Data Cleaning in Eye Image Dataset

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

in

Computer Science and Engineering/Information Technology

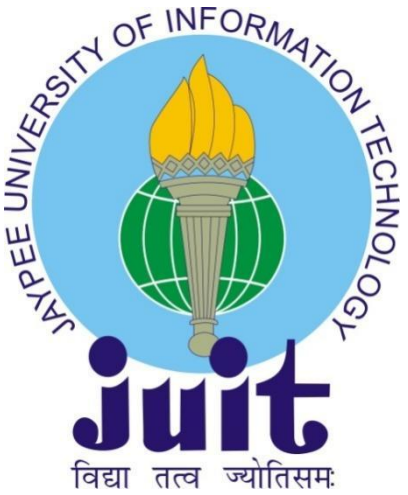
By

Harshit Singh (191264) & Archit Dogra (191277)

Under the supervision of

Mr. Praveen Modi

to



Department of Computer Science & Engineering and Information Technology

Jaypee University of Information Technology
Waknaghat, Solan-173234, Himachal Pradesh

Candidate's Declaration

I hereby declare that the work presented in this report entitled **"Data Cleaning in Eye Image Dataset"** 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 July 2022 to May 2023 under the supervision of **Mr. Praveen Modi** Assistant Professor in Department of Computer Science & Engineering.
The matter embodied in the report has not been submitted for the award of any other degree or diploma.

(Student Signature)
Harshit Singh, 191264
Archit Dogra, 191277

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

(Supervisor Signature)
Supervisor Name: Mr. Praveen Modi
Designation: Assistant Professor
Department name: Computer Science & Engineering
Dated:

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><td>Copy Received on</td><td rowspan="4">• All Preliminary Pages
• Bibliography/Images/Quotes
• 14 Words String</td><td rowspan="3"></td><td>Word Counts</td><td></td></tr>
<tr><td rowspan="2">Report Generated on</td><td>Character Counts</td><td></td></tr>
<tr><td rowspan="2">Submission ID</td><td>Total Pages Scanned</td><td></td></tr>
<tr><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

ACKNOWLEDGEMENT

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

I am really grateful and wish my profound indebtedness to the Supervisor **Mr. Praveen Modi, Assistant Professor**, Department of Computer Science & Engineering in Jaypee University of Information Technology, Waknaghat. Deep Knowledge & keen interest of my supervisor in the field of "**Data Cleaning**" to carry out this project. His endless patience, scholarly guidance, continual encouragement, constant and energetic supervision, constructive criticism, valuable advice, and reading many inferior drafts and correcting them at all stages have made it possible to complete this project.

I would like to express my heartiest gratitude to **Mr. Praveen Modi**, Department of Computer Science & Engineering, for his kind help to finish my 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.

Project Group No. :132
Student Name: Harshit Singh
Roll No.:191264
Student Name: Archit Dogra
Roll No.:191277

TABLE OF CONTENT

Content **Page No.**

LIST OF ABBREVIATIONS

Abbreviations	Full Form	Page Number
PNG	Portable Network Graphics	3
AI	Artificial Intelligence	14
VGG	Visual Geometry Group	6
API	Application Programming Interface	20
SVM	Support Vector Machine	7
RF	Random Forest	7
CNN	Convolutional Neural Network	31
PCA	Principal Component Analysis	8
RGB	Red Green Blue	8
NN	Neural Network	36
ML	Machine Learning	67
FER	Facial Expression Recognition	44
FC Layer	Fully Connected Layer	39
DFT	Discrete Fourier Transform	57

LIST OF FIGURES

LIST OF GRAPHS

LIST OF TABLES

ABSTRACT

There's a common adage that data scientists spend 90% of their time cleaning data and 10% modelling. With image classifiers, it is more like 99% cleaning to 1% modelling. This is because a neural network needs images to be of a standardized size. How many pictures do we come across on a google image search that are all the same size? There are a bevy of different approaches for standardizing images and it is important to remember that no method is necessarily better or worse than another. Each one has its own drawbacks and applications. Oftentimes your ultimate limiter will be computer power.

Although user-generated image data increases more and more quickly on the current Internet, many image methods have attracted widespread attention from industry and academia. Recently, some image classification approaches using deep learning have demonstrated that they can potentially enhance the accuracy of the classification based on the high quality datasets. However, the existing methods only consider the accuracy of the classification and ignore the quality of the datasets. To address these issues, we propose a new image data cleaning framework using deep neural networks, named ImageDC, to improve the quality of the datasets. Image DC not only uses cleaning with the minority class to remove the images of the rare classes, but also adopts cleaning with the low recognition rate to remove the noisy data to enhance the recognition rate of the datasets. Experimental results conducted on a variety of datasets demonstrate that our model significantly outperforms the whole approach.

Detecting incorrect scenes uploaded by users and maintaining the correctness of the database through automatic data cleaning is essential because human inspection is not feasible for verifying massive amounts of data. In this study, we compared different feature extractors using deep convolutional neural networks trained using big data. We designed a multilevel extractor to improve feature extraction. Moreover, a detector based on multiresolution

dissimilarity calculation was designed to overcome the issue of large intra class distances and successfully identify incorrect scenes. The proposed system was validated using a highly challenging dataset with 138 000 images collected from Google Places. The experiments show that the multilevel extractor and the detector based on multiresolution dissimilarity calculation can improve the accuracy in identifying incorrect scenes and achieve satisfying data cleaning results

CHAPTER-1
INTRODUCTION

1.1 Introduction

Data cleaning involves the elimination of erroneous and incorrectly formatted data. It is common for such problems to arise during the data collection process, especially when data is sourced from different sources. This can lead to duplicated and mislabeled data, as well as irrelevant data inputted by human beings during the data collection process. Such anomalies can render a dataset useless in machine learning training as the model will not be able to accurately learn from it. Thus, data cleaning is crucial as it uses various methods to remove these anomalies and produce a cleaned dataset that can be effectively used for model training.

This project focuses on applying these data cleaning methods to a dataset of eye images for the purpose of identifying eye diseases through data analysis.

Data cleaning is a critical aspect of machine learning, as a clean dataset can generate better results even with a less complex algorithm, compared to a complex algorithm processing a dataset with errors. A model's accuracy and reliability depend on the quality of the data used for training. Machine learning models are not intelligent and can only learn from the data presented to them. Therefore, inaccurate or incomplete data can lead to incorrect model features and flawed classification results. In essence, it is better to have better data than fancy algorithms.

1.2 Problem Statement

The process of data cleaning in image datasets is accompanied by several challenges or issues. Images are visual data that are transformed into matrices and learned by machine learning algorithms. However, algorithms require pre-processed or cleaned data to function effectively.

One of the problems in image data cleaning is intra-class variation. When there are different types or variations of images within a single class, such as different brands or models of cars in a vehicle classification algorithm, it is challenging to label and organize the data. The data cleaning process must address this issue by identifying and resolving intra-class variation in the dataset.

Another significant problem is scale variation, where images of the same object or class differ in size and aspect ratio. This can change the matrix of the image substantially while still representing the same object. Appropriate techniques must be used to remove the problem of scale variation while retaining the integrity of the data.

Viewpoint variation is another problem that arises when an object appears in the input data from various viewpoints or angles. For instance, images of the same eye taken from different angles need to be processed with specific constraints to ensure accurate data cleaning.

Occlusion is another issue in image classification where other objects or elements occupy parts of the image, affecting the accuracy of the algorithm's learning process. Effective data cleaning techniques must address the problem of occlusion to enhance the accuracy of image classification systems.

1.3 Objective

In our project we have medical data of eye images, which contains images of human eyes, following are the objectives to achieve the goal of this project.

- After identifying total number of classes in the final classification of the medical data of eye images, it is important to identify any mislabelled images. In the case that there are images with labels that do not match the available classes, it becomes necessary to determine whether to merge this data into a particular class or exclude it from the dataset altogether. The

decision to merge or exclude such data is based on the similarity scores of the images and the total number of similar data elements. The most similar cluster can be identified using similarity scores, and this data can be merged into the rest of the dataset. Ultimately, the goal is to achieve accurate classification of all images in the datasetTo remove scale variation images has to be resized in a common aspect ratio, pixel and size, this has to be done in such a way that no data is lost or modified.

- In order to eliminate differences resulting from different perspectives, it is necessary to impose restrictions on data collection to ensure that only images with the desired orientation are selected. If data has already been collected without this in mind, any elements with an undesired orientation must be identified and removed.

- If there are additional parts in the images like different objects which are not required then such images need to be cropped or removed from the dataset.

1.4 Methodology

The dataset of eyes, which was sourced from Kaggle https://www.kaggle.com/datasets/kayvanshah/eye-dataset , was contributed by Kavyash Shah in the year 2020. The dataset is made up of a total of 14.5 thousand images of eyes, out of which 11.5 thousand images are in the PNG format, while the remaining 2815 images are in the JPG format. The data collection process involved the use of Google Photos, where a certain methodology was employed. Initially, a few images were collected from Google Images, and then they were modified to generate images from different angles. After that, these images were compiled together into various folders to generate the complete dataset of eyes.

The dataset of eyes, which was sourced from Kaggle, was contributed by Kavyash Shah in the year 2020. The dataset is made up of a total of 14.5

thousand images of eyes, out of which 11.5 thousand images are in the PNG format, while the remaining 2815 images are in the JPG format. The data collection process involved the use of Google Photos, where a certain methodology was employed. Initially, a few images were collected from Google Images, and then they were modified to generate images from different angles. After that, these images were compiled together into various folders to generate the complete dataset of eyes.

To ensure that the images in the dataset are standardized and consistent, they were subjected to a process of regularization and augmentation. As a result, the dataset achieved an accuracy of over 96% during the initial training session. The final dataset of eyes is composed of four directories, which are named "close-look," "left-look," "right-look," and "forward-look." These directories contain 3828 close-look images, 3457 forward-look images, 3498 left-look images, and 3577 right-look images, respectively.

To ensure that the images in the dataset are standardized and consistent, they were subjected to a process of regularization and augmentation. As a result, the dataset achieved an accuracy of over 96% during the initial training session. The final dataset of eyes is composed of four directories, which are named "close-look," "left-look," "right-look," and "forward-look." These directories contain 3828 close-look images, 3457 forward-look images, 3498 left-look images, and 3577 right-look images, respectively.

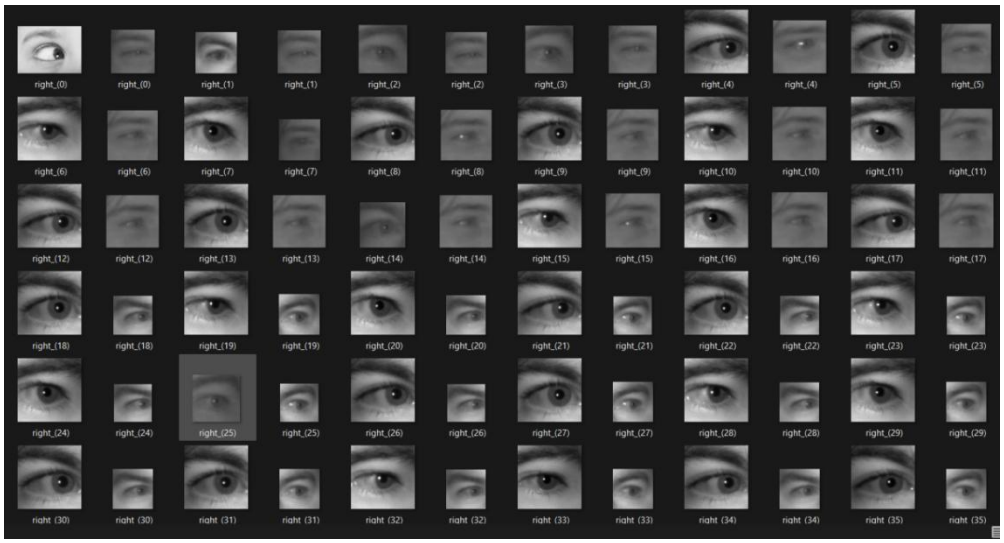


Fig 1: Night Eye Images Dataset [6]

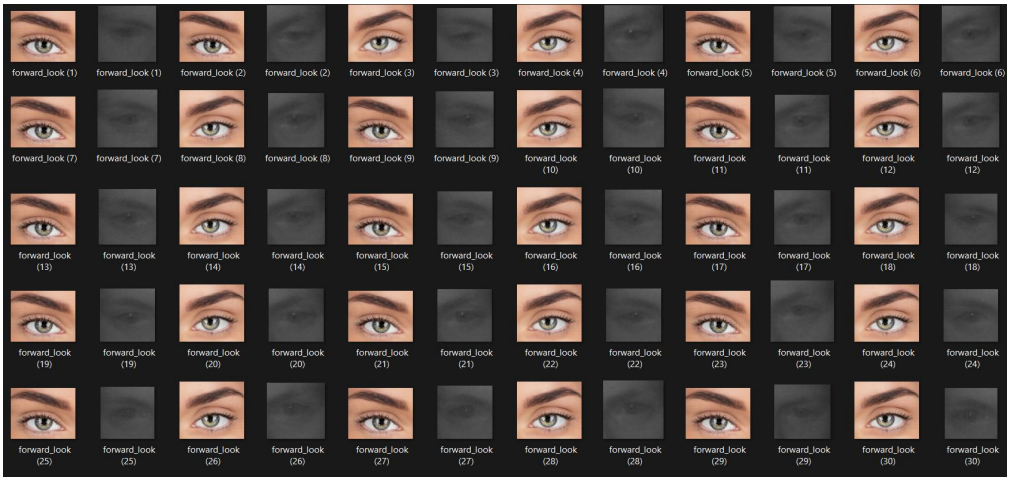
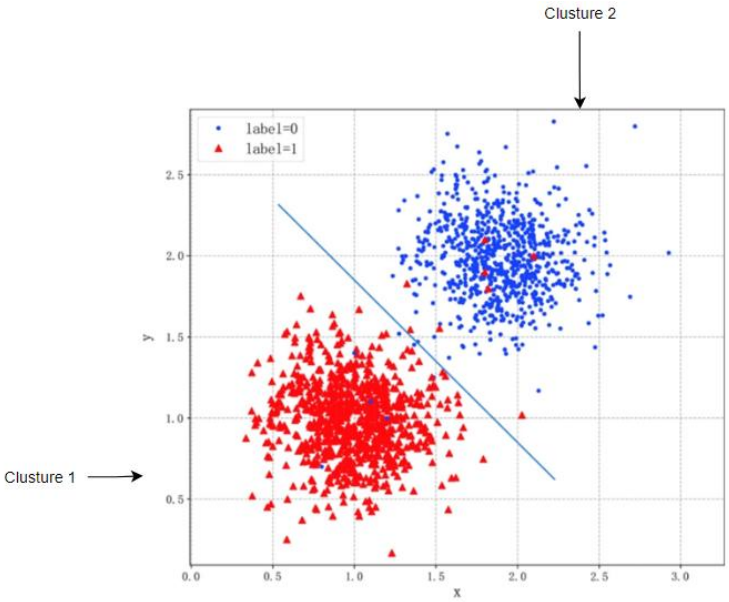


Fig 2: Day Eye Images Dataset [6]

During the image cleaning phase, we utilized various programming tools and libraries to ensure the accuracy and consistency of the dataset. Specifically, we employed Python 3.6.4 as our programming language, and PyTorch 1.0.0 as a deep learning library. Additionally, we utilized Pandas and Numpy to perform basic computation of the image matrix in the cleaning model. The pygenerator library was also used to generate the dataset, while matplotlib was utilized to plot the accuracy curve and some basic comparison curves throughout the project.

