Logo

Description automatically generated

(7088CEM)

Coursework

Artificial Neural Networks Application

**Module Leader: Dr. Marwan Fuad**

**Student Name:** Jyothi Yendamuri

**SID:** 11467683

**Sign Language Recognition Using Machine Learning and Deep Learning Algorithms**

Table of Contents

[**Abstract: 3**](#_Toc87583323)

[**1. Objective/Aim 3**](#_Toc87583324)

[**2. Introduction: 3**](#_Toc87583325)

[**3. Challenges: 4**](#_Toc87583326)

[**4. Installations: 4**](#_Toc87583327)

[**4.0 Python Libraries: 5**](#_Toc87583328)

[**4.1. TensorFlow: 5**](#_Toc87583329)

[**4,2. Keras: 6**](#_Toc87583330)

[**4.5 Pandas: 7**](#_Toc87583331)

[**5. Related Work: 7**](#_Toc87583332)

[**Dataset: 8**](#_Toc87583333)

[**6. Methodology 9**](#_Toc87583334)

[**6.1 Data Processing: 9**](#_Toc87583335)

[**6.2 Training: 10**](#_Toc87583336)

[**6.3 Classify Gesture: 10**](#_Toc87583337)

[**7. THE PROPOSED SYSTEM 10**](#_Toc87583338)

[**7.1 Data Collection: 10**](#_Toc87583339)

[**7.2 Image Pre-Processing: 10**](#_Toc87583340)

[**7.3 Feature Extraction: 11**](#_Toc87583341)

[**7.4. Classification: 11**](#_Toc87583342)

[**8.Algorithms: 11**](#_Toc87583343)

[**Machine Learning Algorithms: 11**](#_Toc87583344)

[**8.1 K-Nearest Neighbor: 11**](#_Toc87583345)

[**8.2 Support Vector Machine: 11**](#_Toc87583346)

[**8.3 Decision Tree: 12**](#_Toc87583347)

[**8.4 Random Forest Algorithm: 12**](#_Toc87583348)

[**Deep Learning Algorithms: 12**](#_Toc87583349)

[**8.5 Convolutional Neural Network (CNN) Algorithm: 12**](#_Toc87583350)

[**8.6 Artificial Neural Network: 13**](#_Toc87583351)

[**8.7. Back propagation: 14**](#_Toc87583352)

[**8.8 . Autoencoder: 14**](#_Toc87583353)

[**9.Experiment Results: 15**](#_Toc87583354)

[**10. Conclusion: 17**](#_Toc87583355)

[**11. Future Work: 17**](#_Toc87583356)

# 

Table of Figures

[Figure 1: Python version 5](#_Toc87621643)

[Figure 2: Pip version 5](#_Toc87621644)

[Figure 3: TensorFlow Installation 6](#_Toc87621645)

[Figure 4. TensorFlow Information 6](#_Toc87621646)

[Figure 5: Installing Keras 7](#_Toc87621647)

[Figure 6. NumPy version 7](#_Toc87621648)

[Figure 7: Checking pandas 7](#_Toc87621649)

[Figure 8 . Picture category 8](#_Toc87621650)

[Figure 9: American Sign Language Alphabet [Ref(2)] 9](#_Toc87621651)

[Figure 10: Software Structure [Anna Deza] 10](#_Toc87621652)

[Figure11 : Model Architecture as implemented in Using Deep Convolutional Networks [Anna Deza] 13](#_Toc87621653)

[Figure 12: Model Architecture [Anna Deza] 14](#_Toc87621654)

[Figure13 : Artificial Neural Network Architecture [InnoArchiTech] 14](#_Toc87621655)

[Figure 14: Autoencoder Architecture (research gate) 15](#_Toc87621656)

[Figure 15: Test Accuracy and Loss 16](#_Toc87621657)

[Figure 16: SVM and RFC confusion matrix 17](#_Toc87621658)

# Abstract:

The advancement of sign language recognition is a significant step forward in enhancing communication between those who are deaf or hearing-impaired and general people. People with hearing disabilities confront a lot of hurdles while communicating with normal people. Many developed countries made systems for resolving a communication problem with deaf people. Sign Language Recognition (SLR) is an advanced platform in contemporary computer research and major breakthrough that bridges the gap between deaf and normal people. This report proposes the recognition of American Sign Language (ASL) gestures using powerful deep learning tools convolutional Neural Networks (CNN), Autoencoders to predict the sign letters. The system contains several stored images that depict the exact sign in this language, used to train a multilayer neural network using a Artificial neural networks algorithm.

***Keywords:***Sign Language Recognition, Convolutional Neural Networks, Artificial Neural Networks, Sign Recognition, Gesture Recognition, Multi-Layer Neural Network.

# 1. Objective/Aim

The main aim of this project is to develop deep learning techniques and machine learning algorithms that can determine which letter of the American Sign Language alphabet is being signed from a picture of a hand motion. The goal is to figure out which model is the most accurate in predicting the signals.

# 2. Introduction:

Sign Language is used by deaf and hard-of-hearing persons. Hand gestures play an important part in communication and provide us with a unique, natural, and user-friendly way of communicating with computers that is more familiar to humans. Sign language has its own syntax and vocabulary which is entirely different from other spoken and written languages. Moreover, SL is not an international language, very few people understand signs. The goal of this project is to advance the field of automated sign language recognition [2]. The development of sign language applications will help those who do not understand signs, and they can also easily communicate with unhearing and dumb people. Many deaf and mute people would benefit significantly from this effort since it would make it easier for them to communicate with others in everyday situations.

Static and dynamic are the two types of sign gestures. Static gestures have a fixed hand position, whereas dynamic gestures have hands and body components that move. In order to construct SLR for human behaviors in spatio-temporal data, there are two essential processes. The first step is to take the frame sequences and extract their features. As a result, a representation made up of one or more feature vectors (also known as descriptors) is created. This representation will assist the computer in distinguishing between the various classes of actions. The second step is the categorization of the action. These representations will be used by a classifier to distinguish between the various signs. Convolutional neural networks are used to automate feature extraction in our research (CNNs). An artificial neural network (ANN) is utilized for the classification.

# 3**. Challenges:**

American Sign Language uses both hands to make motions, resulting in the occlusion of features. Indian Sign Language, British sign language also affected by geographical differences and the presence of several indicators for the same character. Certain characters have the same alphabet, for example, ‘V ‘and 2 have the same sign and ‘W’ and 3 have the same sign. The sign resolution is context sensitive.

# 4. Installations:

We need to download some software to start working on the project. So below are the steps discussed to download all the relevant software and tools.

