IMAGE CLASSIFICATION USING ENSEMBLE MODELING AND DEEP LEARNING

Kaneesha Gandhi
San Jose State University

IMAGE CLASSIFICATION USING ENSEMBLE MODELING AND DEEP LEARNING

A Project Report

Presented to

The Faculty of the Department of Computer Science

San Jose´ State University

In Partial Fulfillment

of the Requirements for The Degree

Master of Science

by

Kaneesha Gandhi

May 2023

The Designated Thesis Committee Approves the Project Titled

IMAGE CLASSIFICATION USING ENSEMBLE MODELING AND DEEP LEARNING

by

Kaneesha Gandhi

APPROVED FOR THE DEPARTMENT OF COMPUTER SCIENCE

SAN JOSÉ STATE UNIVERSITY

May 2023

Dr. Ching-seh Wu, Ph. D.	Department of Computer Science
Dr. William Andreopoulos	Department of Computer Science
Mr. Mohit Gupta	Software Engineer II at Microsoft

ABSTRACT

IMAGE CLASSIFICATION USING ENSEMBLE MODELING AND DEEP LEARNING

by Kaneesha Gandhi

With the advances in technology, image classification has become one of the core areas of interest for researchers in the field of computer vision. We, humans, experience great levels of visuals in our day-to-day lives. The human eye is a powerful tool that not only lets us capture images around us but also aids in remembering, distinguishing, and interpreting these visuals. Comprehending the images that the user perceives is an important application in the fields of artificial intelligence, smart security systems, and areas of virtual reality. Recent advances in machine learning and neural networks have led to more precise and efficient methods for identifying and categorizing visual data. In this project, we focus on the classification of images into different categories by running various ensemble machine-learning models like Decision Tree model, Extra-Tree classifier, XGBoost model and Gradient Boosting model. Our primary goal will be to emphasize two kinds of input data numerical eye-tracking data and image data. Furthermore, we train Deep Convolutional Generative Adversarial Network (DCGAN) to generate new images that resembles our input image dataset. This is particularly to augment our dataset and create images for our image dataset in order to enhance the performance of our ensemble models. A comparative thorough analysis is then performed with our current ensemble models and the previous as well as current literature. Our experiments conclude that for image data classification, our proposed model performs better after data augmentation and for classification using eye-tracking numerical data Decision Tree performs than all the models implemented.

***Keywords* – Ensemble models, eye-tracking, human visuals, image-classification, Hybrid model, Decision Tree model, Extra-Tree classifier, XGBoost model, Gradient Boosting model, DC-GANs**

ACKNOWLEDGEMENTS

Despite being a long and challenging journey, I consider my experience at SJSU to be worthwhile, thanks to the exceptional professors and supportive friends I have encountered along the way. I am deeply grateful for their guidance and would like to take this opportunity to express my appreciation to Dr. Wu, my advisor, who has provided unwavering support throughout my entire master's degree, not just during the thesis. Without his trust in my research abilities, I would not have been able to achieve what I have.

Additionally, I am indebted to Dr. Andreopoulos and Mr. Gupta, who served on my committee and provided invaluable feedback that was instrumental in the success of my project.

Finally, I would like to thank my parents, my sister Hetvi, and my dear friend Dipali for their unwavering support and encouragement during my graduation.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

1 INTRODUCTION

1.1 Introduction

Vision is one of the most important yet one of the most complex senses of the human body. As per popular research, about half of our brain is used for vision [1, 2, 3]. Moreover, of the five senses, sight is the one that has been the most completely studied and understood. Research has clearly shown that we rely more on our sense of sight than any other when interacting with the outside world [4]. As a result, processing visual information occupies a much larger portion, close to 80% of the primate brain, than processing information from any other sense [4, 5]. Human vision and image classification are closely related as image classification tasks involve training machines to mimic the human visual system's ability to recognize and categorize visual information.

In image classification, a machine learning algorithm is trained to recognize and categorize images based on a set of predefined labels [3]. The algorithm analyzes the visual features of the image, such as color, texture, and shape, to make a prediction about the image's content [4].

Image classification has received special interest in the field of computer vision because it can yield methodical explanations for target-oriented eye-movement responses [1, 2]. This phenomenon dates back to the 1960s, when the first computer vision systems were developed. Early techniques for image classification were based on feature extraction, which involved manually identifying and extracting key features from images, such as edges, corners, and textures. These features were then used to train machine learning algorithms, such as support vector machines and decision trees, to classify images [41].

There exists a symbiotic relationship between visual attention and a particular task which indicates that any image that the human eye captures can be successfully categorized [5, 6]. There are a number of characteristics one can deduce from the eye-tracking data and hence realize the different surroundings of a particular subject [6, 7]. These methods are influenced by the organization and operations of the human visual system and have been educated on vast image datasets to accurately identify and sort objects [5, 6, 7].

In the 1990s, neural networks became increasingly popular for image classification, due to their ability to automatically learn and extract features from images. Convolutional neural networks (CNNs) emerged as a particularly effective approach, using layers of filters to detect features at different levels of abstraction [42]. Even today, over the past few years, there has been notable advancement in the creation of deep learning techniques for categorizing images, such as convolutional neural networks (CNNs), which have delivered remarkable outcomes in diverse image classification assignments [4].

In 2012, the ImageNet Large Scale Visual Recognition Challenge (ILSVRC) was introduced, which led to a significant advancement in image classification. The challenge involved categorizing 1.2 million images into 1,000 categories using the deep learning algorithm AlexNet, which achieved a top-5 error rate of 16.4%. Later in 2015, the ResNet architecture surpassed GoogLeNet in accuracy on the ImageNet dataset, achieving a top-5 error rate of 3.57%. Consequently, ResNet has become a popular choice in deep learning for various computer vision tasks, such as object detection and image segmentation [43]. In present year of 2023, Improved Differential Evolution of Convolutional Neural Network (IDECNN) was launched which performed image classification on MNIST dataset. This achieved a top-5 error rate of 0.38%.

The approach employed the Differential Evolution (DE) algorithm to automatically create a layer-based Convolutional Neural Network (CNN) architecture for the purpose of image classification [44]. This comparison of different error rates is shown in the Figure 1 below:

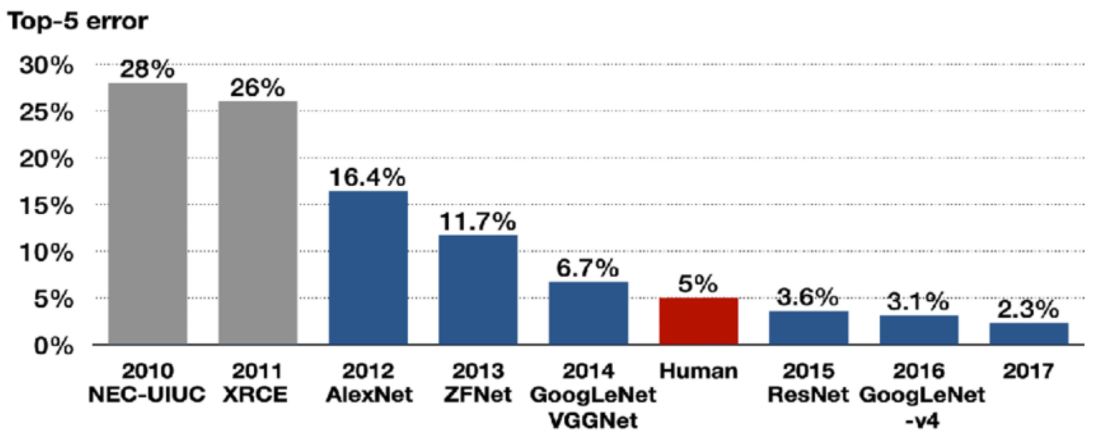


Figure 1: Top-5 error rates of Deep CNN models used for image classification [44]

In image generation, GANs are trained to create new images that are similar to a given set of training images. The training process involves two neural networks: a generator network that creates the images, and a discriminator network that tries to distinguish between the generated images and real images. The two networks are trained simultaneously, with the goal of improving the generator's ability to create realistic images that fool the discriminator. While GANs were originally developed for image generation tasks, they have also been used for image classification in some cases. Our approach is to use a DC-GANs model to generate similar looking images of objects from the original dataset. These synthetic images were added to the training data for all image classification algorithms, which can improve its ability to recognize and classify objects in different contexts.

Our goal is to emphasize the importance image classification using two types of input data: numerical eye tracking data consisting visual displays of user gaze scan paths converted to pixel coordinates of Left and Right eye respectively in CSV files and second, the image dataset.

We enhance our image dataset and boost the precision of the ensemble learning models by training a Deep Convolutional Generative Adversarial Network (DCGAN) to create new images that closely resemble our current dataset.

We then compare the performance of our ensemble models with those used in previous and current research, performing a thorough analysis to assess their efficacy. Our results indicate that image-based classification is more effective for the proposed hybrid model after data augmentation is employed, and that Decision Tree performs comparatively better among all the models for eye tracking data. Figure 2 below shows the detailed diagrammatic representation of the different methodical approaches to be followed for this research.

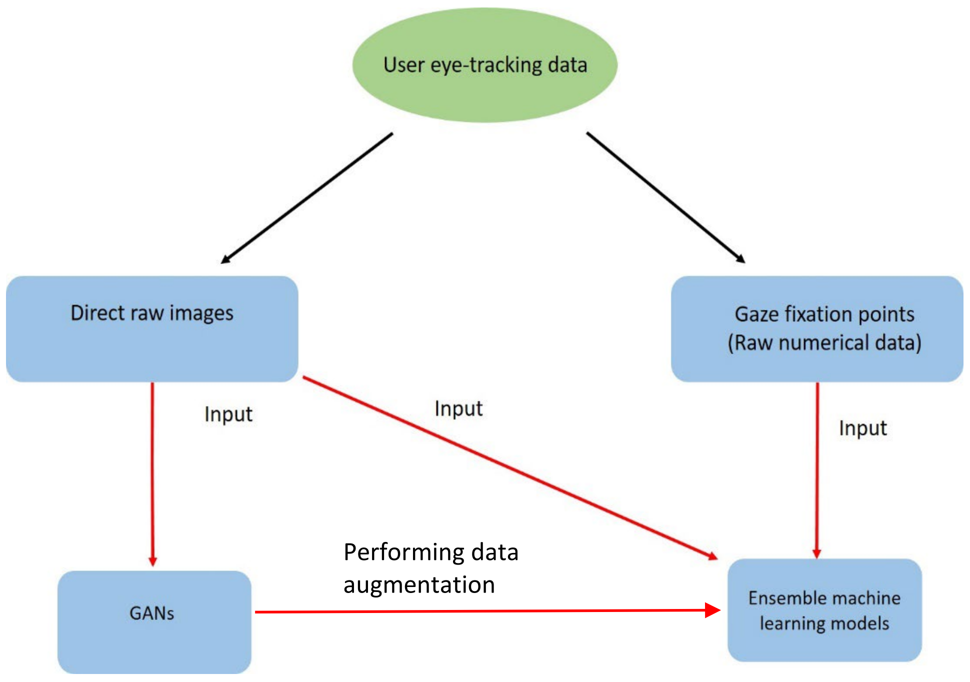

User eye-tracking data
Direct raw images
Gaze fixation points (Raw numerical data)
Input
Input
Input
Performing data augmentation
GANs
Ensemble machine learning models

Figure 2: Organization of different approaches and their inputs

The rest of the report is organized as follows: Chapter 2 covers the background study and literature review of previous researchers in the fields of Image Classification. Chapter 3 describes the different libraries and development tools used for the implementation of the project. Chapter 4 showcases the detailed description of the dataset used. Chapter 5 describes the approaches and implementation of our project by covering the different ensemble models and GANs implementation for the research. Chapter 6 discusses the different metrics used for evaluation of our experiments and the different methods of experimentation for each of the models used. Chapter 7 includes the final results of the research including the comparison of our model with models in other literature . Lastly, chapter 8 discusses the conclusion and future work of the research.

1.2 Research Objective

The objective of this research is to explore different ensemble machine learning approaches for image classification that combines the strengths of multiple classifiers to improve accuracy and robustness. Specifically, the study aims to explore different ensemble techniques of Decision Tree, XGBoost, ExtraTree and Gradient Boosting, and compare their effectiveness on our raw image and eye-tracking datasets. The research also proposes a hybrid ensemble model that combines Decision Tree, XGBoost and ExtraTree classifiers in conjunction and compare the different evaluation metrics with the rest of the models. Additionally, the research will investigate the impact of feature selection and optimization of models through hyper-parameter tuning, on the performance of the ensemble classifiers.

2 LITERATURE REVIEW

2.1 Introduction

Classifying pictures is a difficult task that has been widely researched recently [1, 2] in the field of computer vision. Ensemble learning is a promising method that includes combining predictions from various classifiers to get a more trustworthy and accurate final prediction in order to improve the precision and durability of picture categorization. This section summarizes current studies that explore how ensemble learning may be used to classify images [11]. When employing ensemble learning for image classification, the choice of basic classifiers is also a crucial factor. The performance of the ensemble can be improved by using a variety of classifiers because different classifiers have different strengths and weaknesses [11, 12]. Support vector machines, decision trees, and deep neural networks are just a few examples of basic classifiers that have been studied in studies, and it has been shown that combining them can increase classification accuracy and reduce the danger of overfitting [13].

In recent years, several studies have proposed novel ensemble learning techniques for image classification, such as Bayesian model averaging, mixture of experts, and random forest. These methods have shown promising results on various datasets, and offer new insights into how ensemble learning can be used for image classification [4].

The main subject of this review of existing literature is the classification of images through the utilization of a variety of techniques related to machine learning, deep learning, and DC-GANs. The articles chosen for this review encompass a range of sources such as conference proceedings, academic journals, thesis dissertations, and up-to-date research from companies that specialize in quantitative and qualitative analysis of image classification.

The remainder of the literature review is structured in the following manner: Section II provides a summary of research that has been conducted using conventional machine learning algorithms; Section III details research that has been done using deep learning algorithms, specifically CNNs. Section IV outlines previous studies on image classification and generation that have utilized GANs. Figure 3 illustrates the layout of the literature review.