One of the major challenges that we faced during the image cleaning phase was dealing with mislabelled images present in the dataset. To solve this issue,

we relied on the assumption that the boundary in clustering problems is typically located in a low-density region. In the case of binary image classification, the model learns about the common features of each category from the high-density region. However, the mislabelled data sources are often indistinguishable data hidden in the low-density region and the mislabelled data is often present as background noise data or may be simply due to human errors during the dataset collection and creation. Therefore, we identified and removed this portion of the dataset to ultimately increase the accuracy of the model.



Graph 1: Clustering of data points [2]

In order to identify and eliminate most of the wrong images, we implemented an algorithm that repeatedly screens the dataset until a clean and precise dataset is generated. The steps involved in this algorithm are as follows:

- We used a neural network model called VGG_NIN to fetch all the datasets that need to be cleaned.

- Then, we used this trained model to classify all the images in the dataset and identify the wrongly labelled ones which are deleted from the dataset.

- Steps 1 and 2 are repeated, and a threshold hyperparameter is set to stop the process when the ratio of screen errors is lower than the specified threshold.

The cleaning model was trained on erroneous data sets using the cross entropy loss function and variable learning rate. Each batch of 32 training photos had 224 x 224 pixels in each part. The VGG NIN neural network model was chosen for training. In order to compare the outcomes of each trial to some extent, fixed random seeds were used during the training and cleaning processes for the classification model and error image model.

While accurate image datasets may contain false images, such as background noise images and label noise, this project focuses on eliminating false data and managing mislabelled data, specifically in ocular image datasets. Label noise can be classified into two categories:

- Intra-class label noise images are those with incorrect labels within the two types of images, such as early eye cancer being labelled as non-early cancer and vice versa.

- External label noise images are those outside of the binary image dataset that are labelled as part of the dataset, such as eye photos being included in a classification of eye cancer into two categories: premature and non-premature.

Convolutional Neural Network

There are two main types of categorization methods that are commonly used in machine learning. One type involves exclusionary models, such as support vector machines (SVM), random forests (RF), and convolutional neural networks (CNNs). These models are based on traditional learning techniques that are designed to classify data into different categories. The other type

involves custom feature-based models, which are used to calculate a large number of features in order to accurately classify data using traditional machine learning algorithms. However, calculating these features can be difficult and may result in lower recall rates. To overcome this, more efficient solutions have been developed that use dimensionality reduction techniques like principal component analysis (PCA) or feature selection approaches to reduce the number of features used in classification.

When it comes to brain tumour segmentation, exclusionary approaches are typically used. This is because these procedures rely on extracting a large number of low-level image attributes and don't require much prior knowledge of brain anatomy. This is in contrast to generative modelling techniques, which use more advanced modelling techniques to represent features and specific particle IDs.

In recent research, a convolutional neural network architecture was used to detect and classify brain tumours. CNNs are commonly used to analyze fine mesh data, such as biometrics or images, and are highly effective in identifying patterns and features in complex data.
The unique 3D structure of the neural network allows it to analyze the RGB layer of the image in a highly efficient manner. Unlike previous methods, it analyzes one image at a time and identifies its essential components before using those characteristics to classify the image. The input learning technique automatically detects medium- and high-level representations via convolutional neural networks (ConvNets). This allows for highly accurate brain tumour segmentation and classification, even when dealing with complex and highly variable datasets.

Advantages

Convolutional Neural Networks (CNNs) are a type of neural network that is designed to process and analyze visual data such as images and videos. They have gained popularity in recent years due to their superior performance in

various computer vision tasks, including object recognition, image segmentation, and detection. CNNs have several advantages over traditional machine learning techniques, including:

- High accuracy: CNNs have achieved state-of-the-art performance on several computer vision tasks, such as image classification, object detection, and segmentation. This is because they are designed to learn complex features from data, allowing them to capture the intricate details of images that are essential for accurate classification.

- Ability to learn hierarchical features: One of the key advantages of CNNs is their ability to learn hierarchical features. In traditional machine learning techniques, the features that are used to classify an image are often hand-crafted by domain experts. This can be a time-consuming and challenging task, especially when dealing with large datasets. CNNs, on the other hand, can learn features automatically through a hierarchical process. The first layer of a CNN learns low-level features, such as edges and corners, while the subsequent layers learn more complex features, such as shapes and textures.

- Efficient memory utilization: CNNs are designed to efficiently utilize memory. They use shared weights and biases across multiple neurons in the same layer, which reduces the number of parameters that need to be stored in memory. This makes CNNs computationally efficient and allows them to process large amounts of data with limited resources.

- Robustness to input variations: CNNs are robust to input variations such as changes in scale, orientation, and lighting conditions. This is because they learn features that are invariant to these variations. For example, a CNN trained to recognize faces can recognize a face in an image regardless of whether it is upside down or tilted at an angle.

- Transfer learning: CNNs can be used for transfer learning, which involves reusing a pre-trained model for a new task. This is useful when there is a limited amount of labeled data available for a new task. Instead of training a CNN from scratch, a pre-trained model can be fine-tuned for the new task. This can significantly reduce the training time and improve the performance of the model.

- Parallel processing: CNNs can be easily parallelized, allowing them to process multiple images simultaneously. This is especially useful for real-time applications, such as video analysis, where speed is critical.

- Interpretable features: CNNs learn features that are interpretable, which makes them useful for understanding how the model makes its predictions. For example, the features learned by a CNN trained to recognize faces can provide insight into which facial features are important for recognition.

- Data augmentation: CNNs can be augmented with additional data to improve their performance. Data augmentation involves creating new training data by applying transformations to the existing data. For example, an image can be rotated, cropped, or flipped to create a new image. This can increase the size of the training dataset and reduce overfitting.

- Self-learning: CNNs can be trained in a self-supervised or unsupervised manner, which involves learning features without the need for labeled data. Self-supervised learning involves training a CNN to predict a missing part of an image or to colorize a grayscale image. Unsupervised learning involves training a CNN to learn representations of the data without any explicit labels.

In conclusion, CNNs have several advantages over traditional machine learning techniques, including high accuracy, the ability to learn hierarchical

features, efficient memory utilization, robustness to input variations, transfer learning, parallel processing, interpretable features, data augmentation, and self-learning. These advantages make CNNs an essential tool for computer vision tasks and have enabled significant progress in the

Disadvantages

Although CNNs offer a wide range of benefits, there are also some disadvantages associated with this technique. In this section, we will discuss some of the limitations of CNNs.

- Limited Understanding of Global Structure One of the main limitations of CNNs is their limited understanding of global structure. While CNNs are excellent at detecting local patterns in images, they often fail to capture the global context of an image. This can be a problem when it comes to tasks such as object recognition, where the overall context of an image is crucial for accurate classification.

- Difficulty with Small Datasets Another limitation of CNNs is their difficulty with small datasets. CNNs are designed to learn from large amounts of data, and when the dataset is small, they may not be able to learn the underlying patterns effectively. In such cases, traditional machine learning algorithms may be more effective.

- High Computational Requirements CNNs are computationally expensive and require a significant amount of processing power to train and deploy. This can be a problem for researchers and businesses with limited computational resources.

- Limited Interpretability While CNNs can achieve high levels of accuracy in image recognition tasks, they lack interpretability. It can be challenging to understand why a particular decision was made by a CNN, making it difficult to debug and fine-tune the model.

- Vulnerability to Adversarial Attacks CNNs are vulnerable to adversarial attacks, where small perturbations are added to an image to cause the model to misclassify it. This can be a problem in security-critical applications such as autonomous driving, where a misclassification could lead to severe consequences.

- Difficulty with Rotation and Scaling Invariance While CNNs are effective at detecting patterns in images, they are not inherently rotation or scale invariant. This means that if an object is rotated or scaled, the CNN may not be able to recognize it.

- Limited Generalization CNNs are often trained on a specific dataset and may not generalize well to new datasets or applications. This can be a problem when trying to apply CNNs to real-world problems where the data may be different from what the model was trained on.

In conclusion, CNNs offer significant advantages in image recognition tasks, but they also have some limitations that need to be addressed. Researchers and practitioners should be aware of these limitations and work to overcome them to develop more robust and effective CNN models.

1.4.1 Software Requirements:

Python 3.9

In order to give Python project maintainers more time to plan the removal of Python 2 support and the addition of support for Python 3.9, Python 3.9 is the last version to provide those Python 2 backward compatibility layers.

Advantages of Using Python

Python is a popular programming language used by developers for various reasons. Here are some of the advantages of using Python:

1. Platform Independence: One of the main reasons developers prefer Python over other programming languages is its ability to work on multiple platforms without requiring significant modifications. Python is a cross-platform language, which means it can be used on different operating systems, including Windows, Linux, and macOS, with minimal adjustments. This platform independence makes it easier to deploy software and create independent applications. Moreover, Python is a universal language that works on almost all platforms, so there's no need for a Python specialist to explain the code.

2. Consistency and Simplicity: Python code is concise, clear, and easy to understand, making it easier for developers seeking consistency and simplicity in their work. With Python, developers can write code quickly and effectively, making it easier to get feedback from other developers in the community to improve their program or service. Python is simpler to learn and master than other programming languages, making it suitable for novice developers. Moreover, experienced developers can build reliable systems with Python and concentrate on developing their creativity and applying machine learning to real-world problems.

3. Variety of Frameworks and Libraries: Python has an extensive collection of libraries and frameworks that provide a dependable environment for developing programs. These libraries and frameworks can help speed up program development, particularly on large projects. PyBrain is an example of a Python-based modular machine learning toolkit that offers simple-to-use algorithms for machine learning applications. Python frameworks and libraries provide a proper structure and environment that are necessary for the best and most reliable coding solutions.

Why Python is Most Suitable for our project

The use of machine learning and artificial intelligence (AI) has grown significantly in recent years due to the demand for automation. These technologies can be used to solve common problems, such as developing search engines, spam filters, and recommendation systems. In order to provide intelligent solutions to real-world problems and automate tasks that are otherwise difficult to perform without AI, the development of AI technologies must continue to progress. Python is considered to be the best programming language for automating these tasks, given its simplicity and reliability compared to other programming languages.

One of the primary advantages of Python is its active developer community, which allows programmers to collaborate on projects and provide feedback on how to improve their code. Even experienced programmers and developers have much to learn in the complex world of software development, and having access to a vibrant community where people can discuss and exchange ideas is a valuable asset. Python is a widely used programming language for machine learning, data analysis, regression, web development, and other tasks due to its large developer community.

Python's forums for developers actively encourage the growth of the entire artificial intelligence community. These forums provide resources that help students advance their understanding of Python-based machine learning, which in turn helps to increase the number of specialists in this field. Python is also gaining popularity among large corporations due to its effectiveness and simplicity. Companies such as Google use the programming language to crawl websites, while Spotify uses it to choose songs and entertainment companies use it to create movies.

Jupyter notebook 6.4.12

The Jupyter notebook is a web-based tool that is ideal for documenting every step of the calculation process, allowing users to create, write, and execute code while communicating the results. This represents a significant shift in interactive computing, moving away from the console-based method. The Jupyter notebook is made up of two key components, namely the jupyter web application and notebook documents.The jupyter web application is a browser-based tool that allows users to create documents iteratively with mathematical computations, explanatory text, and the corresponding output in rich media. On the other hand, the notebook documents provide a visual representation of all the materials presented in the web application, including the inputs and outputs of calculations, justification language, mathematics, rich media object representations, and graphics.

Some of the main features of the Jupyter web application include the ability to run browser-based programming, with calculation results linked to the code that generated them. Additionally, rich media representations can be employed to display the results of computation, such as high-quality figures generated by the matplotlib library that can be inserted inline. Furthermore, the in-browser editing of rich text is not limited to plain text, and it can also include code commentary by utilizing the Markdown markup language. Lastly, users can quickly insert mathematical notation using LaTeX and MathJax's native rendering into markdown cells.

Advantages of using Jupyter Notebook in our project –

1. All-in-one solution: Jupyter Notebook is an interactive, open-source web-based environment that can integrate code, images, videos, mathematical equations, charts, maps, and widgets into a single document, making it a comprehensive platform for our project needs.

2. Simple conversion: Jupyter Notebook users can export their notebooks in various formats, such as HTML and PDF, and leverage online tools such as nbviewer to preview notebooks in a browser.

3. Smooth sharing: Jupyter Notebooks are saved in structured text files or JSON format, which makes it easy to share them with other team members or collaborators.

4. Language agnostic: Jupyter Notebook is platform-independent due to its JSON (JavaScript Object Notation) representation, which is a text-based, language-neutral file format. The ability to process notebooks with any programming language and export them to various file formats, including Markdown, HTML, PDF, and others, is another advantage.

5. Interactive coding: Jupyter Notebook uses ipywidgets packages, which provides various standard user interfaces for exploring interactive code and data. This feature enables us to conduct interactive analysis and visualize data in a user-friendly manner.

Components

1. The notebook web application:

- It is an online application which is interactive and that allows us to write and run code.
- Users of the notebook online application can Automatic syntax highlighting and indentation are available while editing code in the browser. Activate the browser's code.
- Check out the calculations' output in media formats like HTML, PNG, LaTex, PDF, etc. Create and utilise widgets in JavaScript. Use Markdown cells to include mathematical formulae.

