Analysis of Sign Language MNIST using Deep Learning

CIND 860: Advanced Data Analytics Project

Section: DJH

Student Name: Muhammad Zaka Shaheryar

Student #: 500648718

Supervisor Name: Tamer Abdou

Date of Submission: April 1st, 2024

GitHub Link: https://github.com/Zaka123456/CIND860

Ryerson University

Table of Contents

Acknowledgement

I want to convey thanks to Dr. Tamer Abdou for supervising me throughout the project. I also would like to thank Riyad Hussain for their help throughout the project. Finally, I would like to thank all my Professors and Teaching assistants, colleagues, Program Director, and Academic coordinator in my certificate program for teaching us valuable skills that we can apply to this project.

1. Revised Abstract

For CIND 860 Capstone project, the dataset chosen initially is changed to "Sign Language MNIST (https://www.kaggle.com/datasets/datamunge/sign-language-mnist). This is done to apply deep algorithms effectively on the dataset. The dataset is taken from Kaggle, and it contains two csv files. First is sign_mnist_train which consists of 27455 rows and 785 columns. Second is sign_mnist_test which has 7172 rows and 785 columns. Each column is a pixel while each row is a image. In total, there are 34627 images of hand gestures.

"American Sign Language (ASL) is a complete, natural language that has the same linguistic properties as spoken languages, with grammar that differs from English. ASL is expressed by the movement of hands and faces. This dataset consists of 27,455 images of hand signs, each image is of 28 x 28 size and in grayscale format. The dataset format is patterned to match closely with the classic MNIST. Images in the dataset belong to a label from 0–25 representing letters from A-Z (but no cases of 9=J or 25=Z as they involve hand motion). The training data (27,455 cases) and the test data (7,172 cases) are approximately half the size of standard MNIST but otherwise similar to a header row of the label, pixel1, pixel…. pixel784. The original hand gesture image data represented multiple users repeating gestures against different backgrounds. The Sign Language MNIST data came from greatly extending the small number (1704) of the color images included as not cropped around the hand region of interest." (Singh, A).

Table 1 below provides the research questions for the project.

Table 1: Research Questions

No.	Research Question (RQ)
1	How does accuracy, computational efficiency, and stability of CNN compare to traditional machine learning techniques such as K-NN, Naïve Bayes, Random Forest, SVM, or, XGBoost in classifying ASL images?
2	How to optimize the CNN model by hyperparameter tuning and better architecture that exhibits the greatest efficiency in classifying ASL images considering accuracy and computational resources?
3	When compared to Naive Bayes (NB), how effective is GNB at identifying the true cause of the categorial target label of English alphabets in the context of the Sign language MNIST dataset?

2. Introduction

Ability to listen and hear enables effective communication. But people who suffer from hearing impairment or hearing loss face difficulty in this regard. According to WHO, there are about 300 million deaf and 1 million mute (Lanxi et al). Therefore, there is a strong demand for a proper way of communicating with these people. Sign language is one of the ways to solve this problem. A deaf-mute is a person who is either deaf or both deaf and dumb. Sign language can help these people to communicate with each other and other people. Hearing impairment can be caused due to age, genes, noise exposure, infections, ear injury, or medicine. (Hasan et al).

Sign language recognition has earned considerable attention due to the large global deaf and mute population. There are 144 different sign languages based on the regional languages (Eman et al). This project will focus on American Sign Language (ASL).

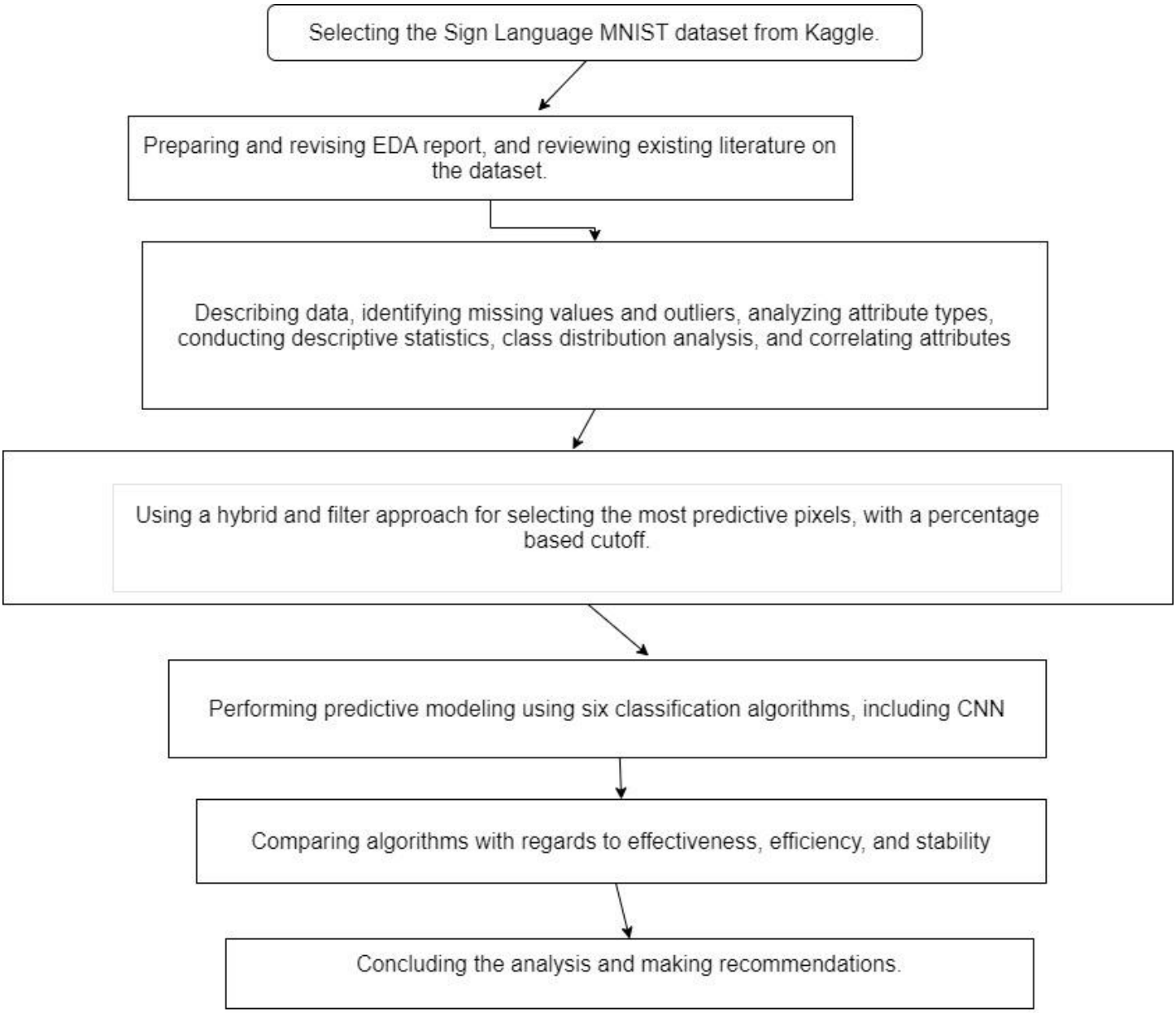
This project's objective is to perform a comprehensive examination of Convolutional Neural Networks (CNNs) on the Sign Language MNIST dataset. In addition to that, various CNN models like MobileNet, and deep CNN will be explored and compared with traditional classifiers like SVM, KNN, etc. The evaluation measures will be accuracy, precision, and recall (Hasan et al).

This study offers a detailed description of the dataset characteristics and outlines the systematic evaluation methodology. Furthermore, it provides conclusive insights into the promise of CNNs over machine learning for the vitally important domain of sign language recognition. By leveraging standardized protocols and Python-based experiments with Keras and TensorFlow, this project aims to facilitate real-time sign language recognition (SLR) systems.

The tools that will be used are Python libraries used for deep learning and traditional machine learning algorithms. The systematic data analysis process approach will be used for the project. After data selection, initial analysis will be carried out followed by the exploratory analysis (EDA). Then experimental design and model building will be carried out. Finally, performance evaluation will be done together with project limitations and conclusion.

The motivation behind this study is to advance American Sign Language (ASL) recognition technology by analyzing the Sign Language MNIST dataset through machine learning and deep learning approaches. This effort aims to improve communication tools for individuals with hearing impairments by enhancing the accuracy of ASL recognition systems. By exploring various algorithms and methodologies, the study seeks to provide valuable insights for educators, technologists, and the broader community, contributing to the development of more effective and accessible ASL educational and communication aids. The project approach is shown in figure 1.

Figure 1: Project Approach



3. Literature Review

This study provides a detailed analysis of the Sign Language MNIST dataset, focusing on enhancing American Sign Language (ASL) recognition through machine learning (ML) and deep learning techniques. By examining 34,627 grayscale images representing ASL alphabets (excluding dynamic letters), our research explores the significance of individual pixels in classification, the effectiveness of various ML algorithms, and the potential of ensemble models for improved accuracy. Initial exploratory data analysis (EDA) revealed equitable label distribution and identified influential pixels through filter-based, wrapper-based, and hybrid methods, enhancing model precision. Comparative analysis between traditional ML algorithms and deep learning models, especially Convolutional Neural Networks (CNNs) will be used to validate the hypothesis that CNNs, despite being time-intensive, offer superior accuracy. Our findings contribute to ASL recognition technology, providing insights for educators, technologists, and the broader community to aid communication for individuals with hearing impairments, emphasizing the importance of feature selection, model choice, and the innovative integration of methodologies for advancing ASL recognition systems. The questions to consider for the literature review are provided in Table 2. Six papers are reviewed and key take-aways and how they can be useful to this project are provided below.