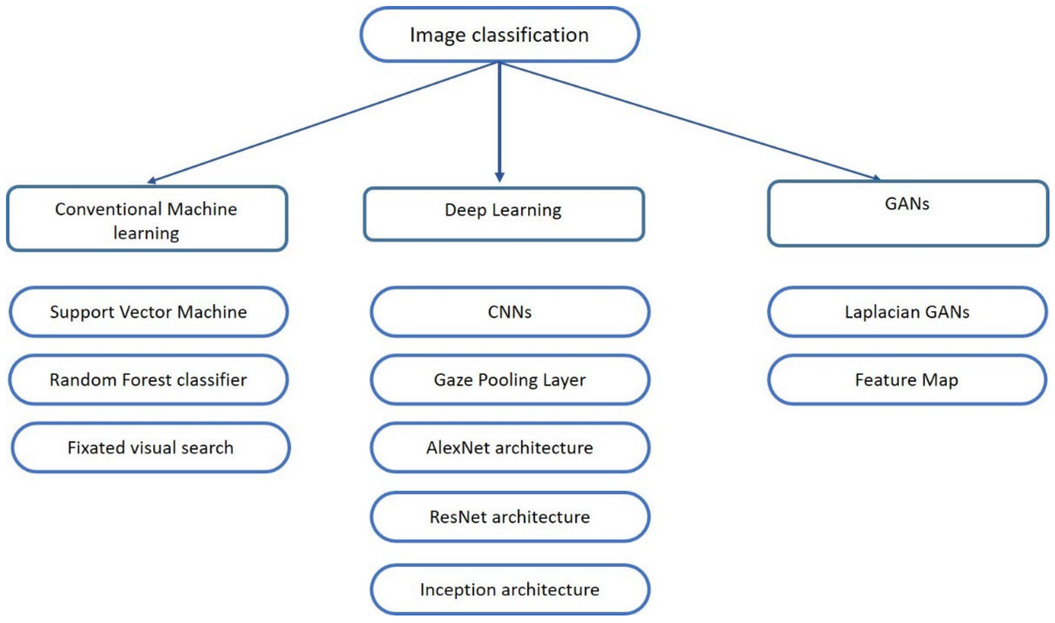


Figure 3: Map of organization of literature review

2.2 Conventional Machine Learning Approach

2.2.1 Support Vector Machine

The process of classifying photographs using their visual features involves dividing them up into various groups or classes. Machine learning has gained appeal as an image classification technique due to its ability to find patterns and features in large datasets [1, 5, 6]. Support Vector Machines (SVMs) and Random Forests are two common examples of traditional machine learning techniques that have been successfully applied to picture categorization applications [4]. For picture classification, these algorithms make use of elements including texture, color, and form. The incapability of these algorithms to understand intricate characteristics and patterns, however, may restrict their usefulness.

Yarbus et al. [2] demonstrated that a person's visual behavior is closely related to the task they are performing when viewing a visual scene. This finding has been highly influential and has led to a large number of subsequent studies in various fields, including human vision, neuroscience, artificial intelligence, and computer vision. These studies often aim to analyze visual behavior, such as fixations and saccades, in order to make predictions about a person's behavior [7, 8]. The researchers in this study have compiled a dataset of fixation data from 18 people searching for specific images within synthesized collages of approximately 80 images, in three different image categories. This is shown in the Figure 4 below. In a controlled experiment, they show that they can predict the correct target image from a group of five candidates. The paper also presents a new approach for predicting search targets in a more open-ended setting, which involves learning the connections between fixations and potential target images [3, 4].

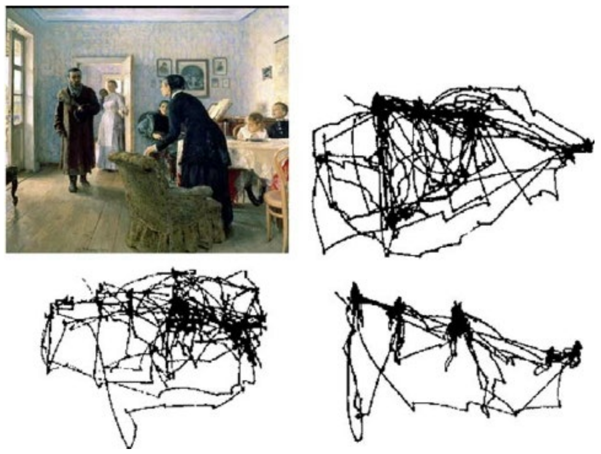


Figure 4: Sample dataset from fixation data [2]

2.2.2 Random Forest Classifier

Samaras et al. [5] conducted a study to decode a person's search behavior during a visual search task. They focused on the objects that a person fixated on during the search, which can provide information about the category of their search target. They found that people tended to fixate on objects that were similar to their target in both search tasks. They also had other participants view the searchers' scan paths, superimposed over displays that did not include the target, and try to decode the target category (bear or butterfly). The bear search team was correctly classified, while the butterfly search team had a classification rate of 77%. To analyze the preferentially fixated objects, the researchers utilized Support Vector Machine (SVM) classifiers and obtained comparable classification rates [6, 7, 10]. By interpreting the fixations made during a search, the researchers demonstrated that it's possible to infer the search target's category. On average, human decoders accurately classified a searcher's target 89% of the time [5].

This high decoding rate differed from the individual trial rates, which were close to or at chance. This indicates that SVM-based classifiers could be beneficial for identifying the characteristics that differentiate various search target categories [11, 13]. The images employed in this study were categorized as high, medium, and low similarity distractors, where similarity was directly linked to the target. This is shown in Figure 5.

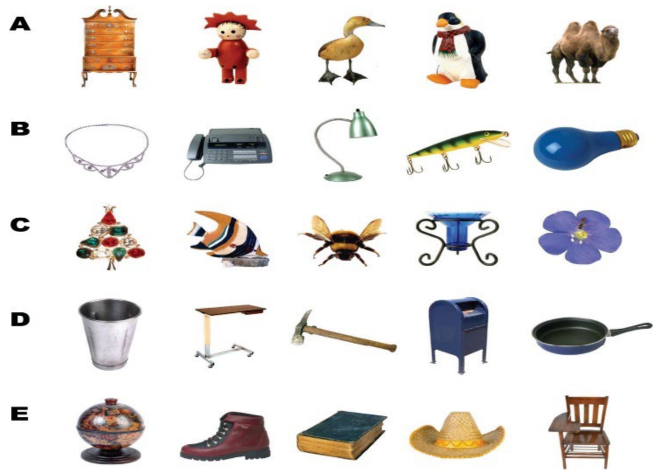


Figure 5: Representative examples of similarity-rated objects used as distractors. (A) Bear-similar objects. (B) Bear-dissimilar objects. (C) Butterfly-similar objects. (D) Butterfly-dissimilar objects. (E) Medium-similar objects [5].

For the labeled dataset, calculations are made using supervised machine learning techniques [11]. In order to construct a computation on a dataset of labeled pictures using supervised machine learning, each image that has been labeled with a significant category or lesson is employed. [12, 14, 16]. Algorithms learn to recognize patterns and attributes inside labeled photos in order to discover new hidden photo categories or histories [17, 18]. Often, guided machine learning for image categorization goes through the initial and testing stages.

2.2.3 Fixated Visual Search

To determine the design and attributes innate in each image history, calculations are done on the annotated dataset. In supervised machine learning for image classification, it is important to have a large and diverse labeled dataset for training the algorithm [23]. The dataset should be representative of the types of images that the algorithm will encounter in the real world. This is because the accuracy of the algorithm depends heavily on the quality of the training data [12, 13].

Another important aspect of supervised machine learning for image classification is the choice of evaluation metrics [9, 10, 11]. Commonly used metrics include accuracy, precision, recall, and F1 score. These metrics provide a quantitative measure of the algorithm's performance on the testing data and help to identify areas for improvement [5].

Overall, fixated visual search has proven to be a powerful tool for image classification, enabling a wide range of applications in areas such as medical imaging, object detection, and autonomous driving [12, 13]. While deep learning has shown remarkable success in image classification tasks, traditional machine learning algorithms still have a role to play, particularly in cases where the dataset size is small or the computational resources are limited [14].

2.3 Deep Learning (CNNs)

2.3.1 CNNs

CNN stands for Convolutional Neural Network, and it is a type of deep learning algorithm commonly used for image classification tasks. It is designed to automatically extract and learn important features from images, such as edges, corners, textures, and patterns, and use them to classify the image into different categories or classes.

CNNs are composed of several layers, including convolutional, pooling, and fully connected layers. During the convolutional layers, filters or kernels are applied to the input image to extract features. The resulting feature maps then undergo non-linear activation functions, such as ReLU. The pooling layers shrink the size of the feature maps and retain only the crucial features. Lastly, the fully connected layers make predictions based on the learned features.

One of the most common applications of CNNs is image classification. The network is trained on a vast dataset of labeled images, where each image corresponds to a particular category or class. The network learns to recognize the distinct patterns and features of each class and uses this knowledge to forecast the class of new, unseen images..

CNNs have proven to be highly successful in image classification tasks, achieving state-of-the-art performance on various benchmark datasets such as ImageNet, CIFAR-10, and MNIST. As a result, they are widely utilized in numerous applications like object detection, facial recognition, and medical image analysis.

2.3.2 Gaze Pooling Layer

Previous studies have explored classification tasks, including those involving human eye-tracking data. Fritz in [1] came up with a new approach called Gaze Pooling Layer. In this, the CNN tampered features were in made to be in conjunction with the gaze data and this was done via a mechanism called attention mechanism.

This mechanism consisted of two different aspects pf the normal human gaze behavior – temporal and spatial. The method was quiet simple. It made use of collection of exhaustive but pre-trained CNN models and hence avoided the use of expensive techniques involving image and gaze data. There were a lot of experiments conducted and these show the accurate prediction of visual target, specifically 10 category and 10 attribute tasks [1]. There were 14 participants and all the researchers showcase this dataset by expensive and exhaustive experimentation showing amazing results using eye-tracking data.

The authors of [3] have introduced a novel method for detecting a person's implicit visual search intention by analyzing their eye movements and pupillary response. The proposed approach can differentiate between task-oriented visual search and task-free visual browsing. To accurately measure the pupillary response, the authors suggest using a hierarchical support vector machine. The model can accurately detect transitions between different intention conditions with high accuracy rates, greater than 90%. However, factors like the intensity and size of the visual stimulus may impact the measurement of the pupillary response. Therefore, the authors suggest a robust baseline model to overcome these effects.

This model can identify a person's implicit intention while viewing real-world scenes in both indoor and outdoor environments as either task-free visual browsing (TFVB) or task-oriented visual search (TOVS) intent. TFVB is when a person is looking for interesting objects without a specific goal in mind, while TOVS is when a person is searching for a particular object or information about its location to achieve a specific task [6, 7]. The block diagram for the implementation is shown in Figure 6.

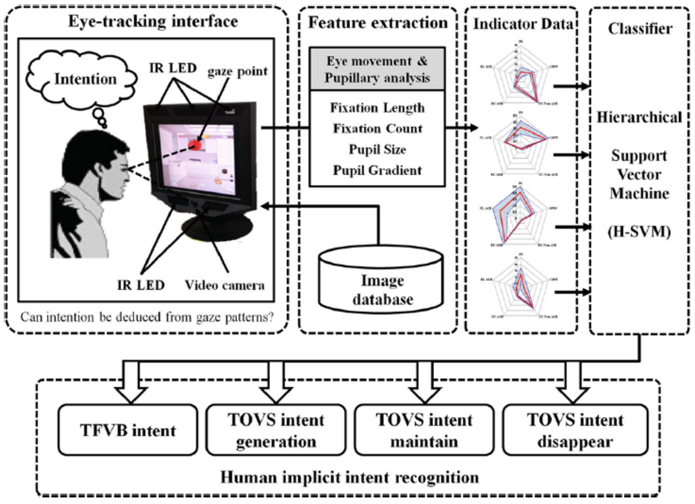


Figure 6: Block diagram of the proposed system for recognizing human's implicit intentions [3]

Kumar et al. [11] analyzed the dataset by creating a scatter plot to visualize the features and data points in order to understand the properties of each task. After eliminating highly correlated features, he trained a support vector machine (SVM) and an Ada Boosting classifier to predict the tasks from the filtered eye movement data. He achieved an accuracy of 95.4% in classifying the tasks, supporting the hypothesis that it is possible to classify tasks based on a person's eye movement data.

2.3.3 AlexNet architecture

In [25], the authors made a significant advancement in deep learning and image classification by introducing the AlexNet architecture, a deep convolutional neural network that consisted of five convolutional layers, max-pooling layers, and three fully connected layers. To train this network, they used the ImageNet dataset that comprised of over 1.2 million high-resolution images categorized into 1000 classes [25].

Their research demonstrated that AlexNet was capable of achieving a top-5 error rate of 15.3%, which was considerably better than the prior best performance of 26.2%. This improvement was largely attributed to the use of a deep convolutional neural network architecture, which was able to automatically learn complex features from the images [1, 3]. The paper also introduced a number of novel techniques for training deep convolutional neural networks, including the use of data augmentation, dropout, and ReLU activation functions. These techniques have since become standard practices in deep learning for computer vision.

2.3.4 ResNet architecture

The paper introduced the ResNet architecture [26, 27], a deep convolutional neural network that addresses the problem of vanishing gradients in very deep networks by using residual connections.

The ResNet architecture consists of multiple residual blocks, each of which contains two or more convolutional layers with shortcut connections that bypass the convolutional layers [26]. The shortcut connections allow the network to learn the difference between the input and output of a residual block, making it easier to learn the residual mapping. By using residual connections, the network can be trained to very deep depths without suffering from the problem of vanishing gradients [27].

The authors assessed the performance of the ResNet architecture on the ImageNet dataset and found that their network outperformed the previous state-of-the-art GoogLeNet, achieving a top-5 error rate of 3.57% with up to 152 layers [26, 27]. As a result, the ResNet architecture has become a popular option for various computer vision tasks, including object detection, image segmentation, and image captioning. Moreover, the ResNet architecture has been modified for use in other areas such as speech recognition and natural language processing [27].

He et al. [27] proposed a solution to the degradation problem in deep neural networks using deep residual learning. They introduced the concept of fitting a residual mapping, represented as *F(x)*, to the desired underlying mapping, represented as *H(x)*, by using stacked nonlinear layers. This residual mapping is defined as the difference between the desired mapping and the input, *F(x) = H(x) - x*. As illustrated in Figure 7 this residual mapping is added back to the input, resulting in the desired mapping *H(x) = F(x) + x*.

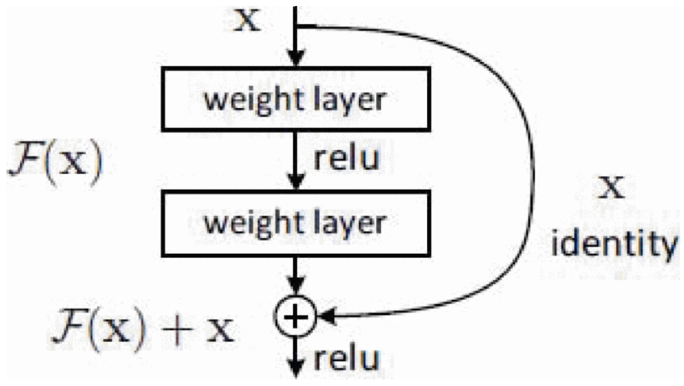


Figure 7: Residual learning building block [27]

In this equation, we assume both *x* and *F(x)* to be the same, i.e., their sizes to be the same. We can see in the above figure 5 that we use different units for map functions between stacked layers and this takes place in a periodic manner. There are various ways of explaining this phenomenon. One such way is, for example, let's say *y= F(x,{Wi}) + x*. Here *x* is the input and *F(x,{Wi})* determines the residual function that is mapped out by different convolutional network layers. *Y* is the output of the entire function.

