based Approach,” *Journal of Advances in Information Technology*, vol. 13, no. 6, 2022, doi: https://doi.org/10.12720/jait.13.6.604-613.

[10] A. Gautam and B. Raman, “Towards effective classification of brain hemorrhagic and ischemic stroke using CNN,” *Biomedical Signal Processing and Control*, vol. 63, p. 102178, Jan. 2021, doi: https://doi.org/10.1016/j.bspc.2020.102178.

[11] M. Kaur, S. R. Sakhare, K. Wanjale, and F. Akter, “Early Stroke Prediction Methods for Prevention of Strokes,” *Behavioural Neurology*, vol. 2022, p. e7725597, Apr. 2022, doi: https://doi.org/10.1155/2022/7725597.

[12] Y. Liu, B. Yin, and Y. Cong, “The Probability of Ischaemic Stroke Prediction with a Multi-Neural-Network Model,” *Sensors*, vol. 20, no. 17, p. 4995, Sep. 2020, doi: https://doi.org/10.3390/s20174995.

[13] Y.-L. Lai, Y.-D. Wu, H.-J. Yeh, Y.-T. Wu, H.-Y. Tsai, and J.-C. Chen, “Using convolutional neural network to analyze brain MRI images for predicting functional outcomes of stroke,” Medical & Biological Engineering & Computing, vol. 60, no. 10, pp. 2841–2849, Aug. 2022, doi: https://doi.org/10.1007/s11517-022-02636-7.

[14] S. Zhang, S. Xu, L. Tan, H. Wang, and J. Meng, “Stroke Lesion Detection and Analysis in MRI Images Based on Deep Learning,” Journal of Healthcare Engineering, vol. 2021, pp. 1–9, Apr. 2021, doi: https://doi.org/10.1155/2021/5524769.

[15] K. Simonyan and A. Zisserman. “Very deep convolutional networks for large-scale image recognition.” arXiv preprint arXiv:1409.1556 (2014). doi: https://doi.org/10.48550/arXiv.1409.1556

[16] L. Wen, X. Li, X. Li, and L. Gao, “A New Transfer Learning Based on VGG-19 Network for Fault Diagnosis,” IEEE Xplore, May 01, 2019. https://ieeexplore.ieee.org/document/8791884 (accessed Sep. 08, 2022).

[17] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, “Rethinking the Inception Architecture for Computer Vision,” 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Jun. 2016, doi: https://doi.org/10.1109/cvpr.2016.308.

[18] M. Bansal, M. Kumar, M. Sachdeva, and A. Mittal, “Transfer learning for image classification using VGG19: Caltech-101 image data set,” Journal of Ambient Intelligence and Humanized Computing, Sep. 2021, doi: https://doi.org/10.1007/s12652-021-03488-z.

[19] R. Feng, M. Badgeley, J. Mocco, and E. K. Oermann, “Deep learning guided stroke management: a review of clinical applications,” Journal of NeuroInterventional Surgery, vol. 10, no. 4, pp. 358–362, Sep. 2017, doi: https://doi.org/10.1136/neurintsurg-2017-013355.

[20] S. Zhang, M. Zhang, S. Ma, Q. Wang, Y. Qu, Z. Sun and T. Yang “Research Progress of Deep Learning in the Diagnosis and Prevention of Stroke,” BioMed Research International, vol. 2021, pp. 1–5, Aug. 2021, doi: https://doi.org/10.1155/2021/5213550.