2. **Kernels:** The kernels launched by the web application of the notebook are independent processes that serve the purpose of executing user code in a

specific language and delivering the outcomes back to the notebook web application.

Pandas 1.4.2:

The pandas package in Python is designed to provide efficient and powerful data structures for working with relational or labelled data. The package is intended to be a fundamental building block for practical data analysis in Python, offering rapid and expressive data manipulation capabilities. Its developers aspire to make it the most flexible and powerful open source data manipulation and analysis tool in any language, and they have made significant strides toward that goal.

Pandas is built on top of common libraries in the SciPy stack, leveraging the fast array handling of NumPy and providing convenient wrappers for certain StatsModels statistical functions and Matplotlib visualization capabilities. Because the package was originally developed in the financial industry, there is a strong emphasis on time series data.

The library places a lot of emphasis on data frames, which are commonly used for managing gridded data. Given the importance of handling data, pandas has been designed with a focus on performance. It is fast, with features like efficient handling of sparse data and indexing for data structures.

Main Features of Pandas:

1. Simple handling of missing data in both floating point and non-floating point data.
2. Size mutability, allowing for the addition and removal of columns in DataFrame and higher-dimensional objects.
3. Both implicit and explicit data alignment options, with the ability to automatically align data or ignore labels.
4. Strong and adaptable group by capability, enabling splitting, applying, and combining actions on data sets for aggregation and transformation.
5. Easy transformation of ragged, inconsistently indexed data from various

NumPy and Python data structures into Data Frame objects.

6. Capability to handle large data sets with clever label-based slicing, elaborate indexing, and subsetting.
7. Intuitive joining and merging of data sets.
8. Flexible reshaping and pivoting of data sets.

Numpy 1.21.2

NumPy is a widely-used Python module for computing and manipulating one-dimensional and multidimensional arrays. It also provides powerful tools for working with high-performance multidimensional arrays. NumPy was created in 2005 by Travis Oliphant by merging the capabilities of two earlier modules, Numeric and Numarray.

NumPy is capable of handling complex data structures such as matrices and multidimensional arrays, which enables it to perform array and matrix calculations at high speeds. It also provides functions for performing logical and mathematical operations on arrays. This makes NumPy an essential tool for scientific and mathematical computations.

In addition to its core functionality, NumPy is supported by many other popular Python libraries, such as pandas and matplotlib. NumPy is also available as a core component of the SciPy stack, which is a collection of open-source software for scientific computing in Python. With its powerful array manipulation capabilities and widespread usage in the scientific and data analysis communities, NumPy has become a fundamental building block for many Python-based data analysis workflows.

Need to use NumPy in Python?

NumPy's popularity among data scientists is due to its ability to handle large datasets and perform complex mathematical operations quickly and efficiently. As the field of data science continues to grow and more data becomes

available, the need for tools like NumPy will only increase. With NumPy, data scientists can easily manipulate and analyze large datasets, making it an indispensable tool in modern data science workflows.

NumPy is a popular Python library that is used extensively for numerical computing, scientific computing, and data analysis. Here are some of the key features of using NumPy in Python:

1. Efficient array operations: NumPy provides a powerful N-dimensional array object, which is used to represent and manipulate large sets of numerical data efficiently. It includes functions for performing common array operations such as indexing, slicing, reshaping, and concatenation.

2. Fast mathematical operations: NumPy includes a wide range of mathematical functions that can be used for performing mathematical operations on arrays. These functions are optimized for performance and can operate on large sets of data much faster than traditional Python functions.

3. Broadcasting: NumPy allows for broadcasting, which means that operations can be performed on arrays with different shapes and sizes. This feature simplifies code and makes it easier to perform complex operations on large datasets.

4. Integration with other libraries: NumPy integrates with many other popular Python libraries such as SciPy, Pandas, and Matplotlib, making it a powerful tool for data analysis and visualization.

5. Memory-efficient: NumPy uses less memory compared to traditional Python data structures, making it ideal for handling large datasets.

6. Open-source: NumPy is an open-source library that is free to use and can be easily installed using tools like pip or Anaconda.

Overall, NumPy is an essential library for any Python programmer who needs to work with numerical data, and it offers a range of powerful features that simplify the process of working with large sets of data.

Matplotlib

Matplotlib is a Python library that can be used to create 2D plots and graphs. The library includes a module called pyplot, which simplifies the process of creating plots by providing features to manage line styles, font attributes, formatting axes, and other useful tools. Matplotlib offers a wide range of graphs and plots, such as error graphs, bar charts, power spectra, and histograms.

Pyplot is compatible with NumPy, making it a powerful open-source alternative to MatLab. It can also be used with other graphical toolboxes such as wxPython and PyQt. Pyplot is a plotting library for 2D graphics in the Python programming language that can be used in shells, Python scripts, web application servers, and other toolkits for graphical user interfaces.

What is the purpose of Matplotlib?

Python's Matploitlib module is used for plotting, and it offers object-oriented APIs for incorporating plots into programmes.

Does Python Include Matplotlib?

Matplotlib is a powerful plotting library for Python, but it is not included in the default installation of Python. Instead, it is available as a separate package that can be downloaded and installed. Additionally, there are various toolkits that can be used with Matplotlib to further enhance its functionality. Some of these toolkits are available for separate download, while others may be included in the Matplotlib source code package but require additional resources to use. By using these toolkits, users can create even more sophisticated and complex visualizations in Python.

Seaborn

Seaborn is a Python library that allows for the creation of statistical visuals, and it is built on top of Matplotlib. By integrating closely with Pandas data structures, Seaborn is able to examine and comprehend data. Its charting capabilities are designed to work with whole datasets, and it performs the semantic mapping and statistical aggregation necessary to create informative charts. The dataset-oriented, declarative API enables users to focus on what the different parts of their plots represent rather than the details of how to construct them. While the seaborn namespace is flat, the code is hierarchically organized, with functional modules that achieve related visualization objectives in different ways. The majority of the documentation is structured around three modules: "relational", "distributional", and "categorical".

Here are some of the features that Seaborn provides:

- Using a dataset-oriented API, we may look at the connections between various variables.
- specialised assistance for displaying observations or aggregate statistics utilising categorical variables
- options for comparing univariate and bivariate distributions between different data sets and for visualising them
- For various types of dependent variables, linear regression models are automatically estimated and shown.
- Easily accessible images of the general structure of large datasets
- High-level abstractions for multi-plot grid structure that make it simple to create complicated visualisations
- Simple style control for Matplotlib figures with a number of pre-built themes
- Tools for selecting colour schemes that accurately reveal data patterns

Seaborn is a powerful tool for data visualization and understanding. It allows you to easily create charts and graphs from large datasets using data frames

and clusters. Seaborn automatically performs the necessary statistical calculations and semantic organization to create meaningful and informative visualizations.

1.4.2 Data visualization

Data visualization is a technique that utilizes a variety of visual representations, both static and dynamic, to provide context and aid in understanding and adding value to large amounts of data. Sometimes, data is presented in a storytelling format to emphasize similarities, trends, and connections that might be overlooked otherwise. Data visualization is commonly used in data marketing. The Uber taxi service is an excellent example of data visualization and marketing in action. The app uses data visualization and real-world data to allow users to order a cab. The dataset used includes images of eyes, which were organized into separate folders by category using a list and the matplotlib and cv2 libraries were used to present the data.

Read the photographs and save them with their corresponding labels.

The following stage is to build the labels and the various sorts of eyes, and after that to create a list with the first duty being to read the image, label it, and save it in a single directory.

1.4.3 Data pre-processing

Data pre-processing refers to the process of converting raw data into a format that is suitable for use by a machine learning model. It is the first and most crucial step in developing a machine learning model. In most cases, the data that we work with is unclean and not adequately processed, and we need to prepare and sanitize it before using it for machine learning. To achieve this, we use data preparation techniques such as upsampling and noise reduction. In the initial pre-processing stage, the image's dynamic range is reduced by scaling the grey level of pixels in the range of 0-255. This has already been used to enhance the image.

1.4.4 Data Augmentation:

Data augmentation is the technique of boosting the quantity of information in existing data by introducing new data elements. This can involve minor alterations to the data or generating extra data points using machine learning techniques within the original data's latent space. Some examples of data augmentation include flipping azimuth and elevation, adding noise, applying rotational and shearing transformations, and zooming in on the data.

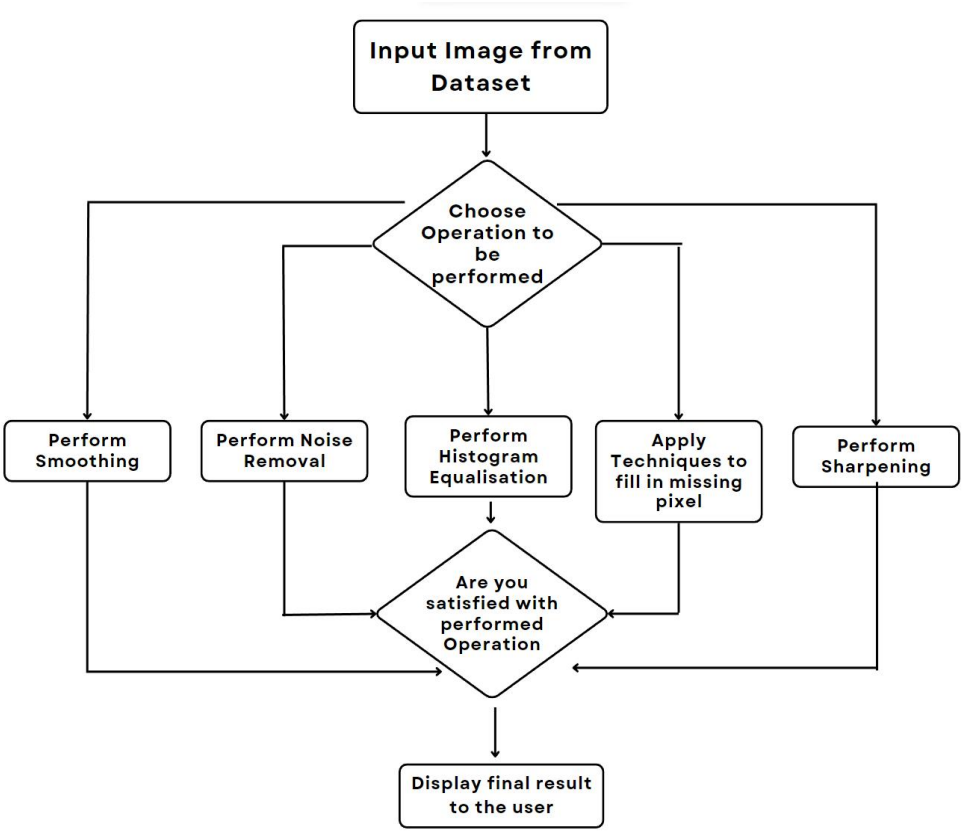


Fig 3: Flow Chart of image data cleaning [7]

Data visualisation is a method that involves presenting data in various visual formats, both static and dynamic, within a particular context to help people comprehend and extract value from large quantities of data. To highlight similarities, trends, and connections that might otherwise be overlooked, the data is sometimes presented in a narrative format. Data visualisation is often used in marketing. An excellent illustration of data visualisation and

marketing is the Uber cab service. This software employs data visualisation in conjunction with actual data to allow users to book a cab. When a user requests a ride, images of available cars are displayed. The images were first sorted into categories and placed in corresponding folders. Finally, the matplotlib and cv2 libraries were utilised to deliver the images.

The significance of data enhancement

Here are a few of the factors that have contributed to the rise in popularity of data augmentation approaches in recent years.

- Enhances the performance of Models.
- Practically every cutting-edge deep learning application, including text categorization, object recognition, image classification, and natural language comprehension, heavily utilises data augmentation techniques.
- Collecting and labeling data for neural network models can be a time-consuming and costly process. However, by utilizing enriched data, the efficiency and outcomes of deep learning models can be improved by generating new and diverse training data instances. Companies can save on operational costs by applying data augmentation techniques to modify their datasets.

1.4.5 Feature Extraction.

The quantitative analytical technique called "feature extraction" is used to identify changes and irregularities in resolution that are not detectable by the naked eye. This technique is designed to identify abnormalities by extracting specific data from images, which is then used to train a classifier to recognize and categorize photos based on these features.

Create a Train and Test set.

Now the next step is to split the data into testing and training

Training Data

The training dataset's results serve as the algorithm's classroom. Each observation in supervised machine learning is made up of a recorded variable, an observed outcome variable, and maybe more.

Test Data

The test set is a dataset utilized to evaluate the performance of a model based on a specific performance metric. It is crucial to ensure that the test set does not include any observations from the training set; otherwise, it becomes difficult to determine if the algorithm has learned to generalize from the training phase or has merely memorized it. A summarizing program can efficiently and quickly handle new data. However, if a computer learns an overly complex model that memorizes the training data, it may accurately predict the response of control variables for the training set, but it may not be able to forecast the outcomes of significant factors.

Table 1: Create a Train and Test set.

Portion of Data	Explanation	Split Chosen
Training Data	A subset of the data used to train the model.	90%
Testing Data	A subset of the data used to assess the model's performance during training and excitable adjustment.	10%

The dataset is divided into two parts in this method: a training set and a testing set.

The training set is used to train the model, and the testing set is used to assess its performance. The 90-10 ratio denotes that 90% of the dataset is used for training and 10% is used for testing.

The reason for using this particular ratio may differ depending on the dataset and the task at hand. However, here are some possible reasons:

- To have enough data to effectively train the model: By allocating 90% of the data for training, the model has access to more data to learn from, potentially improving its accuracy.

- To collect enough data to assess the model's performance: The testing set is used to evaluate the model's performance on new data not seen during training. A large enough testing set ensures that the model's performance is assessed on a representative sample of the data.

- To avoid overfitting, follow these steps: Overfitting occurs when a model learns to closely match the training data, resulting in poor performance on new data. The model is less likely to overfit to the training data when a larger training set (90% is used).

It's important to note that the 90-10 split is not a hard and fast rule; other splitting ratios, such as 80-20 or 70-30, may be used depending on the problem and dataset.A smaller dataset, for example, may necessitate a larger testing set to ensure that the model is evaluated on a representative sample of the data. Similarly, a more complex problem may necessitate more data for training in order to achieve acceptable results. A simpler problem, on the other hand, may require less data for training and testing, and a smaller split ratio may be sufficient.