**Python:**

In this project, we use a python programming language to implement the code. Python is one of the most popular and frequently used programming languages in the business and has displaced a number of other computer languages.

First, we will check whether python is installed in our operating system or not. To check this open the Anaconda prompt and type “python”.

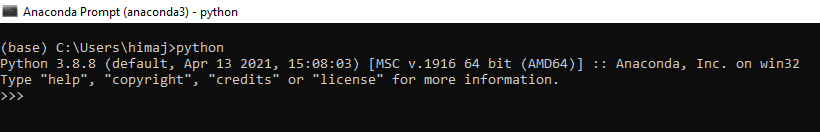


Figure 1: Python version

As shown in the above figure Python 3.8.8 version has been installed.

Next, we will check whether “pip” is installed or not. Pip (Preferred Installer Program) is a python package management system for installing and managing software packages. Pip connects to the python package index, an online repository of public packages. Type below command in Anaconda prompt to see if pip is installed.

*Pip --version*

If pip is installed, we can see the version of the pip like below:

*pip 21.0.1 from C:\Users\himaj\anaconda3\lib\site-packages\pip (python 3.8)*

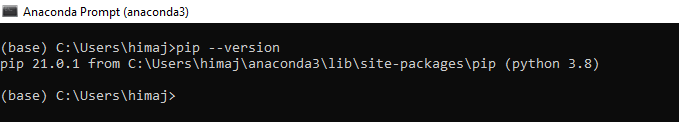


Figure 2: Pip version

## 4.0 Python Libraries:

Python has a number of libraries. Here we use some of the python libraries to develop the code: Keras, NumPy, Pandas, Matplotlib, Seaborn, TensorFlow, Scikit-learn (Sklearn).

## 4.1. TensorFlow:

TensorFlow is an open-source program created by Google in collaboration with the Brain team. TensorFlow is used in practically every machine learning application such as neural networks.

Features: Tensor flow is an open-source library for machine learning computations that enables faster and easier calculations. It is easy to run on a variety of platforms, including Android, CPUs, and GPUs. TensorFlow converts raw data into estimators, which are a type of data that neural networks can interpret. TensorFlow can be installed using the below command on Anaconda prompt:

*Pip install tensorflow*

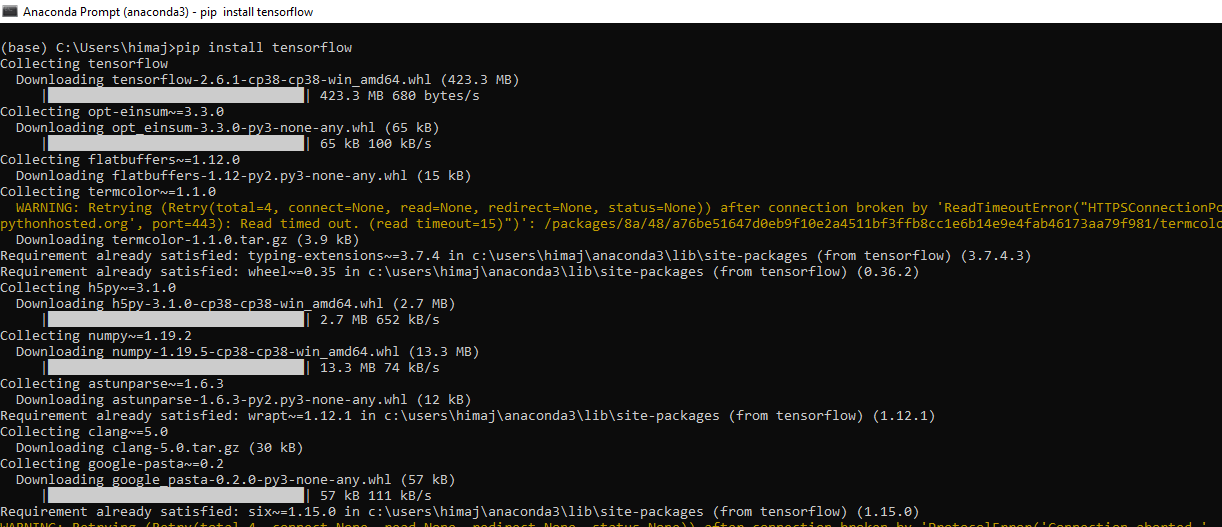


Figure 3: TensorFlow Installation

After installation check the TensorFlow whether it is installed properly or not using the command

*pip show tensorflow*

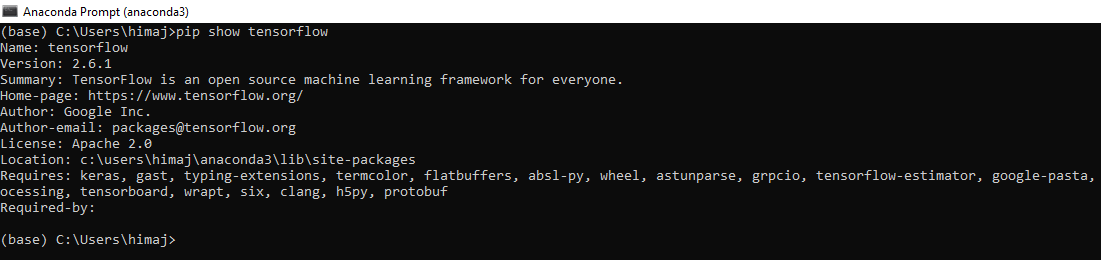


Figure 4. TensorFlow Information

TensorFlow version 2.6.1 is installed successfully. Location: c:\users\himaj\anaconda3\lib\site-packages.

## 4,2. Keras:

Keras is a Python interface to the Keras artificial neural network system, which is open-source. Keras is designed to be user-friendly, modular, and scalable, with the purpose of facilitating rapid deep neural network research. TensorFlow, Microsoft Cognitive Toolkit, Theano, and Plaid ML were among the backends supported by Keras.

***Features:***

Keras comes with a big dataset that has been pre-defined. It gives you access to a range of datasets. You may use this dataset to import and load it directly. Keras enables distributed deep-learning model training on clusters of graphics processing units (GPU) and tensor processing units (TPU), mostly using CUDA.

Keras can be installed using the below command on Anaconda prompt:

*Pip install keras (or) Pip install keras==2.6.*

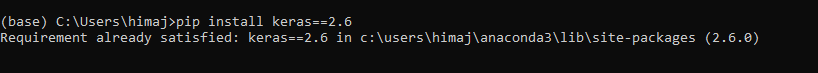


Figure 5: Installing Keras

Already Keras 2.6.0 version has been installed. So it displays the message like a requirement already satisfied with Keras version.