Table 2: Questions to consider for Literature Review

No.	Literature Review Question
1	What do you already know about the topic?
2	What do you have to say critically about what is already known?
3	Has anyone else ever done anything exactly the same?
4	Has anyone else done anything that is related?
5	Where does your work fit in with what has gone before?
6	Why is your research worth doing in the light of what has already been done?

The literature review on enhancing ASL recognition with machine learning and deep learning delves into the substantial groundwork laid by prior research, highlighting significant strides in algorithm application and dataset utilization. It critically acknowledges the existing gap in developing a universally accurate, real-

time recognition system, despite notable efforts. This review positions the novel contributions of the current study within the context of related endeavors, underscoring its potential to address unresolved challenges in gesture variability and real-time processing. It justifies the research's relevance by emphasizing its aim to refine communication tools for the hearing impaired, thereby enriching the technological landscape and making a meaningful societal impact.

The following six papers present advancements in sign language recognition using machine learning and deep learning. They explore various methodologies, including CNNs, MobileNet, and machine learning classifiers, achieving accuracies ranging from 91.7% to 99.95% in recognizing sign languages and gestures. These studies demonstrate the potential of integrating advanced computational techniques for enhancing ASL recognition systems, contributing to improved communication tools for individuals with hearing impairments and educational applications. Their collective findings underscore the importance of optimizing recognition accuracy and real-time processing capabilities in developing accessible and effective ASL interpretation technologies.

Paper 1:

The study titled "Mediapipe and CNNs for Real-Time ASL Gesture Recognition" (Kumar et al) describes a real-time system for ASL movement identification using modern computer vision and machine learning approaches, achieving a remarkable accuracy of 99.95%. This paper will serve as a main paper for the literature review, highlighting the potential of Mediapipe and CNNs in enhancing communication for people with hearing impairments through real-time sign language recognition. The paper presents a highly effective American Sign Language (ASL) recognition model utilizing Convolutional Neural Networks (CNN) for feature extraction and classification, achieving an impressive 99.95% accuracy. Key takeaways include the use of the Mediapipe library for real-time hand tracking and feature extraction, the development of a robust classification system capable of recognizing all ASL alphabets, and the potential extension of this technology to other sign languages. This research is particularly useful for the project as it demonstrates a scalable, highly accurate approach to ASL recognition, suggesting that integrating Mediapipe and CNNs could significantly enhance real-time ASL interpretation systems, improving communication tools for individuals with hearing impairments.

Paper 2

The paper "Sign Language Recognition Using Convolutional Neural Networks" explores the use of CNNs for recognizing Italian sign language gestures with a focus on achieving high accuracy and generalization across different users and environments. Utilizing Microsoft Kinect, the study emphasizes automated feature extraction through CNNs, managing to recognize 20 gestures with a cross-validation accuracy of 91.7%. The approach demonstrates the potential of CNNs in sign language recognition, offering valuable insights for projects aiming to enhance communication for the Deaf community. This research could guide the development of more accessible and effective ASL recognition systems, highlighting the importance of deep learning in automating sign language interpretation.

Paper 3

The paper, "Arabic Sign Language Recognition Using Convolutional Neural Network and MobileNet," focuses on developing a CNN-based model to recognize Arabic sign language with a 94.46% accuracy using the ArASL2018 dataset. This approach demonstrates an improvement over prior models in recognition accuracy. For the project, this study underscores the effectiveness of combining CNNs with MobileNet for sign language recognition, offering a potential framework for enhancing ASL recognition systems by leveraging similar methodologies to achieve high accuracy and performance in real-world applications.

Paper 4

The paper investigates Sign Language Recognition (SLR) using various machine learning algorithms and dimensionality reduction techniques to improve model training speed and classification accuracy. Key takeaways include the effectiveness of PCA and manifold learning in reducing data dimensionality, the superior performance of CNNs in image-based SLR, and the comparative analysis of machine learning models like RFC, KNN, GNB, SVM, and SGD in recognizing sign language gestures. This research could significantly benefit the project by providing insights into optimizing SLR systems for enhanced accuracy and efficiency, especially in real-time applications.

My analysis resembles their analysis since I used four of their models: KNN, SVM, RFC and CNN.

Paper 5

The paper, "Classification of Sign Language Characters by Applying a Deep Convolutional Neural Network," presents a deep CNN model to identify American Sign Language alphabets from the Sign Language MNIST dataset, achieving a 97.62% accuracy. This study outperforms previous approaches by integrating advanced CNN architectures, highlighting the potential of deep learning in sign language recognition. For the project, this research provides a robust framework for improving ASL recognition accuracy, showcasing the effectiveness of deep CNNs in handling complex image classification tasks within sign language interpretation.

Paper 6

The paper focuses on object recognition using NAO robots, emphasizing typed text, color, shape recognition, and sign language gestures with accuracies around 90%. Machine learning classifiers tested on the MNIST dataset showed 87%-98% accuracy. This research contributes to children's visual learning enhancement, suggesting future improvements for broader image applicability. The study highlights the potential of integrating advanced object recognition and machine learning techniques in educational settings, offering insights into developing more interactive and effective learning tools through robotics.

My analysis resembles their analysis since I used similar models: KNN, SVM, RFC, and CNN. Also, I got almost similar results than theirs. Their ranking of model according to accuracy is Neural network, then KNN, and 3rd is SVM. I got KNN as first, SVM as 2nd and RF as third as shown in figure 14.

Paper 7:

The article presents a novel approach to improve content-based image classification systems using Bayesian Networks, specifically focusing on face images. It introduces three variants of Bayesian Networks: Naïve Bayes (NB), Tree Augmented Naïve Bayes (TAN), and Forest Augmented Naïve Bayes (FAN) to classify images based on vector labels derived from image descriptors. The study finds that FAN outperforms both NB and TAN in classification accuracy. This approach emphasizes the importance of optimizing the structure of Bayesian Networks by considering the dependencies between high-level features to accurately represent data. For the research question on using Bayesian networks (RQ 3) for causation and comparing their effectiveness with other causation analysis techniques, this paper demonstrates that Bayesian networks, particularly the FAN variant, can effectively identify the true causes in image classification by optimizing data representation through the analysis of dependencies between features. This could be applied to other domains by comparing Bayesian networks' performance in identifying causal relationships against traditional statistical methods or machine learning approaches, focusing on how well they handle high-dimensional data and uncover underlying dependencies within the dataset.

4. Data Description

American Sign Language (ASL) is a complete, natural language that has the same linguistic properties as spoken languages, with grammar that differs from English. ASL is expressed by the movement of hands and faces. This dataset consists of 27,455 images of hand signs, each image is of 28 x 28 size and in grayscale format. The dataset format is patterned to match closely with the classic MNIST. Images in the dataset belong to a label from 0–25 representing letters from A-Z (but no cases of 9=J or 25=Z as they involve hand motion). The training data (27,455 cases) and the test data (7,172 cases) are approximately half the size of standard MNIST but otherwise like a header row of the label, pixel1 to pixel784. The original hand gesture image data represented multiple users repeating gestures against different backgrounds. The Sign Language MNIST data came from greatly extending the small number (1704) of the color images included as not cropped around the hand region of interest (Kaggle).

The dataset comprises 27455 training and 7172 testing grayscale images distributed over 24 ASL classes or alphabets. Attributes or features represent columns and there are 785 columns. Pixel intensity ranges from 0-255 with 0 being very light and 255 being very dark. A description of them is provided in Table 3. Also, the 24 ASL classes or alphabets (excluding J and Z) are provided in Table 4 and images are shown in Figure 2.

Table 3: Attributes Type

Attribute	**Type**
label	Numeric target column of number 0-25 (excluding 9 and 25)
Pixel (1-784)	Numeric Pixels

Figure 2: Hand Gesture

Table 4: ASL classes

Label Number	Alphabet
0	A
1	B
2	C
3	D
4	E
5	F
6	G
7	H
8	I
10	K
11	L
12	M
13	N
14	O
15	P
16	Q
17	R

18	S
19	T
20	U
21	V
22	W
23	X
24	Y

As seen from above table class 9 representing ‘J’ and class 25 representing ‘Z’ are excluded from dataset.

5. Data Preparation

The Pandas Profiling Report for train dataset provides a comprehensive analysis of a dataset with 27455 observations and 49 variables. After selecting 48 top pixel using filter and hybrid approach. Key findings include a 0% missing data, 0% duplicate rows, and high correlations among certain pixel values, indicating redundancy or strong dependencies between features. The report also identifies ignorable outliers and zeros (extreme lightness in pixel). To avoid data leakage only train dataset is used to train the model and the model is tested on new data, which is test data already in a separate file, so no need to split the dataset into training and testing data.

For feature selection, the filter approach is used first to find the top 25% of 784 pixels using the chi-square test which evaluates the association between each pixel and ASL labels and selects the top-performing labels based on the highest score. This reduces the column size of train data to 195 features. Then top 25% of 195 is found using a hybrid approach which uses recursive feature elimination (RFE) to iteratively build a model to identify a subset of pixels that contributed most to model accuracy. The model used is a Random Forest classifier to find the best pixels related to the target. Therefore, the final features in train data have been reduced to 48 as shown in Figure 3. (also see the final code file).