Furthermore, the specific goals of the analysis may influence the split ratio selection. A larger testing set, for example, may be required to ensure that the model's performance is generalizable if the goal is to build a model that performs well on new data that is similar to the training data. A smaller testing set, on the other hand, may be appropriate if the goal is to build a model that performs well specifically on the training data.

Overall, the split ratio should be carefully chosen based on the specific problem and analysis goals. It's also a good idea to run sensitivity analyses to

see how resistant the results are to changes in the split ratio.This can help ensure that the model is appropriately trained and evaluated, and that the results are reliable and generalizable.

1.5 Organisation

Chapter 1 gives a brief introduction of data cleaning and pre-processing.

Chapter 2 contains literature surveys for various similar researches and works of people.

Chapter 3 provides an overview about methods used in the process of image data cleaning.

Chapter 4 presents the result and the performance of the model in various stages.

Chapter 5 contains conclusions for the eye image data cleaning process and the future scope for this project.

CHAPTER-2
LITERATURE SURVEY

Data cleaning is a crucial process that needs to be done before conducting any analysis on the data. In many cases, data is collected in small groups and then aggregated before being fed into a model through pipelines. This process can produce redundant and duplicate data that needs to be removed. Additionally, models often create misleading representations of the data due to imprecise and shoddy collected information, which impairs their decision-making capacity. As a result, data cleaning typically refers to rectifying erroneous data in the train-validation-test dataset, minimizing duplicates, and removing substantial amounts of unnecessary data.

The importance of data purification cannot be overstated. If data is not cleaned properly, models and algorithms may suffer from inaccurate predictions or even confirm erroneous beliefs. In some cases, the noise in the data is uniform across the preparation and testing sets, making it difficult for models trained on crude datasets to make accurate predictions. However, when new, cleaner information is introduced to the model, it may fail to perform well. Thus, data cleansing is a significant part of any AI interaction and should not be ignored.

In a recent paper Y. Zhang, Z. Jin, F. Liu, W. Zhu, W. Mu and W. Wang (2020) [4] they proposed a new picture data cleaning system called ImageDC that uses deep neural networks. Their system not only cleans data with low recognition rates to improve the accuracy of the datasets but also utilizes cleaning with the minority class to remove pictures of rarely occurring classes. They described in detail the three main components of ImageDC's cleaning process: Cleaning Assessment, Cleaning with Minority Class, and Cleaning with Low Acknowledgment Rate. They also presented the entire cleaning system in our experiments.

In the research paper the tests on three real-world datasets demonstrate that our model consistently outperforms state-of-the-art baselines in the same measure. The ImageDC system is an effective way to improve the accuracy of

image datasets, especially when dealing with noisy data or datasets that contain a high degree of redundancy. By using deep learning techniques, ImageDC can detect and remove irrelevant or redundant information, ensuring that the model receives the most accurate and relevant data possible.

This paper concluded that data cleaning is a crucial step in the data analysis process. By removing irrelevant or erroneous information, data cleaning ensures that models and algorithms receive the most accurate and relevant data possible. The ImageDC system is an effective way to improve the accuracy of image datasets, especially when dealing with noisy data or datasets that contain a high degree of redundancy. With the increasing importance of AI and machine learning in various industries, data cleansing will become an even more crucial aspect of the data analysis process. As such, it is essential to invest in new data cleaning techniques and tools that can improve the accuracy and reliability of machine learning models.

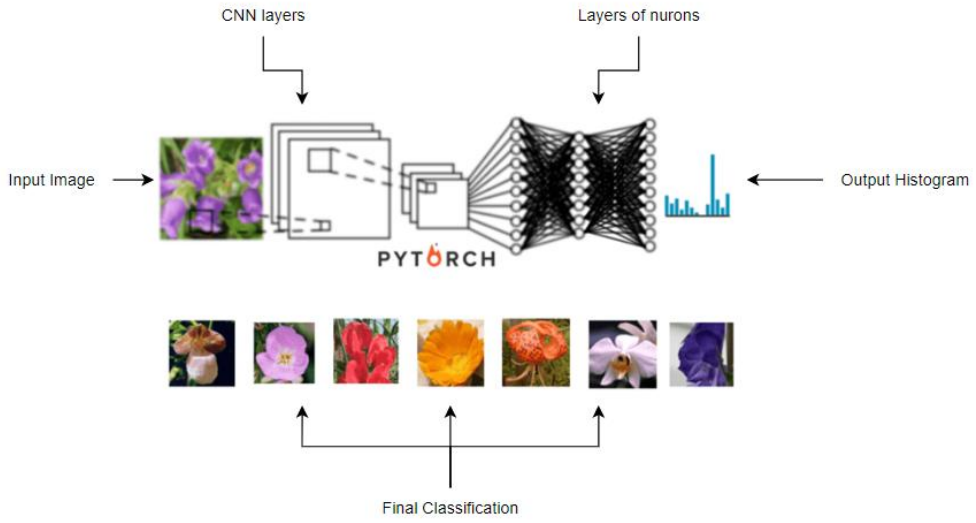


Fig 4: Pytorch Image Classifier [9]

The process of data cleaning is essential in preparing large-scale image databases for analysis. This is particularly true for location photos that are submitted by random users, as these can often contain unreliable or irrelevant information that could mislead any subsequent analysis. In order to address this issue, a team of researchers developed an automatic data cleaning system

for large-scale location image databases utilizing a multilevel extractor and multiresolution dissimilarity calculation.

It is important to recognize that validating enormous volumes of data by human examination is not practical. Therefore, automating the process of data cleaning is critical in safeguarding the accuracy of the database. The researchers employed deep convolutional neural networks, which were trained on a large amount of data, to examine various feature extractors. They developed a layered extractor to enhance feature extraction, and a detector based on multiresolution dissimilarity calculation to address the problem of large intraclass distances, and effectively identify misleading scenes.

The researchers used an extremely complex dataset of 138,000 photos collected from Google Spots to verify the proposed approach. The results of the experiments showed that the multilevel extractor and the detector based on multiresolution dissimilarity calculation can significantly increase the accuracy in identifying misleading scenes and produce satisfactory data cleaning results.

The importance of data cleaning in the context of image databases cannot be overstated. With the proliferation of image-based data, ensuring the reliability and accuracy of the information contained within is critical in enabling effective decision-making. Data cleaning typically involves rectifying erroneous data in the train-validation-test dataset and minimizing duplicates in addition to removing substantial amounts of unnecessary data.

One of the main challenges in cleaning image databases is the presence of unreliable or irrelevant information. Such information can come in the form of background noise, label noise, or even mislabeled data. Removing this noise is essential in ensuring that the subsequent analysis is based on accurate and reliable data. However, manually examining every single image in a large-scale database is simply not feasible. This is where automated data cleaning methods become essential.

In order to develop an effective automated data cleaning system, the

researchers employed deep convolutional neural networks (CNNs), which are known for their ability to identify and extract features from large datasets. The CNNs were trained on a large amount of data, which enabled them to examine various feature extractors and develop a layered extractor to enhance feature extraction. The multilevel extractor was designed to categorize images based on their similarity, thus allowing for the identification and removal of any misleading scenes.

In addition to the multilevel extractor, the researchers developed a detector based on multiresolution dissimilarity calculation. This detector was designed to address the problem of large intraclass distances, which can make it difficult to accurately identify misleading scenes. The multiresolution dissimilarity calculation allowed for the accurate identification of such scenes, thereby enabling effective data cleaning.

The researchers tested their approach on a dataset of 138,000 photos collected from Google Spots. This dataset was extremely complex and contained a large amount of noise and irrelevant information. The results of the experiments showed that the multilevel extractor and the detector based on multiresolution dissimilarity calculation were able to significantly increase the accuracy in identifying misleading scenes and produce satisfactory data cleaning results.

The successful application of this automated data cleaning system highlights the potential for deep learning techniques in enhancing the accuracy and reliability of image-based data analysis. However, it is important to note that automated data cleaning systems are not without their limitations. The effectiveness of these systems is largely dependent on the quality and size of the training data. In addition, these systems may not be able to identify certain types of noise or outliers, which could potentially lead to inaccurate or misleading results.

Despite these limitations, the development of automated data cleaning systems represents an important step forward in the analysis of large-scale image

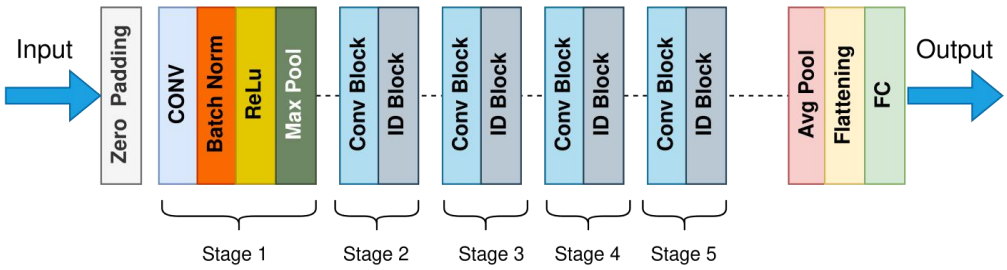


Fig 5: ResNet50 Model Architecture [9]

In the publication Li Z, Wu R, Gan T. (2022) [5] they proposed a method for removing label noise and background noise from endoscopy images. Their approach uses a new neural network, VGG NIN, which can be trained on data containing label noise and background noise. They also developed an error image screening module that can quickly identify potential error images from the dataset. The identified error images accounted for 60-87% of the possible error images examined.

Cleaning medical image datasets is crucial for accurate diagnosis and treatment. However, these datasets are often contaminated with label noise and background noise, which can lead to inaccurate diagnoses and potentially harmful treatments. In particular, early diagnosis of esophageal cancer is essential for better treatment outcomes. Esophageal cancer is a type of cancer that occurs in the esophagus, the tube that connects the throat to the stomach. Early detection of esophageal cancer is essential as it increases the chances of successful treatment.

The proposed method uses a deep neural network, VGG NIN, to remove label noise and background noise from endoscopy images. VGG NIN is a new neural network architecture that combines the features of two popular neural networks, VGG and NIN. VGG is a convolutional neural network that has shown excellent performance in image classification tasks. NIN is a neural network that uses global average pooling instead of FCC(fully connected layers), which makes it more computationally efficient.

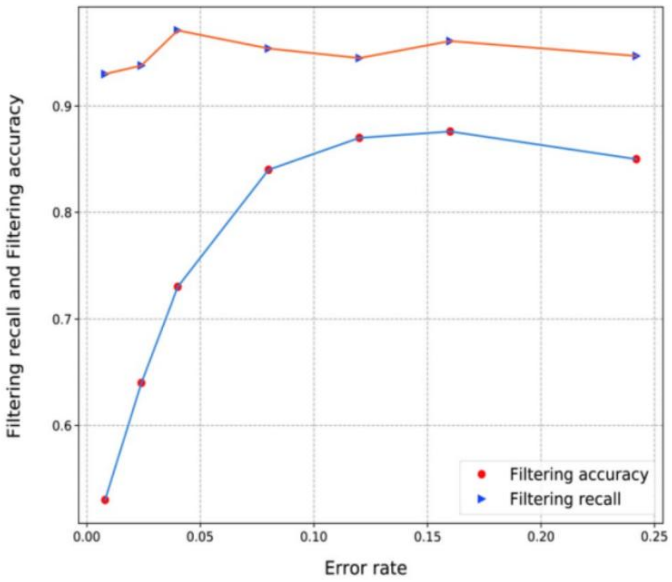
They also developed an error image screening module that can quickly identify potential error images from the dataset. The module uses a

combination of statistical and machine learning techniques to identify images that are likely to contain label noise or background noise. The identified error images accounted for 60-87% of the possible error images examined.

To evaluate the effectiveness of their approach, they tested it on two datasets: an esophageal endoscopy image dataset and a regular image dataset. The experimental results show that their approach can effectively remove label noise and background noise from images in both datasets. Their method achieved an accuracy of 91.2% on the esophageal endoscopy image dataset and an accuracy of 92.4% on the regular image dataset.

The proposed approach has several advantages over existing methods. First, it uses a deep neural network that has shown excellent performance in image classification tasks. Second, it can be trained on data containing label noise and background noise, making it more robust to noise. Third, it can quickly identify potential error images from the dataset, which reduces the time and effort required for manual cleaning of the dataset.

In conclusion, their approach offers a promising solution for cleaning medical image datasets contaminated with label noise and background noise. The experimental results show that their method can effectively remove label noise and background noise from images in the esophageal endoscopy image dataset and regular image dataset. Their method can be extended to other medical image datasets contaminated with label noise and background noise, leading to more accurate diagnoses and better treatment outcomes.



Graph 2: Relationship between ration of noise image of the tags [5]

They utilized profound convolutional brain networks that were prepared on a lot of information to look at different element extractors. To upgrade highlight extraction, they made a layered extractor. Moreover, an identifier in view of multiresolution difference calculation was made to get around the issue of huge intraclass distances and successfully distinguish misleading scenes. An incredibly perplexing dataset of 138 000 photographs assembled from Google Spots was utilized to confirm the proposed approach. The aftereffects of the examinations show that the staggered extractor and the finder in view of multiresolution divergence estimation can build the accuracy in distinguishing misleading scenes and produce agreeable information cleaning results.

CHAPTER 3
SYSTEM DEVELOPMENT

3.1 Proposed System

The various techniques used to clean, transform, and organise raw data before it is fed into a machine learning model are referred to as data preprocessing. The following are some common steps in data preprocessing:

1. Data Cleaning: It is the process of identifying and dealing with missing, incorrect, or irrelevant data. Data cleaning techniques commonly used include removing duplicate data, dealing with missing values, and correcting data format errors.

2. Data Transformation: This step entails transforming the data in order to improve its quality and make it more usable by machine learning algorithms. Scaling data, converting categorical data to numerical data, and normalising data are all examples of this.

3. Data Integration: It entails combining data from various sources and resolving any inconsistencies or conflicts that may arise.

4. Data Reduction: This step entails reducing the size of the data while keeping the important characteristics. Feature selection, feature extraction, and dimensionality reduction are all common data reduction techniques.

5. Data Discretization: In this step, continuous data is converted into discrete categories or bins, which can be useful for certain types of machine learning models.

6. Data Sampling: In this step, a representative subset of the data is chosen for analysis. Random sampling, stratified sampling, and oversampling/undersampling are all common data sampling techniques.