**4.3: Keras Preprocessing**

Keras preprocessing is a deep learning library used for manipulating pictures, text, and sequence data.

**4.4 NumPy**

NumPy (Numerical Python) is one of the library in the python programming language. It is an open-source library. NumPy supports a huge set of high-level mathematical functions to operate on multi-dimensional arrays.

Use the below command to check NumPy installed properly or not

*Pip show numpy*

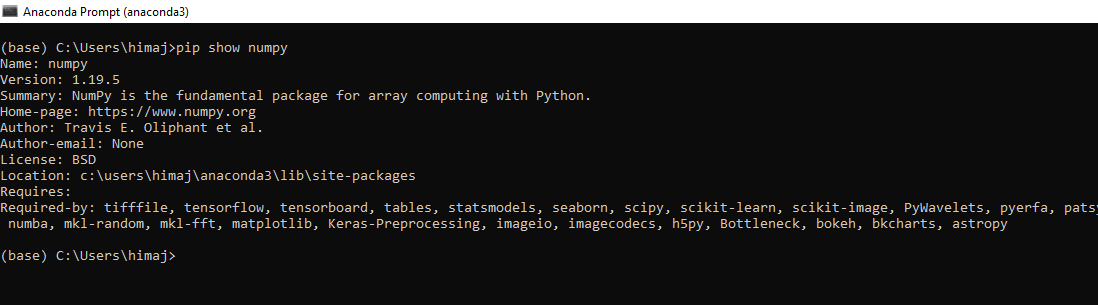


Figure 6. NumPy version

## 4.5 Pandas:

Pandas is an open-source Python library that is frequently used for data analysis and machine learning. Numpy, a library that supports multi-dimensional arrays, is used to build it.

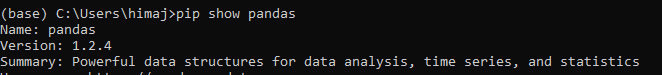


Figure 7: Checking pandas

Pandas contains a number of I/O functions for parsing data, including CSV, HDF5, SQL, JSON, HTML, Excel, and more.

**4.6 Matplotlib:** Matplotlib is a graphical interface and plotting package was created by John Hunter (1968-2012) that is used by thousands of scientists all around the world. Matplotlib numerical extension NumPy runs on all platforms.

# 5. Related Work:

ASL recognition is not a new challenge in computer vision. Researchers have utilized classifiers from a range of categories during the last two decades, which we can group roughly divided into linear classifiers, neural networks, and Bayesian networks. ASL recognition is supplied to a vocabulary of 30 words. They built appearance-based representations as well as a hand tracking system that could be categorized using a hidden Markov model (HMM).

Andrew Ng, Hinton, and LeCun have performed essential research on CNNs in order to increase the performance of CNN algorithms and structural optimization. According to LeCun Deep CNN is a breakthrough in image, video, audio, and voice processing. There has been no comprehensive study done using deep CNN for sign language recognition.

Other approaches for estimating hand positions and recognizing hand gestures have also been developed and applied. Lasonas et al. devised a model-based method for recovering hand posture by comparing a 3D hand model to the picture of the hand. Yeo et al. established a contour shape analysis approach that can distinguish 9 basic custom hand gestures with 86 %. Ren et al [Lorena] introduced a part-based hand gesture detection approach in which the fingers were parsed according to the hand's contour form. G. Balakrishnan, P. S. Rajam, et al. [4] suggested a technique for identifying a 32 set of combinations, 10 for each up and down position of the fingers, in order to generate english letters.

D. Deora and N. Bajaj [7] evolved an Indian Sign Recognition System that identifies 25 English alphabets and nine numerical signs.

J. Rekha and colleagues [8] suggested a method for recognizing ISL double-handed static and dynamic alphabet signs. As training samples, 40 signers provided 23 static ISL alphabet signs, while 22 videos were utilized as testing samples. Principle Curvature Based Region Detector was used to extract from features, Wavelet Packet Decomposition was used to extract texture features from the hand, and complexity defects techniques were used to extract features from the fingers.

M. K. Kar, M. K. Bhuvan, D. Neog et al. [9] suggested a unique technique for hand gesture detection based on assessing the hand's textures and important geometrical properties. The Bayesian classifier method was used to segment the hands. The Gaussian distribution approach was utilized for feature extraction, and the distance measure was employed for closeness measurement.

After studying relevant works by other authors, we discovered several limitations in these works. Previously there is no proper database to research Sign Language. The recognition system we are developing is intended for usage in public spaces where a variety of backgrounds (noises) may be present while obtaining sign pictures. Some writers employed a variety of approaches for categorization and feature extraction that were unclear and for which they were unable to provide justification. The writers of these research publications created several laboratory-based systems that are either expensive or require additional computing power. The systems are not designed to be used in mobile or handheld devices.

**Sign-language Picture Category:** Below figure shows ASL data picture category.

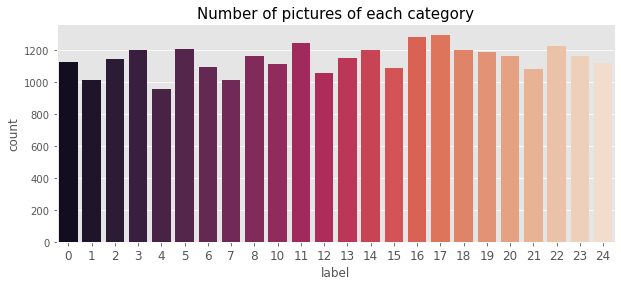


Figure 8 . Picture category

## Dataset:

American Sign Language data set is a primary source of this project taken from “Kaggle's website”.

The University of Surrey’s Center for Vision created the American Sign Language hand gestures dataset. The dataset pattern is matched closely with the MNIST (Modified National Institute Of Standards and Technology). Each training and test data represent a label (0-25) for each alphabetic letter(A-Z).

There are 27,455 photos and 785 columns in the training data, whereas there are 7172 images and 785 columns in the remaining data. The first column of the image contains the label of the image while the rest of the 784 columns represent the flattened 28 by 28-pixel images with grayscale values between 0-255. Data only have values for 24 alphabets, and as per the problem overview letters, J (9) and Z(25) cannot come in image data as it requires multiple gestures.



Figure 9: American Sign Language Alphabet [Ref(2)]

**Key Phrases of Sign Language Recognition:**

Sign Language recognition has three key phrases: hand detection and segmentation, feature, and classifier.

