A
Major Project Report

On

"RETINAL DISEASES DETECTION USING DEEP CNN"

Submitted in partial fulfillment of the

Requirements for the award of the degree of

Bachelor of Technology

In

Computer Science & Engineering - Data Science

By

B Likhitha Reddy - 21R25A6702

Under the guidance of

Mr.A.Kiran Kumar Reddy
Assistant Professor

Department of Data Science & Cyber Security

MARRI LAXMAN REDDY GROUP OF INSTITUTIONS
MLR INSTITUTE OF TECHNOLOGY
(UGC AUTONOMOUS)
Affiliated to JNTUH,Approved by AICTE
Laxman Reddy Avenue, Dundigal, Hyderabad-500 043, Telangana, India

NATIONAL BOARD OF ACCREDITATION

NAAC A

2024

MARRI LAXMAN REDDY GROUP OF INSTITUTIONS MLR INSTITUTE OF TECHNOLOGY
(UGC AUTONOMOUS)
Affiliated to JNTUH, Approved by AICTE
Laxman Reddy Avenue, Dundigal, Hyderabad-500 043, Telangana, India

Department of Data Science & Cyber Security

CERTIFICATE

This is to certify that the project entitled **"Retinal Diseases Detection Using Deep CNN"** has been submitted by **B Likhitha Reddy (21R25A6702)** in partial fulfillment of the requirements for the award of degree of Bachelor of Technology in Computer Science and Engineering-Data Science from Jawaharlal Nehru Technological University, Hyderabad. The results embodied in this project have not been submitted to any other University or Institution for the award of any degree or diploma.

Internal Guide **Head of the Department**

External Examiner

MARRI LAXMAN REDDY GROUP OF INSTITUTIONS MLR INSTITUTE OF TECHNOLOGY
(UGC AUTONOMOUS)
Affiliated to JNTUH, Approved by AICTE
Laxman Reddy Avenue, Dundigal, Hyderabad-500 043, Telangana, India

Department of Data Science & Cyber Security

DECLARATION

We hereby declare that the project entitled **"Retinal Diseases Detection Using Deep CNN"** is the work done during the period from **July 2023 to April 2024** and is submitted in partial fulfillment of the requirements for the award of degree of Bachelor of Technology in Computer Science and Engineering-Data Science from Jawaharlal Nehru Technology University, Hyderabad. The results embodied in this project have not been submitted to any other university or Institution for the award of any degree or diploma.

B Likhitha Reddy - 21R25A6702

MARRI LAXMAN REDDY GROUP OF INSTITUTIONS MLR INSTITUTE OF TECHNOLOGY (UGC AUTONOMOUS) Affiliated to JNTUH,Approved by AICTE Laxman Reddy Avenue, Dundigal, Hyderabad-500 043, Telangana, India NBA NATIONAL BOARD of ACCREDITATION NAAC A

Department of Data Science & Cyber Security

ACKNOWLEDGEMENT

The satisfaction and euphoria that accompany the successful completion of any task would be incomplete without the mention of people who made it possible, whose constant guidance and encouragement crowned our efforts with success. It is a pleasant aspect that we now have the opportunity to express our gratitude for all of them.

First of all, we would like to express our deep gratitude towards our internal guide **Mr.A.KIRAN KUMAR REDDY, Assistant Professor, Department of CSE-CS** for his support in the completion of our dissertation. We wish to express our sincere thanks to **Dr.CHIRANJEEVI MANIKE,** HOD, Dept. of DATA SCIENCE & CYBER SECURITY and also principal **Dr. K. SRINIVAS RAO** for providing the facilities to complete the dissertation.

We would like to thank all our faculty and friends for their help and constructive criticism during the project period. Finally, we are very much indebted to our parents for their moral support and encouragement to achieve goals.

B Likhitha Reddy - 21R25A6702

Department of Computer Science & Engineering

ABSTRACT

Artificial Neural Networks (ANN), Deep learning, Recurrent Neural Networks (RNN), Alex Net, and ResNet can all be seen as lines of study that can help find and classify serious illnesses. CNN and a particular version of it, called U-Net Segmentation, have changed the way diseases are categorized in medicine, especially diseases of the eye. Because feature extraction is so complicated, U-Net makes a big mistake when sending the whole feature map to the right processor. This causes a lot of memory and CPU usage. Putting it together with the unsampled decoder feature map also stops pooling indices from being used again. In this paper, we suggest a convolutional neural network (CNN) model for problems with multiple classes that makes good use of memory. Eye Net has a normal baseline dataset with 32 types of eye diseases that was used to test the proposed model. After running tests, it was found that the suggested model works better when it comes to managing memory and being accurate. The total comparison was done using precision, memory, and accuracy, and each step took a different amount of time and epochs. On the Eye-net dataset, the proposed method worked very well.

LIST OF FIGURES& TABLES

INDEX

CHAPTER 1
INTRODUCTION

1.1 OVERVIEW

People of all ages are getting retinal diseases in large numbers. In the human eye, the retina has a layer of optic nerve tissue that is called photosensitive. This layer turns the light that the lens focuses into signals that the brain can understand. The macula, which is in the middle of the retina, does the sensing. The macula gathers information, which is then processed by the retina and sent to the brain through the optic nerve so that it can be used for vision. People with age-related macular degeneration (AMD), optic disc drusen, Rothspot diabetic macular edema (DME), and other diseases can have trouble seeing. The hardest part is finding eye diseases because they are so different that they can only be accurately diagnosed by an optometrist with a lot of experience. In the same way, eye diseases are easy to spot and treat early on with computer-aided diagnosis tools (CAD). Recently, a lot of cutting-edge machine learning (ML) and deep learning (DL) models have been put forward to help classify, segment, and identify eye illnesses. We see that identifying and collecting data are big problems when using ADDs. This is because many machine learning (ML) and deep learning (DL) models have been created, such as Recurrent Neural Network (RNN), Convolution Neural Network (CNN), Alex Net ResNet, and VGN. These have made it easier for researchers and doctors to find and classify these serious illnesses.

1.2 PURPOSE OF THE PROJECT

The goal of this project is to use advanced neural network designs to solve the problems that come up with classifying serious illnesses, especially diseases of the eye. Techniques like U-Net Segmentation have shown promise, but they make a lot of mistakes when they try to extract features and send feature maps to the processor, which uses a lot of memory and CPU. In answer, this project suggests a new convolutional neural network (CNN) model that works best for jobs that need to classify things into more than one category. It is also designed to use memory efficiently.

The goal of the project is to test how well the proposed model works on Eye Net, a common dataset that includes 32 types of eye diseases. The project's goal is to show that the proposed model is better than current methods at managing memory and making accurate classifications through experiments. When comparing, key measures like recall, precision, and total accuracy are used, taking into account changes in the number of epochs and the amount of time needed for each step in the classification process. In the end, the goal is to use deep learning to make the process of classifying eye illnesses faster and more accurate.

1.3 MOTIVATION

Medical imaging is very important for finding diseases early and planning treatments. This is especially true for eye diseases, where a quick evaluation can have a big effect on how well the patient does. Traditional ways of finding and classifying diseases often depend on reading the results by hand, which can take a long time and lead to mistakes. Since the development of artificial intelligence (AI) and deep learning methods, there has been a change toward automatic diagnosis systems that use neural networks to quickly look at medical pictures.

In recent years, convolutional neural networks (CNNs) have become an important part of medical picture analysis, especially when it comes to classifying eye diseases. Existing CNN designs, like U-Net Segmentation, are useful, but they have problems with memory usage and extra work that needs to be done on the computer, especially when complex feature extraction tasks are involved.

We are working on this because we need to fix these problems and make a better CNN model for classifying eye diseases into the different groups. We want to improve the accuracy and flexibility of disease identification systems by making better use of memory and computing power. Our suggested CNN model is better in a number of ways than current methods. By carefully planning how to best use memory, we not only lower the amount of memory needed, but we also boost performance in areas like precision, remember, and accuracy. Also, our model works well with a variety of epoch numbers, so the performance stays the same over time.

We tested our suggested method on Eye-net, a common dataset that includes 32 types of eye diseases. This shows that it works in real life. By carefully testing our method against other methods, we show that it works better than them, which supports the idea that our approach could completely change how eye diseases are classified. In the end, our work offers a strong answer to the most important problems in medical picture analysis, making the process of classifying diseases faster and more accurate. We make better clinical decision support systems possible by using CNNs to their full potential and making the best use of memory. These systems will help doctors make faster and more accurate diagnoses, which will eventually improve patient results and the level of care.