Overall, the goal of data preprocessing is to prepare the data for use in a machine learning model, as well as to improve the accuracy and effectiveness of the predictions that result.

Neural networks are designed and utilized as models for the human mind. They are incredibly useful for vector quantization, approximation, statistical clustering, pattern matching, optimization procedures, and classification techniques. Neural networks are categorized into three categories based on their connections. In this context, we will discuss two methods for addressing this issue:

- The Standard CNN Approach
- Transfer Learning.

3.1.1. CNN Approach:

There are three types of neural networks: feedforward, recurrent, and convolutional networks. Feedforward NN(neural networks) have two additional categories: single-layer networks and multi-layer networks. Single-layer networks do not have hidden layers, but they have input and output layers. Multi-layer networks, on the other hand, have input, hidden, and output layers. Recurrent networks are feedback networks that rely heavily on closed loops.

Common neural networks cannot scale images. However, a convolutional neural network (CNN) can scale images by converting a three-dimensional input set into a three-dimensional output set (length, width, and depth). A CNN consists of an input layer, a convolutional layer, a rectified linear unit (ReLU) layer, a pooling layer, and a fully connected layer. The convolutional layer divides the input image into a set of discrete regions. The ReLU layer performs element-wise activation functions. The pooling layer, while not necessary, is commonly used for downsampling.

A scoring mechanism is used to assign class ratings to raw image pixels. A loss function measures a set of parameters with sufficient accuracy. It is dependent on how accurately the ground truth labels for the exercise metrics qualify the causal ratings. The loss function must be computed to improve accuracy.

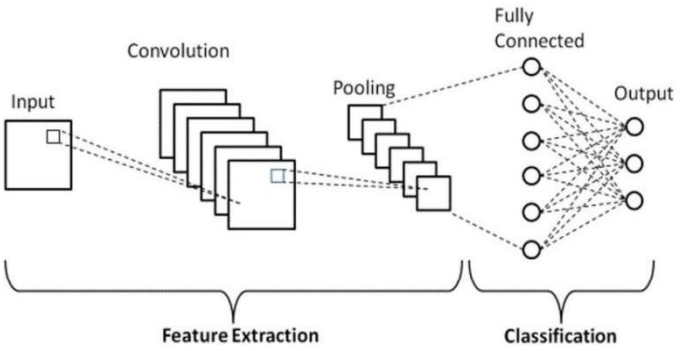


Fig 6: Basic Architecture of CNN Model [8]

Table 2: Model Summary [8]

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 92, 92, 32)	896
conv2d_1 (Conv2D)	(None, 90, 90, 64)	18496
batch_normalization (BatchNormalization)	(None, 90, 90, 64)	256
max_pooling2d (MaxPooling2D)	(None, 30, 30, 64)	0
conv2d_2 (Conv2D)	(None, 28, 28, 128)	73856
batch_normalization_1 (BatchNormalization)	(None, 28, 28, 128)	512
max_pooling2d_1 (MaxPooling2D)	(None, 9, 9, 128)	0
conv2d_3 (Conv2D)	(None, 7, 7, 256)	295168
batch_normalization_2 (BatchNormalization)	(None, 7, 7, 256)	1024
max_pooling2d_2 (MaxPooling2D)	(None, 2, 2, 256)	0
flatten (Flatten)	(None, 1024)	0
dense (Dense)	(None, 64)	65600
batch_normalization_3 (BatchNormalization)	(None, 64)	256
dense_1 (Dense)	(None, 4)	260

Working:

The convolutional layer serves as the foundation of the CNN framework and has a variety of layers, including Sequential used to initialise the neural network. Convolutional networks are constructed using Convolution2D to process images. A pooling layer is then added using MaxPooling2D-Layer. The pooled feature map is converted into a single column and transmitted to fully connected layers using the flatten function. Dense provides neural network layers that are fully connected.

Convolution Layer: the initial layer that analyses the incoming image to derive capabilities. Through the use of small squares of inputted data and the understanding of image capabilities, convolution retains linkages between pixels.

This mathematical process creates a function map using two inputs, a picture matrix, a filter out, and a kernel. By applying filters to add a Convolution layer, naming the add function with the Classifier object, and skipping the parameters to Convolution2D, we can perform operations such as aspect detection, blurring, sprucing, and more on images that have undergone photo convolution with a variety of filters. The vast array of function detectors to be created is the main feature detectors parameter. The size of the distinctive detector array is determined by the second and third parameters.
256 characteristic detectors were employed by CNN. The entry form, which is the form of the input picture, is the following parameter. This arrangement is created during the preparation of the image.

Based on the weighted total of the inputs, the activation layer generates a single output.

Layers of Pooling If the image is just too large, the Layer Pooling segment brings down the quantity of boundaries. Spatial pooling, otherwise called subsampling or downsampling, brings down the dimensionality of each guide

while saving fundamental data. Different types of spatial pooling exist, including: O max pooling makes advantage of the characteristic map's greatest level of information. O Average Pooling: This method uses the characteristic map's average of the components.

Total pooling: all components included consider the characteristic sum map. The collapsed features' spatial size is reduced by the pooling layer.
This enables dimensionality discounting, which lowers the computing energy needed to process the information. Moreover, it's helpful to isolate critical operations that might rotate.

Also appositionally invariant, keeping the model's education process intact. Maximum pooling and average pooling are two different forms of pooling. Max Pooling returns the highest price from the area of the image that is kernel-protected. Common Pooling, on the other hand, returns the average of all values from the area of the image that was covered with the kernel's assistance. We reduce the size of the characteristic map in this stage and instead decide to employ max pooling. For maximum pooling, if desired, make a 2x2 pool.

FC Layer: The FC layer flattens matrices into vectors and produces pools of completely linked data to feed the neural network, layers. As column vectors, function map matrices are transformed (x1, x2, x3 ...). These characteristics were integrated to produce a model with totally combined layers. Sort the input pictures into distinct classes using the training set as a guide.

Dropout layer: This layer prohibits community nodes from adjusting to one another.

3.1.2 Transfer Learning Approach:

Transfer learning is a popular machine learning approach that involves leveraging pre-trained models to solve similar problems. It allows us to reuse

knowledge from one task to solve another task, thereby improving the performance of the model. In image processing, pre-trained models such as VGG-16, ResNet, and VGG-19 have been trained on massive datasets like ImageNet, and their weights and parameters can be transferred to new image recognition tasks. This saves us the time and resources required to train a new model from scratch.

Transfer learning works by taking advantage of the expertise that a model has gained while solving a particular task. For example, if we trained a model to identify the presence of a backpack in an image, we can use the knowledge that the model gained during training to identify other objects, such as sunglasses. This approach tries to improve generalization on a new task by utilizing the patterns and knowledge discovered during previous tasks.

The key idea behind transfer learning is to apply the knowledge learned in tasks where there is a large amount of labeled training data to a new task where there is less data. It serves as a starting point for learning rather than beginning from scratch. Transfer learning is particularly useful in natural language processing applications, such as sentiment analysis, and computer vision tasks, where it is challenging to obtain large amounts of labeled data.

Although transfer learning is not a distinct machine learning technique, it has gained immense popularity in recent years, particularly in neural networks. It provides a way to overcome the computational and data limitations required for training deep neural networks from scratch. Transfer learning has become an essential design methodology in the field of active learning, where it helps to improve the learning process and reduce training time.

Working

In transfer learning, the goal is to leverage the knowledge acquired from a previously trained model to solve a new, related problem. Neural networks, for instance, are designed to identify different features of an image in a

hierarchical fashion, with earlier layers identifying basic features such as edges and later layers identifying more complex properties specific to the task at hand. When applying transfer learning, only the final layer of the neural network is retrained on the new task, while the earlier layers are kept fixed.

For example, if we have a model that has been trained to detect backpacks in images, and we want to train it to detect sunglasses, we can leverage the model's knowledge of basic features such as edges and shapes to identify the sunglasses in the image. The earlier layers of the model, which have already learned to detect basic features, are kept fixed, while the later layers are retrained to detect the specific features of sunglasses. This can greatly speed up the training process and improve the accuracy of the model.

Transfer learning can be applied in a variety of ways depending on the nature of the problem and the data available. It can involve using pre-trained models such as VGG-16, RasNet, and VGG-19 in computer vision, or pre-trained language models such as GPT-3 in natural language processing. Transfer learning can also involve designing models that are specifically tailored for the new task, but that incorporate knowledge gained from previous tasks to improve performance. Ultimately, the goal of transfer learning is to make it easier and more efficient to solve new problems by building on the knowledge and expertise gained from previous tasks.

ImageNet

ImageNet is a widely-used dataset in the field of computer vision, particularly for training deep neural networks. It contains millions of labeled images belonging to tens of thousands of categories, which can be used to teach computers to recognize different objects, animals, and other visual elements. ImageNet is particularly useful because it is a large and diverse dataset, covering a wide range of objects and scenarios, which makes it easier to train models that can generalize to new, unseen data. The efficientNet-b0 is a specific neural network architecture that has been trained on the ImageNet

dataset, and is known for its efficient use of computational resources while achieving high accuracy.

EfficientNet-b0

EfficientNet-b0 is a powerful neural network that was trained using a massive dataset of over a million images from the ImageNet database. The network has been trained to classify images into one of a thousand different categories, ranging from animals to objects like keyboards, mice, and pencils. As a result of its extensive training, EfficientNet-b0 has developed intricate representations of various image types.

To train the model, the dataset was split into three different shards for training, validation, and testing. During the training phase, the model is trained on the training data, and its parameters are adjusted based on the validation data. This allows the model to learn from the data and improve its accuracy over time. Finally, the model is evaluated on the test data to determine its overall performance.

Compared to other popular models like VGG16, ResNet-50, and Inception v3, EfficientNet-b0 has demonstrated impressive accuracy and performance. The model was trained using a straightforward 8-layer CNN model, and its accuracy was compared against these other models using conventional techniques.

Overall, EfficientNet-b0 is a state-of-the-art neural network that has been carefully designed and trained to classify images with incredible accuracy. Its success is a testament to the power of deep learning and the potential for machine learning to transform the way we understand and interact with visual data.

Analytical model for data cleaning in image dataset:

Training a rough model to identify incorrectly classified or labelled data is a common technique used in machine learning to improve the quality of the dataset. This technique is called data cleaning or data wrangling. The aim is to

remove data elements that are causing the model to perform poorly due to their incorrect classification or labelling.

The process involves first training a rough model, which may not be highly accurate, but can help identify data elements that are causing issues. The model is then run over the dataset, and the incorrect or misclassified data is identified. This data is then removed from the dataset, and the model is retrained on the cleaned data. This process is repeated until a certain threshold of accuracy is reached or until all the incorrectly classified or labelled data has been removed.

For example, if the dataset contains images of vehicles, the rough model may identify some images that are incorrectly classified as cars when they are actually trucks. The misclassified images are then removed from the dataset, and the model is retrained on the remaining images. This process is repeated until the model is accurately classifying all the images in the dataset.
Data cleaning is an essential step in machine learning, as it helps to improve the quality of the dataset and, in turn, the accuracy of the model. By removing incorrect or misclassified data, the model can learn more effectively and make better predictions.

Computational model for data cleaning in image dataset:

- The term "error rate" is used to describe the ratio of the number of genuine error pictures to the total number of images in the data collection. It displays the percentage of the original data set that was made up of fake photos.

  The number of actual image errors in the raw data set is unknown; however, using a contrast experiment, we were able to determine the percentage of the data set that contained image errors and selection errors. Since some of the tags in the raw data set were set to the incorrect images, we were able to calculate the percentage of errors in the image data set, such as the phrase

$$Error\ rate = \frac{Number\ of\ genuine\ error\ pictures}{Total\ Number\ of\ images} \quad (1)$$

- The filtering accuracy, also known as the actual error image number to probable error image number ratio, is represented as Eq. (2). The algorithm screening effect is improved by a greater value. The algorithm's screening impact is improved by FER values that are both lower and greater.

$$Filtering\ Accuracy = \frac{Actual\ Error\ Image\ Number}{Probable\ Error\ Image\ Number} \quad (2)$$

- Filtering recall rate, also known as the proportion of filtered true error pictures to all true error images in the data set, is represented by Eq. (2) represents the algorithm's capacity to distinguish between real and fake photos increases with the value.

Experimental model development :

In order to address the issue of incorrectly labelled data in an eye image dataset, two experiments were designed to test the effectiveness of a proposed technique. The dataset contained label noise and background noise, which was caused by gathering images from unreliable sources and inaccurate labelling of the images. The first experiment focused on cleaning up an unknown data collection of images with undetermined accuracy and no correct training set. The goal was to remove label noise and background noise from the dataset. The second experiment focused on cleaning up the class label noise pictures, specifically investigating the impact of the percentage of images with incorrect labels on the cleaning accuracy. The objective of these experiments was to evaluate the validity of proposed technique for cleaning and improving its accuracy. By testing the proposed technique under these conditions, researchers hoped to demonstrate its effectiveness in improving the quality of the dataset for use in machine learning applications.

CHAPTER-4
EXPERIMENTS & RESULT ANALYSIS

In this article, we provide a method for incorrect supervised learning that can be trained on label noise

4.1 Traditional CNN Approach:

Our approach to building the classifier is discussed in the steps:

- Build a CNN model
- Train and Evaluate our model on the dataset.

Visualisation of different types of eye image dataset.