X and Y are the two input and output variables in that layer's settings. For a residual function *F(x)* to exist, the parameters *x* and *F* must match. If the dimensions do not match, the shortcut link must have a *Ws* linear projection to correct the dimensions. The building block output *y* is computed as a function of the residual *F(x,Wi)* multiplied by the linear projection *Ws* multiplied by the input x, or *y = F(x,Wi) + Ws x.*

2.3.5 Inception architecture

The Inception architecture was proposed by Szegedy et al. [11] as a deep convolutional neural network that utilizes multiple convolutional filters at each layer. It introduced the concept of "inception modules" which enables the network to process input data in parallel with filters of various sizes. This allows the network to capture features at different scales and resolutions, which is important for recognizing objects in complex images.

In the Inception architecture, there were way lesser parameters involved. This, hence increased the accuracy of the model in comparison to other previous approaches. This architecture received further light and importance when it was used on the ImageNet dataset. In this, 1x1 convolutions were used to decrease the amount of input channels. This along with bigger filters and global pooling were used in conjunction in place of fully linked layers.

The Inception architecture has since undergone several iterations, namely Inception-v2, Inception-v3, and Inception-v4. Our most current versions substantially improve performance while keeping efficiency by employing state-of-the-art techniques including regularization, residual connectivity, and factorization of huge convolutions.

2.4 GANs

2.4.1 Laplacian GANs

Navab et al. [8] provides a detailed overview of all recent literature that makes use of GANs in solving major shortcomings in the field of medicine. Major applications of GANs in medical analysis involve conditional synthesis, denoising, reconstruction, classification and segmentation, just to name a few [8]. Out of these applications, Navab concludes that maximum number of GANs methods have been employed in the field of conditional and unconditional synthesis, comprising 35.44% of the total number of methods. Comparing the different types of GANs, Navab indicates that Laplacian GAN (LAPGAN) performs the best in determining conditional and unconditional synthesis.

2.4.2 Feature Map

GANs may be utilized for picture classification jobs even if their primary use is image creation. One method is to employ a GAN generator as a feature map, which may be used to take high-level characteristics from an input picture and utilize them to classify the image [19, 20]. This is accomplished by running a picture and through generator network and utilizing the feature vector produced by one of its intermediary layers [21].

An alternative method is to employ a GAN in a semi-supervised learning environment, where the classifier network is utilized to categorize labeled pictures into distinct categories and to identify actual or false images [22]. When just a little quantity of labelled data is available, this method may be advantageous since the discriminator may use the knowledge from both unlabeled and labeled information to enhance performance of the classifier [23, 24].

The table 1 below summarizes all the related works described above in a terse manner including the scheme, results and challenges of each research.

Table 1: Tabular summary of related works

SCHEME	RESULTS	CHALLENGES
Visual Target Classification using Deep Gaze Pooling [1]	1. Attribute accuracy using max-pooling: 32%±1 2. Attribute accuracy using max-pooling: 34%±1	Differing mental models among different subjects leading to different fixation behavior for same search target
Prediction of Search Targets From Fixations in Open-World Settings [2]	Average accuracy in open – world setting for participants in groups of Amazon: 70.33%, O'Reilly: 59.66%, mugshots: 50.83%	Prediction of classes unknown at training time, hence lesser accuracy as compared to closed – world setting (75%)
Identification of human implicit visual search intention based on eye movement and pupillary analysis [3]	Testing performance of H-SVM for TFVB intent to TOVS intent: 90.02±0.48(%)	1. Used pre-defined areas of Interest (AOIs) for Tobii eye tracker in contrast to real world 2. Considers only stationery visual stimuli 3. Uses Tobii 1750 eye tracking system which is expensive and not portable
Decoding gaze fixations to reveal categorical search targets [5]	Human decoders were able to accurately classify a searcher's target with an average accuracy of 89%	Design constraints due to presence of only two target classes and search displays having segmented objects rather than objects embedded in scenes
GANs for medical image analysis [8]	1. Concluded with high scope of success for GANs in its ability to generate realistic-looking images for increasing training datasets in semi- and unsupervised settings. 2. Considerable success has been made in image-to-image translation	1. GAN training is unstable because of numerical reasons and this leads to convergence issues 2. Computer vision related analysis (especially in the field of medicine) rely more on deep learning methods
Task Classification Model for Visual Fixation, Exploration, and Search [11]	Adaboost model accuracy of 95.4% as compared to SVM (accuracy of 80.3%)	1. Classifying algorithms apart from SVM and Adaboost not compared 2. Limited fixed subset of features tried

3 IMPLEMENTATION PLATFORM AND LIBRARIES USED

3.1 Data and Graph Manipulation Libraries

Python 3 is a powerful and versatile programming language that has become a popular choice for data science projects due to its vast array of libraries and support for machine learning and data analysis. Among the libraries utilized in this project, Tensorflow stands out as a crucial tool for building and simulating deep learning models. Its ability to leverage both CPU and GPU compute resources allows for more efficient and faster training of models, which is especially important for complex projects such as image classification using GANs.

To store and manipulate the data, Pandas and NumPy were also used. NumPy is the best option for processing CSV data since it offers scientific computation and vector handling functions, but Pandas is particularly beneficial for it's own built-in features for data manipulation. The scikit-learn toolkit, which offers a comprehensive workflow across all machine learning techniques and makes it simple to create and test different models, also played a significant part in this project. The successful execution of the image recognition project utilizing GANs was made possible by the integration of these potent libraries.

3.2 Development Platform

3.2.1 Setup on local computer (Windows 10)

A local desktop configuration was used in the project's early phases for activities like data preparation and graph building. The desktop has an i7 CPU, 16 GB of Memory, and an Nvidia GeForce RTX GPU, which gave it the processing capacity required for these computationally demanding activities. The Anaconda environment, a complete distribution of well-liked open-source data analysis and machine learning libraries, was used to get the libraries needed for the project. On their website, Anaconda offers setup instructions that make using and installing the required libraries quick and easy. Using a local desktop gave users more control over their computer resources and eliminated the necessity of cloud computing, which may be expensive and has network speed restrictions and availability. Overall, the local desktop setup with Anaconda proved to be an efficient and reliable platform for the initial stages of the project.

3.2.2 Jupyter Notebook (Anaconda)

The open-source online application Jupyter notebook, created by Project Jupyter, enables GPU-accelerated deep learning. This platform's pre-installed libraries, including TensorFlow, Keras, PyTorch, and Pandas, that are crucial for creating deep learning applications, are one of its standout benefits. It is crucial to remember that there are particular limitations on session length and file size, which may make it less useful for some users. Jupyter notebook is a fantastic tool for creating algorithms for deep learning that take advantage of GPUs despite these drawbacks.

For novices, the documentation available on the official website of Jupyter notebook might be a great place to start. It provides descriptions of the platform's many features and functions as well as step-by-step instructions for setting up and using Jupyter notebook. Furthermore, there are a ton of online tutorials and instructions that can teach you how to utilize Jupyter notebook for deep project - based learning, including how to create and train neural networks, handle and analyze massive datasets, and launch your models for practical use.

4 DATASET

The dataset we use for this research is the same as used by Kumar et at. [11] which was inturn borrowed from Otero-Millan et al's study [12]. The four categories of target stimuli are - Blank, Natural, Puzzle and Waldo. Blank scene is merely a plain screen (50% gray) presented to the user. Natural image is one where there is a scene that the user is asked to observe. Puzzle indicates two similar images placed side by side and the user is asked to find the differences between the two. Lastly, Waldo is derived from the game "Where's Waldo" where the user is asked to find the character Waldo, i.e., find an object amidst a group of objects. Figure 8 below shows four sample of images belonging to the four different labels presented to user for different eye scan paths.

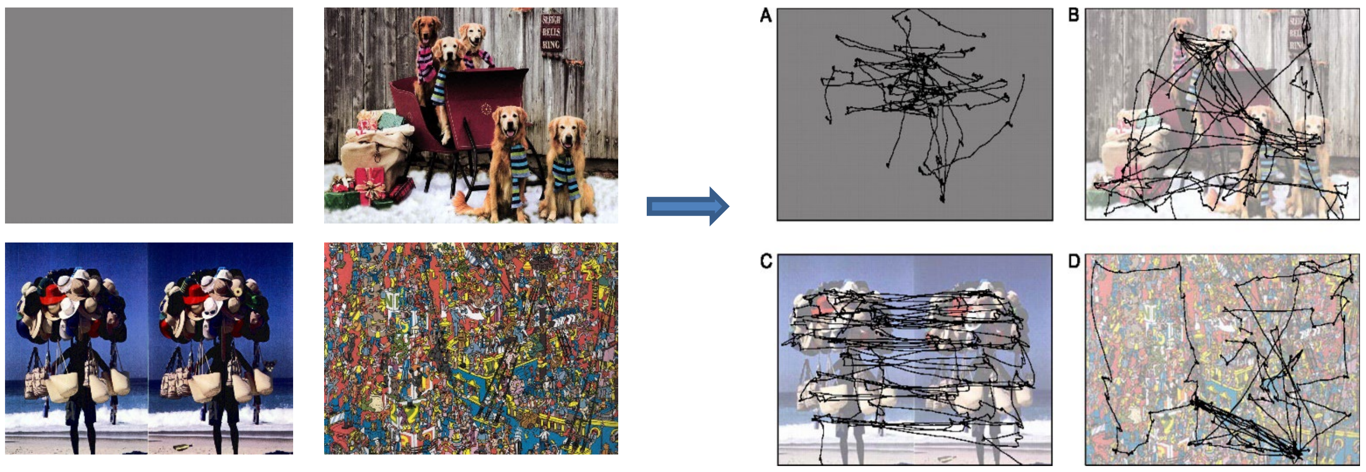
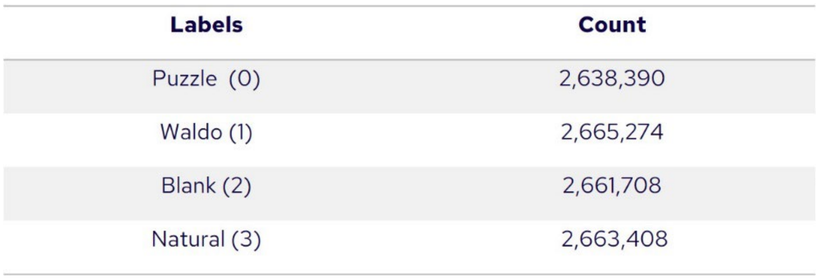


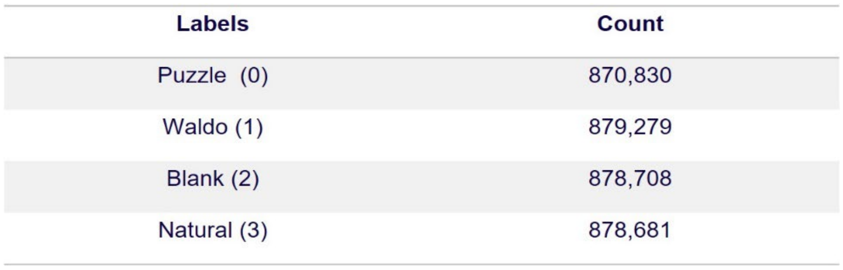
Figure 8: Sample images from the dataset presented to user for different eye scan paths [12]

For this research, the resolution of every image was fixed to 921 x 630 pixels. The images after data augmentation with DC-GANs was at a size of 500 images dataset. Every trial lasted for about 45 seconds gathering a result of approximately 10 million data points for each for the numerical eye-tracking, which are nothing but the data points of user scan paths. This balanced distribution of our dataset before performing train test split is shown in the Figure 9 below. Whereas, the dataset count for each of the labels after test split of 33% is shown in Figure 10.



Labels	Count
Puzzle (0)	2,638,390
Waldo (1)	2,665,274
Blank (2)	2,661,708
Natural (3)	2,663,408

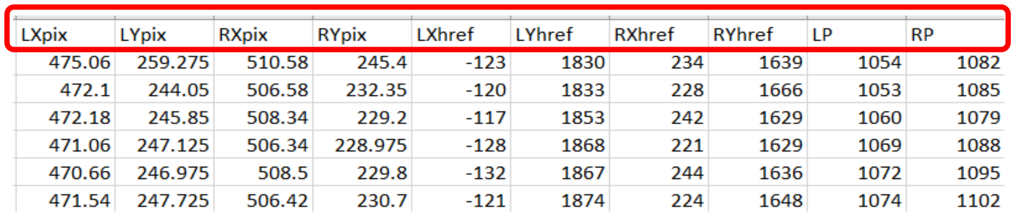
Figure 9: Total dataset count before performing train-test split



Labels	Count
Puzzle (0)	870,830
Waldo (1)	879,279
Blank (2)	878,708
Natural (3)	878,681

Figure 10: Total test - dataset count after performing 33% test split

In our research, we try to implement different types of inputs based on the same user dataset available to us. In the first case, we pass gaze fixation points, i.e., raw numerical data (in CSV format) to different ensemble models and compare their accuracies. A few of the rows of a sample CSV file of our initial eye-tracking dataset is shown in Figure 11.



LXpix	LYpix	RXpix	RYpix	LXhref	LYhref	RXhref	RYhref	LP	RP
475.06	259.275	510.58	245.4	-123	1830	234	1639	1054	1082
472.1	244.05	506.58	232.35	-120	1833	228	1666	1053	1085
472.18	245.85	508.34	229.2	-117	1853	242	1629	1060	1079
471.06	247.125	506.34	228.975	-128	1868	221	1629	1069	1088
470.66	246.975	508.5	229.8	-132	1867	244	1636	1072	1095
471.54	247.725	506.42	230.7	-121	1874	224	1648	1074	1102

Figure 11: Few rows of eye-tracking data stored in a sample CSV file

In the second case, we pass direct raw images in our dataset as input to the DC-GANs model for data augmentation and then to the ensemble learning models.

4.1 Challenges of the given dataset

The provided dataset includes many CSV files relating to the user gaze scan path and raw images. The classification may be done with both CSV data and raw images. The CSV files include several contain records having NaN values and unusual characters. Data processing is required in order to use the CSV files as input to our machine learning models. The raw images provided in the dataset that were used as input to our ensemble models were restricted in number, with a dataset of only 50 images using DenseNet201 with transfer learning produced results with extremely low accuracy, almost close to 1%.

The figure 12 indicates the training and validation accuracy across 11 epochs. To improve accuracy, we undertook data augmentation to boost the number of images.