***1. Hand Detection and Segmentation:*** Hand segmentation is the process of collecting hand pixels from a picture and segmenting them from the backdrop before identification. A lot of restrictions are placed on the background, user, and imagery throughout the segmentation phase. A background controller simplifies the task.

***2. Feature:*** Sign Language recognition can be done using Manual signs and Non-Manual signs. Nonmanual signs include face characteristics, head, and body mobility, whereas manual signs include hand form, hand position, and hand motion.

***3. Sign Recognition:*** After hand detection and features have been computed, sign recognition can be detected. Sign recognition can be divided into two parts, recognition of isolated signs and continuous signs.

# 6. Methodology

We used different models to train the pre-trained model for translating the picture to the appropriate alphabet. The project will be divided into three independent functional blocks, as shown in the below figure, Data Processing, Training, and Classify Gesture. The diagram is simplified in detail to abstract some of the intricacy.



Figure 10: Software Structure [Anna Deza]

## 6.1 Data Processing:

The raw image data has been loaded using the load-data.py script function and saved as NumPy arrays into file storage. The process data loads the image data from data.npy and preprocess it by resizing the image and applying some filters to enhance the features [Anna Deza]. The processed image data is divided into training and testing data. An individual image is imported and processed from the disc to use the trained model in identifying gestures.

## 6.2 Training:

In the training phase, the individual image is pre-processed after the reference image is retrieved. Filters are used in this preprocessing to increase the important content of the frame information while reducing the unnecessary data as much as feasible. After that, using CNN, all of the characteristics of the processed picture are retrieved and saved in a database linked with that sign. The same method is performed for each sign to be added to the system. In addition, the entire reference database has been prepared.

## 6.3 Classify Gesture:

Once training has been finished, it can be used to classify the new American Sign gesture that is available in the file system. The user enters the gesture image file path as well as the test data. Then the file path will be sent to process the data.py which will load and preprocess the file in the same way that the model has been trained.

# 7. THE PROPOSED SYSTEM

The report is organized in the following order:

1. Data collection
2. Image pre-processing
3. Feature extraction
4. Classification Algorithms
5. Experiment Results
6. Conclusion
7. Future work

## **7.1 Data Collection**:

Data set: “American Sign Language” dataset has taken from Kaggle’s website.

CSV files:

1. sign\_mnist\_train.csv
2. sign\_mnist\_test.csv

The training data contains 27,455 images and 785 columns, while the remaining data contains 7172 images and 785 columns. Data have 24 alphabets, and as per the problem overview, letters J(9) and Z(25) requires multiple gestures so not shown in image data.

## 7.2 Image Pre-Processing:

The main aim of image pre-processing is to improve the image data, which reduced the unwanted variations or enhancing image features for further processing. Cropping, resizing, and grey scaling are examples of image preprocessing. It is an attempt to capture the significant pattern that expresses the unique data without noise or unwanted data.

Cropping will remove the unwanted parts of the image to improve the framing and change the image aspect ratio. Images will be resized to suit the space that has been allocated. Image resizing is a great tip for preserving the quality of an original image.

## 7.3 Feature Extraction:

The most significant module in a sign language recognition system is feature extraction. Because the nature of each sign language and the signs evaluated differs, the most trustworthy elements must be chosen [Madhuri Sharma]. Here feature extraction is used for image compression or dimensionality reduction. Because the input pictures are too large to analyze in a reasonable amount of time, we use feature extraction to lower the shape of the input image. Feature extraction is the process of transforming raw data into a feature. The feature extraction method is chosen such that picture information is preserved. Pattern recognition and machine learning issues need feature extraction. Feature extraction is the process of reducing the number of resources needed to correctly describe a large set of data.

## 7.4. Classification:

Classification techniques are useful to recognize the gestures. The purpose of image classification is to recognize and classify images. The most important aspect of digital image analysis is picture categorization. The goal of classification systems is to produce a suitable answer for all potential inputs and to conduct the most probable matching of the inputs while accounting for statistical variation.

Various techniques are proposed to recognize the American Sign Language (ASL) gestures. Powerful artificial intelligence tools convolutional Neural Networks (CNN), Artificial Neural Networks(ANN), Autoencoders, Backpropagation and Machine Learning Classification techniques, KNN, SVM, Decision Tress, Random Forest, are used to predict the sign letters.

# 8.Algorithms:

# Machine Learning Algorithms:

## 8.1 K-Nearest Neighbor:

K-Nearest Neighbor is a machine learning approach that is utilized in a variety of companies. This is a supervised learning algorithm used for classification and regression problems. Objects are classified using the K-nearest neighbor classifier, which uses feature space. It encapsulates the concept of similarity and calculating the distance between similar data points.

Here to predict the signs we are using the KNN algorithm. first, initialize “k” to choose a number of neighbors. Split the data set into X\_ train and Y\_ train to train the data. Scikit function “train\_test\_split” is used to split the data set into train and test. Train\_test\_split have four parameters. first 2 parameters are input x\_ train and output X\_ test. Setting test\_ size as 0.2, and random\_ state=0 to reproduce the result. Once the KNN model has been trained we use the predict function to our model to create predictions on the test data. Finally, the accuracy of the KNN algorithm is tested and displayed.

## 8.2 Support Vector Machine:

Support Vector Machine is a supervised machine learning technique utilized for classification and Regression. However mostly used for classification challenges. We do classification by locating the hyperplane that best distinguishes the two classes. It is simple to create a linear hyper-plane between these two classes in the SVM classifier’s algorithm has kernel trick technique. The kernel is a function that turns a low-dimensional input space into a higher-dimensional space. The kernel has a number of possibilities, including "linear," "rbf," "poly," and others. For non-linear hyper-planes, "rbf" and "poly" are beneficial. Here we test both linear and non-linear(poly) hyper-planes to find out the accuracy of the ASL recognition.

## 8.3 Decision Tree:

A decision tree is a well-known technique for classification that is widely utilised. It is a graphical representation of all potential solutions to a decision, with the decisions being made under certain parameters. Inner nodes represent dataset properties, leaves represent outcomes, and branches represent decision rules in this tree-structured classifier. It sorts or arranges instances from root to leaves to separate them. During the traversal of each segment of the node, the algorithm must make a decision based on a set of conditions. Here, we predict the ASL accuracy using the decision tree classifier. Criterion Gini has taken. Train the data giving input X train and output Y test. Predict function is used to predict the score.