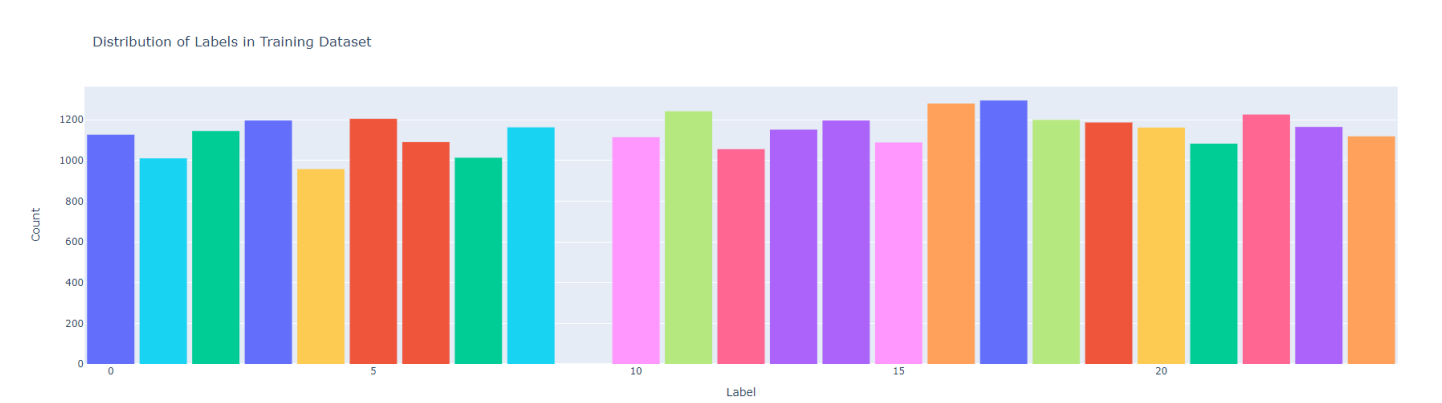
The further reduction of feature size using the hybrid method is done so that the algorithms for predictive modelling will work without crashing and taking a long time.

Figure 3: Selected Feature with their Importance Score

	Features	Importance
1	pixel354	0.003464
2	pixel520	0.003418
3	pixel211	0.003319
4	pixel239	0.003303
5	pixel246	0.003300
6	pixel413	0.003277
7	pixel355	0.003240
8	pixel266	0.003197
9	pixel238	0.003179
10	pixel274	0.003138
11	pixel321	0.003137
12	pixel441	0.003133
13	pixel493	0.003003
14	pixel273	0.002956
15	pixel237	0.002880
16	pixel576	0.002837
17	pixel265	0.002814
18	pixel382	0.002796
19	pixel302	0.002778
20	pixel385	0.002768
21	pixel245	0.002762
22	pixel159	0.002709
23	pixel376	0.002702
24	pixel519	0.002699
25	pixel327	0.002686
26	pixel352	0.002653
27	pixel215	0.002635
28	pixel353	0.002625
29	pixel350	0.002611
30	pixel772	0.002582
31	pixel465	0.002574
32	pixel187	0.002568
33	pixel437	0.002553
34	pixel240	0.002553
35	pixel217	0.002548
36	pixel381	0.002542
37	pixel466	0.002540
38	pixel377	0.002533
39	pixel469	0.002524
40	pixel210	0.002512
41	pixel411	0.002505
42	pixel440	0.002501
43	pixel348	0.002479
44	pixel605	0.002475
45	pixel387	0.002464
46	pixel326	0.002451
47	pixel438	0.002451
48	pixel468	0.002450

The Pandas profiling report shows that all 48 selected features Gaussian like or almost normal distribution. Furthermore, label column which is target column show uniform distribution which also means almost is no class imbalance as shown in figure 4.

Figure 4: Class Distribution in train dataset



6. Modelling

Six models are used in this project, namely K-Nearest Neighbours, Naïve Bayes, Support Vector Machine, Random Forest, XGBoost, and Convolutional Neural Network. KNN is taken as Baseline model to be used as reference for other models. Other models' performance is checked compared to KNN. To check the stability of most model K fold cross validation is used and K is taken as 10. Therefore, mean validation accuracy and variance is reported together with test accuracy. Also Mathew correlation coefficient (MCC) which capture true negative is used together with precision and recall to measure the effectiveness of the model. The formula of MCC is provided in provide below.

Figure 5: MCC formula

$$MCC = \frac{TN \times TP - FN \times FP}{\sqrt{(TP + FP)(TP + FN)(TN + FP)(TN + FN)}}$$

Also, confusion matrix is drawn which shows true positive, true negative, and false positive false negative. For each class positive class is taken as a true class and the rest of the 23 classes are taken as negative. Furthermore, GPU is used instead of CPU as run time which increases the processing speed since it is capable of parallel processing and image data set and image classification and some model like CNN gains a lot due to it.

To answer the first research question, each model is being trained and tested. The summary of each model is provided below.

K-Nearest Neighbours

For the baseline KNN model, k is taken as 5 which is the number of nearest neighbors. The model has the best accuracy of 70% in this project and the lowest cross-validation variance of 0.0000018 which shows its stability. Due to time constraints, some other evaluation measures such as train/test time, and Mattew Correlation coefficient are not checked.

Figure 6: KNN results

```
Accuracy scores for each fold (KNN Baseline): [0.99635834 0.99708667 0.99599417 0.99344501 0.99708667 0.99854281
 0.99781421 0.99744991 0.99708561 0.99562842]
Mean accuracy (KNN Baseline): 0.9966491813891907
Variance (KNN Baseline): 1.797982324817986e-06
Test Accuracy 70.32
```

```
KNN Confusion Matrix:
[[326   0   0   0   1   0   0   0   0   0   0   3   1   0   0   0   0   0
    0   0   0   0   0   0]
 [  0 402   0   0   0   0   0   0   0   4   0   0   0   0   0   0   3   0
   13  10   0   0   0   0]
 [  0   0 288   0   0   1   0  14   0   0   0   0   0   0   5   0   0   0
    0   0   0   0   0   2]
 [  0   0   0 244   1   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0]
 [  2   1   0   0 470   0   0   0   1   0   0   3   2   0  18   0   0   1
    0   0   0   0   0   0]
 [  0   0   0   0   0 208  13   0   0   0   0   0   0   0   4   0   0   0
    2   0   0   0  20   0]
 [  0   0   0   1   0   1 257   0   4   0   0   0   0   0   1   1  15   0
   40   0  16   0   9   3]
 [ 15   0   0   0   0   4  61 306   0   0   0   0   0   0   0   5   0   0
   42   0   0   0   2   1]
 [ 21   0   0  15   0   0   0   5 128   0  36   0  18   0   4   9   3  13
    0   0   0   0   1  35]
 [  0   1   0   7   1   0   1   0   0 146   3   2   0   0  12  23  57   0
   10  60   3   5   0   0]
 [  0   0   0   0   0   0   0   0   0   0 207   0   0   1   0   0   0   0
    0   0   0   0   1   0]
 [ 50   0   0   1  29   0   0   1   9   1   0 227   2   0   0  19   5  26
    1   1  11   0   0  11]
 [ 70   0   0  12   0   0  21   0   9   0   0  37  77   0   0  32   0  10
    0  14   0   3   0   6]
 [  0   0   5   0   0   0   0   0   0  16   0   0   0 172   0  24   1   0
   28   0   0   0   0   0]
 [  0   0   0   1   0   0   0  14   4   3   2   0   0   0 274  20   2   0
    4   0  20   2   0   1]
 [  8   0   0   0   0   0   0   0   0   0   0   0   0   0   4 142   0  10
    0   0   0   0   0   0]
 [  0   3   0   0   0   0   2   0   0  14  12   0   0   0   0   0  71   0
    0  38   0   4   0   0]
 [  0   0   0   4   3   2   0   2   1   0  25   4   0   0   0  15   5 169
    5   0   0   0   0  11]
 [  0   0   0   0   0  15  38  10   0   0  14   0   0   0   8   0   9   0
  138   0   0   6  10   0]
 [  0   6   0   3   9   0  19   0   1  24   0   0   0   0   0   0  34   0
    0 166   0   0   0   4]
 [  0   0   0  18   0  10  15  21  16   8   5   0   0   0   6   1  22   1
    0   5 194  12   2  10]
 [  0   0   0   0   0   1   0   0  12  27   0   0   0   0   0   0   0   0
    3   6  26 131   0   0]
 [  0   1   0   3   0   1   0   0   1   1   3   0   0   0   3   7   1  18
   22   0  18  23 148  17]
 [  0   0   0   0   0   0   0  22  66   0  16   6   0   0   4   0   1  15
   28   2  18   0   2 152]]
```

Classification Report (KNN):

	precision	recall	f1-score	support
0	0.66	0.98	0.79	331
1	0.97	0.93	0.95	432
2	0.98	0.93	0.96	310
3	0.79	1.00	0.88	245
4	0.91	0.94	0.93	498
5	0.86	0.84	0.85	247
6	0.60	0.74	0.66	348
7	0.77	0.70	0.74	436
8	0.51	0.44	0.47	288
10	0.60	0.44	0.51	331
11	0.64	0.99	0.78	209
12	0.80	0.58	0.67	394
13	0.77	0.26	0.39	291
14	0.99	0.70	0.82	246
15	0.80	0.79	0.79	347
16	0.48	0.87	0.61	164
17	0.31	0.49	0.38	144
18	0.64	0.69	0.66	246
19	0.41	0.56	0.47	248
20	0.55	0.62	0.58	266
21	0.63	0.56	0.60	346
22	0.70	0.64	0.67	206
23	0.76	0.55	0.64	267
24	0.60	0.46	0.52	332
accuracy			0.70	7172
macro avg	0.70	0.70	0.68	7172
weighted avg	0.72	0.70	0.70	7172

Naïve Bayes

This model performs the worst in term of accuracy as it has accuracy of only 39%, but it has good cross validation variance of 0.00015 which is second best after KNN. The lower the variance, the more stable the model is.