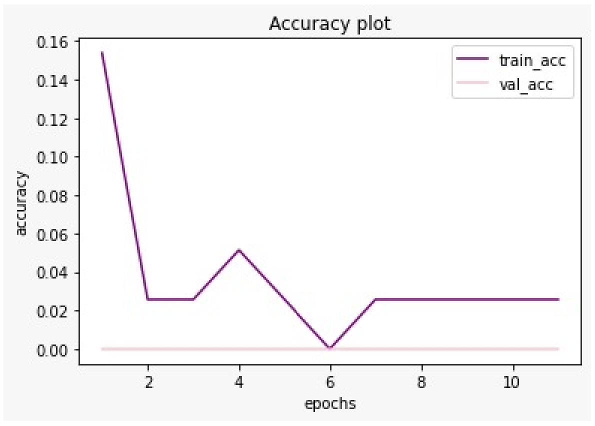
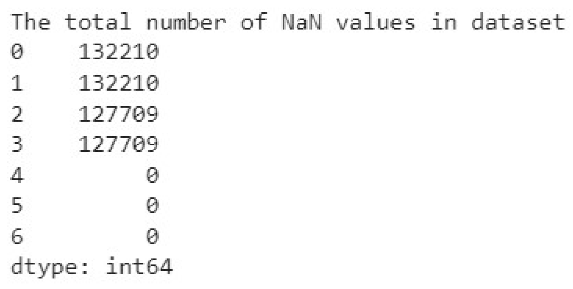


Figure 12: Accuracy plot of DenseNet201

4.2 Data Cleaning

Data integration is performed to aggregate data from multiple CSV files into one dataframe. The dataset contains multiple NaN values and unusual characters. The missing values corresponded to the data points where the user looked outside of the viewing area. Since this affected model accuracy, we eliminated such rows using basic Python libraries. These significantly reduce the accuracy of a model, hence data cleaning is necessary. Figure 13 below indicates the number of NaN across different columns of data belonging to the whole dataset. After data cleaning, the below result is achieved showing successful removal of NaN values.

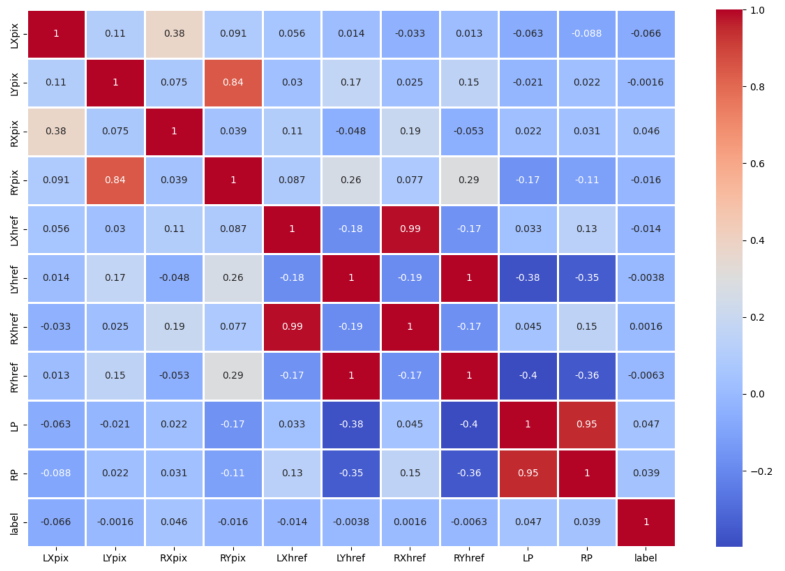


```
The total number of NaN values in dataset
0    132210
1    132210
2    127709
3    127709
4         0
5         0
6         0
dtype: int64
```

Figure 13: Total number of NaN values in the entire dataset

4.3 Data Pre-processing – Correlation Analysis

We used the Pearson correlation matrix to plot the correlation between different features of our initial eye-tracking dataset. The Pearson correlation coefficient is a measure of the linear relationship between two variables, which ranges from -1 to 1. A correlation coefficient of -1 indicates a perfect negative correlation, meaning that as one variable increases, the other decreases in a perfectly linear fashion. A coefficient of 0 indicates no linear correlation between two variables, while a coefficient of 1 indicates a perfect positive correlation, meaning that as one variable increases, the other also increases in a perfectly linear fashion. The correlation between features of the user gaze fixation points is plotted to shows the dependency of different attributes on one another as shown below in Figure 14.



	LXpix	LYpix	RXpix	RYpix	LXhref	LYhref	RXhref	RYhref	LP	RP	label
LXpix	1	0.11	0.38	0.091	0.056	0.014	-0.033	0.013	-0.063	-0.088	-0.066
LYpix	0.11	1	0.075	0.84	0.03	0.17	0.025	0.15	-0.021	0.022	-0.0016
RXpix	0.38	0.075	1	0.039	0.11	-0.048	0.19	-0.053	0.022	0.031	0.046
RYpix	0.091	0.84	0.039	1	0.087	0.26	0.077	0.29	-0.17	-0.11	-0.016
LXhref	0.056	0.03	0.11	0.087	1	-0.18	0.99	-0.17	0.033	0.13	-0.014
LYhref	0.014	0.17	-0.048	0.26	-0.18	1	-0.19	1	-0.38	-0.35	-0.0038
RXhref	-0.033	0.025	0.19	0.077	0.99	-0.19	1	-0.17	0.045	0.15	0.0016
RYhref	0.013	0.15	-0.053	0.29	-0.17	1	-0.17	1	-0.4	-0.36	-0.0063
LP	-0.063	-0.021	0.022	-0.17	0.033	-0.38	0.045	-0.4	1	0.95	0.047
RP	-0.088	0.022	0.031	-0.11	0.13	-0.35	0.15	-0.36	0.95	1	0.039
label	-0.066	-0.0016	0.046	-0.016	-0.014	-0.0038	0.0016	-0.0063	0.047	0.039	1

Figure 14: Feature correlation between different attributes of the dataset

5 APPROACHES AND IMPLEMENTATION

There were many machine learning techniques and manually constructed feature extraction methods that were previously used to classify images. Image preprocessing to eliminate noise and reduce the photos to a uniform size was one phase in the process. Another step was feature extraction, which entailed manually creating a collection of features to describe the image. Local Binary Patterns, Scale-Invariant Feature Transforms, and Histogram of Oriented Gradients were frequently used feature extraction techniques. Following the extraction of the features, a machine learning algorithm—such as SVM Classifier, Random Forest, or k-Nearest Neighbors—was trained on a labeled dataset of photos. After being trained, the model was able to identify the class of fresh photos by identifying their attributes. The development of the proper feature extraction algorithms for this method took a lot of effort and skill. However, deep learning, on the other hand, has made the procedure easier by enabling computers to pick up the features on their own.

In this project, two broad methods are utilized for achieving comparative analysis between them. The first method consist of using numerical eye-tracking dataset stored in CSV files and passing them as input to train our ensemble models. The second method consists of taking raw images as input and passing them to the ensemble models to see in order to form strong comparative analysis between the working and performance of different ensemble approaches. We also propose a novel approach to perform a cohesive incorporation three out of four ensemble models as to determine the accuracy of this hybrid model. Additionally, the research also takes into account the use of DC-GAN for the purpose of data augmentation. The flowchart for the complete execution of the project is shown below in Figure 15.

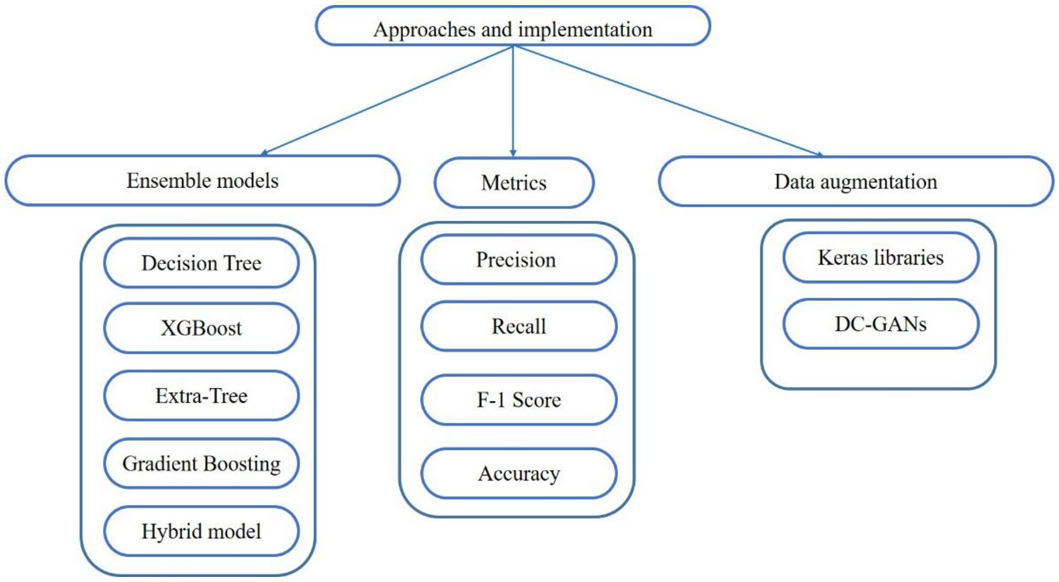


Figure 15: Experimental flowchart of the different technical aspects of project

5.1 Decision Tree Ensemble model

A very powerful ensemble learning approach for classification and regression is the decision tree. With ‘n’ nodes the data characteristics and edges representing the decision rules, this supervised learning technique divides the data into smaller groups using binary judgments. The most informative features are found and a stopping requirement, such as a maximum depth or a minimum sample size, is satisfied while this process iteratively proceeds [6]. Figure 16 below illustrates the same.

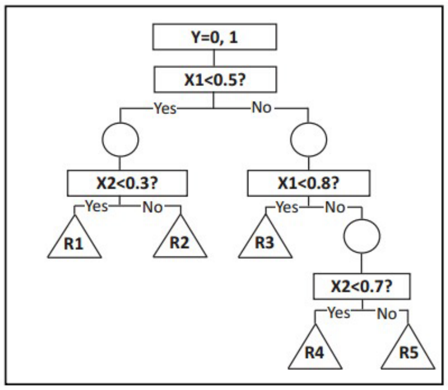


Figure 16: Sample decision tree based on binary target variable Y [28]

The fundamental decision tree model presented in Figure 15 consists of two dependent variable, x1 and x2, having values ranging from 0 to 1, as well as one goal variable Y (which might have a value of zero or one). The nodes and branches of a decision tree structure are its crucial components, and crucial processes including data division, setting halting criteria, and pruning are necessary for its formation [7].

In the Decision Tree model, every eye tracking data is read and file and features are extracted along with label values for each data sample in our dataset. This is done for all four label classes namely “Blank”, “Natural”, “Waldo” and “Puzzle”, for the numerical eye-tracking data as well as image data. For this model, the dataset is split into test and train dataset with 67% train and 33% test sizes respectively.

5.1.1 Definitions and nomenclature

Nodes: Several sorts of nodes may be recognized in a decision tree. The root of the tree, which is sometimes called the decision node, comes first [12]. The selection represented by this node will divide the data into several, mutually exclusive subgroups. Second, internal nodes exist, which are also known as risk junctions [9, 13, 28]. These nodes reflect one of the options that are accessible at that particular branch in the binary tree. They are connected to its parent node at the top and their leaf node or leaf nodes at the bottom via an edge. The leaf nodes, often referred to as end nodes, come last. These nodes show how a chain of choices ultimately turned out [28].

Branches: The root nodes and intermediate nodes of the decision tree model have a network of branches that reflect potential outcomes or occurrences [14, 15]. A classification rule is represented by each route from the root node via the hidden layers to a leaf node. If-then statements, where a given combination of criteria results in a certain conclusion, can be used to define these decision rules [28]. For instance, the outcome will be outcome j if a specific set of circumstances, like condition 1, condition 2, etc., are satisfied.

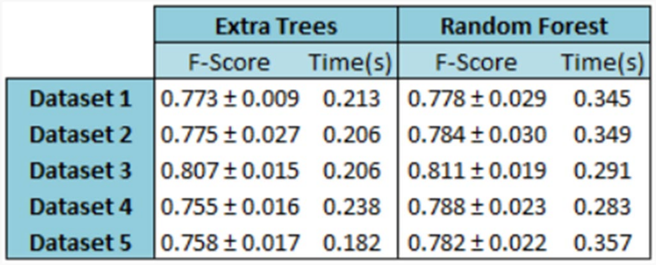
Splitting: Only input parameters that have a connection with the attribute value are taken into account when dividing the parent nodes, which results in purer child nodes that depend on the target parameter [12, 13, 15]. It is required to decide which input variables would be most crucial before dividing the data at the root node and succeeding internal nodes into different categories or 'bins' according to the current state of these variables [28].

Stopping: Stopping in a decision tree refers to the process of determining when the tree should stop growing and instead become a final decision model. Complexity and robustness are conflicting qualities that must be taken into account when developing a statistical model [28]. In some instances, a decision tree model could be constructed in such a way as to provide pure leaf nodes, where every record has the desired outcome. This strategy is severe, though, and it might not work in every circumstance [28, 29].

5.2 Extra-Tree Ensemble Model

This algorithm has been widely used in a variety of applications, including the classification of land cover using Extremely Randomized Trees [31, 32, 33] and the development of a multi-layer detection system for intrusion using Extra Trees feature selection, an ensemble learning machine outfit, and softmax aggregation. The comparison between the ExtraTree classifier and the Random Forest classifier using the F-1 score as a measure and for five distinct datasets is shown in the table 2 below.

Table 2: Tabular summary of F-1 score and execution time between Random Forest and Extra Trees algorithm



	Extra Trees		Random Forest	
	F-Score	Time(s)	F-Score	Time(s)
Dataset 1	0.773 ± 0.009	0.213	0.778 ± 0.029	0.345
Dataset 2	0.775 ± 0.027	0.206	0.784 ± 0.030	0.349
Dataset 3	0.807 ± 0.015	0.206	0.811 ± 0.019	0.291
Dataset 4	0.755 ± 0.016	0.238	0.788 ± 0.023	0.283
Dataset 5	0.758 ± 0.017	0.182	0.782 ± 0.022	0.357

In our Extra-Tree classifier, our ensemble model is hence constructed. The dataset is divided with a train-test split of 67% and 33% respectively. This is done for all four label classes namely "Blank", "Natural", "Waldo" and "Puzzle", for the numerical eye-tracking data as well as image data.

The Extra-Trees classifier uses a majority vote to aggregate all of the trees' predictions to get the final result [29, 30, 31]. This method's basic premise is that ensemble averaging and fully randomizing the picking of characteristics and cut-points together substantially minimize variation compared to weaker randomization techniques utilized in other approaches [33]. The approach does not employ bootstrap replicas but rather all of the unique training data to minimize bias.

One of the main advantages of the Extra-Trees classifier is that it is recognized for being computationally efficient [31, 32]. This algorithm has been widely used in a variety of applications, including the classification of land cover using Extremely Randomized Trees [15, 16, 33] and the development of a multi-layer detection system for intrusion using Extra Trees feature selection, an ensemble learning machine outfit, and softmax aggregation.

5.3 Gradient Boosting Ensemble Model

Gradient boosting machines are a collection of powerful machine learning techniques that have shown notable success across a range of real-world applications [1, 2, 34]. They are quite adaptable when it comes to conforming to the particular application's requirements, such as being learned using different loss functions.

Gradient boosting machines, or GBMs for short, employ a sequential learning procedure in which new models are fitted one after the other to increase the precision of the predicted dependent variables [24, 35, 36]. Designing the new base-learners to have a strong correlation with the ensemble's extremely negative gradient of the error function is the key idea behind this technique [36]. Although the selection of the loss function might be arbitrary, it is important to keep in mind that the learning process would include fitting mistakes one after the other if the utility function is the conventional squared-error loss [34, 37].