## 8.4 Random Forest Algorithm:

A random forest is a machine learning approach for dealing with classification and regression issues. Random forest constructs decision trees from several samples and uses the majority classification and less for regression. The random forest algorithm's produced 'tree' is trained via bagging or bootstrap aggregation. It outperforms the decision tree in terms of accuracy. It is a useful tool for dealing with missing data. Without hyper-parameter adjustment, it can provide a fair forecast. Random forest overcomes the problems of decision tree overfitting.

# Deep Learning Algorithms:

## 8.5 Convolutional Neural Network (CNN) Algorithm:

Convolutional neural networks (CNNs) were first introduced in 1995 by Yann LeCun and Yoshua Bengio. This is most widely used deep learning architecture also known as convolutional networks or CNNs. CNN is a type of multi-layer neural network that uses an apparent grid-like structure to analyze input. Fewer uses include image and video recognition, image analysis, recommendation systems, natural language processing, computing interfaces, financial time-series, and a number of others.

A CNN's architecture is built to make use of the two-dimensional structure of an input picture (or other two-dimensional input, such as a voice signal). Local connections and linked weights are used, followed by some type of pooling, to provide translation-invariant characteristics. CNNs are also easier to train than fully linked networks with the same number of hidden units because they have many fewer parameters.

CNNs take their inspiration from the human brain's visual cortex. A receptive field, which is a tiny portion of the visual field, will be coupled to the artificial neurons in a CNN. This is accomplished by filtering the image with discrete convolutions and using the filter outputs as trainable weights. Many filters are applied to each channel, which, when coupled with the neurons' activation functions, form feature maps.

Proposed Architecture:

The proposed CNN model contain convolutional blocks with 2- 2D Convolutional Layers with ReLU activation, followed by Max Pooling and Dropout layers made up this model. These convolutional blocks are repeated three times before being followed by Fully Connected layers that categorize the data into the appropriate categories [Towards AI]. Throughout the model, the kernel sizes are kept constant at 3 X 3. Below figure depicts our original proposed model, which is similar to the one from the a forementioned study.

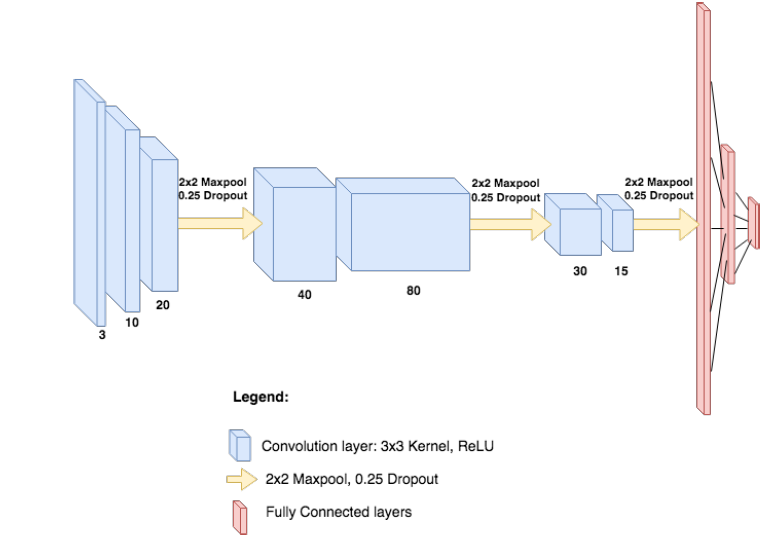


Figure11 : Model Architecture Using Deep Convolutional Networks [Anna Deza]

This model was created with the goal of being faster to train and establishing a benchmark for problem complexity. This model was created with only one convolutional layer block, which consisted of two convolutional layers with varying kernel sizes with ReLU activation, and the typical Max Pooling and Dropout.

Here we are using the input shape is 28 X 28 with one channel. The output will have 32 features and the kernel size is 3 by 3. Padding given as same means same dimensional output is required as input.

This was fed into three completely linked layers, each of which generated different letter classes. Our dataset included the backdrop, but the study pre-processed their data to eliminate the background, which led to the variance in kernel sizes. The bigger kernel will capture a mixture of the smaller information, such as finger crossing, angles, and hand placement, among others. This model architecture is shown below. Convolutional Neural Network Classification Accuracy is: 93.76%

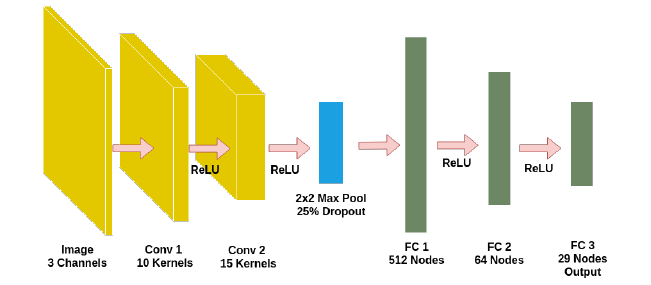


Figure 12: Model Architecture [Anna Deza]

## **8.6 Artificial Neural Network**:

Each layer of an Artificial Neural Network (ANN) has a collection of neurons. A Feed-forward Neural Network is one that processes ANN inputs in the forward direction. Artificial Neural Networks include three layers: input, hidden, and output. The data is received by the input layer, processed by the hidden layer, and generated by the output layer. Each layer is essentially attempting to learn certain weights.ANN was used to solve the issues on image data, tabular data, and text data. Any non-linear function can be learned by ANN very easily. As a result, Universal Function Approximators are a typical name for these networks. Weights that map the input to the intended output can be learned by ANNs.

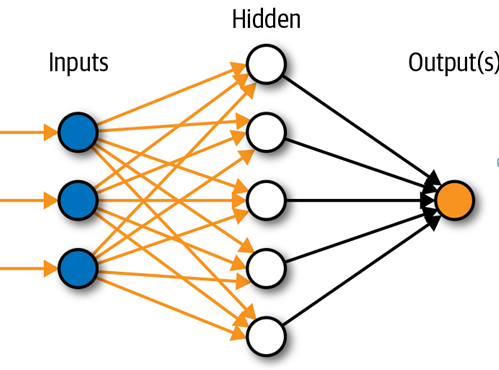


Figure13 : Artificial Neural Network Architecture [InnoArchiTech]

One of the main reasons for universal approximation is the activation function. The network's nonlinear, the activation function is one of the main key reasons for universal approximation. The network's nonlinear features are introduced using activation functions. Activation functions are used to introduce the network's nonlinear properties. This makes it easier for the network to learn any intricate input-output relationship. As we can see in the above figure the output of each perceptron or neuron is the activation of the sum of inputs. So, this clearly shows activation is the power of ANN. Image classification using ANN is one of the challenges in ANN first it converts 2-dimensional images into one-dimensional vector.it has two drawbacks. The number of parameters will be increased with the image size.