CHAPTER 2
LITERATURE SURVEY

A thorough review of the literature has been done by looking at current methods for verifying and making certificates. Before this poll was made, a lot of research papers, magazines, and other media were also looked at.

2.1EXISTING SYSTEM

Medical picture labeling is an important part of evaluation and treatment that must be done correctly. Many medical imaging jobs, like classifying skin cancer, diabetic retinopathy, age-related macular degeneration (AMD), and COVID-19 detection from chest X-rays, have been done amazingly well with deep learning methods. This study looks at the latest progress in using deep learning to classify medical images. It does this by looking at the methods, datasets, and performance measures that have been described in the literature.

This is because medical picture analysis is a key part of helping doctors diagnose diseases quickly and correctly. Deep learning, especially convolutional neural networks (CNNs), has become a strong tool for medical picture analysis that can be used on a large scale and with high accuracy.

Deep Learning Models for Skin Cancer Classification: Esteva et al. [1] suggested using deep neural networks to classify skin cancer at the level of a physician. They had great success telling the difference between different types of skin tumors.

Classification of Diabetic Retinopathy: Shankar et al. [2] created an automated system using a synergic deep learning model to find and categorize diabetic retinopathy. This shows that it could be useful for early detection and planning treatment.

Age-related Macular Degeneration (AMD) Classification: Farsiu et al. [5] created a mathematical way for using optical coherence tomography to tell the difference between eyes with and without intermediate AMD. This method gives us important information about how the disease progresses.

Finding COVID-19 in Chest X-rays: Jabra et al. [11] and Guefrechi et al. [12] suggested deep learning-based methods for finding COVID-19 in chest X-rays, showing that these methods could help healthcare systems during the pandemic.

Using Deep Learning to Protect Privacy for COVID-19 Detection: Boulila et al. [13] suggested a way to use deep learning to identify COVID-19 in lung X-ray pictures while protecting patient data privacy.

Classification of eye illnesses Using Optical Coherence Tomography: Perdomo et al. [14] created a deep learning method for sorting photos of optical coherence tomography to identify eye illnesses connected to diabetes. This shows the possibility for early diagnosis and treatment.

Analysis of Retinal Diseases Using Machine Learning Algorithms: Mahendran et al. [15] looked at retinal diseases using different machine learning algorithms. This was a comparison of different ways to classify illnesses.

Conclusion: Deep learning methods have changed the way medical images are classified, making it easier and more accurate to diagnose a wide range of illnesses. By finding problems early and tailoring treatments to each person, more study in this area could lead to even better health results. Haidar et al. [14] look into how blockchain technology could be used to run tests, keep records, and make certificates all in one piece of software. If this were to happen, it could completely change how exams are run now. But this idea only looked at the possible uses; it didn't say what technology would be needed to make it happen.

They describe in the papers an automatic recognition and classification model for fundus DR pictures that is based on deep learning. Their method includes a number of steps, such as preparation, segmentation, and classification. The first step in their process is preparation, which gets rid of any noise that isn't needed in the edges. The next step is histogram-based segmentation, which pulls out the useful parts of the picture. Then, the Synergic Deep Learning (SDL) model was used to sort the DR fundus images into different levels of intensity. The given SDL model was tested and proven to work on the Messidor DR dataset.

2.2DISADVANTAGES OF EXISTING SYSTEM:

In short, here are the problems with the above implementations:

- The current work uses histogram-based segmentation to take useful parts out of fundus DR pictures. This method might not be as good at picking out fine lines and features, especially when working with complicated pictures.
- The current work uses the Synergic Deep Learning (SDL) model to group DR fundus images into different categories. More modern neural network designs may be better at scaling and adapting than the SDL model, even though it may have got good results.

- The current work doesn't say exactly how much memory their method uses or how well it works. Using too much memory and too little computer power could be a problem, depending on how complicated the SDL model is and how the data is prepared.

CHAPTER 3

PROPOSED SYSTEM

3.1 PROPOSED SYSTEM

We suggest a convolutional neural network (CNN) model for problems with multiple classes that makes good use of memory. Eye Net has a normal baseline dataset with 32 types of eye diseases that was used to test the proposed model. The CNN model based on deep learning has been used to improve the standard way of diagnosing eye illnesses that are very important. There are similarities between the suggested CNN model and current state-of-the-art methods. The total comparison was done using precision, memory, and accuracy, and each step took a different amount of time and epochs.

3.2 ADVANTAGES OF PROPOSED SYSTEM

The following are good things about the suggested system:

- Based on the details given, our work doesn't say exactly what segmentation method was used. It might be able to find important traits more accurately, though, if it uses a more complicated segmentation method.
- Using a convolutional neural network (CNN) model in our work is common and flexible when it comes to picture recognition tasks. CNNs have been used for a long time and have been successful in many areas, which makes them a good choice for classifying eye diseases.
- The CNN model is made to make good use of limited RAM. This means that our work might provide a better and more resource-efficient answer, which is very important for deploying and expanding in the real world.

3.3 SYSTEM REQUIREMENTS

This part lists the system requirements that must be met in order for the project to be developed and made available as an app. Do not mix up these needs with the hardware requirements for end users. The application is cross-platform, which means it should work on platforms of all shapes and sizes. There are no special needs for end users.

3.3.1 SOFTWARE REQUIREMENTS

Software requirements list the computer resources and system requirements needed for a program to run well. These needs are seldom supplied with program downloads. Install these individually before loading the program.

Platform - Software runs on computer platforms. Hardware or software may form the framework. Platforms often involve computer design, operating systems, programming languages, and runtime tools.

When you talk about system needs (software), one of the first things that comes up is the operating system. Different models of the same line of operating systems may not be able to work with the same software, but past compatibility is usually kept. Most Windows XP applications won't operate on Windows 98, but occasionally it may. Similar to how software created with the latest Linux Kernel v2.6 features doesn't operate or compile on v2.2 or v2.4.

APIs and drivers— Specialized software that uses multiple hardware devices, such as advanced display ports, needs custom drivers or APIs. DirectX, a collection of Microsoft APIs primarily used for multimedia and game development, is an example of this.

Web browser—The basic browser that comes with a computer is used by most web apps and software that relies on the Internet. Users often choose Microsoft Internet Explorer to run on Microsoft Windows. This software uses ActiveX features, which are known to be vulnerable.

1) Software: Anaconda

2) Primary Language: Python

3) Frontend Framework: Flask

4) Back-end Framework: Jupyter Notebook

5) Database: Sqlite3

6) Front-End Technologies: HTML, CSS, JavaScript and Bootstrap4

3.3.2HARDWARE REQUIREMENTS

Every OS and software need hardware, or computer tools. Hardware requirements lists frequently include a hardware compatibility list (HCL). In particular, operating systems. Hardware included in an HCL has been tested with a certain operating system or software. Following articles discuss hardware requirements.

Architecture –Every computer operating system is made to work with a certain type of computer architecture. Most software programs can only run on certain operating systems and computer platforms. While some operating systems and programs are architecture-independent, most need to be recompiled for a new architecture. Here is a list of popular operating systems and the platforms they are compatible with.

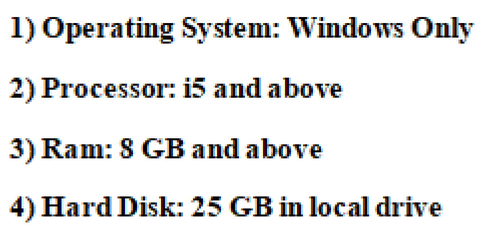
Processing power—It is important for any software that the central processing unit (CPU) has enough power. In most programs that use the x86 design, processing power is shown by the CPU type and clock speed. Many people overlook factors such as bus speed, cache, and MIPS when considering the speed and power of a CPU. Despite having the same clock speed, Athlon and Pentium CPUs can have varying levels of performance. Intel Pentium CPUs are frequently discussed in this context due to their widespread popularity.

Memory– Random access memory is responsible for storing all software when a computer is turned on. It is important to take into account the memory requirements of the application, operating system, supporting software, files, and ongoing processes in order to ensure optimal performance of multiple applications running simultaneously on a multitasking machine.

Secondary storage– The amount of hard drive space needed for a software package is determined by factors such as its size, the number of temporary files generated during installation or use, and the use of swap space if there is insufficient RAM.

Display adapter— Graphics tools and high-end games that demand a better computer graphics display typically require high-end display adapters.

Peripherals – Some software applications require extra power or functionality to utilize particular tools in a unique manner. Accessories include laptops, CD-ROM players, pointing devices, and network gadgets.