With the variety of activation functions which have been produced to date and the ability to create a tailored loss function suited to the particular job, choosing which loss function value to utilize is typically left up to the researcher's choice [36, 37, 40].

In our project, Gradient Boosting algorithm is combined with ***MinMaxScaler*** from the ***sklearn*** module. When using ***MinMaxScaler*** on a feature, each value in the feature is first subtracted by the minimum value of the feature, and then divided by the range [35]. The range, in turn, is calculated as the difference between the original maximum and minimum values of the feature. Importantly, ***MinMaxScaler*** does not alter the shape of the original distribution of the feature, but rather scales it uniformly to fall within the range of 0 and 1 [36, 37].

The figure 17 below shows a diagrammatic explanation of simplistic working of Gradient Boosting algorithm.

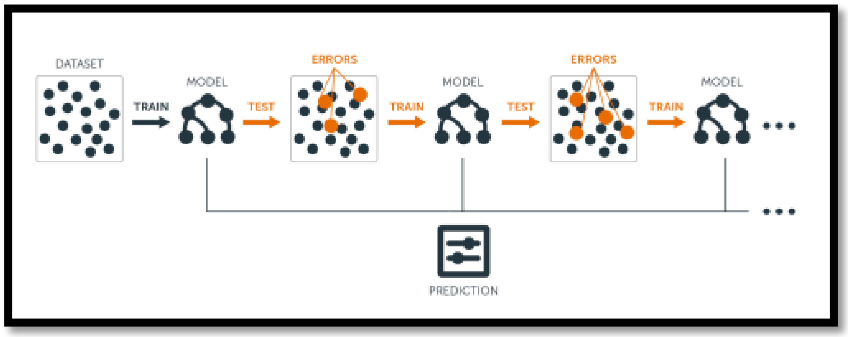


Figure 17: Diagrammatic explanation of Gradient Boosting [38]

Iteratively building an ultimate ensemble model while employing a loss function that has been tuned via gradient descent, gradient boosting is a remarkable ensemble approach since it acknowledges the limits of weak models.

Gradient boosting frequently uses decision trees as the weak learners, thus the other term "gradient tree boosting" for the method. Gradient boosting regression uses a front stagewise modeling method to calculate the actual values, in contrast to AdaBoost, which modifies the weights of training instances to improve prediction performance. Additional decision trees are then fitted to minimize the disparity between the expected and actual values. To lessen the residual's negative gradient, the newly created weak learners are added to the ensemble. This process is called “gradient descent”.

5.3.1 Procedure

1. Setting the loss function is the first step in creating a gradient boosting regression model.

2. Next, determine the negative gradient to determine the current regression model's residual error (r_{ij}).

3. Create a regression tree using the residual r_{ij} as a starting point, and then identify the leaf

region R_{ij} for tree i.

4. To get the loss function's minimal value, do a linear search.

5. Replace the previous ensemble model of the regression tree with a new decision tree.

6. Keep iterating over n cycles to build the ultimate gradient boosting regression model.

In this model too, raw data is passed as input for image classification for four labels/classes and for the image data, the classification is performed for three labels/classes. We use the ***MinMaxScaler*** to scale both the X and Y train sets of the same size as vectors.

5.4 XGBoost Ensemble Model

Despite its superior performance compared to existing gradient boosting implementations, XGBoost can take a very long time to start [39]. A typical task can take hours or days to complete. Generating highly accurate models using gradient boosting requires careful parameter tuning. During this process, we need to run the algorithm multiple times to explore the effect of parameters such as learning rate and L1/L2 regularization conditions on cross-validation accuracy [39, 40]. The basic architecture of XGBoost model is shown in Figure 18.

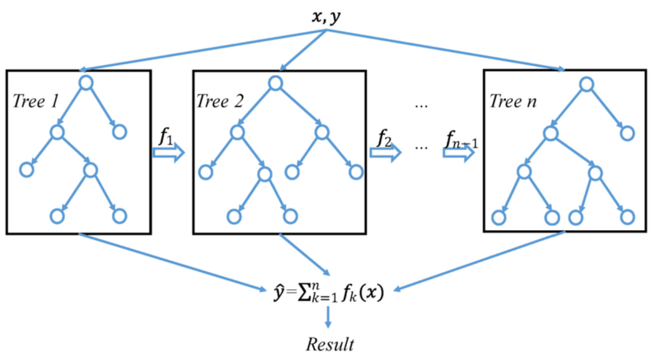


Figure 18: Architecture of XGBoost ensemble model [39]

In the XGBoost ensemble model for this project, the four labels/classes are taken into account with the '***unicode_escape***' encoding and labels taken as 0, 1, 2, 3. For this model, the dataset is split as 70% for training and 30% for testing datasets. At first the learning rate was set to 0.05 and then slowly increased during hyper-parameter tuning.

XGBoost [40] is one of the boosting tree models with the strongest expansion and adaptability. To create a more robust learner model, it incorporates many tree models. A further characteristic of XGBoost is its capacity to automatically utilize the CPU's multithreading for parallel processing, which can accelerate the calculation. Just the first derivative information is used in conventional Boosting Tree models [40, 41]. The residual of the previous n-1 trees is employed while training the nth tree, making dispersed training challenging to implement.

The loss function is expanded by XGBoost in a second-order Taylor manner, and it may automatically utilize the CPU's multithreading for parallel processing. Moreover, XGBoost employs a number of techniques to prevent overfitting [39, 40]. All the above ensemble models are first run standalone on the inputs of image dataset and raw CSV file dataset and then in integration through the same datasets described above.

5.5 Proposed Hybrid Model

In addition to the previously mentioned ensemble models, we proposed a new ensemble model consisting of three of the above models – Decision Tree, Extra Tree and XGBoost models. This was done to improve the accuracy of our predictions and evaluate the impact of increasing model complexity. The hybrid model approach aims to leverage the unique strengths of each algorithm and combine them in a way that maximizes performance. We attempted to combine all four models but it proved too computationally expensive and we were unable to complete the training of the entire dataset given our available resources.

Firstly, the decision tree model is useful for handling non-linear data and can generate understandable rules for feature selection. It splits data based on the most significant feature, which makes it easy to interpret and visualize. However, decision trees are prone to overfitting, where the model learns too much from the training data and performs poorly on the test data.

To address this issue, we use the extra tree model, which is similar to the decision tree model but reduces variance by randomly selecting features and split points. This helps to improve the accuracy of the decision tree model and prevent overfitting.

Finally, the XGBoost model is a popular gradient boosting algorithm that can provide better regularization and prevent overfitting. It considers the weights of the samples during training, which can improve accuracy. Additionally, XGBoost can handle missing values, which is particularly useful for image classification tasks where there may be missing pixels or corrupted data. By combining these models in a hybrid ensemble, we can leverage the strengths of each individual algorithm and achieve better performance than using a single model alone.

The Figure 19 below shows a simple conceptual diagrammatic representation of the conjunction of all the three ensemble models. This was essentially called an "ensemble of ensemble models".

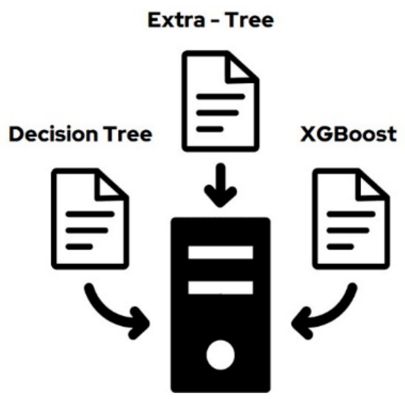


Figure 19: Diagrammatic representation of hybrid model

5.6 DC-GANs Model

The creation of GANs that use generative models to generate new data is one of a million ways to generate massive amounts of data [17]. GANs are also used to transform low-resolution photos into high-resolution ones. Ledig et al. [18] introduced SRGAN for super-resolution (SR) imaging.

The authors of [20] developed a conditional GAN to generate high-resolution images without a trained network. Most of the experiments mentioned above focused only on turning low-resolution photos into high-resolution images, but GANs can also generate fake data that looks real. Denton et al. [19] used a convolutional network to build an image inside a Laplacian pyramid.

Deep Convolutional GANs (DCGANs), which employ typical and fractional convolutional layers for downsampling and upsampling, respectively, to produce high-resolution pictures, were first described by Radford et al. [22]. By applying GANs, Rawat and Shen [23, 24] created novel topology design techniques to provide the best designs for buildings.

In our DC-GAN model, we perform data augmentation to increase the size of our dataset, as our dataset was limited in size. We utilize convolutional layers with sigmoid and ReLU as loss and activation functions respectively. The DC-GAN model consists of generator and discriminator following the same loss and activation functions described above. After training the Discriminator and Generator in separate functions we download the generated images in a directory, compute the loss, and real score parameters. We train these images for a total of 340 epochs and then test the images generated. The ultimate goal is to generate similar looking images as our dataset so as to increase our total dataset size to 500 and hence increase accuracy. The figure 20 below gives a brief description of the GANs generator and discriminator model. It clearly depicts how random noise is fed to the generator and then the generated images are compared with the training set till a point where the discriminator cannot differentiate between the real and fake images.

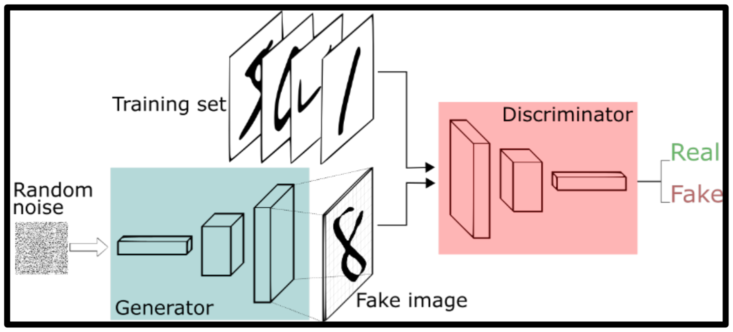


Figure 20: Working of GANs [40]

GANs generates both valid and invalid data, and then the discriminator performs a data analysis pattern to spurn out the valid from the invalid data.

This process continues till the discriminator is no longer able to differentiate between valid and invalid data. This can be done subdividing and bifurcating the data into new branches. One of the ways to perform this is clustering analysis [17]. This helps the users understand the entire profile of the dataset available by the process of coalescing similar data into one group and so on. In the past too, there have been multiple studies to determine clusters of such groups [20, 21].

The authors of [20] applied different methods to detect World Wide Web unauthorized data by using the method of K-means clustering and self-organizing map (SOM). This was in turn done for visualizations purposes in the typhoon image collection. There have been multiple studies in the past that have combined GANs model along with clustering analysis for various kinds of topology optimization problems.

6 EXPERIMENTS

6.1 Metrics

In this project we have built multiple models for both our eye-tracking and raw image dataset. The resolution of every image was fixed to 921 x 630 pixels. Every trial lasted for about 45 seconds gathering a result of 22500 data points per CSV file, which are nothing but the data points of user scan paths.

In our research, we attempt to develop several input formats based on the same user dataset that is at our disposal. In the first instance, we subject several ensemble models to gaze fixation points, or raw numerical data (in CSV format), and compare the accuracy of the results. In the second scenario, we feed DC-GANs models and ensemble learning models directly raw pictures from our dataset.

In this project we have used the following metrics:

1) Precision
2) Recall
3) F-1 Score
4) Accuracy

- Precision

  There are many ways to calculate the performance of a model and one of them is precision. It is calculated by the total number of positives divided by the total number of positives summed with the total number of false positives (Figure 21).

$$precision = \frac{tp}{tp + fp}$$

Figure 21: Precision in machine learning

- Recall

  Similarly, there are multiple ways of finding out recall for different multi-label classification problems. It is calculated by the total number of true positives divided by the sum of true positives and false negatives (Figure 22).

$$recall = \frac{tp}{tp + fn}$$

Figure 22: Recall in machine learning

- F-1 Score

  It combines a model's accuracy and recall ratings. It defines the number of times a model can correctly predict the full dataset. It is computed as the harmonic mean of precision and recall (Figure 23).

$$F_1 = 2 * \frac{Precision * Recall}{Precision + Recall}$$

Figure 23: F-1 score in machine learning

- Accuracy

  The accuracy of a model determines the total number of correct predictions made by the respective model(s). That is, the sum of true negatives and the true positives divided by the sum of all the predictions (Figure 24).

$$\text{Accuracy} = \frac{\text{Number of correct predictions}}{\text{Total number of predictions}}$$

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN}$$

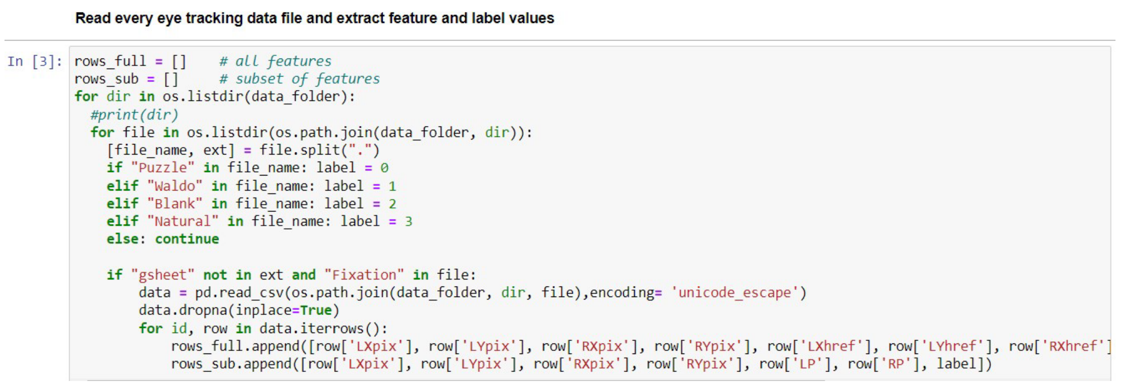
Figure 24: Accuracy in machine learning

6.2 Experiments

Experiments were conducted by passing input data first to the four ensemble models – both raw and image data and then data augmentation was performed using DC-GANs model, after which the data was again passed as input to the four ensemble models for comparing the metrics described above. Moreover, a hybrid model was proposed comprising of the four ensemble models in which both raw and image data was passed as input.

- Ensemble models

In the Decision Tree model, read every eye tracking data and file and extract feature is performed along with label values. This is done for both numerical eye-tracking data as well as the image dataset. The dataset is split into test and train dataset with 67% train and 33% test sizes respectively.


Read every eye tracking data file and extract feature and label values

```
In [3]: rows_full = []    # all features
        rows_sub = []     # subset of features
        for dir in os.listdir(data_folder):
          #print(dir)
          for file in os.listdir(os.path.join(data_folder, dir)):
            [file_name, ext] = file.split(".")
            if "Puzzle" in file_name: label = 0
            elif "Waldo" in file_name: label = 1
            elif "Blank" in file_name: label = 2
            elif "Natural" in file_name: label = 3
            else: continue

            if "gsheet" not in ext and "Fixation" in file:
                data = pd.read_csv(os.path.join(data_folder, dir, file),encoding= 'unicode_escape')
                data.dropna(inplace=True)
                for id, row in data.iterrows():
                    rows_full.append([row['LXpix'], row['LYpix'], row['RXpix'], row['RYpix'], row['LXhref'], row['LYhref'], row['RXhref']
                    rows_sub.append([row['LXpix'], row['LYpix'], row['RXpix'], row['RYpix'], row['LP'], row['RP'], label])
```

Figure 25: Reading eye tracking data file and extracting feature and label values

The code in the Figure 25 above illustrates how to parse eye tracking data files and extract feature and label values from them. Eye tracking data files generally comprise a time-stamped series of gaze coordinates from which different parameters such as fixation length, saccade velocity, and pupil diameter may be extracted.