Figure 7: Naïve Bayes Result

```
Accuracy scores for each fold (Naive Bayes): [0.42753095 0.41150765 0.44136926 0.44246176 0.43554261 0.44626594
 0.4276867  0.43497268 0.456102   0.44808743]
Mean accuracy (Naive Bayes): 0.43715269900779674
Variance (Naive Bayes): 0.00014548578397972483
Test Accuracy 39.12
```

```
NB Confusion Matrix:
[[239   0  13   0   0   0   0   0   0   0   0   0  47  26   0   0   0   6
    0   0   0   0   0   0]
 [  0 391   0   0  15   0   0   0   0  21   0   5   0   0   0   0   0   0
    0   0   0   0   0   0]
 [  0   0 212   0   0  13  15   0   0   0   9   0   0  40   0   0   0   0
    0   0  16   5   0   0]
 [  4  20   0  54  19   0   0   0   0   0   0   0  34   0   0   2   6   1
    0  36   0   0  67   2]
 [ 35   6   1   0 379   0   0  16   0   0   0  54   0   0   0   0   0   6
    0   0   0   1   0   0]
 [  0  10   0   0   0 108  44  21   0   0  41   0   0   0   0   0   0   0
    0   0  20   3   0   0]
 [  0   0   2   0   0  21 140  21  26  21   6   0   0   0   0   0  46   0
   55   1   0   0   9   0]
 [  6   0   0  21  70   0  22 144   0   0  19   0   0   0   0  45   0  26
   55   0   4   3  21   0]
 [ 23   0   0   0   0   0   0   0  62   8   0   0  60   0   0  23  20   0
    0  12  18  42   0  20]
 [  0  20   0   3   0   0  66   0   0  77   2   0   0   0  18  74   6   0
    0   3   1  37   7  17]
 [  0   0  10   0   0  92   5   0   0   0  51   0   0  34   0   0   0   0
   17   0   0   0   0   0]
 [ 35   1   0   0  83  14   0  38  10   0   0 109  16   3   0  42   0  23
    0   0   1  19   0   0]
 [110  38   0  21   0  21   4   0   2  17   0  21  26   0   0  20   0  11
    0   0   0   0   0   0]
 [  0   0  59   0  18   0  21   0   0   0   0   0   0  88   0  41   0   0
    0   0  19   0   0   0]
 [  0  42   0   0   0  15   0   0  41   0  18   0   0   1  72   1  26   0
    1   0   0  57  53  20]
 [  0   0   0   0   3  21  20   1   0   0   0  18   0   0   0 101   0   0
    0   0   0   0   0   0]
 [  1  20   0   0   0   0   4   0  21  20  17   0  20   0   0   0   9  20
    0   0  12   0   0   0]
 [ 25   0   0   0   1   0   0  62  20   0   0   0   4   7   0  41   0  33
   41   0   0  12   0   0]
 [  0   0   0   0   0   4  39   0  21   0   2   0   0   2   2  16  41   0
   88   0   0  14  19   0]
 [  0   0   0   0   0   0   1   0  44 100   0   0   1   0   0   2   6  37
   24  35   0   0  16   0]
 [  0   0   0   0   0   0  33   8  42   0   1   0   0   0  32   6  13   0
    0  34  89  20  39  29]
 [  0   0   0   0   0   0   0   0  20  20   0   0   0   0  27   0   1   0
    0   0  26 103   9   0]
 [  0   0   0   0   0   0  19   0   0   0  26   0   0   0  31   0   0   0
    0   0   0  49 142   0]
 [  0   0   0  21   0   0   4  21  62  21   0   0   0   0  36  21   0   0
   17  21  31  15   8  54]]
```

Classification Report (NB):

	precision	recall	f1-score	support
0	0.50	0.72	0.59	331
1	0.71	0.91	0.80	432
2	0.71	0.68	0.70	310
3	0.45	0.22	0.30	245
4	0.64	0.76	0.70	498
5	0.35	0.44	0.39	247
6	0.32	0.40	0.36	348
7	0.43	0.33	0.37	436
8	0.17	0.22	0.19	288
10	0.25	0.23	0.24	331
11	0.27	0.24	0.25	209
12	0.53	0.28	0.36	394
13	0.12	0.09	0.10	291
14	0.44	0.36	0.39	246
15	0.33	0.21	0.25	347
16	0.23	0.62	0.34	164
17	0.05	0.06	0.06	144
18	0.20	0.13	0.16	246
19	0.30	0.35	0.32	248
20	0.25	0.13	0.17	266
21	0.38	0.26	0.31	346
22	0.27	0.50	0.35	206
23	0.36	0.53	0.43	267
24	0.38	0.16	0.23	332
accuracy			0.39	7172
macro avg	0.36	0.37	0.35	7172
weighted avg	0.39	0.39	0.38	7172

SVM

Following are the results for SVM RBF radial based function is used as kernel this model shows decent results compared to other models.

Figure 8: SVM Results

```
Accuracy scores for each fold: [99.59941733430443, 99.74508375819373, 99.74508375819373, 99.56300072833211, 99.67225054624909, 99.70856102003644, 99.48998178506375, 99.5264116575592, 99.5264116575592, 99.85428051001821]
Mean accuracy: 99.64304827555097
Variance: 0.012953504929125873
```

For clarity of above picture, the results are also typed as follows.

Accuracy scores for each fold: [99.59941733430443, 99.74508375819373, 99.74508375819373, 99.56300072833211, 99.67225054624909, 99.70856102003644, 99.48998178506375, 99.5264116575592, 99.5264116575592, 99.85428051001821]

Mean accuracy: 99.64304827555097

Variance: 0.012953504929125873

Test accuracy on the test set: 69.49247071946458

```
SVM Confusion Matrix:
[[317   0   0   0   0   0   0   0   0   0   0   0  14   0   0   0   0   0
    0   0   0   0   0   0]
 [  0 408   0   0   0   0   0   0   0   0   0   0   0   0   0   0   1   0
    5  12   1   3   0   2]
 [  0   0 305   0   0   0   0   5   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0]
 [  0   0   0 228   0   0   0   0   0   0   0   4   0   0   0   0   1   0
    0   4   8   0   0   0]
 [  1   0   0   7 447   0   0   1   0   0   0  35   2   0   5   0   0   0
    0   0   0   0   0   0]
 [  0   0   0   1   0 199  36   2   0   0   4   0   0   0   5   0   0   0
    0   0   0   0   0   0]
 [  0   0   0   1   0   0 234   1  17  15   6   0   0   0   0   4  16   0
   21   7   0   0   7  19]
 [ 22   0   0  17   1   3  41 314   2   9   0   5   0   0   0   8   0   0
   14   0   0   0   0   0]
 [ 12   2   0   0   0   0   0   0 147   0  17   0  29   0   0  11   6  14
    0   0  17   6   0  27]
 [  0  12   0   0   0  19   1   0  21 158   3   0   0   0   6  14  57   3
    1   0   0   2   6  28]
 [  0   0   0   0   0   2   0   0   0   0 207   0   0   0   0   0   0   0
    0   0   0   0   0   0]
 [  0   0   0  18  21   0   0   1   1   0   0 254  48   0   0   7  15  29
    0   0   0   0   0   0]
 [ 31   0   0  30   0  14  21   0  22   0   0  26 102   0   0  11   0  34
    0   0   0   0   0   0]
 [  0   0   0   0  11   0   1   0   0  16   4   0   0 193   0  19   0   0
    2   0   0   0   0   0]
 [  0  10   0   0   0   4   0   0  10  18   0   0   0   0 223  14  13   0
    5   0   3  23   4  20]
 [  0   0   0   0   0   0   0   1   0   0   0   0   0   0   0 142   0  21
    0   0   0   0   0   0]
 [  0   0   0   0   0   0   1   0   3   2   0   0   0   0   0   0  70   3
    0  41   0  24   0   0]
 [  0   0   0   1  16   0   5   1   1   0  15   1   8   0   0  24   0 166
    0   0   0   0   0   8]
 [  0   0   0   0   0  12  22   0  27   0   0   0   0   0  11   0  21   0
  139   0   0   6  10   0]
 [  0   2   0  18   0   0   0   0   0  21   3   2   0   0   0   0  74   5
    0 124   0   0   0  17]
 [  0   0   0  11   0  15  26   0  20   8   0   0   0   0  16   0  18   0
    0  23 146  41   2  20]
 [  0   0   0   0   0   4   0   0   4  20   0   0   0   0  14   0   0   0
    0   0  22 142   0   0]
 [  0   0   0   0   0   0   0   0   0   6  18   0   0   0   1   1   0   0
   15   0  20  42 161   3]
 [  0   0   0  16   0   0   1  55  57   0   0   4   0   0   5   0  23   2
    5   0   6   0   0 158]]
```