## 8.7. Back propagation:

Backpropagation, which stands for "backward propagation of errors," is a method for updating weights using gradient descent. One of the most crucial ideas in a neural network is backpropagation. It is most widely used algorithm for training feedforward neural network. For a single input–output example, backpropagation calculates the gradient of the loss function in terms of network weights, and it does it fast, as opposed to a naive direct computation of the gradient with respect to each weight separately. We'll utilise the Batch Gradient Descent optimization method to figure out which way the weights should be adjusted to achieve a smaller loss.

## 8.8**. Autoencoder:**

Autoencoder is a form of Artificial Neural Network used to develop effective code for unstructured input. Autoencoder compresses the data effectively and then reconstructs the data back for dimensionality reduction. Autoencoders are generally used for image generation, feature extraction, image denoising, image compressing.

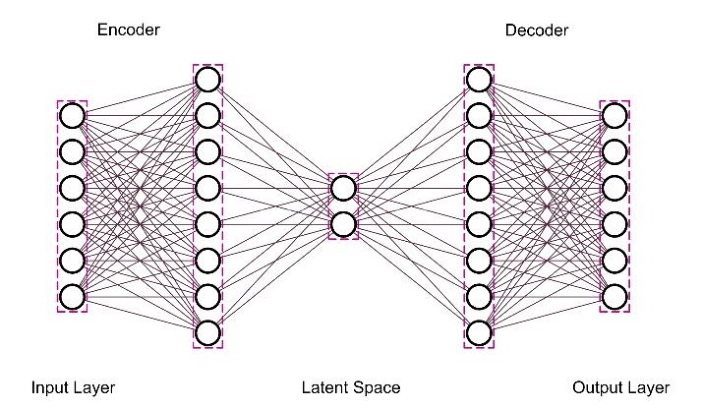


Figure 14: Autoencoder Architecture (research gate)

autoencoders are unsupervised learning in which the network can reveal the unlabelled data structure. Neural networks with autoencoders are a form of deep neural network. They are made up of four layers. An input layer, intermediate layer, hidden layer, output layer.

**Optimizer (Adam):** Adam is a deep neural network training system that uses an adaptive learning rate optimization approach. To identify individual learning rates for each parameter, the algorithms make use of adaptive learning rate approaches.

# 9.Experiment Results:

Machine learning algorithms and Deep learning algorithms are successfully applied to American sign language dataset to predict the letters. The results for each model is shown below. Each of the model or technique has been trained on letter ‘a’ to ‘y’ excluding j. For categorization, the proposed CNN model is used.

**Deep Learning Algorithm Results:**

In this report CNN, ANN models are used for classification. CNN is trained in three layers. CNN process is performed by some filters in a convolutional layer. First layer is 128 filters are taken with kernel size (5x5) and the input shape is 28 x 28. Second layer is 64 filter with 2x2 Kernel size. Third layer with 32 filters with Kernal size 2x2. Padding is taken to same. Activation Relu is selected. Activation represents the activation function. Max pooling layer is used to reduce the dimensionality of the image. First layer of max pool size is taken as 3 by 3 and for remaining 2 x 2 pool size is selected. After training and testing completed successfully, Model accuracy has been calculated and got 97 %. Autoencoders are used for dimensionality reduction.

|  |  |
| --- | --- |
| Model | Accuracy |
| CNN | 97% |
| ANN | 93% |

Auto encoders are used and tested successfully, and we get test loss is 0.17 %. Below figure shows the test accuracy and loss.

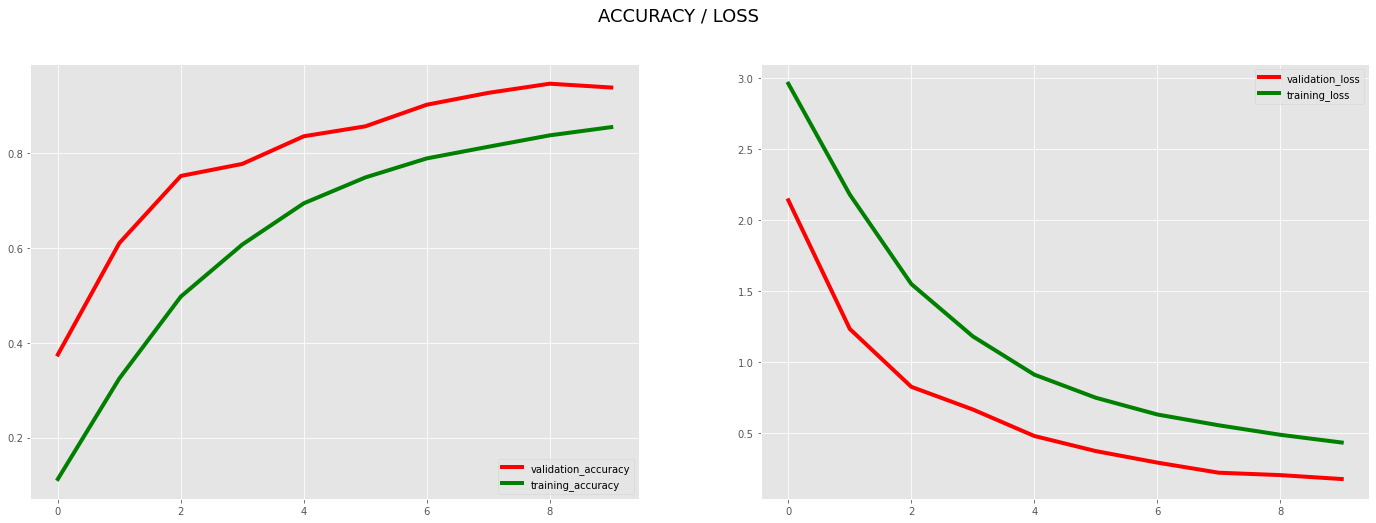


Figure 15: Test Accuracy and Loss

Machine Learning Algorithm Results:

|  |  |
| --- | --- |
| Model | Accuracy |
| KNN Classifier | 96.8% |
| SVM-Linear | 81.07% |
| SVM- Rbf | 84.71 |
| SVM-Poly | 974.3% |
| Random Forest | 98 .6% |
| Decision Tree Classifier | 89.6 % |

**Confusion Matrix:**

Confusion matrix obtained using the American sign language dataset. Below figures shows confusion matrix for SVM and RFC models.