1) Operating System: Windows Only

2) Processor: i5 and above

3) Ram: 8 GB and above

4) Hard Disk: 25 GB in local drive

3.3.3IMPLEMENTATION TECHNOLOGIES

DEEP LEARNING:

What is deep learning?

A three-level neural network is deep learning in machine learning. Though far from perfect, these neural networks strive to replicate the brain. This enables them "learn" from plenty of data. A neural network with one layer may generate imprecise estimates, but additional hidden layers can enhance accuracy.

Deep learning helps many AI apps and services automate physical and mental processes without human intervention. Digital assistants, voice-enabled TV remotes, and credit card fraud detection use deep learning. Self-driving cars use deep learning.

Deep learning vs. machine learning

How are deep learning and machine learning different? Deep learning utilizes different data and learns differently from typical machine learning.

Structured, labelled data helps machine learning algorithms predict. This involves describing and tabulating model features from raw data. Unstructured data is frequently pre-processed to arrange it. It utilizes unstructured data sometimes.

Deep learning does not need all the data pre-processing of machine learning. These algorithms handle unstructured text and images. Experts are required less since they can automate feature extraction. Suppose we wanted to arrange pet photos into categories like "cat," "dog," "hamster," etc. Deep learning algorithms can determine which animal features, such ears, are most distinguishing. Experts manually rank this list of qualities in machine learning.

The deep learning algorithm then uses gradient descent and backpropagation to enhance accuracy. This helps it predict a new animal photo better.

Machine learning and deep learning models learn differently. Guided, uncontrolled, and reinforcement learning. Labeled datasets for supervised learning to categorize or predict need human involvement to name the incoming data. Unsupervised learning doesn't need named datasets. Instead, it finds data patterns and organizes them by distinct qualities. Reinforcement learning improves a model's performance in a particular situation by providing feedback. Win the largest reward.

To go more into the inconspicuous varieties among the different advances, see "AI vs. Machine Learning vs. Deep Learning vs. Neural Networks: What’s the Difference?"

For a closer look at the specific differences between supervised and unsupervised learning, see "Supervised vs. Unsupervised Learning: What's the Difference?"

How deep learning works

Deep learning neural networks strive to mimic the human brain using data sources, weights, and bias. These sections collaborate to discover, organize, and explain data.

Many layers of connected nodes make up deep neural networks. Layers enhance prediction and categorization by building on each other. Results spread over a network via forward propagation. Deep neural networks have input and output layers. Data is entered and processed at the output layer by the deep learning model. The final prediction or categorization is produced in the output layer.

Backpropagation trains a model by moving backwards through the levels and modifying function weights and biases depending on prior estimate errors. Gradient descent is used for this.

Forward propagation and backpropagation assist neural networks predict and correct errors. The software improves with time.

The simplest deep neural network requires no more explanation. Deep learning techniques are sophisticated, and different neural networks are utilized for different tasks or data sets. For instance,

- CNNs, often used in computer vision and image classification, can identify characteristics and patterns in images, enabling object recognition and identification. CNN defeated a human in object recognition for the first time in 2015.
- Recurrent neural networks (RNNs) are often utilized in natural language and voice recognition applications due to their ability to incorporate linear or time series data.

Deep learning Applications examples

We use deep learning every day, but most of the time, users don't even notice that complicated data processing is going on in the background because the apps are so well built into goods and services. The following are some examples of these:

Law enforcement

Deep learning algorithms can identify problematic financial patterns that may indicate fraud or other criminal activity. Speech recognition, computer vision, and other deep learning technologies may speed up and improve investigations by discovering patterns and evidence in audio and video recordings, photographs, and documents. Because they process vast volumes of data more correctly and rapidly.

Financial services

Predictive analytics is often used by financial companies to help clients handle their credit and investment accounts, trade stocks automatically, and decide whether to lend money to a business.

Customer service

An increasing number of businesses use deep learning technology to help with customer service. A simple example of AI is the chatbot, which is used in many apps, services, and customer service sites. Traditional chatbots use natural language and even image recognition, which you can see in choices that look like call centers. Chatbots that are smarter, on the other hand, try to learn whether there are more than one answer to an unclear question. The robot then tries to answer these questions directly or send the conversation to a real person based on the answers it gets.

The idea of a robot is expanded by virtual helpers like Apple's Siri, Amazon's Alexa, or Google Assistant, which can recognize speech. This gives you a new way to interact with people in a personalized way.

Healthcare

Deep learning has been very helpful for the healthcare business ever since medical data and pictures were digitized. Medical imaging experts and doctors can use image recognition software to help them look at and judge more pictures in less time.

Deep learning hardware requirements

A huge amount of computer power is needed for deep learning. High-performance graphical processing units (GPUs) are best because they have a lot of memory and can handle a lot of operations on multiple parts. Managing multiple GPUs on-premises, on the other hand, can put a lot of stress on internal resources and be very expensive to grow.

PYTHON LANGUAGE:

Python is a dynamic, object-oriented language that is ideal for Rapid Application Development due to its high-level data structures, dynamic types, and dynamic binding capabilities. It may be used to program or connect items. Simple Python syntax is easy to learn. It focuses on readability, which lowers the cost of maintaining programs. Python lets you use modules and packages, which makes it easier to break up programs into smaller pieces and reuse code. For all major systems, you can get the Python engine and the large standard library for free in code or binary form, and you can share them with anyone else. A lot of the time, coders fall in love with Python because it helps them get more done. The loop of change, test, and fix is very fast because there is no assembly step. It's easy to find bugs in Python programs because a segmentation fault can't be caused by a bug or bad data. Instead, an exception is raised by the translator when it finds a mistake. A stack trace is shown by the processor when the computer doesn't catch the bad code. A source level debugger in Python allows you to examine variables, execute code, set breakpoints, step through code line by line, and perform other tasks.

Writing code for the debugger in Python is an example of self-analysis. Adding print lines to source code is often the easiest way to make a program better because it makes changing, testing, and fixing go faster. Python is a dynamic, high-level, free, open-source computer language that lets you write both object-oriented and procedural code. Since it doesn't need variable types, x could be either a string or a number.

Features in Python:

Python is a programming language with many functions. Here are some of them:

1. Open Source and Free

Python is available for free on its website. Click the get Python phrase below and the link to get it. Get Python here. Open-source implies anybody may view the source code. Thus, you may get, utilize, and share it.

2. Easy to code

Python is a high-level language. C, C#, JavaScript, Java, etc. are tougher to learn than Python. Anyone can learn Python fundamentals in hours or days. The language is easy. The language is very simple for developers.

3. Easy to Read

You will see that it's easy to learn Python. As was already said, Python's grammar is very easy to understand. Instead of semicolons or braces, the indentations show what the code block is.

4. Object-Oriented Language

Object-oriented computing is one of the most important parts of Python. Python works with object-oriented languages and the ideas of classes, isolation, and so on.

5. GUI Programming Support

Graphical In Python, you can use a package like PyQt5, PyQt4, wxPython, or Tk to make a user interface. The most popular way to use Python to make graphics apps is with PyQt5.

6.High-Level Language

Python is advanced. Python simplifies program writing without requiring system or memory knowledge.

7. Extensible feature

Python is extensible. Python code may be written in C or C++ and built in those languages.

8. Easy to Debug

Great expertise for detecting errors. After reading Python's error traces, you can easily detect and repair most software issues. The code's purpose is obvious at a glance.

9. Python is a Portable language

Python may be used anywhere. Python code written for Windows will function on Linux, Unix, and Mac without modification.

10. Python is an integrated language

Python is also a combined language, which means it's easy to use with other languages like C, C++, and so on.

11. Interpreted Language:

Python executes code line by line, making it interpreted. Unlike C, C++, Java, and other languages, Python code doesn't require compilation. This simplifies code bug detection. Python source code is converted to fast byte code.

12. Large Standard Library

Python has a large standard library of modules and functions, so you don't have to create everything. Python includes packages for regular expressions, unit tests, web tools, and more.

13. Dynamically Typed Language

Python types vary dynamically. Variable types (int, double, long, etc.) are selected during runtime, not beforehand. This means we don't need to explain the variable's type.

14. Frontend and backend development

Use basic tags like <py-script> and <py-env> to execute and write Python programs in HTML. Your new project is py script. This lets you develop JavaScript-like webpages using Python. Python excels at back-end tasks, and Django and Flask make it easier.

15. Allocating Memory Dynamically

Python doesn't need data type. A variable gets memory instantly when you give it a number at runtime. If y equals 15, developers don't need to put int y = 18. Just write 18 for y.