Extraction of features and labels from eye tracking data generally consists of many processes, including data cleaning and preprocessing, feature extraction, and labeling. Data cleaning and preprocessing may include the removal of outliers, the filtering of noise, and the interpolation of missing data points. Typically, feature extraction entails extracting several statistical measurements and metrics from raw eye tracking data, such as mean fixation length or frequency of saccades.

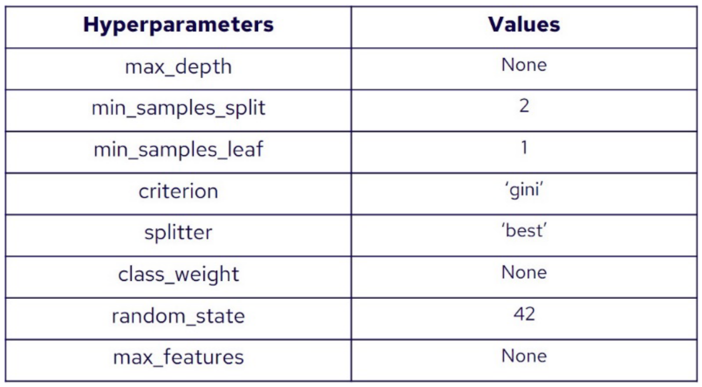
This code defines two empty lists called ***rows_full*** and ***rows_sub***. It then iterates over the files in the directory specified by ***data_folder***.

For each file, it checks if the file name contains certain keywords *(**"Puzzle", "Waldo", "Blank"**, or **"Natural"**)* to determine the label for the data.

For each row in the DataFrame, the code appends the values of the columns ***LXpix***, ***LYpix, RXpix, RYpix, LXhref, LYhref, RXhref, RYhref, LP, RP***, and the label to the ***rows_full*** list as a list. It also appends the values of the columns ***LXpix, LYpix, RXpix, RYpix, LP, RP***, and the label to the ***rows_sub*** list as a list.

Overall, this code reads fixation data from files in a directory and extracts certain columns, creating two lists: one with all columns and one with a subset of columns.

The different hyper-parameters and their corresponding values for the Decision Tree are shown below in Figure 26.



Hyperparameters	Values
max_depth	None
min_samples_split	2
min_samples_leaf	1
criterion	‘gini’
splitter	‘best’
class_weight	None
random_state	42
max_features	None

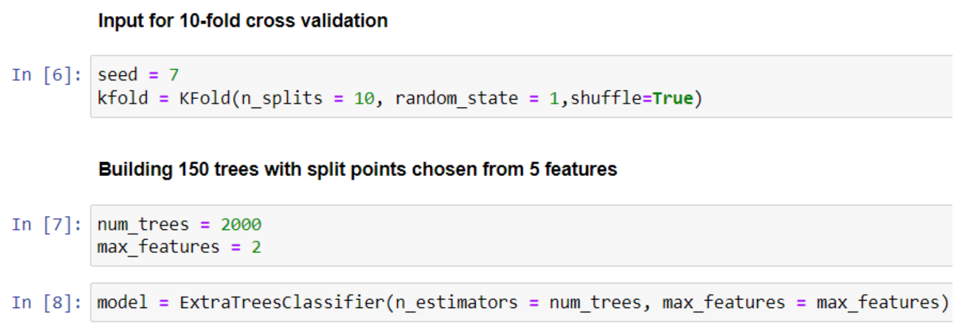
Figure 26: Different hyper-parameters and their values for Decision Tree

We employed ***GridSearchCV*** for decision tree-based image classification, a parameter grid is usually defined to specify the range of values to explore for each hyperparameter. The ***GridSearchCV*** algorithm performs a systematic search of the hyperparameter space, evaluating the performance of the decision tree model with each combination of hyperparameters using cross-validation.

We experimented with different hyper-parameters and their values. To arrive at the optimal values for the hyper-parameters, we used a hit-and-try method and ran experiments across multiple epochs. Through this process, we carefully selected values that resulted in the best performance compared to other values that we tested. Here are the hyper-parameters and their respective values for the decision tree model.

The criterion was set to “gini” which means the algorithm tries to minimize the Gini impurity of the samples in each node. Gini impurity is a measure of the probability of misclassifying a randomly chosen element from the set. This worked for our dataset as compared to the “entropy” value because we had a large amount of raw data with balanced number of labels. Splitter determines the strategy used to choose the split at each node. The best splitter (splitter='best') chooses the best split amongst all possible splits and hence leads to homogenous subsets.

In our Extra-Tree classifier, we initially used an input of 10-fold cross validation is used with a combination of a total of 2000 individual decision trees and a maximum of 2 features. Our ensemble model is hence constructed. The dataset is divided with a train-test split of 67% and 33% respectively. Due to resource limitations and constraints, these parameters did not prove to be so efficient and was causing overfitting, hence we had to employ hyper-parameter tuning and set suitable values to achieve considerable performance out of the model.



Input for 10-fold cross validation

```
In [6]: seed = 7
        kfold = KFold(n_splits = 10, random_state = 1,shuffle=True)
```

Building 150 trees with split points chosen from 5 features

```
In [7]: num_trees = 2000
        max_features = 2

In [8]: model = ExtraTreesClassifier(n_estimators = num_trees, max_features = max_features)
```

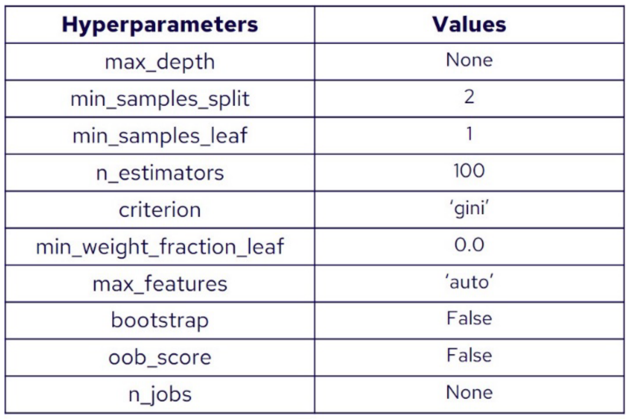
Figure 27: Building Extra-Tree model with 2000 trees

In the provided code in Figure 27, the variable `***seed***` is set to 7 to ensure reproducibility. Then, a `***KFold***` object called `***kfold***` is initialized to split the data into 10 folds for cross-validation. The `***random_state***` parameter is set to 1 to ensure that the same splits are used each time, and `***shuffle=True***` indicates that the data should be shuffled before splitting.

The number of trees in the Extra Trees Classifier is set to 2000 using the variable `***num_trees***`, and the maximum number of features used at each split is set to 2 using `***max_features***`. Extra Trees Classifier model is then initialized with these parameters and assigned to the variable `model`. This code initializes a `***KFold***` object and the model with specified parameters, which can be used for cross-validation and classification tasks.

During hyper-parameter tuning, for the experimentation process of extra tree classifier, the hyper-parameter values were used as shown in Figure 28. In particular, the out of bag score which is an inbuilt mechanism to calculate the performance of the model , was set to false because we wanted to test the performance of the model during the testing phase and computing the out-of-bag score increases the computational cost, as the model needs to be trained multiple times to generate the out-of-bag samples.

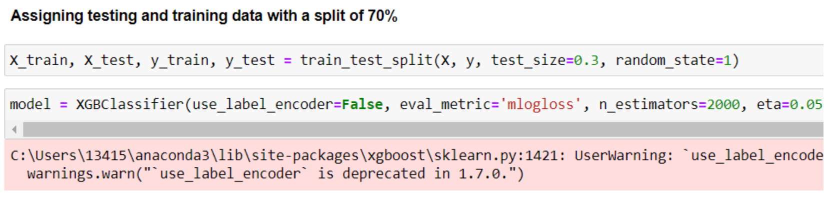
The boostrap parameter is set to False, because we had a huge training dataset and this was beneficial as each learner was built on the entire training dataset without any resampling.



Hyperparameters	Values
max_depth	None
min_samples_split	2
min_samples_leaf	1
n_estimators	100
criterion	'gini'
min_weight_fraction_leaf	0.0
max_features	'auto'
bootstrap	False
oob_score	False
n_jobs	None

Figure 28: Different hyper-parameters and their values for Extra Tree

In the XGBoost ensemble model for this project, the four labels/classes are taken into account with the 'unicode_escape' encoding and labels taken as 0, 1, 2, 3. For this model, the dataset split is done as 70% for training and 30% for testing datasets. At first, the learning rate was set to 0.05 and then slowly increased during hyper-parameter tuning.


Assigning testing and training data with a split of 70%

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=1)
```

```
model = XGBClassifier(use_label_encoder=False, eval_metric='mlogloss', n_estimators=2000, eta=0.05
```

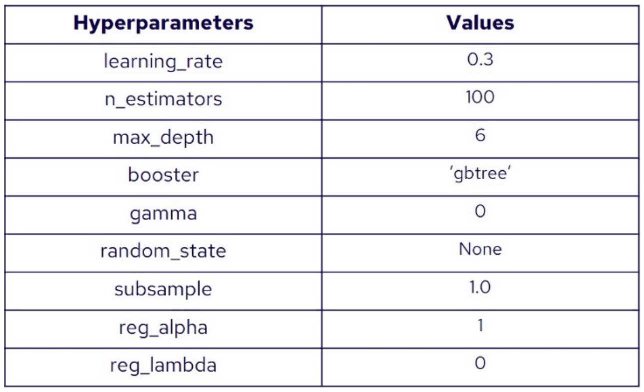
```
C:\Users\13415\anaconda3\lib\site-packages\xgboost\sklearn.py:1421: UserWarning: `use_label_encode
  warnings.warn("`use_label_encoder` is deprecated in 1.7.0.")
```

Figure 29: Initial run of XGBoost model before hyper-parameter tuning

The given code utilizes the ***train_test_split*** function from scikit-learn's ***model_selection*** module to split the data into training and testing sets. The feature matrix is stored in ***X***, and the corresponding labels are stored in \`***y***\`. The data is split in a 70:30 ratio between training and testing sets using the ***test_size*** parameter, with a fixed ***random_state*** value of 1 for reproducibility.

XGBClassifier model is then initialized using the values shown below in Figure 28. By using these values, we tried to combat overfitting as far as possible whilst maintaining model performance. The parameter ***booster*** is the type of model to use for boosting. In XGBoost, there are two types of boosters: ***gbtree*** and ***gblinear***. ***gbtree*** trains decision tree-based models. On the other hand, ***gblinear*** trains a linear model. Setting the parameter to ***gbtree*** worked better in our case because of its higher complexity as compared to ***gblinear*** that is generally used for smaller datasets that require faster models. The parameter ***gamma*** is the minimum loss reduction required to partition a leaf node. We set it to 0 which means no minimum loss reduction is required and a split is made at every leaf node. This encouraged more randomization for our model.

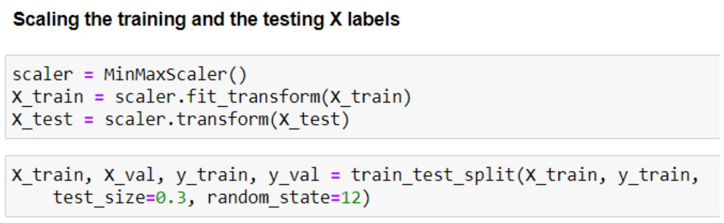
reg_alpha and ***reg_lambda*** are the L1 and L2 regularization terms on weights to prevent overfitting. Since we have a large dataset, to enhance better feature selection we set the ***reg_alpha*** to 1 and ***reg_lambda*** to 0 to reduce expensive computation.



Hyperparameters	Values
learning_rate	0.3
n_estimators	100
max_depth	6
booster	'gbtree'
gamma	0
random_state	None
subsample	1.0
reg_alpha	1
reg_lambda	0

Figure 30: Different hyper-parameters and their values for XGBoost

In our project, Gradient Boosting algorithm is combined with ***MinMaxScaler*** from the sklearn module. When using ***MinMaxScaler*** on a feature, each value in the feature is first subtracted by the minimum value of the feature, and then divided by the range [35]. The range, in turn, is calculated as the difference between the original maximum and minimum values of the feature. Importantly, ***MinMaxScaler*** does not alter the shape of the original distribution of the feature, but rather scales it uniformly to fall within the range of 0 and 1 [36, 37] as shown in Figure 31.



Scaling the training and the testing X labels

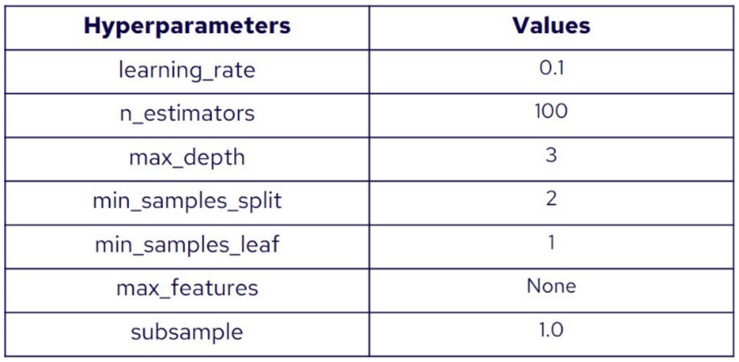
```
scaler = MinMaxScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

```
X_train, X_val, y_train, y_val = train_test_split(X_train, y_train,
    test_size=0.3, random_state=12)
```

Figure 31: MinMax Scaler for Gradient Boost model

In the given code, a ***MinMaxScaler*** object is initialized and stored in the variable scaler from the preprocessing module of scikit-learn. Then, ***X_train*** is transformed using the ***fit_transform()*** method of the scaler to scale the feature values between 0 and 1. The ***transform()*** method of the scaler is then called on ***X_test*** to scale the test data with the same scaling factors as the training data.

After that, the ***train_test_split()*** function is used to split the transformed ***X_train*** into training and validation sets with an initial of 70:30 ratio. The training and validation sets are stored in ***X_train**, **X_val**, **y_train**, and **y_val***.Finally, a Gradient Boosting Classifier ensemble model is created using the values tuned as shown below in Figure 32.