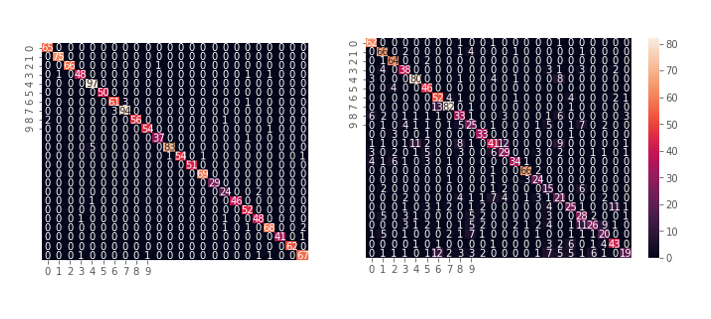


Figure 16: SVM and RFC confusion matrix

# 10. Conclusion:

Both Machine learning and Neural networks are most powerful methods in pattern recognition and identification systems. The technologies perform admirably in recognising the gesture of the alphabet.

We constructed and trained an American Sign Language translator for letters a-y, to make a strong model. In this report we used deep learning algorithms CNN, ANN for pattern classification. Each convolutional layer is evaluated with varied filtering window widths, which enhances identification speed and accuracy. And Max pooling technique is used for better performance. The training is done in different models to determine the best accuracy. This study demonstrates that, when compare to other deep learning techniques ANN, Autoencoders, Back-propagation, “convolutional neural networks” can effectively distinguish distinct signals in a American sign language with high accuracy.

Various Machine learning techniques also applied to ASL dataset to find the best accuracy. After applying the KNN, SVM (Linear, Non-linear, Poly), Random forest, Decision tree techniques Random forest is the best model. From the tests a maximum accuracy is attained by CNN from deep learning algorithms and Random forest technique got high accuracy from machine learning algorithms.

# 11. Future Work:

In future if the images are heavily pre-processed, the categorization effort might be much simplified. This might involve things like contrast correction, background reduction, and cropping. Using another CNN to localise and crop the hand would be a more robust technique. This advancement in image pre-processing will have a variety of applications in the future, including discovering and extracting information about human hands for use in sign language recognition and transcription to voice or text, robotics, gaming technology, virtual controllers, and remote control in the industry, among others.

**References**

Vogler, C., & Metaxas, D. (2001). A framework for recognizing the simultaneous aspects of American sign language. Computer Vision and Image Understanding, 81(3), 358-384.

Cao Dong.,(2015).American sign language recognition using Microsoft Kinect. Student theses and Dissertations

Shailesh b., Shubham dixit, Rohin Chadha., and Prof. Avinashi Bagul.,(2020). Sign Language Recognition Using Neural Networks.

Mitchell, Ross; Young, Trevas; Bachleda, Bellamie; Karchmer, Michael (2006). Sign Language Studies (Gallaudet University Press.) 6 (3). ISSN 0302-1475. Retrieved November 27, 2012.

Anna Deza, Danial Hasan., (2018). MIE324 Final Report: Sign Language Recognition

Deora, D. and N. Bajaj(2012)., Indian sign language recognition. International Conference on Emerging Technology Trends in Electronics Communication and Networking (ET2ECN).

Garcia, B., & Viesca, S. A. (2016). Real-time American sign language recognition with convolutional neural networks. Convolutional Neural Networks for Visual Recognition, 2, 225-232.

Farnaz D. Notash and Elahe Elhamki, InnoArchiTech. \Comparing loneliness, depression and stress in students

impaired and normal students studying in secondary schools of Tabriz". In: International Journal of Humanities and Cultural Studies February 2016 Special Issue (2016). issn: 2356-5926.

Lorena P. Varga,Leiner Barba, C O Torres and L Mattos[2011],Sign Language Recognition System using Neural Network for Digital Hardware Implementation.

Madhuri Sharma, Ranjna Pal and Ashok Kumar Sahoo, Department of Computer Science and Engineering, School of Engineering and Technology, Sharda University

Sharma, R., Nemani, Y., Kumar, S., Kane, L., & Khanna, P. (2013, July). Recognition of single handed sign language gestures using contour tracing descriptor. In Proceedings of the World Congress on Engineering (Vol. 2, pp. 3-5).

RuchiKawatra, S. A. (2009). Application of Fast Fourier Transformation on image processing software.International journal of Artificial TEM Journal – Volume 3 / Number 4 / 2014. 301www.temjournal.com Intelligence and Knowledge Discovery ,Vol 1, No 1(2011).

Appendix A

1. #import libraries
2. import pandas as pd
3. import numpy as np
4. from pandas.plotting import scatter\_matrix
5. import matplotlib.pyplot as plt
6. from sklearn import model\_selection
7. from sklearn.metrics import classification\_report
8. from sklearn.metrics import confusion\_matrix
9. from sklearn.tree import DecisionTreeClassifier
10. from sklearn.neighbors import KNeighborsClassifier
11. from sklearn.svm import SVC
12. from sklearn.preprocessing import LabelBinarizer
13. import seaborn as sns
14. from sklearn.model\_selection import train\_test\_split
15. import warnings
16. warnings.filterwarnings('ignore')
17. import keras
18. from keras.models import Sequential
19. from keras.layers import Dense,Flatten,Conv2D,MaxPool2D,Dropout
20. from tensorflow.keras.preprocessing.image import ImageDataGenerator

#import the dataset

1. train\_sign\_mnist=pd.read\_csv(r"C:\Users\himaj\Downloads\ANN\sign\_mnist\_train.csv")
2. train\_sign\_mnist.head()
3. test\_sign\_mnist=pd.read\_csv(r"C:\Users\himaj\Downloads\ANN\sign\_mnist\_test.csv")
4. test\_sign\_mnist.head()
5. train\_sign\_mnist.describe()
6. test\_sign\_mnist.describe()

**# Display the information of the datase**t

1. train\_sign\_mnist.info()
2. test\_sign\_mnist.info()

**# Check the duplicates**

1. train\_sign\_mnist.isnull().sum()
2. test\_sign\_mnist.isnull().sum()

**# Display the number of categories in each category**

1. plt.style.use("ggplot")
2. plt.figure(figsize =(10,4))
3. sns.countplot(x=train\_sign\_mnist['label'],data =train\_sign\_mnist,palette = "rocket")
4. plt.title("Number of pictures of each category", fontsize = 15)
5. plt.xticks(fontsize = 12)
6. plt.show()