LIBRARIES/PACKAGES :-

Tensor flow

This dataflow and differentiable computing tool is free and open source, and it can be used for many things. The tool is used for math with signs. For neural networks and other jobs that use machine learning. Google does work and school on it.

The Google Brain team made TensorFlow just for people who work at Google. It came out on November 9, 2015, under Apache 2.0.

Numpy

You can do a lot of different things with the array package Numpy. This class gives you both a fast multidimensional array object and tools for working with them.

You need this package to use Python for science computing. There are many parts to it, but these are the most important ones:

The object is a complex N-dimensional collection with methods for combining C/C++ and Fortran code. It has features that are useful for linear algebra, Fourier transform, and random numbers.

NumPy has many science uses and can store general data in a number of different ways. Because it can explain any data type, Numpy can connect to a lot of applications right away.

Pandas

Pandas is an open-source Python tool that has strong data structures that make it easy to work with and examine big amounts of data. Python was used most of the time to get data ready and load it.

It didn't really help with the study of the facts. Pants were able to solve this issue. Without a doubt, Pandas lets us handle and study data from any source by letting us load, prepare, edit, model, and examine it. Some of the business and scholarly fields that use Python and Pandas are finance, economics, statistics, analytics, and more.

Matplotlib

Matplotlib is a Python tool for making 2D charts. It can make graphs that can be printed in a variety of forms and used interactively on multiple computers. You can use Python scripts, Python and IPython tools, Jupyter Notebook, four GUI toolkits, and web application servers with this package. To make tough tasks possible and easy tasks simple. You can make bar charts, error charts, scatter plots, histograms, and power spectra with just a few lines of code. You can get ideas from the sample plots and pictures.

The pyplot package looks like MATLAB and lets you make simple plots when you use it with IPython. Power users in MATLAB know how to use a set of tools that let them change things like line styles, font properties, axes properties, and more.

Scikit – learn

The Scikit-learn library gives you a consistent way to use a variety of supervised and unsupervised learning algorithms in Python. It comes with a simple, permissive BSD license and is included in many Linux distributions, which makes it easy for businesses and schools to use.

CHAPTER 4

SYSTEM DESIGN

4.1 PROPOSED SYSTEM ARCHITECTURE

The suggested system structure uses a convolutional neural network (CNN) model that is best for sorting eye diseases into multiple groups. Innovative memory management methods are built into the design to improve memory efficiency. The design is made up of separate parts that work together to make feature extraction and classification work well. Unlike other designs, this one reduces the amount of extra memory needed by making the flow of data more efficient. An analysis of the Eye Net dataset, which includes 32 types of eye diseases, shows that it performs better in terms of accuracy and resource use. Key measures like memory, precision, and total accuracy show that the suggested method works. This system design looks like it will be a strong way to find and classify important diseases in medical images.

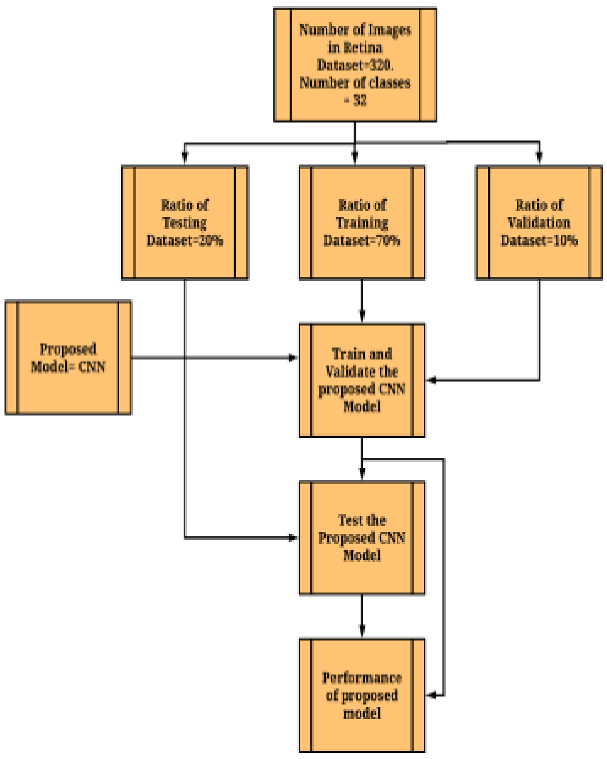

Number of Images in Retina Dataset=320. Number of classes = 32
Ratio of Testing Dataset=20%
Ratio of Training Dataset=70%
Ratio of Validation Dataset=10%
Proposed Model= CNN
Train and Validate the proposed CNN Model
Test the Proposed CNN Model
Performance of proposed model

Fig.4.1.1 System architecture

4.2 APPLICATION MODULES

In general, the program includes the main parts

Data Exploration Module:

Description: This module is all about putting data into the system so that it can be processed further.

Features: Load dataset: Download the Eye-net dataset, which has 32 groups of eye illnesses.

Data visualization:

Give people ways to see the information so they can better understand its organization and features.

Data preprocessing:

Clean up, normalize, and divide the data into training and testing sets as part of the first steps of preparation.

Image Processing Module:

Description: Images are turned into digital files and processes are carried out to gather useful information by the Image Processing Module.

Features: Image loading: Bring up pictures from the collection.

Image transformation: Change pictures into a computer file that can be processed.

Feature extraction:

Use edge recognition, segmentation, and morphological processes to get useful features out of retinal images.

Model Generation Module:

Description: This module's job is to create various convolutional neural network (CNN) models for problems with multiple classes that use little memory.

Features: Choose a model: You can pick from different CNN designs that have already been trained, such as ResNet101, VGG19, Xception, MobileNet, and mixtures like Xception and MobileNet.

Training: Use the Eye-net dataset to train the chosen model.

Model evaluation: Use measures like accuracy, precision, and memory to judge the learned models.

User Signup & Login Module:

Description: This module is in charge of registering users and making sure they are who they say they are when they log in to the system.

Features: User registration: Let users make accounts by giving the required information.

User authentication: Use details like a username and password to verify a user's identity when they log in.

User management: Give users the ability to handle their own accounts by resetting passwords and deleting accounts.

User Input Module:

Description: This tool lets people enter data to make predictions.

Features: Input interface: Give people an easy-to-use way to enter retinal pictures or other related data.

Input validation: Check the style and limitations of user input to make sure it fits the needs.

Prediction Module:

Description: This section makes the final guesses about eye diseases using the raw data and the CNN models that have been trained.

Features: Prediction engine: Use the CNN models that have been taught to guess what kind of eye diseases the input data will be.

Display results: Show the user the expected results, along with the projected class and confidence numbers that go with it.

These parts work together to make an app that uses convolutional neural networks to find and classify serious eye diseases while efficiently controlling memory use.

CHAPTER 5

IMPLEMENTATION

5.1 IMPLEMENTATION WITH HYPOTHETICAL SCENARIOS

We've looked into different deep learning models, like Artificial Neural Networks (ANN), Deep Learning, Recurrent Neural Networks (RNN), AlexNet, and ResNet, as part of our work to find and define serious eye diseases. But we've been focusing on Convolutional Neural Networks (CNN) and a variation of it called U-Net Segmentation. These have made a big difference in how eye diseases are classified. U-Net works well, but it has problems with mistakes and using a lot of memory and CPU because the feature extraction process is so complicated. This is especially true when sending the whole feature map to the parser. To solve this problem, we suggest a CNN model that is designed to work well with multi-class classification problems and make good use of memory. The Eye Net standard dataset, which has 32 types of eye diseases, was used to carefully test our model. Through a lot of testing, we've shown that our approach is better at managing memory and being accurate than other methods that are already out there. We compared full sets of data based on precision, memory, and accuracy over a range of epochs, checking how much time was needed at each step. Notably, the Eye-net dataset showed that our suggested method worked very well. Our solution includes exploring data to load it, processing pictures to get useful information from them, and making models using ResNet101, VGG19, Xception, MobileNet, and a combination of Xception and MobileNet. We also offer signup and login options for users to access the system, allow users to make guesses, and finally show the results that were expected. We hope that this unified method will provide a strong and effective way to handle the important job of classifying eye diseases.

Dataset:- https://www.kaggle.com/datasets/drsaeedmohsen/braintumordatasets

Extension

In the base paper, the author talked about using Single Image Super Resolution with the GAN model to change low-resolution images to high-resolution images and then using Image Data Generator to get features out of an image so that the VGG19 and ResNet101 models could be used for analysis. These models were tested and found to be 99% and 100% accurate, respectively. We can improve speed even more, though, by looking into Xception, MobileNet, and MobileNet with the Xception Transfer Learning model. MobileNet got 100% accuracy in testing, and as an add-on, we can use the Flask framework to build the front end for user tests and authentication.