Hyperparameters	Values
learning_rate	0.1
n_estimators	100
max_depth	3
min_samples_split	2
min_samples_leaf	1
max_features	None
subsample	1.0

Figure 32: Different hyper-parameters and their values for GradientBoost

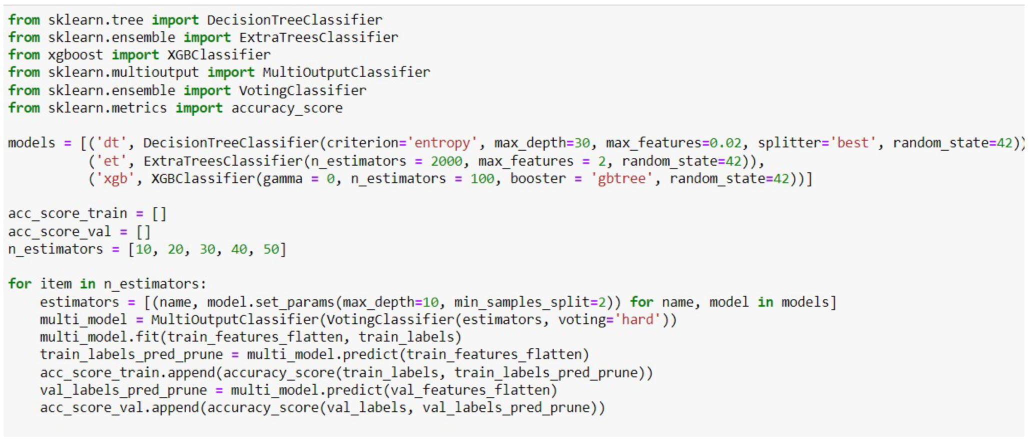
For gradient boost, the most important hyper-parameter values that brought a significant change in the performance of the model were as follows: ***Learning_rate*** of 0.1 means that each decision tree in the gradient boosting model will contribute a fraction of 0.1 to the final prediction. This worked for our dataset because a value lower than this required more boosting iterations to achieve the same reduction in training error, and a value larger than this resulted in faster convergence and lead to overfitting.

We set the ***max_depth*** of 3 meaning that each decision tree in the gradient boosting model had at most three levels of nested decisions. Increasing this factor was causing degradation of model performance by increasing overfitting. Lastly, we set the value of ***subsample*** parameter to 1.0 which means that all samples are used for fitting the individual base learners. When we tried to set a value less than 1, we observed that the model was not able predict all classes of images efficiently and it was resulting in a very skewed output.

For the proposed hybrid model, the Figure 33 below provides a part code snippet of hybrid model that we used. The hybrid ensemble model was created using the ***VotingClassifier*** function from the ***sklearn.ensemble*** module. The purpose of the ***VotingClassifier*** was to combine the predictions of multiple base classifiers to make a final prediction.

The ***VotingClassifier*** uses a majority voting scheme by default, where each classifier's prediction is treated equally and the most frequent prediction is chosen as the final prediction. In the code, the ensemble model combines the predictions of three different classifiers, namely Decision Tree Classifier, Extra Trees Classifier, and XGB Classifier.

The three classifiers were added to a list called ***estimators***, and the list is passed to the ***VotingClassifier*** function to create the ensemble hybrid model.



```
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import ExtraTreesClassifier
from xgboost import XGBClassifier
from sklearn.multioutput import MultiOutputClassifier
from sklearn.ensemble import VotingClassifier
from sklearn.metrics import accuracy_score

models = [('dt', DecisionTreeClassifier(criterion='entropy', max_depth=30, max_features=0.02, splitter='best', random_state=42))
          ('et', ExtraTreesClassifier(n_estimators = 2000, max_features = 2, random_state=42)),
          ('xgb', XGBClassifier(gamma = 0, n_estimators = 100, booster = 'gbtree', random_state=42))]

acc_score_train = []
acc_score_val = []
n_estimators = [10, 20, 30, 40, 50]

for item in n_estimators:
    estimators = [(name, model.set_params(max_depth=10, min_samples_split=2)) for name, model in models]
    multi_model = MultiOutputClassifier(VotingClassifier(estimators, voting='hard'))
    multi_model.fit(train_features_flatten, train_labels)
    train_labels_pred_prune = multi_model.predict(train_features_flatten)
    acc_score_train.append(accuracy_score(train_labels, train_labels_pred_prune))
    val_labels_pred_prune = multi_model.predict(val_features_flatten)
    acc_score_val.append(accuracy_score(val_labels, val_labels_pred_prune))
```

Figure 33: Hybrid model using VotingClassifier

In the above figure, the three classifiers are defined: DecisionTreeClassifier, ExtraTreesClassifier, and XGBClassifier, with their hyperparameters specified. These classifiers are then added to a list called ***models***.

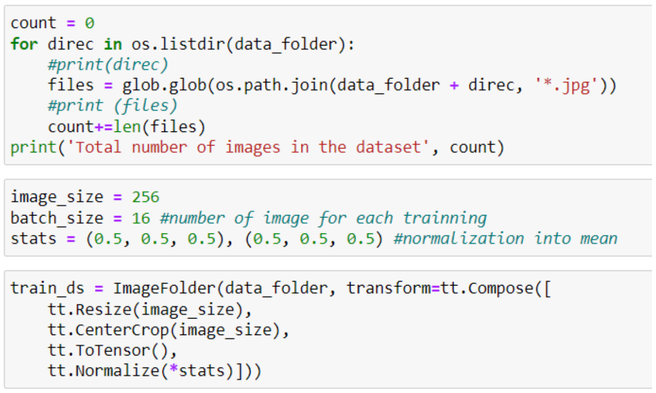
Next, a loop is set up with a range of ***n_estimators*** from 10 to 50. Within the loop, the models are initialized with updated parameters, and the multi-output classification is performed using the ***VotingClassifier*** and ***MultiOutputClassifier*** classes.

The accuracy scores for the training and validation sets are stored in the ***acc_score_train*** and ***acc_score_val*** lists, respectively, and can be used to evaluate the performance of the ensemble model for different numbers of estimators.

- DC-GANs model

In our DC-GAN model, we perform data augmentation to increase the size of our dataset, as our dataset consists of 50 raw images with their corresponding labels of “Natural”, “Blank”, “Puzzle” and “Waldo”. The dataset was increased to a size of 500. We utilize convolutional layers with sigmoid and ReLU as loss and activation functions respectively.

The DC-GAN model consists of generator and discriminator following the same loss and activation functions described above. After training the Discriminator and Generator in separate functions we download the generated images in a directory, compute the loss, and real score parameters. We train these images for a total of 340 epochs and then test the images generated. The ultimate goal is to generate similar looking images as our dataset so as to increase our total dataset size and hence increase accuracy. The basic training of this network is shown in Figure 34.



```
count = 0
for direc in os.listdir(data_folder):
    #print(direc)
    files = glob.glob(os.path.join(data_folder + direc, '*.jpg'))
    #print (files)
    count+=len(files)
print('Total number of images in the dataset', count)
```

```
image_size = 256
batch_size = 16 #number of image for each trainning
stats = (0.5, 0.5, 0.5), (0.5, 0.5, 0.5) #normalization into mean
```

```
train_ds = ImageFolder(data_folder, transform=tt.Compose([
    tt.Resize(image_size),
    tt.CenterCrop(image_size),
    tt.ToTensor(),
    tt.Normalize(*stats)]))
```

Figure 34: Training DC-GANs model

The code initializes a variable count to zero. Then, for each directory in ***data_folder***, it uses the ***glob.glob()*** function to obtain a list of all ***.jpg*** files in that directory and stores it in the variable ***files***. The length of this list is added to the count variable. Next, the code sets the ***image_size*** variable to 256 and ***batch_size*** variable to 16. Two tuples ***stats*** are defined which represent the mean and standard deviation values for each color channel of the images.

An ***ImageFolder*** object is created with the ***data_folder*** as the root directory and a set of image transformations applied to each image. These transformations include resizing and cropping the image to ***image_size***, converting it to a tensor, and normalizing its values using the ***stats*** tuple. The resulting transformed dataset is stored in the ***train_ds*** variable. Using the ImageFolder class from the ***torchvision.datasets*** library, the script reads in the dataset and applies a series of image transformations using the ***tt.Compose*** method which further normalize pixel values of the images as part of pre-processing.

7 EVALUATION AND RESULTS

7.1 Experimental Results

During the initial run of experiments with all the ensemble models, before we encompassed different hyper-parameters for testing out the performance of each respective model, and plotted their confusion matrices to better understand this performance and devise strong conclusive about the different evaluation metrics of precision, recall, f-1 score and accuracy. Figures 35 through 39 represent these confusion matrices for each of the ensemble models.

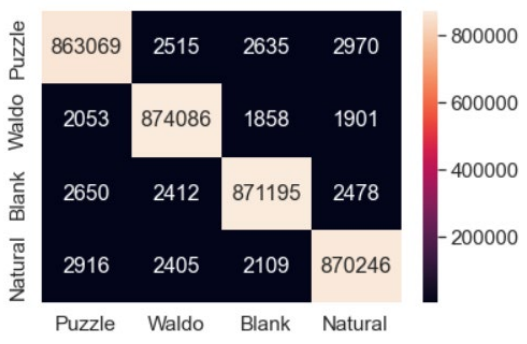


Figure 35: Confusion matrix for Extra-Tree classifier

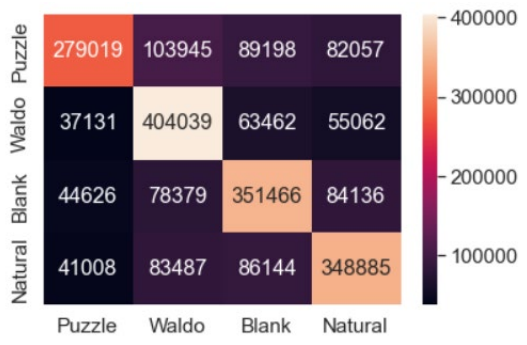


Figure 36: Confusion matrix for Gradient Boost model

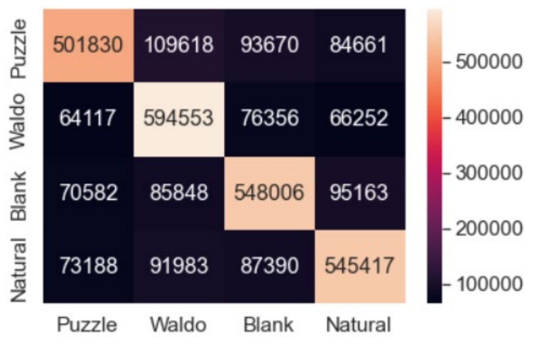


Figure 37: Confusion matrix for XGBoost model

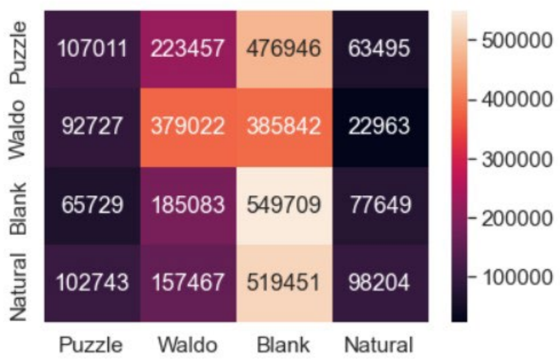


Figure 38: Confusion matrix for Decision Tree classifier

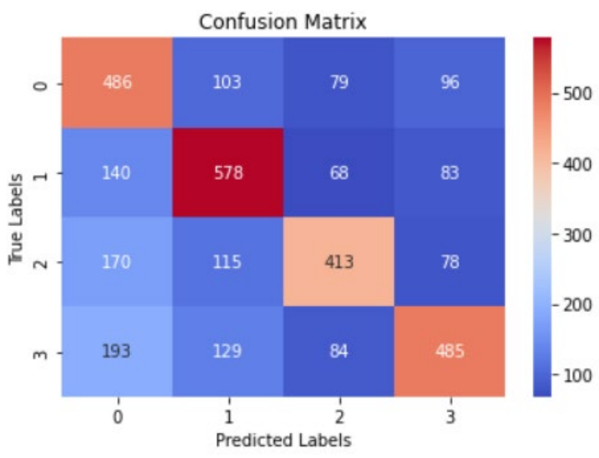


Figure 39: Confusion matrix for proposed hybrid model

From the Extra-Tree confusion matrix, it can be inferred that the model has relatively high accuracy in classifying the images as belonging to the first, second, and fourth classes (as evidenced by the high numbers on the diagonal for those classes). However, it appears to struggle more with the third class, as there are more misclassifications between this class and the others.

Based on the confusion matrix of the Gradient Boosting model and XGBoost model, it can be deduced that the model has a higher accuracy rate in identifying images that belong to the second, third, and fourth classes as indicated by the high number of correct classifications on the diagonal for those classes. However, it appears that the model faces challenges in accurately classifying images in the first class (Puzzle) as there are more instances of incorrect classifications between this class and the others.

The confusion matrix of the Decision Tree model suggests that the model has a better accuracy rate in identifying images belonging to the first, second, and third classes, indicated by a high number of correct classifications on the diagonal. However, it seems to struggle in accurately classifying images in the third class (Natural) as there are more instances of incorrect classifications between this class and others, as shown in Figure 31.

The confusion matrix of the proposed hybrid model indicates that the model is more accurate in classifying images belonging to the first, second, and fourth classes, as there are high numbers on the diagonal for those classes. The class "Waldo" has the highest accuracy compared to others. However, the model faces challenges in accurately classifying images in the third class, as there are more instances of incorrect classifications between this class and the others.

Hence, to resolve the issues discussed above, we undertook careful hyper-parameter tuning of different parameters and devised the results in the tables 3 and 4 shown below.

Table 3: Comparison of ensemble models with all evaluation metrics for eye-tracking data

Sr. No.	**Model**	**Learning Rate**	**Precision**	**Recall**	**F-1 Score**	**Accuracy**
1	Decision Tree	0.1	0.89	0.88	0.88	90.3%
2	XGBoost ensemble model	0.1	0.70	0.73	0.71	86.0%
3	Gradient Boosting model	0.1	0.75	0.78	0.76	75.0%
4	ExtraTree Classifier	0.1	0.86	0.89	0.87	86.4%

The above table 3 conclude that XGBoost, Gradient Boosting, Extra Tree Classifier, and Decision Tree are among the popular ensemble models used for image classification. These models have their strengths and weaknesses, and their performance depends on the dataset and the specific problem at hand.

In this particular case, it was observed that the Decision Tree model outperformed the other models in terms of precision, recall, f-1 score, and accuracy. This means that the Decision Tree model was better able to correctly identify and classify images compared to the other models. This observation was marked due to several reasons.

One, decision tree model was better able to capture the most important features for predicting the outcome variable. Since, decision trees are known for their ability to identify important features and make decisions based on them, this was an advantage for this dataset. Second, Decision trees are good at modeling non-linear relationships between variables, which was beneficial for our research because there are several non-linear relationships between the features and the outcome variable. F-1 score of 0.88 and accuracy of 90.3% suggests that somewhere the model was unable to capture the nuances of certain images but at the same time was able to classify the majority of the images in the dataset.

The XGBoost model does not perform so well, since it has the least f-1 score of 0.71 but is showing an accuracy of fairly high amount of 86%, which means it is correctly identifying the negative instances but is not doing as well in identifying the positive instances. This can be reflected in a high accuracy score, but lower precision and recall scores.