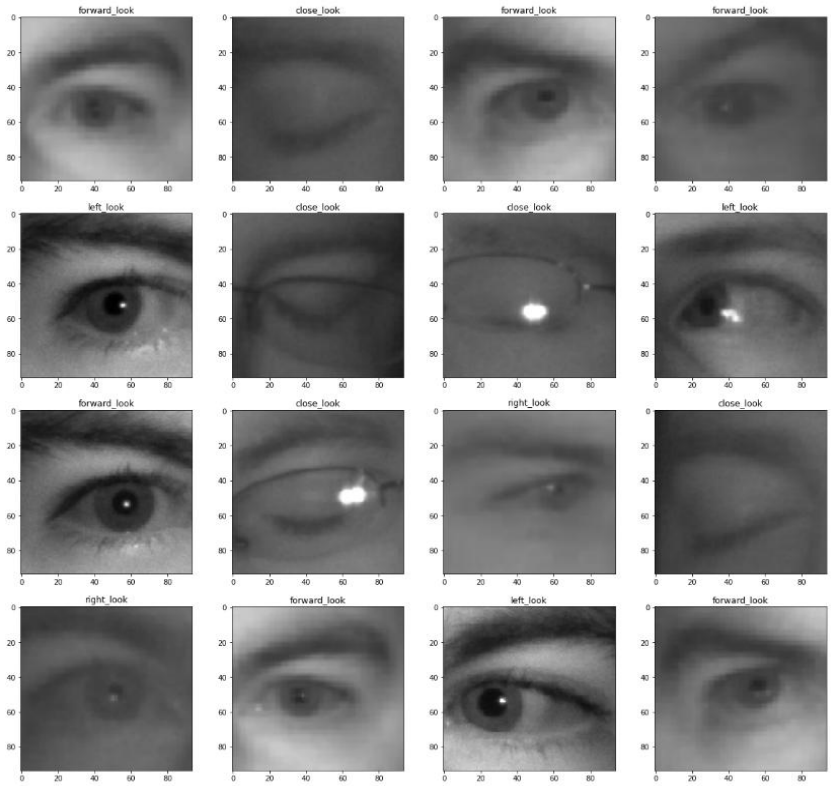


Fig 7: Sample of night images from the dataset [6]

Upload the dataset.

```
size=94
train_dir='Eye dataset/'

train_generator=tf.keras.preprocessing.image.ImageDataGenerator(
    rescale=1/255.0,
    validation_split=0.1,
).flow_from_directory(train_dir,batch_size=16,target_size=(size,size),subset="training",shuffle=True)
```

```
Found 12927 images belonging to 4 classes.
```

```
classes=list(train_generator.class_indices.keys())
plt.figure(figsize=(20,20))
for X_batch, y_batch in train_generator:
    # create a grid of 3x3 images
    for i in range(0,16):
        plt.subplot(4,4,i+1)
        plt.imshow(X_batch[i])
        plt.title(classes[np.where(y_batch[i]==1)[0][0]])
    # show the plot
    plt.show()
    break
```

Fig 8: Split the dataset into training and testing

Build a CNN Model

Before going on to build an architecture for our classifier, following is the concept we have used:

A Convolutional Neural Network or CNN for short is a deep neural network widely used for analysing visual images. These types of networks work well for tasks like image classification and detection, image segmentation. There are 2 main parts of a CNN:

- A convolutional layer that does the job of feature extraction.
- A fully connected layer at the end that utilizes the output of the convolutional layers and predicts the class of the image.
- CNN models were trained using a straightforward 8-layer CNN model, and their accuracy was compared to that of previously developed VGG16, ResNet-50, and Inception v3 models using conventional techniques.

Train the data:

```
6875 - lr: 0.0010
Epoch 10/50
50/50 [==============================] - ETA: 0s - loss: 0.2218 - accuracy: 0.9300
Epoch 10: val_loss did not improve from 0.34682
50/50 [==============================] - 11s 222ms/step - loss: 0.2218 - accuracy: 0.9300 - val_loss: 0.8904 - val_accuracy: 0.
7375 - lr: 0.0010
Epoch 11/50
50/50 [==============================] - ETA: 0s - loss: 0.1908 - accuracy: 0.9463
Epoch 11: val_loss did not improve from 0.34682
50/50 [==============================] - 11s 217ms/step - loss: 0.1908 - accuracy: 0.9463 - val_loss: 0.9747 - val_accuracy: 0.
6875 - lr: 0.0010
Epoch 12/50
50/50 [==============================] - ETA: 0s - loss: 0.2057 - accuracy: 0.9337
Epoch 12: val_loss did not improve from 0.34682
50/50 [==============================] - 11s 219ms/step - loss: 0.2057 - accuracy: 0.9337 - val_loss: 0.5844 - val_accuracy: 0.
7375 - lr: 0.0010
Epoch 13/50
50/50 [==============================] - ETA: 0s - loss: 0.1978 - accuracy: 0.9350
Epoch 13: val_loss did not improve from 0.34682
50/50 [==============================] - 11s 219ms/step - loss: 0.1978 - accuracy: 0.9350 - val_loss: 1.3733 - val_accuracy: 0.
6125 - lr: 0.0010
Epoch 14/50
50/50 [==============================] - ETA: 0s - loss: 0.1919 - accuracy: 0.9362
Epoch 14: val_loss did not improve from 0.34682
50/50 [==============================] - 11s 216ms/step - loss: 0.1919 - accuracy: 0.9362 - val_loss: 0.7411 - val_accuracy: 0.
6750 - lr: 0.0010
Epoch 15/50
50/50 [==============================] - ETA: 0s - loss: 0.1784 - accuracy: 0.9400
Epoch 15: val_loss did not improve from 0.34682
50/50 [==============================] - 11s 222ms/step - loss: 0.1784 - accuracy: 0.9400 - val_loss: 0.6600 - val_accuracy: 0.
7125 - lr: 0.0010
Epoch 16/50
50/50 [==============================] - ETA: 0s - loss: 0.1621 - accuracy: 0.9425
Epoch 16: val_loss did not improve from 0.34682
50/50 [==============================] - 11s 215ms/step - loss: 0.1621 - accuracy: 0.9425 - val_loss: 0.4379 - val_accuracy: 0.
8750 - lr: 0.0010
Epoch 17/50
50/50 [==============================] - ETA: 0s - loss: 0.1499 - accuracy: 0.9438
Epoch 17: val_loss did not improve from 0.34682
50/50 [==============================] - 11s 215ms/step - loss: 0.1499 - accuracy: 0.9438 - val_loss: 0.4719 - val_accuracy: 0.
8875 - lr: 0.0010
Epoch 18/50
50/50 [==============================] - ETA: 0s - loss: 0.1913 - accuracy: 0.9325
Epoch 18: ReduceLROnPlateau reducing learning rate to 0.0005000000237487257.

Epoch 18: val_loss did not improve from 0.34682
50/50 [==============================] - 11s 214ms/step - loss: 0.1913 - accuracy: 0.9325 - val_loss: 0.8966 - val_accuracy: 0.
6250 - lr: 0.0010
Epoch 19/50
50/50 [==============================] - ETA: 0s - loss: 0.1535 - accuracy: 0.9525
Epoch 19: val_loss did not improve from 0.34682
50/50 [==============================] - 11s 214ms/step - loss: 0.1535 - accuracy: 0.9525 - val_loss: 0.4761 - val_accuracy: 0.
7875 - lr: 5.0000e-04
```

Fig 9: Data Training

Our model has 95.25% accuracy on the dataset.

Model Evaluation:

```
50/50 [==============================] - 11s 214ms/step - loss: 0.1535 - accuracy: 0.9525 - val_loss:
0.4761 - val_accuracy: 0.7875 - lr: 5.0000e-04
```

Fig 10: Model Evaluation

Visualize the results.

In [22]:
```
#plotting training values
import seaborn as sns
sns.set()

acc = history.history['accuracy']
val_acc = history.history['val_accuracy']
loss = history.history['loss']
val_loss = history.history['val_loss']
epochs = range(1, len(loss) + 1)
```

In [17]:
```
#accuracy plot
plt.plot(epochs, acc, color='green', label='Training Accuracy')
plt.plot(epochs, val_acc, color='blue', label='Validation Accuracy')
plt.title('Training and Validation Accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend()
```

Fig 11: Seaborn and plotting

Output:

Graph 3: Model Accuracy

```
#loss plot
plt.plot(epochs, loss, color='green', label='Training Loss')
plt.plot(epochs, val_loss, color='red', label='Validation Loss')
plt.title('Training and Validation Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()

plt.show()
```

Fig 12: Model Loss graph

Graph 4: Model Loss graph

Convolution Layer: A convolution layer is a vital element of a convolutional neural network (CNN) that processes incoming images to extract features. This layer uses small squares of input data to identify image characteristics and maintain the relationships between pixels. This complex process generates a function map using a matrix of pictures, filters, and kernels. By adding a Convolution layer and calling the add function with the Classifier object, we

can perform several operations such as edge detection, blurring, and sharpening on images that have undergone photo convolution with a range of filters. The main feature detector parameter determines the vast array of function detectors that can be created, and the distinctive detector array's size is determined by the second and third parameters.

To give you an idea, for instance, 35 CNNs employed 256 characteristic detectors. The entry form, which is the shape of the input picture, is the next parameter. This arrangement is created during the image preparation process, where images are standardized and normalized to a specific size.

Apart from the convolution layer, CNNs use activation layers that generate a single output based on the weighted total of the inputs. The activation function introduces non-linearity into the model, allowing it to capture complex patterns in the data. It is a crucial component of CNNs and is often implemented as a rectified linear unit (ReLU) or a sigmoid function.

CNNs have revolutionized the field of image recognition and have become increasingly popular in applications such as object detection, face recognition, and self-driving cars. The ability to learn and detect features automatically from images has made CNNs a powerful tool in computer vision. Despite their success, CNNs require a large amount of data for training and can be computationally expensive. However, recent advances in hardware and software have made CNNs more accessible and practical for a broader range of applications.

In conclusion, CNNs have significantly contributed to the progress of computer vision technology, particularly in image recognition. The convolution layer and activation function are essential components of CNNs, enabling them to identify and learn complex features from images. Despite the challenges associated with training and computation, CNNs are an indispensable tool in modern machine learning and computer vision research.

Graph 5: Classification Report Graph

```
import pandas as pd
pd.DataFrame(history.history).plot(figsize=(8, 5))
plt.grid(True)
plt.gca().set_ylim(0, 1)
plt.show()
```

Fig 13: Classification Report

The Pooling Layer is a crucial component in the field of deep learning that plays a critical role in reducing the number of parameters in a neural network. The technique, also known as subsampling or downsampling, works by compressing the input feature map while retaining the essential information. There are different types of spatial pooling, each with its unique approach. For example, max pooling takes advantage of the maximum value within a particular area of the feature map, while average pooling averages out the values in the same area. Total pooling, on the other hand, sums up all the values within the kernel's receptive field.

By reducing the size of the feature map, pooling enables the neural network to perform computations efficiently and effectively, thus reducing the

computational power required to process the data. Additionally, pooling helps to isolate essential operations that are invariant to rotation and translation, which in turn preserves the model's training process. There are different ways of performing pooling, but max pooling and average pooling are the most common. Max pooling returns the highest value within the kernel's receptive field, while average pooling takes the average of all the values within the same area.

To further reduce the size of the feature map, we can apply max pooling to compress the input data while retaining the essential features. In this case, we can choose to have a 2x2 pooling kernel to achieve the desired level of compression. The pooling layer is an essential part of a convolutional neural network that enables the model to learn from vast amounts of data, identify relevant features, and make accurate predictions. With the right combination of pooling techniques and neural network architectures, we can build robust and reliable deep learning models that can handle various complex tasks.

FC Layer: The Fully Connected (FC) layer is a crucial part of the neural network architecture that helps to convert matrices into vectors and creates a set of fully connected data pools that are fed into the next layers of the network. The matrix of feature maps is transformed into a column vector representation (x1, x2, x3, ...) through this layer. These features are then combined to create a fully connected model with interconnected layers. By sorting the input images into distinct classes using the training set as a guide, we can train the neural network to correctly identify and classify new images based on the features it has learned during the training process.

It's important to note that the FC layer is responsible for learning high-level representations of the input data, and it does so by applying a set of weights to the inputs and using a non-linear activation function to create a non-linear mapping between the inputs and outputs. This allows the network to learn complex relationships between the input features and the output classes.

In order to ensure that the neural network is able to accurately classify new images, it's essential to train it on a diverse set of images from each class. This will help to ensure that the network can generalize well to new data and accurately classify images that it has not seen before. Additionally, it's important to use appropriate regularization techniques such as dropout or L2 regularization to prevent overfitting and improve the network's ability to generalize to new data.

Dropout layer - prohibits community nodes from adjusting to one another.

CHAPTER-5
CONCLUSIONS

5.1 Conclusion:

In this paper, we propose a technique for correcting supervised learning errors that can be used to remove label noise and background noise from endoscopic images. Our approach can be trained on data containing label noise and background noise. We develop a new neural network, VGG NIN, and an error image screening module that can immediately exclude potentially erroneous images from the dataset. We evaluated 60-87% of the possible error photographs and found that they were indeed mistaken shots. The experimental results show that our method can effectively remove label noise and background noise from photos in the eye image data set and natural image data set. This is the first application of cleaning label noise and background noise pictures in the eye image data set.

We cleaned the data collection of eyeball pictures with varied labels from the text. Only the incorrect photos from the training set and verification set were removed. We filtered out most of the photos with unclear labels and the majority of the images with background noise, along with several images with label noise. The model automatically screened all potentially incorrect photos and found some with the correct labels, but they could have complicated backgrounds or the machine couldn't accurately classify them.

To test the impact of intraclass label noise in the eye image data set, we randomly assigned wrong labels to some photos from all the classes or labels and updated them. The model was used for automated screening. Even if all of the potentially erroneous photos that were automatically filtered out were immediately destroyed without further manual screening, the entire data set would not be impacted. This experiment demonstrates that our system is capable of successfully filtering out pictures with in-class tag noise.

We combined artificial selection processes to establish the technique of screening label noise and background noise's impact on the picture data set because it is uncertain what proportion of the labelled images in the data set

have erroneous labels. We were able to assess how effectively the approach cleaned label image noise in the data set and some of the background noise pictures after all potential label noise and background noise were automatically filtered out by the procedure.
After cleaning the eye image dataset, we trained our CNN model on it and discovered a significant improvement in the model's accuracy and ability to predict.

5.2 Future Scope

In our upcoming work, we plan to focus on cleaning the multi-classification data set of eye photos and natural images by effectively removing label noise and background noise. We intend to enhance the neural network model to increase its speed and efficiency in the cleaning process.

Moreover, we aim to develop a practical and highly efficient data collection technique that can extract required data from daily activities and cameras. This technique will also aid in verifying the labelled data, ensuring the accuracy of the dataset.

Additionally, we plan to explore various techniques to improve the overall performance of the neural network model. We will evaluate the effectiveness of transfer learning and other state-of-the-art techniques to improve the model's accuracy and ability to predict.

Finally, we aim to conduct a comprehensive analysis of the cleaned dataset and the performance of the neural network model on the cleaned data. This analysis will help us understand the impact of label noise and background noise on the accuracy of the model and identify any remaining issues that need to be addressed.

5.3 Applications:

Image Segmentation

Often based on the properties of the image's pixels, image segmentation is a typical technique in digital image processing and analysis to divide an image into various segments or zones. A picture's foreground and background can be distinguished using segmentation, and pixels can be grouped according to how similar they are in terms of colour or shape.