Algorithms:

ResNet101: ResNet-101 is a convolutional neural network with 101 level of depth. There is a version of the network that has already been trained on more than a million pictures that you can get from the ImageNet collection [1]. The network has already been taught to put pictures into 1000 different groups of objects, such as keyboards, mice, pencils, and many animals.

There are 19 levels in VGG-19, which is a convolutional neural network. There is a version of the network that has already been trained on more than a million pictures that you can get from the ImageNet collection [1]. The network has already been taught to put pictures into 1000 different groups of objects, such as keyboards, mice, pencils, and many animals.

Xception: Xception has 71 levels and is a convolutional neural network. There is a version of the network that has already been trained on more than a million pictures that you can get from the ImageNet collection [1]. The network has already been taught to put pictures into 1000 groups of objects, such as keyboards, mice, pencils, and many animals.

MobileNet is a computer vision model that was made by Google and is available to everyone. It is meant to teach classifiers. It makes a lightweight deep neural network by using depthwise convolutions to cut down on the number of parameters compared to other networks. Tensorflow's first mobile computer vision model is called MobileNet.

The term "Xception" refers to the version of "InceptionV3" that has fewer factors but better features extraction. Combining the collected features from MobileNetV2 and Xception in this study suggests an ensemble that can improve the performance of finding plant diseases.