Classification Report (SVM):

	precision	recall	f1-score	support
0	0.83	0.96	0.89	331
1	0.94	0.94	0.94	432
2	1.00	0.98	0.99	310
3	0.66	0.93	0.77	245
4	0.90	0.90	0.90	498
5	0.73	0.81	0.77	247
6	0.60	0.67	0.64	348
7	0.82	0.72	0.77	436
8	0.44	0.51	0.47	288
10	0.58	0.48	0.52	331
11	0.75	0.99	0.85	209
12	0.77	0.64	0.70	394
13	0.50	0.35	0.41	291
14	1.00	0.78	0.88	246
15	0.78	0.64	0.70	347
16	0.56	0.87	0.68	164
17	0.22	0.49	0.31	144
18	0.60	0.67	0.63	246
19	0.67	0.56	0.61	248
20	0.59	0.47	0.52	266
21	0.65	0.42	0.51	346
22	0.49	0.69	0.57	206
23	0.85	0.60	0.70	267
24	0.52	0.48	0.50	332
accuracy			0.69	7172
macro avg	0.69	0.69	0.68	7172
weighted avg	0.71	0.69	0.70	7172

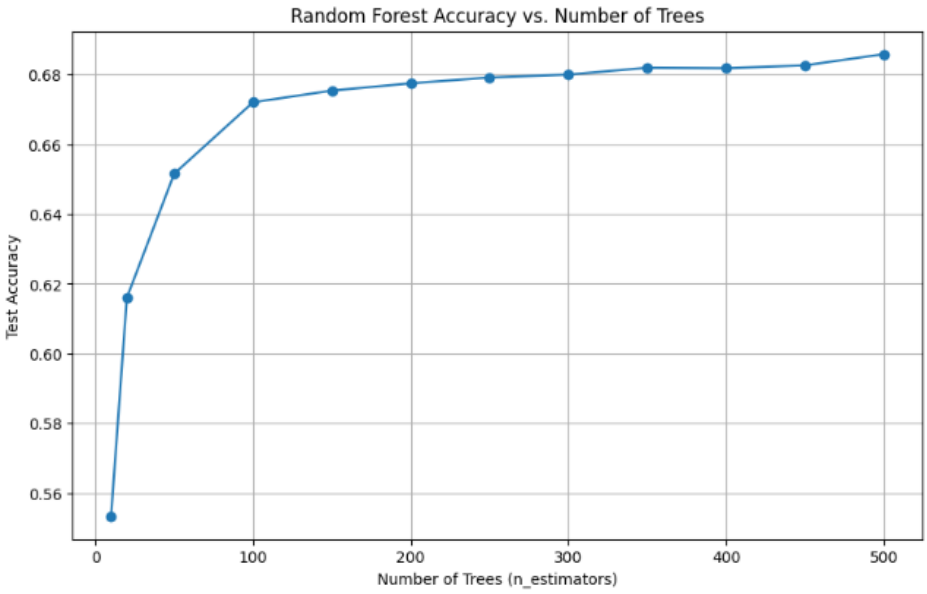
SVM MCC: 0.6818563170464946

Training Time - SVM: 13.56 seconds
Testing Time - SVM: 9.26 seconds

Random Forest

Random forest is an ensemble model which creates multiple decision trees to classify the inputs. Following figures shows using more trees can increase the efficiency. But the accuracy gain is very slow after 300 trees so 300 is taken as number trees as the rest of the model.

Figure 9: Random Forest Graph and accuracy




```
Test accuracy (n_estimators=10): 55.34%
Test accuracy (n_estimators=20): 61.60%
Test accuracy (n_estimators=50): 65.16%
Test accuracy (n_estimators=100): 67.21%
Test accuracy (n_estimators=150): 67.54%
Test accuracy (n_estimators=200): 67.75%
Test accuracy (n_estimators=250): 67.92%
Test accuracy (n_estimators=300): 68.00%
Test accuracy (n_estimators=350): 68.20%
Test accuracy (n_estimators=400): 68.18%
Test accuracy (n_estimators=450): 68.27%
Test accuracy (n_estimators=500): 68.59%
```

The cross-validation result and test accuracy reported below together with MCC and train test time.

Random Forest Results are listed below.
Accuracy scores for each fold: [99.63583394027677, 99.45375091041515, 99.63583394027677, 99.63583394027677, 99.49016751638747, 99.59927140255009, 99.672131147541, 99.38069216757741, 99.30783242258651, 99.48998178506375]
Mean accuracy: 99.53013291729518
Variance: 0.013925535160247682

Test accuracy on the test set: 68.05633017289459

Random Forest MCC: 0.6663156151791096

Training Time - Random Forest (300 trees): 23.35 seconds
Testing Time - Random Forest (300 trees): 0.50 seconds

Peak memory: 12217.96 MiB
Memory increment: 0.00 MiB

Figure 10: Random Forest Confusion Matrix and Classification Report

```
Random Forest Confusion Matrix:
[[319   0   0   0   0   0   0   0   0   0   0   0   1   9   0   0   0   2
    0   0   0   0   0   0]
 [  0 409   0   0   0   0   0   0   0  10   0   0   0   0   0   0   2   0
    0   1   2   2   0   6]
 [  0   0 300   0   0   1   0   1   0   0   0   0   0   1   7   0   0   0
    0   0   0   0   0   0]
 [  0   1   0 232   5   0   0   0   0   0   0   0   1   0   0   0   0   0
    0   0   3   0   3   0]
 [  6   8   0   2 468   0   0   0   0   0   0   2   0   3   3   1   0   4
    0   0   0   1   0   0]
 [  0   5   0   1   0 211   6   0   0   0  17   0   0   0   7   0   0   0
    0   0   0   0   0   0]
 [  0   0   0   2   0   2 229   0  19   0   0   0   0   1   0   3  21   0
   45   0   0   0  21   5]
 [  1   0   0   0   9   0  48 332   7   0   1   0   0   1   0   0   0   0
   25   0   0   0  12   0]
 [ 12   0   0   0   0   0   0   2 135   5  18   0  31   0   3  11  19  13
    5   0   0   3   0  31]
 [  0  20   0   2   0   6   0   3   5 142  18   0   0   0  20  25  52   1
    4  16   0  15   0   2]
 [  0   0   1   0   0   3   0   0   0   0 205   0   0   0   0   0   0   0
    0   0   0   0   0   0]
 [ 15   0   0   8  25   0   0  19   1   1   1 228  21   0   0  17   3  35
    0   0  17   2   0   1]
 [ 45   0   0  16   0  19   9   0   1  11   1  35  74   0   0  23   0  34
    0   3   0   1   0  19]
 [  0   0   2   0  19   0  18   0   0   2   0   0   0 169   0   6   8   0
   21   0   1   0   0   0]
 [  0   6   0   0   0   2   1   0   0   9   3   0   0   0 237  27   9   0
    1   2  14  15   3  18]
 [  0   0   0   1   0   0   0   0   0   0   0   5   0   0   2 143   0  13
    0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0  23  17   0   0   0   0   0  62   1
    0  40   0   1   0   0]
 [  0   0   0   3   2   2   1   5  14   2   2   2   0   0   6  17   2 151
   17  17   0   0   2   1]
 [  0   0   0   0   0   7  20   3  16   0  13   0   0   0   6   2  22   0
  137   0   1   0  21   0]
 [  0  17   0  23   0   0   0   0   0  33   0   0   0   0   0   0  61   0
    0 132   0   0   0   0]
 [  0   2   0   0   0   4  34   0  21  12  16   0   0   0   7   1   9   1
    3  30 158  10  15  23]
 [  0   2   0   0   0   7   0   0   3  31   0   0   0   0   4   0   4   0
    0   0  28 126   1   0]
 [  0   0   0   0   0   3   1   0   0   1   7   0   3   0   2   1   1  21
   19   0  17  21 158  12]
 [  0   1   0  17   0  15   0  18  69   0   0   3   0   0   8   0  23   2
   39   0  12   5   0 120]]
```

Random Forest Classification Report:

	precision	recall	f1-score	support
0	0.80	0.96	0.88	331
1	0.87	0.95	0.91	432
2	0.99	0.97	0.98	310
3	0.76	0.95	0.84	245
4	0.89	0.94	0.91	498
5	0.75	0.85	0.80	247
6	0.62	0.66	0.64	348
7	0.87	0.76	0.81	436
8	0.46	0.47	0.47	288
10	0.50	0.43	0.46	331
11	0.64	0.98	0.78	209
12	0.83	0.58	0.68	394
13	0.56	0.25	0.35	291
14	0.92	0.69	0.79	246
15	0.76	0.68	0.72	347
16	0.52	0.87	0.65	164
17	0.21	0.43	0.28	144
18	0.54	0.61	0.58	246
19	0.43	0.55	0.49	248
20	0.55	0.50	0.52	266
21	0.62	0.46	0.53	346
22	0.62	0.61	0.62	206
23	0.67	0.59	0.63	267
24	0.50	0.36	0.42	332
accuracy			0.68	7172
macro avg	0.66	0.67	0.65	7172
weighted avg	0.69	0.68	0.68	7172

XGBoost

XGBoost is also ensemble model (boosting ensemble method). This model give error once on training the data using y_train which doesn't include a class 9 or J therefore label encoder is used to number the class continuously 0 to 23 rather than 0 to 24 excluding 9. This will make the code work and following the results.