Table 4: Comparison of ensemble models with all evaluation metrics for image data – After data augmentation using DC-GAN

Sr. No.	Model	Learning Rate	Precision	Recall	F-1 Score	Accuracy
1	Decision Tree	0.1	0.67	0.67	0.67	66.6%
2	XGBoost ensemble model	0.1	0.72	0.70	0.71	72%
3	Gradient Boosting model	0.1	0.82	0.80	0.81	82.7%
4	ExtraTree Classifier	0.1	0.67	0.65	0.66	67.8%
5	Proposed Hybrid model	0.1	0.85	1.00	0.92	92.5%

On the other hand, for the image data, the proposed hybrid model performed better than the rest of the models. This observation was marked due to a number of predicted reasons.

Working with an image dataset contributed to the better performance of the hybrid model. Since the image dataset was large and complex, with a high number of features and non-linear relationships between the features and the target variable, it was easier for a complex model that comprised of several models to capture all the nuances in the data. In short, the hybrid model was able to leverage the strengths of different models and capture a broader range of relationships in the data. For example, the decision tree model was good at capturing non-linear relationships in the data, while the xgboost model was good at reducing bias and variance in the model. Combining the two models in a hybrid approach could help capture the non-linear relationships while reducing bias and variance, resulting in better performance.

Moreover, for the image data, decision tree showed the poorest performance with an accuracy of 66.6% and an F-1 score of 0.67 because images are high-dimensional data with many features, and decision trees tend to work better on low-dimensional datasets. When used for image classification, decision trees may not be able to effectively capture the complex relationships between the many features of an image. In addition, decision trees are prone to overfitting, where the model learns too much from the training data and performs poorly on the test data.

7.2 Comparative Analysis with other literature

Our proposed model is compared with one of the previous literature by Kumar et al. [11] and two of the current literature that has been published in the recent years. This enhanced comparison is shown below in Table 6.

Table 5: Qualitative and quantitative comparison of our approach with other literature

SCHEME	RESULTS	CHALLENGES
Proposed hybrid model using a coalition of Decision Tree, Extra-Tree and XGBoost model	1. The hybrid model provided accuracy of 92.5% accuracy on the image data after performing data augmentation. 2. The data augmentation was performed using DC-GAN with the generator generating fake images similar to our input dataset	Limited image dataset was augmented using DC-GANs model however, the DC-GANs model had to be run on 340 total epochs to produce similar looking images due to varying resolutions and image sizes
Task classification model for visual fixation, exploration, and search [11]	1. Performed studies on fixation tasks, where the participant is asked to fixate their gaze in the center of the image, no matter the type of image. 2. Achieved a classification accuracy of about 95.4%	1. High variability of human behavior and the large amount of noise in the eye-tracking data. 2. Removal artifacts and correct for calibration errors to improve the quality of the data 3. Issue of imbalanced data by using techniques such as oversampling and undersampling
A Lightweight Deep Learning Model for Real-time Plant Disease Detection [45]	1. Two popular benchmark datasets for image classification, namely CIFAR-10 and CIFAR-100 2. Achieved an accuracy of 97.33% on the MNIST dataset and 92.1% on the CIFAR-10 dataset	1. Very high computational complexity of deep neural networks including high memory usage. 2. Used an architecture that leverages the strengths of CNNs and LSTM networks to achieve high accuracy with reduced computational complexity and memory usage
XAI-Net model [46]	1. Utilized a multi-scale feature extraction module followed by a feature selection module to extract informative features 2. Evaluated on a hold-out test set, consisting of 1,000 IFE images. The model achieved an accuracy of 96.9%	The number of images in each class initially was not equal, which could lead to biased predictions and poor model performance.

8 CONCLUSION AND FUTURE WORK

8.1 Conclusion

In this project, a novel approach for image classification is proposed using a coalition of different ensemble models for four different labels – “Natural”, “Blank”, “Waldo” and “Puzzle” image classes. Moreover, in this project, a novel approach for image classification is proposed using a coalition of three different ensemble models namely – Decision Tree, Extra-Tree, XGBoost classifier.

Decision tree model was better able to capture the most important features for predicting the outcome variable since they are known for their ability to identify important features and make decisions based on them, which was an advantage in this dataset. Also, there was a huge amount of non-linear relationships in our eye-tracking dataset. Decision trees are good at classification when there are huge non-linear relationships because they recursively split the data based on the most significant feature, which can capture complex, non-linear relationships between features. Additionally, decision trees can handle both categorical and numerical data and this proved to be beneficial in our case because our eye-tracking dataset was numerical and large in size.

We employed another model, DC-GANs for the purpose of data augmentation of the image dataset since it was limited in size. This was done primarily to increase the efficiency of all models during training and testing. Working with an image dataset contributed to the better performance of the hybrid model. Though the hybrid model combines the strengths of the three base models in our case, other possible reasons for high accuracy can also lead possible conclusions of the small image dataset size of 500 images and possibly lesser features in our images.

8.2 Future work

Looking ahead, there are several directions for future research that build on the findings of this study. There is scope of using deep learning along with reinforcement learning in this field for improvised accuracy. Reinforcement learning involves training a model to make decisions based on trial and error, with the goal of maximizing a reward. Deep learning involves training neural networks with multiple layers to recognize complex patterns in data.

We can also extend focus to leveraging the sequential information in the data and employing algorithms like Hidden Markov Models (HMMs) and Long Short-Term Memory (LSTMs). HMMs are a type of statistical model that can be used to analyze sequential data, while LSTMs are a type of neural network that can be trained on sequential data. Transfer learning can be used to improve the performance of the models by leveraging pre-trained models such as VGG, ResNet, or Inception, and fine-tune it for the specific image classification. Transfer learning has been shown to improve the performance of models and reduce the need for large amounts of labeled data.

We can explore other multi-label classification techniques such as binary relevance and classifier chains to handle multi-label classification. Multi-label classification involves assigning multiple labels to a single instance, which can be challenging but is relevant in many applications such as medical diagnosis.

REFERENCES

[1] H. Sattar, A. Bulling and M. Fritz, "Predicting the category and attributes of visual search targets using deep gaze pooling," in Proceedings of the IEEE International Conference on Computer Vision Workshops, 2017.

[2] H. Sattar et al, "Prediction of search targets from fixations in open-world settings," in Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2015.

[3] Y. Jang, R. Mallipeddi and M. Lee, "Identification of human implicit visual search intention based on eye movement and pupillary analysis," User Modeling and User-Adapted Interaction, vol. 24, (4), pp. 315-344, 2014.

[4] E. F. Keller and C. R. Grontkowski, "The mind's eye," in Discovering Reality Anonymous 1983.

[5] G. J. Zelinsky, Y. Peng and D. Samaras, "Eye can read your mind: Decoding gaze fixations to reveal categorical search targets," Journal of Vision, vol. 13, (14), pp. 10, 2013.

[6] A. Krizhevsky, I. Sutskever and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," Commun ACM, vol. 60, (6), pp. 84-90, 2017.

[7] D. Lunga et al, "Manifold-learning-based feature extraction for classification of hyperspectral data: A review of advances in manifold learning," IEEE Signal Process. Mag., vol. 31, (1), pp. 55-66, 2013.

[8] S. Kazeminia et al, "GANs for medical image analysis," Artif. Intell. Med., vol. 109, pp. 101938, 2020.

[9] Y. Zhu et al, "Heterogeneous transfer learning for image classification," in Twenty-Fifth Aaai Conference on Artificial Intelligence, 2011.

[10] X. Zhou et al, "Eye tracking data guided feature selection for image classification," Pattern Recognition, vol. 63, pp. 56-70, 2017.

[11] A. Kumar et al, "Task classification model for visual fixation, exploration, and search," in Proceedings of the 11th ACM Symposium on Eye Tracking Research Applications, 2019, .

[12] J. Otero-Millan et al, "Saccades and microsaccades during visual fixation, exploration, and search: foundations for a common saccadic generator," Journal of Vision, vol. 8, (14), pp. 21, 2008.

[13] N. Pinto, D. Doukhan, J. J. DiCarlo, and D. D. Cox, "A high-throughput screening approach to discovering good forms of biologically inspired visual representation," PLoS Comput. Biol., vol. 5, no. 11, p. e1000579, Nov. 2009.

[14] F. Rosenblatt, "The perceptron: a probabilistic model for information storage and organization in the brain," Psychol. Rev., vol. 65, no. 6, pp. 386–408, Nov. 1958.

[15] J. Li, J. Jia, and D. Xu, "Unsupervised representation learning of image-based plant disease with deep convolutional generative Adversarial Networks," 2018 37th Chinese Control Conference (CCC), 2018.

[16] “National Aeronautics and Space Administration (NASA)” (2018) The Grants Register 2018, pp. 529–529. Available at: https://doi.org/10.1007/978-1-349-94186-5_816.

[17] J.-Y. Zhu, T. Park, P. Isola, and A. A. Efros, “Unpaired image-to-image translation using cycle-consistent adversarial networks,” in 2017 IEEE International Conference on Computer Vision (ICCV), Venice, Oct. 2017, pp. 2223–2232.

[18] M. Mirza and S. Osindero, “Conditional Generative Adversarial Nets,” arXiv [cs.LG], Nov. 06, 2014.

[19] P. Isola, J.-Y. Zhu, T. Zhou, and A. A. Efros, “Image-to-image translation with conditional adversarial networks,” arXiv [cs.CV], pp. 1125–1134, Nov. 21, 2016. Accessed: Mar. 13, 2023.

[20] X. Mao, Q. Li, H. Xie, R. Y. K. Lau, Z. Wang, and S. Paul Smolley, “Least squares generative adversarial networks,” in Proceedings of the IEEE international conference on computer vision, 2017, pp. 2794–2802.

[21] A. Odena, C. Olah, and J. Shlens, “Conditional image synthesis with auxiliary classifier GANs,” arXiv [stat.ML], pp. 2642–2651, 06--11 Aug 2017.

[22] I. Gulrajani, F. Ahmed, M. Arjovsky, V. Dumoulin, and A. C. Courville, “Improved training of wasserstein gans,” Adv. Neural Inf. Process. Syst., vol. 30, 2017.

[23] N. Kodali, J. Abernethy, J. Hays, and Z. Kira, “On Convergence and Stability of GANs,” arXiv [cs.AI], May 19, 2017.

[24] H. Zhang et al., “StackGAN++: Realistic Image Synthesis with Stacked Generative Adversarial Networks,” IEEE Trans. Pattern Anal. Mach. Intell., vol. 41, no. 8, pp. 1947–1962, Aug. 2019.

[25] A. Krizhevsky, I. Sutskever, and G. E. Hinton, “ImageNet classification with deep convolutional neural networks,” Commun. ACM, vol. 60, no. 6, pp. 84–90, May 2017.

[26] K. He, X. Zhang, S. Ren, and J. Sun, “Deep residual learning for image recognition,” arXiv [cs.CV], pp. 770–778, Dec. 10, 2015.

[27] X. Li, L. Ding, L. Wang, and F. Cao, “FPGA accelerates deep residual learning for image recognition,” in 2017 IEEE 2nd Information Technology, Networking, Electronic and Automation Control Conference (ITNEC), Dec. 2017, pp. 837–840.

[28] Y.-Y. Song and Y. Lu, “Decision tree methods: applications for classification and prediction,” Shanghai Arch Psychiatry, vol. 27, no. 2, pp. 130–135, Apr. 2015.

[29] H. H. Patel and P. Prajapati, “Study and analysis of decision tree based classification algorithms,” International Journal of Computer Sciences and Engineering, vol. 6, no. 10, pp. 74–78, 2018.

[30] E. K. Ampomah, Z. Qin, and G. Nyame, “Evaluation of Tree-Based Ensemble Machine Learning Models in Predicting Stock Price Direction of Movement,” Information, vol. 11, no. 6, p. 332, Jun. 2020.

[31] P. Geurts, D. Ernst, and L. Wehenkel, “Extremely randomized trees,” Mach. Learn., vol. 63, no. 1, pp. 3–42, Apr. 2006.

[32] A. Zafari, R. Zurita-Milla, and E. Izquierdo-Verdiguier, “Land Cover Classification Using Extremely Randomized Trees: A Kernel Perspective,” IEEE Geoscience and Remote Sensing Letters, vol. 17, no. 10, pp. 1702–1706, Oct. 2020.

[33] J. Sharma, C. Giri, O.-C. Granmo, and M. Goodwin, “Multi-layer intrusion detection system with ExtraTrees feature selection, extreme learning machine ensemble, and softmax aggregation,” EURASIP Journal on Information Security, vol. 2019, no. 1, pp. 1–16, Oct. 2019.

[34] A. Natekin and A. Knoll, “Gradient boosting machines, a tutorial,” Front. Neurorobot., vol. 7, p. 21, Dec. 2013.

[35] G. Ke et al., “LightGBM: A highly efficient gradient boosting decision tree,” Adv. Neural Inf. Process. Syst., vol. 30, 2017, Accessed: Mar. 23, 2023.

[36] J. H. Friedman, “Greedy Function Approximation: A Gradient Boosting Machine,” Ann. Stat., vol. 29, no. 5, pp. 1189–1232, 2001.

[37] J. H. Friedman, “Stochastic gradient boosting,” Comput. Stat. Data Anal., vol. 38, no. 4, pp. 367–378, Feb. 2002.

[38] B. Boehmke and B. M. Greenwell, Hands-On Machine Learning with R. CRC Press, 2019.

[39] J. Yang, J. Li, Y. Wang, X. Li and Y. Li, "A hybrid ensemble method for pulsar candidate classification," 2019 IEEE 5th Intl Conference on Big Data Security on Cloud (BigDataSecurity), IEEE Intl Conference on High Performance and Smart Computing, (HPSC) and IEEE Intl Conference on Intelligent Data and Security (IDS), 2019, pp. 138-14.

[40] W. Li, Y. Yin, X. Quan, and H. Zhang, “Gene Expression Value Prediction Based on XGBoost Algorithm,” Front. Genet., vol. 10, p. 1077, Nov. 2019.

[41] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in Advances in neural information processing systems, 2012, pp. 1097-1105.

[42] R. Szeliski, Computer vision: algorithms and applications. London, U.K.: Springer Science & Business Media, 2010.

[43] Deng, J., Dong, W., Socher, R., Li, L. J., Li, K., & Fei-Fei, L. (2009). Imagenet: A large-scale hierarchical image database. In 2009 IEEE conference on computer vision and pattern recognition (pp. 248-255).

[44] He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep residual learning for image recognition. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 770-778).

[45] A. H. M. Kamal, J. Lu, F. Sifat, and X. Chen, "A Lightweight Deep Learning Model for Real-time Plant Disease Detection," in Proceedings of the 2018 IEEE International Conference on Imaging Systems and Techniques (IST), Krakow, Poland, 2018, pp. 1-5

[46] Y. Zhang, J. Zhu, Y. Liu, Q. Guo, X. Zhu, Z. Liu and X. Wang, "Expert-Level Immunofixation Electrophoresis Image Recognition based on Explainable and Generalizable Deep Learning," in IEEE Journal of Biomedical and Health Informatics