5.2 SOURCE CODE

```
import argparse
import datetime
import numpy as np
import time
import torch
import torch.nn as nn
import torch.backends.cudnn as cudnn
import json
import os

from pathlib import Path
from timm.data.mixup import Mixup
from timm.models import create_model
from timm.loss import LabelSmoothingCrossEntropy,SoftTargetCrossEntropy
    from timm.utils import ModelEma
from optim_factory import create_optimizer, LayerDecayValueAssigner


from dataset import build_dataset

from engine import train_one_epoch,evaluate

from model.resnext import ReXNetV1

from model.rexnetv2 import ReXNetV2

from model.mpvit import mpvit_base


from model.convnext import convnext_base,convnext_large,convnext_small,convnext_tiny,convnext_xlarge

from utils import NativeScalerWithGradNormCount as NativeScaler

import utils


def str2bool(v):

    if isinstance(v,bool):

        return v

    if v.lower() in ('yes', 'true', 't', 'y', '1'):

        return True

    elif v.lower() in ('no', 'false', 'f', 'n', '0'):

        return False

    else:

        raise argparse.ArgumentTypeError('Boolean value expected.')


def get_args_parser():

    parser = argparse.ArgumentParser('ConvNeXt training and evaluation script for image classification',
add_help=False)

    parser.add_argument('--batch_size', default=32, type=int,

                        help='Per GPU batch size')
    parser.add_argument('--epochs', default=1000, type=int)
    parser.add_argument('--update_freq', default=1, type=int,
                        help='gradient accumulation steps')

    # Model parameters
    parser.add_argument('--model', default='convnext_tiny', type=str, metavar='MODEL',
                        help='Name of model to train')
    parser.add_argument('--drop_path', type=float, default=0.3, metavar='PCT',
                        help='Drop path rate (default: 0.0)')
    parser.add_argument('--input_size', default=512, type=int,
                        help='image input size')
    parser.add_argument('--layer_scale_init_value', default=1e-6, type=float,
                        help="Layer scale initial values")

    # EMA related parameters
    parser.add_argument('--model_ema', type=str2bool, default=True)
    parser.add_argument('--model_ema_decay', type=float, default=0.9999, help='')
    parser.add_argument('--model_ema_force_cpu', type=str2bool, default=False, help='')
    parser.add_argument('--model_ema_eval', type=str2bool, default=True, help='Using ema to eval during
training.')

    # Optimization parameters
    parser.add_argument('--opt', default='adamw', type=str, metavar='OPTIMIZER',
                        help='Optimizer (default: "adamw"')
    parser.add_argument('--opt_eps', default=1e-8, type=float, metavar='EPSILON',
                        help='Optimizer Epsilon (default: 1e-8)')
    parser.add_argument('--opt_betas', default=None, type=float, nargs='+', metavar='BETA',
                help='Optimizer Betas (default: None, use opt default)')
parser.add_argument('--clip_grad', type=float, default=None, metavar='NORM',
                help='Clip gradient norm (default: None, no clipping)')
parser.add_argument('--momentum', type=float, default=0.9, metavar='M',
                help='SGD momentum (default: 0.9)')
parser.add_argument('--weight_decay', type=float, default=0.05,
                help='weight decay (default: 0.05)')
parser.add_argument('--weight_decay_end', type=float, default=None, help="""Final value of the
    weight decay. We use a cosine schedule for WD and using a larger decay by
    the end of training improves performance for ViTs.""")

parser.add_argument('--lr', type=float, default=1e-3, metavar='LR',
                help='learning rate (default: 4e-3), with total batch size 4096')
parser.add_argument('--layer_decay', type=float, default=1.0)
parser.add_argument('--min_lr', type=float, default=1e-6, metavar='LR',
                help='lower lr bound for cyclic schedulers that hit 0 (1e-6)')
parser.add_argument('--warmup_epochs', type=int, default=20, metavar='N',
                help='epochs to warmup LR, if scheduler supports')
parser.add_argument('--warmup_steps', type=int, default=-1, metavar='N',
                help='num of steps to warmup LR, will overload warmup_epochs if set > 0')

# Augmentation parameters
parser.add_argument('--color_jitter', type=float, default=0.4, metavar='PCT',
                help='Color jitter factor (default: 0.4)')
parser.add_argument('--aa', type=str, default='rand-m9-mstd0.5-inc1', metavar='NAME',
                help='Use AutoAugment policy. "v0" or "original". " + "(default: rand-m9-mstd0.5-inc1)'),
parser.add_argument('--smoothing', type=float, default=0.1,
          help='Label smoothing (default: 0.1)')
parser.add_argument('--train_interpolation', type=str, default='bicubic',
          help='Training interpolation (random, bilinear, bicubic default: "bicubic")')

# Evaluation parameters
parser.add_argument('--crop_pct', type=float, default=None)

# * Random Erase params
parser.add_argument('--reprob', type=float, default=0.25, metavar='PCT',
          help='Random erase prob (default: 0.25)')
parser.add_argument('--remode', type=str, default='pixel',
          help='Random erase mode (default: "pixel")')
parser.add_argument('--recount', type=int, default=1,
          help='Random erase count (default: 1)')
parser.add_argument('--resplit', type=str2bool, default=False,
          help='Do not random erase first (clean) augmentation split')

# * Mixup params
parser.add_argument('--mixup', type=float, default=0.8,
          help='mixup alpha, mixup enabled if > 0.')
parser.add_argument('--cutmix', type=float, default=1.0,
          help='cutmix alpha, cutmix enabled if > 0.')
parser.add_argument('--cutmix_minmax', type=float, nargs='+', default=None,
          help='cutmix min/max ratio, overrides alpha and enables cutmix if set (default: None)')
parser.add_argument('--mixup_prob', type=float, default=1.0,
                help='Probability of performing mixup or cutmix when either/both is enabled')
    parser.add_argument('--mixup_switch_prob', type=float, default=0.5,
                help='Probability of switching to cutmix when both mixup and cutmix enabled')
    parser.add_argument('--mixup_mode', type=str, default='batch',
                help='How to apply mixup/cutmix params. Per "batch", "pair", or "elem"')

    # * Finetuning params
    parser.add_argument('--finetune', default='Experiment/1/checkpoint-best-ema.pth',
                help='finetune from checkpoint')
    parser.add_argument('--head_init_scale', default=1.0, type=float,
                help='classifier head initial scale, typically adjusted in fine-tuning')
    parser.add_argument('--model_key', default='model|module', type=str,
                help='which key to load from saved state dict, usually model or model_ema')
    parser.add_argument('--model_prefix', default='', type=str)

    # Dataset parameters
    parser.add_argument('--data_path', default='/datasets01/imagenet_full_size/061417/', type=str,
                help='dataset path')
    parser.add_argument('--eval_data_path', default=None, type=str,
                help='dataset path for evaluation')
    parser.add_argument('--nb_classes', default=5, type=int,
                help='number of the classification types')
    parser.add_argument('--imagenet_default_mean_and_std', type=str2bool, default=False)
    parser.add_argument('--data_set', default='MICCAI', choices=['EyePacs','messidor1',
'messidor2','rfmid','apots','rsna','MICCAI'],
                type=str, help='ImageNet dataset path')
    parser.add_argument('--output_dir', default='Experiment/3',
                    help='path where to save, empty for no saving')

parser.add_argument('--fold_train', default='kfold_csv/train_fold_1.csv',
                    help='path where to save, empty for no saving')
parser.add_argument('--fold_test', default='kfold_csv/test_fold_1.csv',
                    help='path where to save, empty for no saving')

parser.add_argument('--log_dir', default='Experiment/3/log',
                    help='path where to tensorboard log')
parser.add_argument('--device', default='cuda',
                    help='device to use for training / testing')
parser.add_argument('--seed', default=0, type=int)

parser.add_argument('--resume', default='',
                    help='resume from checkpoint')
parser.add_argument('--auto_resume', type=str2bool, default=True)
parser.add_argument('--save_ckpt', type=str2bool, default=True)
parser.add_argument('--save_ckpt_freq', default=1, type=int)
parser.add_argument('--save_ckpt_num', default=3, type=int)

parser.add_argument('--start_epoch', default=0, type=int, metavar='N',
                    help='start epoch')
parser.add_argument('--eval', type=str2bool, default=False,
                    help='Perform evaluation only')
parser.add_argument('--dist_eval', type=str2bool, default=True,
                    help='Enabling distributed evaluation')
```

```
parser.add_argument('--disable_eval', type=str2bool, default=False,
                    help='Disabling evaluation during training')
  parser.add_argument('--num_workers', default=0, type=int)
  parser.add_argument('--pin_mem', type=str2bool, default=True,
                    help='Pin CPU memory in DataLoader for more efficient (sometimes) transfer to GPU.')

  # distributed training parameters
  parser.add_argument('--world_size', default=1, type=int,
                    help='number of distributed processes')
  parser.add_argument('--local_rank', default=-1, type=int)
  parser.add_argument('--dist_on_itp', type=str2bool, default=False)
  parser.add_argument('--dist_url', default='env://',
                    help='url used to set up distributed training')
  parser.add_argument('--main_eval', default='f1', type=str,
                    help="The Save the model weight besed on which metric.")
  parser.add_argument('--use_amp', type=str2bool, default=False,
                    help="Use PyTorch's AMP (Automatic Mixed Precision) or not")

  # Weights and Biases arguments
  parser.add_argument('--enable_wandb', type=str2bool, default=False,
                    help="enable logging to Weights and Biases")
  parser.add_argument('--project', default='convnext', type=str,
                    help="The name of the W&B project where you're sending the new run.")
  parser.add_argument('--wandb_ckpt', type=str2bool, default=False,


                  help="Save model checkpoints as W&B Artifacts.")

return parser
```

```
def main(args):
    utils.init_distributed_mode(args)
    print(args)
    device = torch.device(args.device)

    # fix the seed for reproducibility
    seed = args.seed + utils.get_rank()
    torch.manual_seed(seed)
    np.random.seed(seed)
    cudnn.benchmark = True

    dataset_train, args.nb_classes = build_dataset(is_train=True, args=args)
    if args.disable_eval:
        args.dist_eval = False
        dataset_val = None
    else:
        dataset_val, _ = build_dataset(is_train=False, args=args)

    num_tasks = utils.get_world_size()
    global_rank = utils.get_rank()

    sampler_train = torch.utils.data.DistributedSampler(
        dataset_train, num_replicas=num_tasks, rank=global_rank, shuffle=True, seed=args.seed,
    )
    print("Sampler_train = %s" % str(sampler_train))
    if args.dist_eval:
[1]         if len(dataset_val) % num_tasks != 0:
        print('Warning: Enabling distributed evaluation with an eval dataset not divisible by process number. '
              'This will slightly alter validation results as extra duplicate entries are added to achieve '
              'equal num of samples per-process.')
    sampler_val = torch.utils.data.DistributedSampler(
        dataset_val, num_replicas=num_tasks, rank=global_rank, shuffle=False)
else:
    sampler_val = torch.utils.data.SequentialSampler(dataset_val)

if global_rank == 0 and args.log_dir is not None:
    os.makedirs(args.log_dir, exist_ok=True)
    log_writer = utils.TensorboardLogger(log_dir=args.log_dir)
else:
    log_writer = None

if global_rank == 0 and args.enable_wandb:
    wandb_logger = utils.WandbLogger(args)
else:
    wandb_logger = None

data_loader_train = torch.utils.data.DataLoader(
    dataset_train, sampler=sampler_train,
    batch_size=args.batch_size,
    num_workers=args.num_workers,
    pin_memory=args.pin_mem,
    drop_last=True,
[1]  )
```

```
if dataset_val is not None:
    data_loader_val = torch.utils.data.DataLoader(
        dataset_val, sampler=sampler_val,
        batch_size=int(1.5 * args.batch_size),
        num_workers=args.num_workers,
        pin_memory=args.pin_mem,
        drop_last=False
    )
else:
    data_loader_val = None

mixup_fn = None