XGBoost Results are listed below.
Accuracy scores for each fold (XGBoost): [98.5797523670794, 98.61616897305171, 98.39766933721778, 98.06991988346687, 98.94391842680263, 98.5063752276867, 98.90710382513662, 98.54280510018215, 97.77777777777777, 98.39708561020036]
Mean accuracy (XGBoost): 98.4738576528602
Variance (XGBoost): 0.1105328242522369

Test accuracy on the test set (XGBoost): 64.07%
XGBoost MCC: 0.6252974061811201

Training Time - XGBoost: 10.17 seconds
Testing Time - XGBoost: 0.07 seconds

Peak memory: 10192.00 MiB
Memory increment: 1.80 MiB

Figure 11: XGBoost Confusion Matrix and Classification Report

```
XGBoost Confusion Matrix:
[[304   0   0   5   0   0   0   0   2   0   0   1   7   5   0   0   0   7
    0   0   0   0   0   0]
 [  0 386   0   0   0   0   0   0   4   3   0   0   0   0   0   0   5   0
    6   9   2  17   0   0]
 [  0   0 256   3   0   7   0   1   0   0   2   1   0  27   6   0   0   0
    0   0   1   1   0   5]
 [  0   0   0 195   7   1   0   0   0   0   0   0   5   0   0   3  12   1
    0   2  15   0   4   0]
 [ 23   1   0   0 440   0   0   2   3   0   0  11   1   1   6   1   0   4
    1   0   0   4   0   0]
 [  0  13   0   4   0 177  24   0   5   0  16   0   0   0   8   0   0   0
    0   0   0   0   0   0]
 [ 14   0   0   4   4   0 211  18   0   3   0   0   0   2   3   2  27   0
   32   7   0   2  10   9]
 [  1   0   0   3  11   0  39 327   3   3   0   0   0   0   0   8   0   3
   22   1   0   0   1  14]
 [ 12   0   0   1   0   0   0   0 148   7   7   0  23   0   0  12   0  12
    8   0  21   8  12  17]
 [  0  20   0   0   0   8   1   0   7 143   6   0   0   0  17   2  72   3
   19   4   1   9   7  12]
 [  0   0   1   0   0   3   0   0   0   0 205   0   0   0   0   0   0   0
    0   0   0   0   0   0]
 [ 25   1   0  21  26   0   0   0   8   1   0 185  46   0  10   9  11  40
    0   0  10   0   0   1]
 [ 27   0  19  20   0   6   3   0  20   7   2  18  92   2   0  35   0  24
    3   0   0   3   0  10]
 [  0   0   7   0  15  15   1   0   0   3   1   0   0 146  18  17   1   2
   20   0   0   0   0   0]
 [  0   4   0   0   0   2   0   1   3  19   1   0   0   0 207  16   3   0
    5   0  18  49   0  19]
 [  1   0   0   2   0   4   0   0   0   0   0   2   0   0   0 136   0  18
    0   0   0   0   0   1]
 [  0   0   0   0   0   1   0   0   0  20   2   1   0   0   2   0  59   2
    0  38   0  19   0   0]
 [  0   2   0  10   6   5   2   1   5   1   0   8   0   0   8  17   3 140
   11  20   1   0   0   6]
 [  0   0   4   0   0   0  23   4  14   0  15   0   0   0   9   0  38   2
  123   0   0   0  16   0]
 [  0   1   0   0   0   0   0   0   3  26   0   3   0   0   0   0  67   8
    0 137   0   4   0  17]
 [  0   5   0   0   0  10  19   2  22   0  10   0   0   0  15   0  23   3
   12  31 154  14   2  24]
 [  0   0   0   0   0  15   0   0   2  21   1   0   0   0   3   3   0   0
    0   0  21 120   0  20]
 [  0   0  11   0   0   0   3   0   0   1   2   0   0   0  20   0   5  18
    6   0   3  26 157  15]
 [  0   0   0  15   0   8   6  20  65   0   4   1   1   0   1   0  17   2
   37   0   2   1   5 147]]
```

Classification Report (XGBoost):

	precision	recall	f1-score	support
0	0.75	0.92	0.82	331
1	0.89	0.89	0.89	432
2	0.86	0.83	0.84	310
3	0.69	0.80	0.74	245
4	0.86	0.88	0.87	498
5	0.68	0.72	0.70	247
6	0.64	0.61	0.62	348
7	0.87	0.75	0.81	436
8	0.47	0.51	0.49	288
9	0.55	0.43	0.49	331
10	0.75	0.98	0.85	209
11	0.80	0.47	0.59	394
12	0.53	0.32	0.39	291
13	0.80	0.59	0.68	246
14	0.62	0.60	0.61	347
15	0.52	0.83	0.64	164
16	0.17	0.41	0.24	144
17	0.48	0.57	0.52	246
18	0.40	0.50	0.44	248
19	0.55	0.52	0.53	266
20	0.62	0.45	0.52	346
21	0.43	0.58	0.50	206
22	0.73	0.59	0.65	267
23	0.46	0.44	0.45	332
accuracy			0.64	7172
macro avg	0.63	0.63	0.62	7172
weighted avg	0.66	0.64	0.64	7172

CNN

First step in CNN model is preprocessing step which include reshaping of input image to 4D tensor of shape (width, height, number of channels, 1). The number of channels is taken as 1. Also, the last dimension is 1 because it is a pixel are in greyscale format. Therefore, X_train_reshape has shape of (27455, 48, 1, 1) Secondly input tensor, (x_train_reshape) is converted into float data type and normalize to have value between 0 and 1. Furthermore target column is numerical, so it is converted to categorical using one hot coding.

Configuration of simple CNN model

Following is Configuration for simple CNN model

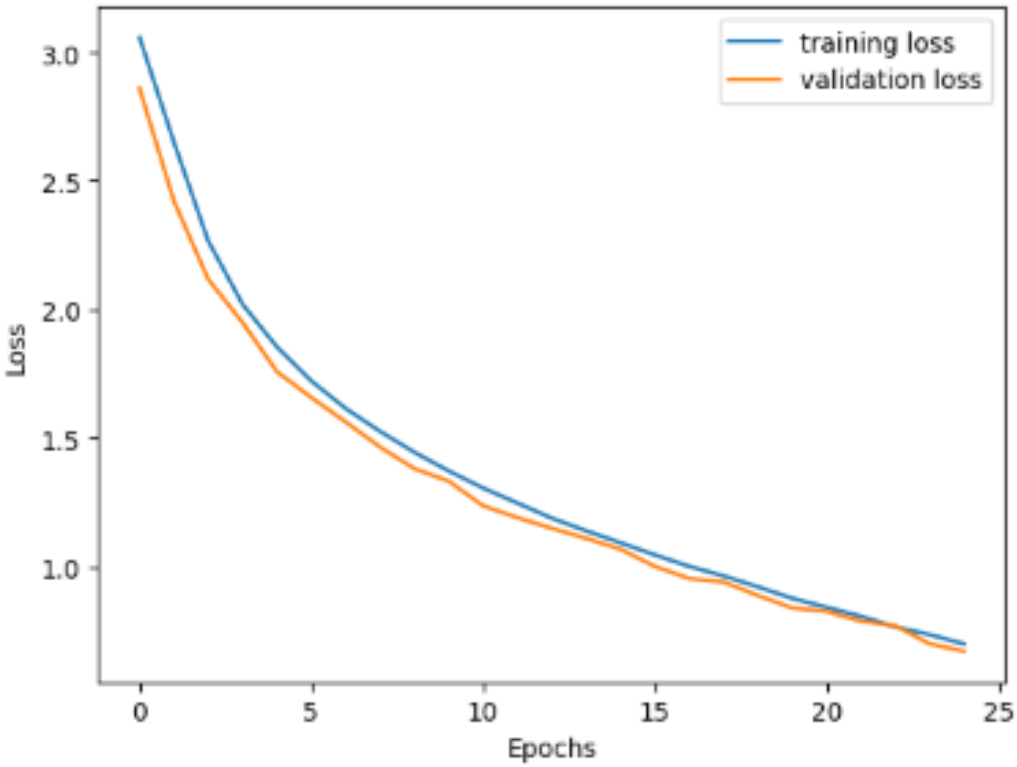
optimizer='rmsprop', epochs=25, batch_size = 512 validation ratio=0.2

simple_cnn_model: 1 conv layers with 32 channels and (3x1 kernels) followed by a max pooling layer (2x1). And a final dense layer of 128 neurons before the output layer.