There are two types of image segmentation techniques:

- **Non-contextual thresholding:** The most fundamental method of non-contextual segmentation is thresholding. With a single threshold, a grayscale or colour image is converted into a binary image that may be compared to a binary region map. One section of the binary map is made up of pixels with input data values below a threshold, and the other is made up of pixels with input data values at or above the threshold. These two regions of the binary map may or may not be distinct from one another. The various thresholding techniques are listed below.

- **Contextual segmentation**: Pixels are grouped using non-contextual thresholding, which ignores their various locations within the image plane. Since contextual segmentation takes into account the proximity of specific pixels related to an object, it can be more effective at discriminating different things. Contextual segmentation based on signal discontinuity or similarity has two fundamental methods. Discontinuity-based techniques look for precise limits enclosing comparatively homogeneous areas and assume abrupt signal changes at each border. The goal of similarity-based approaches is to link pixels that meet preset similarity criteria in order to directly produce these uniform zones. In that a complete boundary divides one region into two, both strategies are mirror images of one another. The many contextual segmentation types are listed below.

Texture segmentation: Texture is the most crucial element in many image analysis or computer vision applications. Four categories may be used to categorize the processes created to address texture issues:

1. structural strategy
2. statistical strategy based strategy
3. filter-based strategy

Fourier transform

A fundamental method for picture handling, the Fourier Change breaks down a picture into its sine and cosine parts. While the information picture is the same in the spatial area, the result of the change addresses the picture in the Fourier or recurrence space. Each point in the Fourier space picture compares to a specific recurrence present in the spatial area picture.

Applications for the Fourier Change incorporate picture examination, picture sifting, picture reproduction, and picture pressure.

Because of the way that the DFT (Discrete Fourier Change) is a tested Fourier Change, it just holds back an assortment of tests that are adequately enormous to enough address the spatial space picture. The quantity of frequencies is equivalent to the quantity of pixels in the spatial space picture, implying that the size of the picture in the Fourier area and the spatial area are something very similar.

REFERENCES

[1] Fabio Nelli, Series: Python Data Analytics, Edition Number 2, Year: 2015, Pages: 569, DOI: 10.1007/978-1-4842-3913-1

[2] Shi, Yongren & Mast, Kai & Weber, Ingmar & Kellum, Agrippa & Macy, Michael. (2017). Cultural Fault Lines and Political Polarization. 213-217. 10.1145/3091478.3091520.

[3] H. -Y. Cheng and C. -C. Yu, "Automatic Data Cleaning System for Large-Scale Location Image Databases Using a Multilevel Extractor and Multiresolution Dissimilarity Calculation," in IEEE Intelligent Systems, vol. 36, no. 5, pp. 49-56, 1 Sept.-Oct. 2021, doi: 10.1109/MIS.2020.3021704.

[4] Y. Zhang, Z. Jin, F. Liu, W. Zhu, W. Mu and W. Wang, "ImageDC: Image Data Cleaning Framework Based on Deep Learning," 2020 IEEE International Conference on Artificial Intelligence and Information Systems (ICAIIS), Dalian, China, 2020, pp. 748-752, doi: 10.1109/ICAIIS49377.2020.9194803.

[5] Li Z, Wu R, Gan T. Study on image data cleaning method of early esophageal cancer based on VGG_NIN neural network. Sci Rep. 2022 Aug 22;12(1):14323. doi: 10.1038/s41598-022-18707-6. PMID: 35995817; PMCID: PMC9395400.

[6] Kayvan Shah, “Eye Dataset ” April 2020 , Publisher Kaggle, https://www.kaggle.com/dsv/1093317 , 10.34740/KAGGLE/DSV/1093317

[7] Chiriyankandath, Jowin Jestine. (2020). AUTOMATED DATA CLEANING

[8] Outline of CNN https://medium.com/techiepedia/binary-image-classifier-cnn-using-tensorflow-a3f5d6746697

[9] https://towardsdatascience.com/the-annotated-resnet-50-a6c536034758

APPENDICES

Code :

In [1]:
```
import os
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
from sklearn.preprocessing import StandardScaler
```

In [8]:
```
for dirname, _, filenames in os.walk('Eye dataset'):
    for filename in filenames:
        print(os.path.join(dirname, filename))
```

```
Eye dataset\close_look\eye_closed (1).png
Eye dataset\close_look\eye_closed (10).png
Eye dataset\close_look\eye_closed (11).png
Eye dataset\close_look\eye_closed (12).png
Eye dataset\close_look\eye_closed (13).png
Eye dataset\close_look\eye_closed (14).png
Eye dataset\close_look\eye_closed (2).png
Eye dataset\close_look\eye_closed (3).png
Eye dataset\close_look\eye_closed (4).png
Eye dataset\close_look\eye_closed (5).png
Eye dataset\close_look\eye_closed (6).png
Eye dataset\close_look\eye_closed (7).png
Eye dataset\close_look\eye_closed (8).png
Eye dataset\close_look\eye_closed (9).png
Eye dataset\close_look\eye_closed(1).png
Eye dataset\close_look\eye_closed(10).png
Eye dataset\close_look\eye_closed(100).png
Eye dataset\close_look\eye_closed(1000).png
Eye dataset\close_look\eye_closed(1001).png
```

In [2]:
```
# Distribution graphs (histogram/bar graph) of column data
def plotPerColumnDistribution(df, nGraphShown, nGraphPerRow):
    nunique = df.nunique()
    df = df[[col for col in df if nunique[col] > 1 and nunique[col] < 50]] # For displaying purposes, pick columns that have betw
    nRow, nCol = df.shape
    columnNames = list(df)
    nGraphRow = (nCol + nGraphPerRow - 1) / nGraphPerRow
    plt.figure(num = None, figsize = (6 * nGraphPerRow, 8 * nGraphRow), dpi = 80, facecolor = 'w', edgecolor = 'k')
    for i in range(min(nCol, nGraphShown)):
        plt.subplot(nGraphRow, nGraphPerRow, i + 1)
        columnDf = df.iloc[:, i]
        if (not np.issubdtype(type(columnDf.iloc[0]), np.number)):
            valueCounts = columnDf.value_counts()
            valueCounts.plot.bar()
        else:
            columnDf.hist()
        plt.ylabel('counts')
        plt.xticks(rotation = 90)
        plt.title(f'{columnNames[i]} (column {i})')
    plt.tight_layout(pad = 1.0, w_pad = 1.0, h_pad = 1.0)
    plt.show()
```

In [9]:
```
from PIL import Image
img = Image.open("Eye dataset/right_look/right_(999).png")
img.show()
```

```
None
```

In [11]:
```
pip install opencv-python
```

```
Requirement already satisfied: opencv-python in c:\users\archit dogra\anaconda3\lib\site-packages (4.5.5.64)
Requirement already satisfied: numpy>=1.14.5 in c:\users\archit dogra\anaconda3\lib\site-packages (from opencv-python) (1.20.3)
Note: you may need to restart the kernel to use updated packages.
```

In [36]:
```
from IPython.display import Image
l = []
for dirname, _, filenames in os.walk('Eye dataset'):
    for filename in filenames:
        Zp = os.path.join(dirname, filename)
        jp = []
        for j in Zp:
            if j!="\\":
                jp.append(j)
            else:
                jp.append("/")
        X = "".join(jp)
        l.append(X)

print(l)
```

```
['Eye dataset/close_look/eye_closed (1).png', 'Eye dataset/close_look/eye_closed (10).png', 'Eye dataset/close_look/eye_close
d (11).png', 'Eye dataset/close_look/eye_closed (12).png', 'Eye dataset/close_look/eye_closed (13).png', 'Eye dataset/close_l
ook/eye_closed (14).png', 'Eye dataset/close_look/eye_closed (2).png', 'Eye dataset/close_look/eye_closed (3).png', 'Eye data
set/close_look/eye_closed (4).png', 'Eye dataset/close_look/eye_closed (5).png', 'Eye dataset/close_look/eye_closed (6).png',
'Eye dataset/close_look/eye_closed (7).png', 'Eye dataset/close_look/eye_closed (8).png', 'Eye dataset/close_look/eye_closed
(9).png', 'Eye dataset/close_look/eye_closed(1).png', 'Eye dataset/close_look/eye_closed(10).png', 'Eye dataset/close_look/ey
e_closed(100).png', 'Eye dataset/close_look/eye_closed(1000).png', 'Eye dataset/close_look/eye_closed(1001).png', 'Eye datase
t/close_look/eye_closed(1002).png', 'Eye dataset/close_look/eye_closed(1003).png', 'Eye dataset/close_look/eye_closed(1004).p
ng', 'Eye dataset/close_look/eye_closed(1005).png', 'Eye dataset/close_look/eye_closed(1006).png', 'Eye dataset/close_look/ey
e_closed(1007).png', 'Eye dataset/close_look/eye_closed(1008).png', 'Eye dataset/close_look/eye_closed(1009).png', 'Eye datas
et/close_look/eye_closed(101).png', 'Eye dataset/close_look/eye_closed(1010).png', 'Eye dataset/close_look/eye_closed(1011).p
ng', 'Eye dataset/close_look/eye_closed(1012).png', 'Eye dataset/close_look/eye_closed(1013).png', 'Eye dataset/close_look/ey
e_closed(1014).png', 'Eye dataset/close_look/eye_closed(1015).png', 'Eye dataset/close_look/eye_closed(1016).png', 'Eye datas
et/close_look/eye_closed(1017).png', 'Eye dataset/close_look/eye_closed(1018).png', 'Eye dataset/close_look/eye_closed(1019).
png', 'Eye dataset/close_look/eye_closed(102).png', 'Eye dataset/close_look/eye_closed(1020).png', 'Eye dataset/close_look/ey
e_closed(1021).png', 'Eye dataset/close_look/eye_closed(1022).png', 'Eye dataset/close_look/eye_closed(1023).png', 'Eye datas
et/close_look/eye_closed(1024).png', 'Eye dataset/close_look/eye_closed(1025).png', 'Eye dataset/close_look/eye_closed(1026).
png', 'Eye dataset/close_look/eye_closed(1027).png', 'Eye dataset/close_look/eye_closed(1028).png', 'Eye dataset/close_look/e
ye_closed(1029).png', 'Eye dataset/close_look/eye_closed(103).png', 'Eye dataset/close_look/eye_closed(1030).png', 'Eye datas
```

In [39]:
```
pip install ipyplot
```

```
Collecting ipyplot
  Downloading ipyplot-1.1.1-py3-none-any.whl (13 kB)
Requirement already satisfied: pillow in c:\users\archit dogra\anaconda3\lib\site-packages (from ipyplot) (8.4.0)
Requirement already satisfied: numpy in c:\users\archit dogra\anaconda3\lib\site-packages (from ipyplot) (1.20.3)
Collecting shortuuid
  Downloading shortuuid-1.0.11-py3-none-any.whl (10 kB)
Requirement already satisfied: IPython in c:\users\archit dogra\anaconda3\lib\site-packages (from ipyplot) (7.29.0)
Requirement already satisfied: jedi>=0.16 in c:\users\archit dogra\anaconda3\lib\site-packages (from IPython->ipyplot) (0.18.0)
Requirement already satisfied: decorator in c:\users\archit dogra\anaconda3\lib\site-packages (from IPython->ipyplot) (5.1.0)
Requirement already satisfied: pickleshare in c:\users\archit dogra\anaconda3\lib\site-packages (from IPython->ipyplot) (0.7.5)
Requirement already satisfied: backcall in c:\users\archit dogra\anaconda3\lib\site-packages (from IPython->ipyplot) (0.2.0)
Requirement already satisfied: setuptools>=18.5 in c:\users\archit dogra\anaconda3\lib\site-packages (from IPython->ipyplot) (5
8.0.4)
Requirement already satisfied: prompt-toolkit!=3.0.0,!=3.0.1,<3.1.0,>=2.0.0 in c:\users\archit dogra\anaconda3\lib\site-package
s (from IPython->ipyplot) (3.0.20)
Requirement already satisfied: matplotlib-inline in c:\users\archit dogra\anaconda3\lib\site-packages (from IPython->ipyplot)
(0.1.2)
Requirement already satisfied: pygments in c:\users\archit dogra\anaconda3\lib\site-packages (from IPython->ipyplot) (2.10.0)
Requirement already satisfied: colorama in c:\users\archit dogra\anaconda3\lib\site-packages (from IPython->ipyplot) (0.4.4)
Requirement already satisfied: traitlets>=4.2 in c:\users\archit dogra\anaconda3\lib\site-packages (from IPython->ipyplot) (5.
1.0)
Requirement already satisfied: parso<0.9.0,>=0.8.0 in c:\users\archit dogra\anaconda3\lib\site-packages (from jedi>=0.16->IPyth
on->ipyplot) (0.8.2)
Requirement already satisfied: wcwidth in c:\users\archit dogra\anaconda3\lib\site-packages (from prompt-toolkit!=3.0.0,!=3.0.
1,<3.1.0,>=2.0.0->IPython->ipyplot) (0.2.5)
Installing collected packages: shortuuid, ipyplot
Successfully installed ipyplot-1.1.1 shortuuid-1.0.11
Note: you may need to restart the kernel to use updated packages.
```

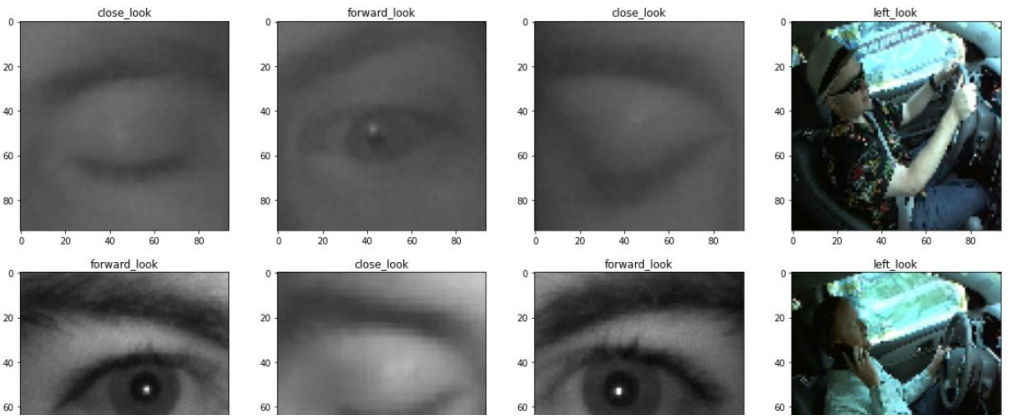
In [17]:
```
import ipyplot

ipyplot.plot_images(l, max_images=1500, img_width=150)
```