mixup_active = args.mixup > 0 or args.cutmix > 0. or args.cutmix_minmax is not None
if mixup_active:
    print("Mixup is activated!")
    mixup_fn = Mixup(
        mixup_alpha=args.mixup, cutmix_alpha=args.cutmix, cutmix_minmax=args.cutmix_minmax,
        prob=args.mixup_prob, switch_prob=args.mixup_switch_prob, mode=args.mixup_mode,
        label_smoothing=args.smoothing, num_classes=args.nb_classes)

# model = create_model(
#     args.model,
#     pretrained=False,
#     num_classes=args.nb_classes,
#     drop_path_rate=args.drop_path,
    #     layer_scale_init_value=args.layer_scale_init_value,
#    head_init_scale=args.head_init_scale,
#    )

model = ReXNetV1(width_mult=3.0,classes=args.nb_classes,dropout_path=args.drop_path)
#model.load_state_dict(torch.load('pretrain-weight/rexnetv1_1.0.pth'),strict=False)
model.load_state_dict(torch.load('rexnet_3.0.pth'),strict=False)



#model = mpvit_base(num_classes=5)

#input = torch.randn(1,3,512,512)
#checkpoint = torch.load('pretrain_weight/mpvit_base.pth')
#model.load_state_dict(checkpoint["model"],strict=False)
model.to(device)

model_ema = None
if args.model_ema:
  # Important to create EMA model after cuda(), DP wrapper, and AMP but before SyncBN and DDP wrapper
  model_ema = ModelEma(
    model,
    decay=args.model_ema_decay,
    device='cpu' if args.model_ema_force_cpu else '',
    resume='')
  print("Using EMA with decay = %.8f" % args.model_ema_decay)

model_without_ddp = model
    n_parameters = sum(p.numel() for p in model.parameters() if p.requires_grad)
    #print("Model = %s" % str(model_without_ddp))
    #print('number of params:', n_parameters)

    total_batch_size = args.batch_size * args.update_freq * utils.get_world_size()
    num_training_steps_per_epoch = len(dataset_train) // total_batch_size
    print("LR = %.8f" % args.lr)
    print("Batch size = %d" % total_batch_size)
    print("Update frequent = %d" % args.update_freq)
    print("Number of training examples = %d" % len(dataset_train))
    print("Number of training training per epoch = %d" % num_training_steps_per_epoch)

    if args.layer_decay < 1.0 or args.layer_decay > 1.0:
        num_layers = 12 # convnext layers divided into 12 parts, each with a different decayed lr value.
        assert args.model in ['convnext_small', 'convnext_base', 'convnext_large', 'convnext_xlarge'], \
             "Layer Decay impl only supports convnext_small/base/large/xlarge"
        assigner = LayerDecayValueAssigner(list(args.layer_decay ** (num_layers + 1 - i) for i in range(num_layers
+ 2)))
    else:
        assigner = None

    if assigner is not None:
        print("Assigned values = %s" % str(assigner.values))

    if args.distributed:
        print(args.gpu)
        model = torch.nn.parallel.DistributedDataParallel(model, device_ids=[args.gpu],
find_unused_parameters=False)
        model_without_ddp = model.module
```

```
optimizer = create_optimizer(
    args, model_without_ddp, skip_list=None,
    get_num_layer=assigner.get_layer_id if assigner is not None else None,
    get_layer_scale=assigner.get_scale if assigner is not None else None)

loss_scaler = NativeScaler() # if args.use_amp is False, this won't be used

print("Use Cosine LR scheduler")
lr_schedule_values = utils.cosine_scheduler(
    args.lr, args.min_lr, args.epochs, num_training_steps_per_epoch,
    warmup_epochs=args.warmup_epochs, warmup_steps=args.warmup_steps,
)

if args.weight_decay_end is None:
    args.weight_decay_end = args.weight_decay
wd_schedule_values = utils.cosine_scheduler(
    args.weight_decay, args.weight_decay_end, args.epochs, num_training_steps_per_epoch)
print("Max WD = %.7f, Min WD = %.7f" % (max(wd_schedule_values), min(wd_schedule_values)))

if mixup_fn is not None:
    # smoothing is handled with mixup label transform
    criterion = SoftTargetCrossEntropy()
elif args.smoothing > 0.:
    criterion = LabelSmoothingCrossEntropy(smoothing=args.smoothing)
else:
        criterion = torch.nn.CrossEntropyLoss()
```

```
print("criterion = %s" % str(criterion))

utils.auto_load_model(
    args=args, model=model, model_without_ddp=model_without_ddp,
    optimizer=optimizer, loss_scaler=loss_scaler, model_ema=model_ema)

if args.eval:
    print(f"Eval only mode")
    test_stats = evaluate(data_loader_val, model, device, use_amp=args.use_amp)
    print(f"kappa of the network on {len(dataset_val)} test images: {test_stats['kappa']:.5f}%")
    return

max_accuracy = 0.0
if args.model_ema and args.model_ema_eval:
    max_accuracy_ema = 0.0

print("Start training for %d epochs" % args.epochs)
start_time = time.time()
for epoch in range(args.start_epoch, args.epochs):
    print(epoch)

    if args.distributed:
        data_loader_train.sampler.set_epoch(epoch)
    if log_writer is not None:
        log_writer.set_step(epoch * num_training_steps_per_epoch * args.update_freq)
    if wandb_logger:
            wandb_logger.set_steps()
```

```
train_stats = train_one_epoch(
  model, criterion, data_loader_train, optimizer,
  device, epoch, loss_scaler, args.clip_grad, model_ema, mixup_fn,
  log_writer=log_writer, wandb_logger=wandb_logger, start_steps=epoch * num_training_steps_per_epoch,
  lr_schedule_values=lr_schedule_values, wd_schedule_values=wd_schedule_values,
  num_training_steps_per_epoch=num_training_steps_per_epoch, update_freq=args.update_freq,
  use_amp=args.use_amp
)
if args.output_dir and args.save_ckpt:
  if (epoch + 1) % args.save_ckpt_freq == 0 or epoch + 1 == args.epochs:
    utils.save_model(
      args=args, model=model, model_without_ddp=model_without_ddp, optimizer=optimizer,
      loss_scaler=loss_scaler, epoch=epoch, model_ema=model_ema)
if data_loader_val is not None:
  test_stats = evaluate(data_loader_val, model, device, use_amp=args.use_amp)
  print(f"kaapa of the model on the {len(dataset_val)} test images: {test_stats['kappa']:.4f}%")
  if args.main_eval == 'f1':
    print('f1')
    if max_accuracy < test_stats["f1"]:
      max_accuracy = test_stats["f1"]
      if args.output_dir and args.save_ckpt:
        utils.save_model(
          args=args, model=model, model_without_ddp=model_without_ddp, optimizer=optimizer,
          loss_scaler=loss_scaler, epoch="best", model_ema=model_ema)
    print(f'Max f1: {max_accuracy:.2f}%')


    elif args.main_eval == 'auc':
```

```
    if max_accuracy < test_stats["auc"]:
        max_accuracy = test_stats["auc"]
        if args.output_dir and args.save_ckpt:
            utils.save_model(
                args=args, model=model, model_without_ddp=model_without_ddp, optimizer=optimizer,
                loss_scaler=loss_scaler, epoch="best", model_ema=model_ema)
    print(f'Max auc: {max_accuracy:.2f}%')

elif args.main_eval == 'kappa':
    if max_accuracy < test_stats["kappa"]:
        max_accuracy = test_stats["kappa"]
        if args.output_dir and args.save_ckpt:
            utils.save_model(
                args=args, model=model, model_without_ddp=model_without_ddp, optimizer=optimizer,
                loss_scaler=loss_scaler, epoch="best", model_ema=model_ema)
    print(f'Max kappa: {max_accuracy:.4f}%')

elif args.main_eval == 'average':

    temp = (test_stats["kappa"] + test_stats["f1"] + test_stats["spec"]) / 3

    if max_accuracy < temp:
        max_accuracy = temp
        if args.output_dir and args.save_ckpt:
            utils.save_model(
                args=args, model=model, model_without_ddp=model_without_ddp, optimizer=optimizer,
                loss_scaler=loss_scaler, epoch="best", model_ema=model_ema)
    print(f'Max average: {max_accuracy:.4f}%')
else:
    if max_accuracy < test_stats["accuracy_score"]:
        max_accuracy = test_stats["accuracy_score"]
        if args.output_dir and args.save_ckpt:
            utils.save_model(
                args=args, model=model, model_without_ddp=model_without_ddp, optimizer=optimizer,
                loss_scaler=loss_scaler, epoch="best", model_ema=model_ema)
    print(f'Max accuracy: {max_accuracy:.4f}%')



if log_writer is not None:
    log_writer.update(test_acc1=test_stats['accuracy_score'], head="perf", step=epoch)
    log_writer.update(test_acc1=test_stats['auc'], head="perf", step=epoch)
    log_writer.update(test_acc5=test_stats['kappa'], head="perf", step=epoch)
    log_writer.update(test_acc5=test_stats['f1'], head="perf", step=epoch)
    log_writer.update(test_loss=test_stats['loss'], head="perf", step=epoch)

log_stats = {**{f'train_{k}': v for k, v in train_stats.items()},
             **{f'test_{k}': v for k, v in test_stats.items()},
             'epoch': epoch,
             'n_parameters': n_parameters}

# repeat testing routines for EMA, if ema eval is turned on
if args.model_ema and args.model_ema_eval:
    test_stats_ema = evaluate(data_loader_val, model_ema.ema, device, use_amp=args.use_amp)
        print(f'AUC of the model EMA on {len(dataset_val)} test images: {test_stats_ema['kappa']:.1f}%')
```

```
if args.main_eval == 'f1':
    if max_accuracy_ema < test_stats_ema["f1"]:
        max_accuracy_ema = test_stats_ema["f1"]
        if args.output_dir and args.save_ckpt:
            utils.save_model(
                args=args, model=model, model_without_ddp=model_without_ddp, optimizer=optimizer,
                loss_scaler=loss_scaler, epoch="best-ema", model_ema=model_ema)
        print(f'Max f1 : {max_accuracy_ema:.4f}%')
# if log_writer is not None:
#     log_writer.update(test_acc1_ema=test_stats_ema['accuracy_score'], head="perf", step=epoch)
    log_stats.update({**{f'test_{k}_ema': v for k, v in test_stats_ema.items()}})

elif args.main_eval == 'auc':
    if max_accuracy_ema < test_stats_ema["auc"]:
        max_accuracy_ema = test_stats_ema["auc"]
        if args.output_dir and args.save_ckpt:
            utils.save_model(
                args=args, model=model, model_without_ddp=model_without_ddp, optimizer=optimizer,
                loss_scaler=loss_scaler, epoch="best-ema", model_ema=model_ema)
        print(f'Max auc : {max_accuracy_ema:.4f}%')
    log_stats.update({**{f'test_{k}_ema': v for k, v in test_stats_ema.items()}})

elif args.main_eval == 'kappa':
    if max_accuracy_ema < test_stats_ema["kappa"]:
        max_accuracy_ema = test_stats_ema["kappa"]
        if args.output_dir and args.save_ckpt:
            utils.save_model(
                args=args, model=model, model_without_ddp=model_without_ddp, optimizer=optimizer,
                loss_scaler=loss_scaler, epoch="best-ema", model_ema=model_ema)
        print(f'Max kappa : {max_accuracy_ema:.4f}%')
    log_stats.update({**{f'test_{k}_ema': v for k, v in test_stats_ema.items()}})

  elif args.main_eval == 'average':
    temp = (test_stats["kappa"] + test_stats["f1"] + test_stats["spec"]) / 3
    if max_accuracy_ema < temp:
        max_accuracy_ema = temp
        if args.output_dir and args.save_ckpt:
            utils.save_model(
                args=args, model=model, model_without_ddp=model_without_ddp, optimizer=optimizer,
                loss_scaler=loss_scaler, epoch="best-ema", model_ema=model_ema)
        print(f'Max average : {max_accuracy_ema:.4f}%')
    log_stats.update({**{f'test_{k}_ema': v for k, v in test_stats_ema.items()}})

  else:
    if max_accuracy_ema < test_stats_ema["accuracy_score"]:
        max_accuracy_ema = test_stats_ema["accuracy_score"]
        if args.output_dir and args.save_ckpt:
            utils.save_model(
                args=args, model=model, model_without_ddp=model_without_ddp, optimizer=optimizer,
                loss_scaler=loss_scaler, epoch="best-ema", model_ema=model_ema)
        print(f'Max accuracy : {max_accuracy_ema:.4f}%')
    log_stats.update({**{f'test_{k}_ema': v for k, v in test_stats_ema.items()}})

else:
        log_stats = {**{f'train_{k}': v for k, v in train_stats.items()},
                     'epoch': epoch,
                     'n_parameters': n_parameters}

        if args.output_dir and utils.is_main_process():
            if log_writer is not None:
                log_writer.flush()
            with open(os.path.join(args.output_dir, "log.txt"), mode="a", encoding="utf-8") as f:
                f.write(json.dumps(log_stats) + "\n")

        if wandb_logger:
            wandb_logger.log_epoch_metrics(log_stats)

    if wandb_logger and args.wandb_ckpt and args.save_ckpt and args.output_dir:
        wandb_logger.log_checkpoints()



    total_time = time.time() - start_time
    total_time_str = str(datetime.timedelta(seconds=int(total_time)))
    print('Training time {}'.format(total_time_str))

if __name__ == '__main__':
    parser = argparse.ArgumentParser('retinal dataset training and evaluation script', parents=[get_args_parser()])
    args = parser.parse_args()
    if args.output_dir:
        Path(args.output_dir).mkdir(parents=True, exist_ok=True)
    main(args)
```