Figure 12: CNN Model Training

```
Epoch 1/25
43/43 [==============================] - 1s 8ms/step - loss: 3.0583 - accuracy: 0.1350 - val_loss: 2.8596 - val_accuracy: 0.2473
Epoch 2/25
43/43 [==============================] - 0s 4ms/step - loss: 2.6508 - accuracy: 0.2869 - val_loss: 2.4216 - val_accuracy: 0.3626
Epoch 3/25
43/43 [==============================] - 0s 4ms/step - loss: 2.2661 - accuracy: 0.3938 - val_loss: 2.1173 - val_accuracy: 0.4236
Epoch 4/25
43/43 [==============================] - 0s 4ms/step - loss: 2.0200 - accuracy: 0.4485 - val_loss: 1.9491 - val_accuracy: 0.4628
Epoch 5/25
43/43 [==============================] - 0s 4ms/step - loss: 1.8541 - accuracy: 0.4877 - val_loss: 1.7580 - val_accuracy: 0.5183
Epoch 6/25
43/43 [==============================] - 0s 4ms/step - loss: 1.7219 - accuracy: 0.5172 - val_loss: 1.6575 - val_accuracy: 0.5312
Epoch 7/25
43/43 [==============================] - 0s 4ms/step - loss: 1.6150 - accuracy: 0.5427 - val_loss: 1.5629 - val_accuracy: 0.5611
Epoch 8/25
43/43 [==============================] - 0s 4ms/step - loss: 1.5260 - accuracy: 0.5676 - val_loss: 1.4652 - val_accuracy: 0.5842
Epoch 9/25
43/43 [==============================] - 0s 4ms/step - loss: 1.4447 - accuracy: 0.5872 - val_loss: 1.3808 - val_accuracy: 0.6066
Epoch 10/25
43/43 [==============================] - 0s 4ms/step - loss: 1.3720 - accuracy: 0.6040 - val_loss: 1.3322 - val_accuracy: 0.6046
Epoch 11/25
43/43 [==============================] - 0s 4ms/step - loss: 1.3055 - accuracy: 0.6234 - val_loss: 1.2365 - val_accuracy: 0.6547
Epoch 12/25
43/43 [==============================] - 0s 4ms/step - loss: 1.2471 - accuracy: 0.6352 - val_loss: 1.1902 - val_accuracy: 0.6520
Epoch 13/25
43/43 [==============================] - 0s 4ms/step - loss: 1.1883 - accuracy: 0.6528 - val_loss: 1.1488 - val_accuracy: 0.6616
Epoch 14/25
43/43 [==============================] - 0s 4ms/step - loss: 1.1393 - accuracy: 0.6658 - val_loss: 1.1098 - val_accuracy: 0.6675
Epoch 15/25
43/43 [==============================] - 0s 4ms/step - loss: 1.0926 - accuracy: 0.6785 - val_loss: 1.0679 - val_accuracy: 0.6687
Epoch 16/25
43/43 [==============================] - 0s 4ms/step - loss: 1.0455 - accuracy: 0.6920 - val_loss: 1.0007 - val_accuracy: 0.7044
Epoch 17/25
43/43 [==============================] - 0s 4ms/step - loss: 1.0008 - accuracy: 0.7077 - val_loss: 0.9534 - val_accuracy: 0.7090
Epoch 18/25
43/43 [==============================] - 0s 4ms/step - loss: 0.9637 - accuracy: 0.7187 - val_loss: 0.9401 - val_accuracy: 0.7237
Epoch 19/25
43/43 [==============================] - 0s 4ms/step - loss: 0.9212 - accuracy: 0.7290 - val_loss: 0.8874 - val_accuracy: 0.7489
Epoch 20/25
43/43 [==============================] - 0s 4ms/step - loss: 0.8765 - accuracy: 0.7423 - val_loss: 0.8392 - val_accuracy: 0.7596
Epoch 21/25
43/43 [==============================] - 0s 4ms/step - loss: 0.8417 - accuracy: 0.7515 - val_loss: 0.8256 - val_accuracy: 0.7585
Epoch 22/25
43/43 [==============================] - 0s 4ms/step - loss: 0.8065 - accuracy: 0.7634 - val_loss: 0.7876 - val_accuracy: 0.7698
Epoch 23/25
43/43 [==============================] - 0s 4ms/step - loss: 0.7655 - accuracy: 0.7783 - val_loss: 0.7701 - val_accuracy: 0.7658
Epoch 24/25
43/43 [==============================] - 0s 4ms/step - loss: 0.7354 - accuracy: 0.7845 - val_loss: 0.6996 - val_accuracy: 0.8039
Epoch 25/25
43/43 [==============================] - 0s 4ms/step - loss: 0.6992 - accuracy: 0.7961 - val_loss: 0.6706 - val_accuracy: 0.8141
```

Figure 13: CNN Model Training and Validation Loss



225/225 [==============================] - 1s 2ms/step –

loss: 1.6922 - accuracy: 0.5165

Test accuracy: 51.65%

From above graph, it seems underfitting as validation loss has decreasing trend and increasing number of epochs might reduce validation loss further. Therefore, regarding research question 2, hyperparameter tuning is done by increasing the number of epochs. The number of epochs that is chosen considering memory and time consumption is 50 and model is run again. There is 5% increase in accuracy. Different hyperparameter tuning was also carried out but since the code is giving errors, therefore they have been left out of the analysis.

The following is the list of model training after hyperparameter tuning.

Epoch 1/50

54/54 [==============================] - 1s 3ms/step - loss: 3.0043 - accuracy: 0.1705
Epoch 2/50
54/54 [==============================] - 0s 3ms/step - loss: 2.4857 - accuracy: 0.3265

```
Epoch 3/50
54/54 [==============================] - 0s 3ms/step - loss: 2.1170 - accuracy: 0.4118
Epoch 4/50
54/54 [==============================] - 0s 3ms/step - loss: 1.8893 - accuracy: 0.4704
Epoch 5/50
54/54 [==============================] - 0s 3ms/step - loss: 1.7285 - accuracy: 0.5145
Epoch 6/50
54/54 [==============================] - 0s 3ms/step - loss: 1.5970 - accuracy: 0.5465
Epoch 7/50
54/54 [==============================] - 0s 3ms/step - loss: 1.4830 - accuracy: 0.5776
Epoch 8/50
54/54 [==============================] - 0s 3ms/step - loss: 1.3832 - accuracy: 0.6006
Epoch 9/50
54/54 [==============================] - 0s 3ms/step - loss: 1.2958 - accuracy: 0.6251
Epoch 10/50
54/54 [==============================] - 0s 3ms/step - loss: 1.2109 - accuracy: 0.6451
Epoch 11/50
54/54 [==============================] - 0s 3ms/step - loss: 1.1408 - accuracy: 0.6676
Epoch 12/50
54/54 [==============================] - 0s 3ms/step - loss: 1.0697 - accuracy: 0.6847
Epoch 13/50
54/54 [==============================] - 0s 3ms/step - loss: 1.0074 - accuracy: 0.7021
Epoch 14/50
54/54 [==============================] - 0s 3ms/step - loss: 0.9481 - accuracy: 0.7207
Epoch 15/50
54/54 [==============================] - 0s 3ms/step - loss: 0.8951 - accuracy: 0.7354
Epoch 16/50
54/54 [==============================] - 0s 3ms/step - loss: 0.8446 - accuracy: 0.7527
Epoch 17/50
54/54 [==============================] - 0s 3ms/step - loss: 0.7943 - accuracy: 0.7654
Epoch 18/50
54/54 [==============================] - 0s 3ms/step - loss: 0.7443 - accuracy: 0.7839
Epoch 19/50
54/54 [==============================] - 0s 3ms/step - loss: 0.7021 - accuracy: 0.7965
Epoch 20/50
54/54 [==============================] - 0s 3ms/step - loss: 0.6585 - accuracy: 0.8081
Epoch 21/50
54/54 [==============================] - 0s 3ms/step - loss: 0.6163 - accuracy: 0.8246
Epoch 22/50
54/54 [==============================] - 0s 3ms/step - loss: 0.5800 - accuracy: 0.8347
Epoch 23/50
54/54 [==============================] - 0s 3ms/step - loss: 0.5374 - accuracy: 0.8499
Epoch 24/50
54/54 [==============================] - 0s 3ms/step - loss: 0.5022 - accuracy: 0.8616
Epoch 25/50
54/54 [==============================] - 0s 3ms/step - loss: 0.4656 - accuracy: 0.8735
```

```
Epoch 26/50
54/54 [==============================] - 0s 3ms/step - loss: 0.4328 - accuracy: 0.8862
Epoch 27/50
54/54 [==============================] - 0s 3ms/step - loss: 0.3973 - accuracy: 0.8995
Epoch 28/50
54/54 [==============================] - 0s 3ms/step - loss: 0.3712 - accuracy: 0.9085
Epoch 29/50
54/54 [==============================] - 0s 3ms/step - loss: 0.3392 - accuracy: 0.9202
Epoch 30/50
54/54 [==============================] - 0s 3ms/step - loss: 0.3137 - accuracy: 0.9295
Epoch 31/50
54/54 [==============================] - 0s 3ms/step - loss: 0.2876 - accuracy: 0.9364
Epoch 32/50
54/54 [==============================] - 0s 3ms/step - loss: 0.2643 - accuracy: 0.9455
Epoch 33/50
54/54 [==============================] - 0s 3ms/step - loss: 0.2411 - accuracy: 0.9532
Epoch 34/50
54/54 [==============================] - 0s 3ms/step - loss: 0.2202 - accuracy: 0.9570
Epoch 35/50
54/54 [==============================] - 0s 3ms/step - loss: 0.1984 - accuracy: 0.9653
Epoch 36/50
54/54 [==============================] - 0s 3ms/step - loss: 0.1805 - accuracy: 0.9690
Epoch 37/50
54/54 [==============================] - 0s 3ms/step - loss: 0.1627 - accuracy: 0.9767
Epoch 38/50
54/54 [==============================] - 0s 3ms/step - loss: 0.1465 - accuracy: 0.9797
Epoch 39/50
54/54 [==============================] - 0s 3ms/step - loss: 0.1310 - accuracy: 0.9827
Epoch 40/50
54/54 [==============================] - 0s 3ms/step - loss: 0.1204 - accuracy: 0.9854
Epoch 41/50
54/54 [==============================] - 0s 3ms/step - loss: 0.1047 - accuracy: 0.9879
Epoch 42/50
54/54 [==============================] - 0s 3ms/step - loss: 0.0961 - accuracy: 0.9897
Epoch 43/50
54/54 [==============================] - 0s 3ms/step - loss: 0.0841 - accuracy: 0.9922
Epoch 44/50
54/54 [==============================] - 0s 3ms/step - loss: 0.0780 - accuracy: 0.9918
Epoch 45/50
54/54 [==============================] - 0s 3ms/step - loss: 0.0690 - accuracy: 0.9946
Epoch 46/50
54/54 [==============================] - 0s 3ms/step - loss: 0.0616 - accuracy: 0.9948
Epoch 47/50
54/54 [==============================] - 0s 3ms/step - loss: 0.0551 - accuracy: 0.9957
Epoch 48/50
54/54 [==============================] - 0s 3ms/step - loss: 0.0485 - accuracy: 0.9968
```