In [3]:
```
classes=list(train_generator.class_indices.keys())
plt.figure(figsize=(20,20))
for X_batch, y_batch in train_generator:
    # create a grid of 3x3 images
    for i in range(0,16):
        plt.subplot(4,4,i+1)
        plt.imshow(X_batch[i])
        plt.title(classes[np.where(y_batch[i]==1)[0][0]])
    # show the plot
    plt.show()
    break
```

In [4]:
```
valid_generator=tf.keras.preprocessing.image.ImageDataGenerator(
    rescale=1/255.0,
    validation_split=0.1,
).flow_from_directory(train_dir,batch_size=16,target_size=(size,size),subset='validation',shuffle=True)
```

```
Found 1781 images belonging to 4 classes.
```

In [5]:
```
classes=list(valid_generator.class_indices.keys())
plt.figure(figsize=(20,20))
for X_batch, y_batch in valid_generator:
    # create a grid of 3x3 images
    for i in range(0,16):
        plt.subplot(4,4,i+1)
        plt.imshow(X_batch[i])
        plt.title(classes[np.where(y_batch[i]==1)[0][0]])
    # show the plot
    plt.show()
    break
```



In [6]:
```
test_generator=tf.keras.preprocessing.image.ImageDataGenerator(
    rescale=1/255.0,
    validation_split=0.1
).flow_from_directory(train_dir,batch_size=16,target_size=(size,size),subset='validation',shuffle=True)
```

```
Found 1781 images belonging to 4 classes.
```

In [7]:
```
classes=list(test_generator.class_indices.keys())
plt.figure(figsize=(20,20))
for X_batch, y_batch in test_generator:
    # create a grid of 3x3 images
    for i in range(0,16):
        plt.subplot(4,4,i+1)
        plt.imshow(X_batch[i])
        plt.title(classes[np.where(y_batch[i]==1)[0][0]])
    # show the plot
    plt.show()
    break
```

In [8]:
```
classes
```

Out[8]:
```
['close_look', 'forward_look', 'left_look', 'right_look']
```

In [9]:
```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense ,Flatten ,Conv2D ,MaxPooling2D ,Dropout ,BatchNormalization ,GlobalMaxPool2D
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.callbacks import EarlyStopping ,ReduceLROnPlateau ,ModelCheckpoint
```

In [10]:
```
optimizer_adam=Adam(learning_rate=0.001,beta_1=0.9,beta_2=0.99)
```

In [11]:
```
EarlyStop=EarlyStopping(patience=10,restore_best_weights=True)
Reduce_LR=ReduceLROnPlateau(monitor='val_accuracy',verbose=2,factor=0.5,min_lr=0.00001)
model_check=ModelCheckpoint('model.hdf5',monitor='val_loss',verbose=1,save_best_only=True)
callback=[EarlyStop , Reduce_LR,model_check]
```

In [12]:
```
model=Sequential([
                  Conv2D(32,3,activation='relu',kernel_initializer='he_normal',input_shape=(size,size,3)),
                  Conv2D(64,3,activation='relu',kernel_initializer='he_normal'),
                  BatchNormalization(),
                  MaxPooling2D(3),

                  Conv2D(128,3,activation='relu',kernel_initializer='he_normal'),
                  BatchNormalization(),
                  MaxPooling2D(3),

                  Conv2D(256,3,activation='relu',kernel_initializer='he_normal'),
                  BatchNormalization(),
                  MaxPooling2D(3),

                  Flatten(),
                  Dense(64,activation='relu',kernel_initializer='he_normal'),
                  BatchNormalization(),
                  Dense(4,activation='softmax',kernel_initializer='glorot_normal')

])
```

In [14]:
```
model.compile(optimizer=optimizer_adam,loss='categorical_crossentropy',metrics=['accuracy'])
```

In [15]:
```
history=model.fit(train_generator,validation_data=valid_generator,epochs=50,batch_size=16,
                  steps_per_epoch=len(train_generator)//16,validation_steps=len(valid_generator)//16,
                  callbacks=callback, verbose=1)
```

```
Epoch 1/50
62/62 [==============================] - ETA: 0s - loss: 0.8293 - accuracy: 0.6653
Epoch 1: val_loss improved from inf to 1.49987, saving model to model.hdf5
62/62 [==============================] - 14s 217ms/step - loss: 0.8293 - accuracy: 0.6653 - val_loss: 1.4999 - val_accuracy: 0.
2500 - lr: 0.0010
Epoch 2/50
62/62 [==============================] - ETA: 0s - loss: 0.5808 - accuracy: 0.7732
Epoch 2: val_loss improved from 1.49987 to 1.38125, saving model to model.hdf5
62/62 [==============================] - 13s 209ms/step - loss: 0.5808 - accuracy: 0.7732 - val_loss: 1.3813 - val_accuracy: 0.
2946 - lr: 0.0010
Epoch 3/50
62/62 [==============================] - ETA: 0s - loss: 0.5550 - accuracy: 0.7802
Epoch 3: val_loss did not improve from 1.38125
62/62 [==============================] - 13s 209ms/step - loss: 0.5550 - accuracy: 0.7802 - val_loss: 1.4023 - val_accuracy: 0.
3482 - lr: 0.0010
Epoch 4/50
62/62 [==============================] - ETA: 0s - loss: 0.5218 - accuracy: 0.8065
Epoch 4: val_loss improved from 1.38125 to 1.14414, saving model to model.hdf5
62/62 [==============================] - 13s 208ms/step - loss: 0.5218 - accuracy: 0.8065 - val_loss: 1.1441 - val_accuracy: 0.
4554 - lr: 0.0010
Epoch 5/50
62/62 [==============================] - ETA: 0s - loss: 0.4717 - accuracy: 0.8206
Epoch 5: val_loss improved from 1.14414 to 0.96410, saving model to model.hdf5
62/62 [==============================] - 13s 209ms/step - loss: 0.4717 - accuracy: 0.8206 - val_loss: 0.9641 - val_accuracy: 0.
7054 - lr: 0.0010
Epoch 6/50
62/62 [==============================] - ETA: 0s - loss: 0.4774 - accuracy: 0.8185
Epoch 6: val_loss did not improve from 0.96410
62/62 [==============================] - 13s 209ms/step - loss: 0.4774 - accuracy: 0.8185 - val_loss: 1.6617 - val_accuracy: 0.
3125 - lr: 0.0010
Epoch 7/50
62/62 [==============================] - ETA: 0s - loss: 0.4694 - accuracy: 0.8185
Epoch 7: val_loss did not improve from 0.96410
62/62 [==============================] - 13s 209ms/step - loss: 0.4694 - accuracy: 0.8185 - val_loss: 1.5948 - val_accuracy: 0.
4732 - lr: 0.0010
Epoch 8/50
62/62 [==============================] - ETA: 0s - loss: 0.4250 - accuracy: 0.8347
```

In [16]:
```
#plotting training values
import seaborn as sns
sns.set()

acc = history.history['accuracy']
val_acc = history.history['val_accuracy']
loss = history.history['loss']
val_loss = history.history['val_loss']
epochs = range(1, len(loss) + 1)
```

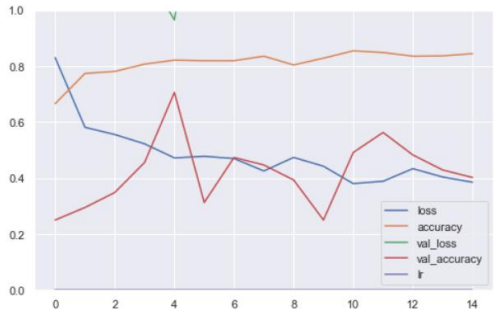
In [33]:
```
#accuracy plot
plt.plot(epochs, acc, color='green', label='Training Accuracy')
plt.plot(epochs, val_acc, color='blue', label='Validation Accuracy')
plt.title('Training and Validation Accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend()
```

Out[33]:
```
<matplotlib.legend.Legend at 0x12161d367f0>
```

In [17]:
```
#loss plot
plt.plot(epochs, loss, color='green', label='Training Loss')
plt.plot(epochs, val_loss, color='red', label='Validation Loss')
plt.title('Training and Validation Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()

plt.show()
```

Epoch

In [18]:
```
import pandas as pd
pd.DataFrame(history.history).plot(figsize=(8, 5))
plt.grid(True)
plt.gca().set_ylim(0, 1)
plt.show()
```



Data Cleaning

Data cleaning, also known as data cleansing, is the process of locating and fixing (or removing) inaccurate or corrupted records from a record set, table, or database. It also refers to the identification of missing, incorrect, inaccurate, or irrelevant portions of data and the replacement, modification, or deletion of impure or raw data.Data cleaning can be carried out in batches using scripting or a data quality firewall or interactively using data wrangling tools.

The data set should be consistent with other sets that are similar in the system after cleaning. User input errors, transmission or storage corruption, or disparate data dictionary definitions of comparable entities in separate repositories could all have contributed to the inconsistencies that were later discovered or fixed. Data validation is different from data cleansing in that it nearly usually results in data being rejected from the system at the time of input rather than being conducted on batches of data.

Typographical errors may be removed, or values may be validated and corrected in accordance with a known list of entities, as part of the actual data cleansing process. Invalid addresses can be rejected, for example, or records can be corrected when they only partially match other known records. Validation can also use fuzzy or approximate string matching. Some data cleansing programmes purge the data by cross-referencing it with a set of approved data. Data enhancement, which involves making data more comprehensive by adding relevant information, is a frequent technique in data cleansing. For instance, relating phone numbers to addresses that correspond to those numbers.

A simple example of data normalisation is the extension of abbreviations. Data normalisation is the process of merging data from "different file formats, naming conventions, and columns" into a single coherent data set.

The data set should be consistent with other sets that are similar in the system after cleaning. User input errors, transmission or storage corruption, or disparate data dictionary definitions of comparable entities in separate repositories could all have contributed to the inconsistencies that were later discovered or fixed.

Dataset

A machine learning dataset is a collection of data that is used to train a model. The data set acts as an example that teaches the machine learning algorithm

how to make predictions. Common data types include:

- Text data
- Image data
- Audio data
- Video data
- Numerical data

Purpose of AI/ML datasets:

Preparing and selecting the right dataset is crucial for the success of an AI/ML development project as it directly impacts the accuracy and effectiveness of the trained model. The dataset serves as the foundation for the machine learning model and acts as a representative sample of the real-world scenarios the model will encounter.

A well-prepared dataset should accurately represent the problem domain and contain enough relevant examples to provide a diverse range of data for the model to learn from. The dataset should also be free of biases and inaccuracies that could negatively impact the model's performance or lead to incorrect conclusions.

Choosing the right dataset involves considering factors such as the size and complexity of the data, the quality and relevance of the features, and the availability and reliability of the data source. The dataset should also be balanced, meaning that it should have an even distribution of examples for each class or category to prevent the model from being biased towards one particular outcome.

In summary, the selection and preparation of a high-quality dataset are critical steps in training an effective machine learning model. Investing time and effort into this stage can greatly increase the chances of success for an AI/ML development project.

Key purposes of an AI/ML dataset:

To train the model

To measure the accuracy of the model once it is trained

What are the types of ML datasets?

The entire collected data set is divided into 3 subsets, which are as follows:

1. Training dataset

This is one of the most important subsets of the entire data set, accounting for roughly 60% of it. This set contains the data that will be used to train the model at first. To put it another way, it teaches the algorithm what to look for in the data. A vehicle licence plate recognition system, for example, will be trained using image data with location labels (e.g., front or back of the car) and the vehicle licence plate data format:

2. Validation data file

This subset accounts for approximately 20% of the total dataset and is used to evaluate all model parameters once the training phase is complete. Validation data is known data that aids in the identification of potential flaws in the model. This information is also used to determine whether the model is over or under fit.

3. Test data file

This subset, which comprises the final 20% of the data set, is used as input in the training process. This subset's data are used to assess the model's precision and are not known to it. This dataset will demonstrate how much your model has learned from the previous two subsets, to put it another way.

Convolutional Neural Network

Convolutional neural networks (CNNs) are a type of artificial neural network (ANN) that is primarily used for visual imagery analysis. Because of their shared-weight architecture of convolution kernels or filters that provide translation-equivariant responses called feature maps, they are also known as

shift invariant or space invariant artificial neural networks. Despite their name, most CNNs are not completely translation invariant due to the down sampling operation they use. Image and video recognition, recommendation systems, image classification, image segmentation, medical image analysis, natural language processing, brain-computer interfaces, and financial time series analysis are just a few of the applications for CNNs.

CNNs are regularised versions of multilayer perceptrons, which typically have fully connected networks in which every neuron in one layer is connected to every neuron in the next layer. However, the "full connectivity" of these networks makes them prone to overfitting, which can be mitigated by regularisation techniques such as weight decay or trimming connectivity. CNNs take a different approach to regularisation by utilising the hierarchical pattern in data and building patterns of increasing complexity by embossing smaller and simpler patterns on their filters. As a result, CNNs are less connected and complex than other types of networks.

CNNs are inspired by biological processes, specifically the animal visual cortex's cooperation. CNN connectivity patterns resemble cortical neurons' receptive fields, which respond to stimuli only in a limited area of the visual field. Different neurons' receptive fields partially overlap to cover the entire visual field.

CNNs require little pre-processing when compared to other image classification algorithms, and the network learns to optimise its filters through automated learning. This lack of reliance on prior knowledge and human intervention in feature extraction is a significant benefit.

Data Cleaning in Eye Image Dataset

ORIGINALITY REPORT

8%	6%	5%	3%
SIMILARITY INDEX	INTERNET SOURCES	PUBLICATIONS	STUDENT PAPERS

PRIMARY SOURCES

1	www.ncbi.nlm.nih.gov Internet Source	2%
2	Submitted to Middle East College of Information Technology Student Paper	1%
3	Submitted to Liverpool John Moores University Student Paper	1%
4	data-flair.training Internet Source	1%
5	scholars.ncu.edu.tw Internet Source	1%
6	"Recent Developments in Electronics and Communication Systems", IOS Press, 2023 Publication	1%
7	10footer.com Internet Source	1%
8	polynoe.lib.uniwa.gr Internet Source	1%