CHAPTER 6

RESULTS

The study presents a new CNN model that is intended to efficiently use memory for multi-class classification of eye diseases. Looking at the EyeNet collection, which has 32 eye diseases, shows encouraging results. Compared to current state-of-the-art methods, the suggested model does a better job of managing memory and being accurate. Researchers did some tests and found that the model had good validation accuracy across a range of epochs. It did especially well at 10 and 15 epochs, with only a small change in validation loss (0.0279) between them. This shows how strong and stable the model is. The study shows that the suggested CNN design has a lot of promise for making it easier to classify eye diseases, which is good news for future progress in the field. Continuous model improvement and training with updated datasets should make it even more useful, taking advantage of new developments in deep learning techniques and the growing availability of datasets that show different eye diseases.

CHAPTER 7

CONCLUSION

We show a CNN model based on deep learning that can sort the different eye diseases into groups. EyeNet, a collection with 32 different eye diseases, is used to put the model into action. Different epochs are used to train the proposed model and test how well it works. At first, the model was trained over 10 iterations and had good validation accuracy. Then, it was trained over 15 iterations and again had good validation accuracy with a validation loss of 0.0279, which is different each time. The model's overall performance is much better than that of other models that are thought to be the best on the market. That being said, the model that was given might help with the classification of eye illnesses. Updating the model often and retraining it with new data will keep making it work better in the future. This will be done by taking advantage of improvements in deep learning methods and the growing number of datasets that include different types of eye diseases.

FUTURE ENHANCEMENTS AND DISCUSSIONS

There are a number of ways that the CNN model for classifying eye diseases can be improved in the future. To begin, using transfer learning methods on bigger and more varied picture datasets can improve performance even more by using models that have already been taught. By mixing results from different models, ensemble learning methods could also improve the stability and generalizability of models. Adding self-attention or attention processes may also help the model focus on important parts of retinal pictures, which could lead to better classification accuracy. It is important to keep improving models by fine-tuning them on new and bigger datasets so that they can adapt to changing trends in eye diseases. Using new deep learning structures and methods, like graph neural networks or capsule networks, could help us learn new things and make our classifications more accurate. Lastly, putting the model to use in hospital settings and getting feedback from doctors and nurses can help make more changes that fit the needs of real-world applications.

REFERENCES

[1] A. Esteva, B. Kuprel, R. A. Novoa, J. Ko, S. M. Swetter, H. M. Blau, and S. Thrun, ‘‘Dermatologist-level classification of skin cancer with deep neural networks,’’ Nature, vol. 542, no. 7639, pp. 115–118, Feb. 2017.

[2] K. Shankar, A. R. W. Sait, D. Gupta, S. K. Lakshmanaprabu, A. Khanna, and H. M. Pandey, ‘‘Automated detection and classification of fundus diabetic retinopathy images using synergic deep learning model,’’ Pattern Recognit. Lett., vol. 133, pp. 210–216, May 2020.

[3] R. Arunkumar and P. Karthigaikumar, ‘‘Multi-retinal disease classification by reduced deep learning features,’’ Neural Comput. Appl., vol. 28, no. 2, pp. 329–334, Feb. 2017.

[4] T. Shanthi and R. S. Sabeenian, ‘‘Modified Alexnet architecture for classification of diabetic retinopathy images,’’ Comput. Electr. Eng., vol. 76, pp. 56–64, Jun. 2019.

[5] S. Farsiu, S. J. Chiu, R. V. O’Connell, F. A. Folgar, E. Yuan, J. A. Izatt, and C. A. Toth, ‘‘Quantitative classification of eyes with and without intermediate age-related macular degeneration using optical coherence tomography,’’ Ophthalmology, vol. 121, no. 1, pp. 162–172, Jan. 2014.

[6] R. F. Mullins, S. R. Russell, D. H. Anderson, and G. S. Hageman, ‘‘Drusen associated with aging and age-related macular degeneration contain proteins common to extracellular deposits associated with atherosclerosis, elastosis, amyloidosis, and dense deposit disease,’’ FASEB J., vol. 14, no. 7, pp. 835–846, May 2000.

[7] Y. Kanagasingam, A. Bhuiyan, M. D. Abràmoff, R. T. Smith, L. Goldschmidt, and T. Y. Wong, ‘‘Progress on retinal image analysis for age related macular degeneration,’’ Prog. Retinal Eye Res., vol. 38, pp. 20–42, Jan. 2014.

[8] D. S. Kermany, ‘‘Identifying medical diagnoses and treatable diseases by image-based deep learning,’’ Cell, vol. 172, no. 5, pp. 1122–1131, Feb. 2018.

[9] M. M. M. S. Fathy and M. T. Mahmoudi, ‘‘A classified and comparative study of edge detection algorithms,’’ in Proc. Int. Conf. Inf. Technol., Coding Comput., Apr. 2002, pp. 117–120.

[10] C.-H. H. Yang, J.-H. Huang, F. Liu, F.-Y. Chiu, M. Gao, W. Lyu, M. D. I.-H. Lin, and J. Tegner, ‘‘A novel hybrid machine learning model for auto-classification of retinal diseases,’’ 2018, arXiv:1806.06423.

[11] M. B. Jabra, A. Koubaa, B. Benjdira, A. Ammar, and H. Hamam, ‘‘COVID- 19 diagnosis in chest X-rays using deep learning and majority voting,’’ Appl. Sci., vol. 11, no. 6, p. 2884, Mar. 2021.

[12] S. Guefrechi, M. B. Jabra, A. Ammar, A. Koubaa, and H. Hamam, ‘‘‘Deep learning based detection of COVID-19 from chest X-ray images,’’ Multimedia Tools Appl., vol. 80, no. 2021, pp. 31803–31820.

[13] W. Boulila, A. Ammar, B. Benjdira, and A. Koubaa, ‘‘Securing the classification of COVID-19 in chest X-ray images: A privacy-preserving deep learning approach,’’ in Proc. 2nd Int. Conf. Smart Syst. Emerg. Technol. (SMARTTECH), May 2022, pp. 220–225.

[14] O. Perdomo, H. Rios, F. J. Rodríguez, S. Otálora, F. Meriaudeau, H. Müller, and F. A. González, ‘‘Classification of diabetes-related retinal diseases using a deep learning approach in optical coherence tomography,’’ Comput. Methods Programs Biomed., vol. 178, pp. 181–189, Sep. 2019.

[15] G. Mahendran, M. Periyasamy, S. Murugeswari, and N. K. Devi, ‘‘Analysis on retinal diseases using machine learning algorithms,’’ Mater. Today, Proc., vol. 33, pp. 3102–3107, Jan. 2020.