Epoch 49/50
54/54 [==============================] - 0s 3ms/step - loss: 0.0441 - accuracy: 0.9969
Epoch 50/50
54/54 [==============================] - 0s 3ms/step - loss: 0.0391 - accuracy: 0.9976
225/225 [==============================] - 1s 2ms/step - loss: 2.2860 - accuracy: 0.5648
test_loss: 2.286, test_acc: 0.565

7. Results and Discussion

Regarding Q1, KNN shows better accuracy (70%) compared to CNN (52%) as shown in Figure 14 below. Also, KNN is the most stable model having the lowest cross-validation variance.

The second model that performs after KNN is SVM which has an accuracy of 69 % and is third best stable model having cross validation variance of 0.013 furthermore it has an MCC of 0.68 which is the best among all models. The training and test times respectively 13.6 seconds and 9.3 seconds which is second best among models.

Third best is Random Forest which has an accuracy of 68% and MCC of 0.67. Furthermore, it has a cross-validation variance of 0.014 also train/test times are respectively 23.4 and 0.5 seconds. Moreover, it has a peak memory consumption of 12218 MiB.

The next best model is XGBoost with having accuracy of 64 % and MCC of 0.63 also it has cross validation variance of 0.11. Furthermore, it is the top method concerning time efficiency with 10.2 seconds of training time and 0.07 seconds of testing time respectively. Moreover, it has peak memory consumption of 10192 MiB

Figure 14: Model Comparison.

Model	Test Accuracy (%)	Crosss validation variance	MCC	Train time (sec)	Test time (sec)	Peak memory (MiB)	Overall Rank
KNN Baseline	70	0.0000018					1st
NB	39	0.00015					
SVM	69	0.013	0.68	13.6	9.3		2nd
RF (300 tress)	68	0.014	0.67	23.4	0.5	12218	3rd
XGBoost	64	0.11	0.63	10.2	0.07	10192	
CNN	52						
CNN tuned	57						

As seen from the above table the accuracy of CNN model and other traditional machine learning models including ensemble models like random forest and XGBoost all show very low accuracy which means there is a need for more observation or adding new variables that improve more accuracy

8. Analysis Limitations

CNN shows very less accuracy and is unable to beat the baseline model which contradicts previous research results therefore it needs further investigation regarding configuration and hyperparameter tuning as discussed before. More data is needed, or data augmentation can help in better prediction. Furthermore, new features or combing features can be added that can help in increasing the efficiency of the model. Other variants of CNN can also be explored such as MobilNet, AlexNet, LeNet, VGG, etc. Furthermore, during CNN modeling some of the code gives errors such as in calculating MCC, Test train time, and some of the hyperparameter tuning therefore they are excluded from the analysis due to time constraints.

Another limitation of the analysis is the exclusion of research question 3 from the analysis due to time constraints which is about using the Bayesian network to find the causation of the target variable.

9. Conclusion and Future Directions

Since the human hand is involved which is the personal data care should be taken while taking more data therefore privacy of personal data should be kept. Three different categories of machine learning models have been used in the project. The first one is traditional ML models such as KNN baseline, SVM, and Naïve Bayes, the second category is the ensemble methods which combine the prediction of multiple models to increase the accuracy. Random forest which is a bagging ensemble method and XBoost which is the boosting ensemble model have been used in the project. The third category is deep learning which includes CNN has been used in the project. The results and analysis limitation show that KNN is the best-performing model for this project with a test accuracy of 70%. CNN doesn't perform well as discussed before which can be a topic to explore for future studies. In particular grid search can be used available in scikit-learn python library to tune the hyperparameters of keras’s CNN model. (Jason). In addition to that Bayesian network can be explored to find the cause of labels in the sign language MNIST dataset.

10. References

[1] Kumar et al (2023). Mediapipe and CNNs for Real-Time ASL Gesture Recognition. Retrieved from https://arxiv.org/abs/2305.05296

[2] Pigou et al (2015). Sign Language Recognition Using Convolutional Neural Networks. Retrieved from https://link-springer-com.ezproxy.lib.torontomu.ca/chapter/10.1007/978-3-319-16178-5_40

[3] Eman et al (2023). Arabic Sign Language Recognition using Convolutional Neural Network and MobileNet. Retrieved from https://journals-scholarsportal-info.ezproxy.lib.torontomu.ca/details/2193567x/v48i0002/2147_aslrucnnam.xml

[4] Lanxi et al (2022). American Sign Language Recognition Based on Machine learning and Neural Network. Retrieved from https://ieeexplore-ieee-org.ezproxy.lib.torontomu.ca/document/9943235

[5] Hasan et al. (2020). Classification of Sign Language Characters by Applying a Deep Convolutional Neural Network. Retrieved from https://ieeexplore-ieee-org.ezproxy.lib.torontomu.ca/document/9333456

[6] Karayaneva et al (2018). Object Recognition in Python and MNIST Dataset Modification and Recognition with Five Machine Learning Classifiers. Retrieved from https://www.joig.net/uploadfile/2018/0717/20180717055805469.pdf

[7] Khlifia et al.(2012). New approach using Bayesian Network to improve content based image classification systems. Retrieved from https://arxiv.org/abs/1204.1631

[8] OpenAI. (2023). ChatGPT (GPT-4 model) [Large language model]. https://chat.openai.com/chat

[9] Kaggle (2018). Sign Language MNIST. Retrieved from https://www.kaggle.com/datasets/datamunge/sign-language-mnist

[10] Jason. (2022). How to Grid Search Hyperparameters for Deep Learning Models in Python with Keras. Retrieved from https://machinelearningmastery.com/grid-search-hyperparameters-deep-learning-models-python-keras/

[11] Yadav et al. (2019). Recognition of Static Hand Gestures in Indian Sign Language.

[12] Singh, A. (2020, July). Sign language MNIST problem: American Sign Language. Retrieved from https://medium.com/@abhkmr30/sign-language-mnist-problem-american-sign-language-48896ea960e0

11. Appendix

Nomenclature

ASL: American Sign Language

AI: Artificial Intelligence

AMI: Advanced Metering Infrastructure

ANN: Artificial Neural Network

BIC: Bayesian Criterion

BP: Back-Propagation

BPN: Back-Propagation Network

GBRT: Gradient Boosted Regression Trees

CBIR: Content-Based Image Retrieval

CNN: Convolution Neural Networks

CPU: Central Processing Unit

CV: Computer Vision

DNN: Deep Neural Networks

FAN: Forest Augmented Naive Bayes

FCN: Fully Connected Network

GA: Genetic Algorithm

GLCM: Grey Level Co-occurrence Matrix

GMM: Gaussian Mixture Model

GNB: Gaussian Naive Bayes

GPU: Graphical Processing Unit

ICA: Independent Component Analysis

KNN: K-Nearest Neighbor

KPI: Key Performance Indicator

LDA: Linear Discriminate Analysis

MNIST: Modified National Institute of Standards and Technology

MAP: Maximum a posteriori

MAE: Mean Absolute Error

MAPE: Mean Absolute Percentage Error

MCC: Matthew's Correlation Coefficient

ML: Machine Learning

MLP: Multi-layer Perceptron

MRF: Markov Random Field

NB: Naive Bayesian Network

PCA: Principal Component Analysis

PCC: Percentage of correct classification

PDF: Probability Density Function

ReLU: Rectified Linear Unit

RMSE: Root Mean Square Error

RMSPE: Root Mean Square Percentage Error

RFC: Random Forest Classification

RFI: Random Forest Importance

RBF: Radial Basis Function

RFE: Recursive Feature Elimination

RNN: Recurrent Neural Network

SLR: Sign Language Recognition

SVM: Support Vector Machine

SVR: Support Vector Machine Regression

SGD: Stochastic Gradient Descent

TAN: Tree Augmented Naive Bayes