#Reshaping the data

#Now we will add another dimension to the data

1. Y\_train\_image=train\_sign\_mnist['label'].values
2. Y\_test\_image=test\_sign\_mnist['label'].values

#Drop the label in train data and test data

1. X\_train\_image =train\_sign\_mnist.drop(["label"],axis=1).values
2. X\_test\_image=test\_sign\_mnist.drop(["label"],axis=1).values

#Display the shape of the train and test data

1. print("X train values",X\_train\_image.shape)
2. print("X test values",X\_test\_image.shape)

#Reshaping the train and test data

1. mnist\_training\_images = X\_train\_image.reshape(-1,28,28,1)
2. mnist\_testing\_images = X\_test\_image.reshape(-1,28,28,1)

#Printing the train and test values after reshaping the data

1. print("training images after reshaping", mnist\_training\_images.shape)
2. print("testing images after reshaping",mnist\_testing\_images.shape)
3. print("Y train labels",Y\_train\_image.shape)
4. print("Y test labels",Y\_test\_image.shape)
5. #Normalizing the data
6. mnist\_training\_images = mnist\_training\_images/255.0
7. mnist\_testing\_images =mnist\_testing\_images/255.0
8. print(mnist\_training\_images.shape)
9. print(mnist\_testing\_images.shape)
10. #display the normal images
11. figure, axis = plt.subplots(2,4)
12. figure.set\_size\_inches(8,8)
13. n = 0
14. for i in range(2):
15. for j in range(4):
16. axis[i,j].set\_xlabel(chr(Y\_test\_image[n] + 65))
17. axis[i,j].imshow(mnist\_testing\_images[n].reshape(28,28)
18. ,cmap='gray')
19. n+= 1
20. plt.tight\_layout()
21. #Label Encoding
22. from sklearn.preprocessing import LabelBinarizer
23. labelbinar=LabelBinarizer()
24. Y\_train\_image=labelbinar.fit\_transform(Y\_train\_image)
25. Y\_test\_image=labelbinar.fit\_transform(Y\_test\_image)
26. print(Y\_train\_image)
27. print('First image label after encoding: ',Y\_train\_image[0])
28. #Using unique to find out different numeric intrepretation of symbols
29. unique\_label= train\_sign\_mnist.label.values
30. unique\_value = np.array(unique\_label)
31. np.unique(unique\_value)
32. #Data Visualization
33. figure,axe=plt.subplots(2,2)
34. figure.suptitle('Dataset Preview')
35. figure.set\_size\_inches(5,5)
36. axe[0,0].imshow(mnist\_training\_images[0],cmap='gray')
37. axe[0,0].set\_title('label: 3 letter: D')
38. axe[0,1].imshow(mnist\_training\_images[1],cmap='gray')
39. axe[0,1].set\_title('label: 6 letter: G')
40. axe[1,0].imshow(mnist\_training\_images[2],cmap='gray')
41. axe[1,0].set\_title('label: 2 letter: C')
42. axe[1,1].imshow(mnist\_training\_images[4],cmap='gray')
43. axe[1,1].set\_title('label: 11 letter: M')
44. plt.tight\_layout()
45. #Data Augmentation
46. #Now creating the Image generator object for train images
47. train\_image\_datagen= ImageDataGenerator(rotation\_range = 0.2,
48. height\_shift\_range=0.2,
49. width\_shift\_range=0.2,
50. shear\_range=0.2,
51. zoom\_range=0.2,
52. horizontal\_flip=True,
53. fill\_mode='nearest')
54. #for validation data
55. validation\_datagenerator = ImageDataGenerator()
56. print(mnist\_training\_images.shape)
57. print(mnist\_testing\_images.shape)

#Convolutional Neural Network

1. model=Sequential()
2. model.add(Conv2D(filters=128,kernel\_size=(5,5), strides=1,padding='same',activation='relu',input\_shape=(28,28,1)))
3. model.add(MaxPool2D(pool\_size=(3,3),strides=1,padding='same'))
4. model.add(Conv2D(filters=64,kernel\_size=(2,2),

strides=1,activation='relu',padding='same'))

1. model.add(MaxPool2D((2,2),strides=2,padding='same'))
2. model.add(Conv2D(filters=32,kernel\_size=(2,2),

strides=1,activation='relu',padding='same'))

1. model.add(MaxPool2D((2,2),strides=2,padding='same'))

1. model.add(Flatten())
2. model.add(Dense(units=512,activation='relu'))
3. model.add(Dropout(0.2))
4. model.add(Dense(units=24,activation='softmax'))
5. model.compile(optimizer='adam',loss='categorical\_crossentropy',metrics=['accuracy'])
6. model.summary()

# Train the Model

1. result = model.fit(train\_image\_datagen.flow(mnist\_training\_images,Y\_train\_image, batch\_size=200),

epochs=10, validation\_data=(mnist\_testing\_images,Y\_test\_image),

shuffle=1)

#printing the accuracy

1. (ls,acc)=model.evaluate(mnist\_testing\_images, Y\_test\_image)
2. print('CNN MODEL ACCURACY = {}%'.format(acc\*100))

#finding the training accuracy and loss

1. plt.figure(figsize=(24, 8))
2. plt.subplot(1, 2, 1)
3. plt.plot(result.history['val\_accuracy'],label="validation\_accuracy",c="red",linewidth=4)
4. plt.plot(result.history['accuracy'],label="training\_accuracy",c="green",linewidth=4)
5. plt.legend(loc='lower right')
6. plt.subplot(1,2,2)
7. plt.plot(history.history['val\_loss'],label="validation\_loss",c="red",linewidth=4)
8. plt.plot(history.history['loss'],label="training\_loss",c="green",linewidth=4)
9. plt.legend(loc='upper right')
10. plt.suptitle("ACCURACY / LOSS",fontsize=18)
11. plt.show()

#Machine learning algorithms

#KNN Algorithm

1. from sklearn.model\_selection import train\_test\_split
2. x\_train, x\_test, y\_train, y\_test = train\_test\_split(X\_test\_image,Y\_test\_image, test\_size = 0.2, random\_state=0)
3. from sklearn.neighbors import KNeighborsClassifier
4. from sklearn.model\_selection import cross\_val\_score
5. from numpy import mean
6. KNN\_Model\_Classifier= KNeighborsClassifier(n\_neighbors=10)
7. KNN\_Model\_Classifier.fit(x\_train, y\_train)

#predicting the accuracy

1. y\_predKNN = KNN\_Model\_Classifier.predict(x\_test)
2. scores = cross\_val\_score(KNN\_Model\_Classifier, X\_test\_image, Y\_test\_image, scoring='accuracy')

#Finding the accuracy

1. accuracy\_KNN = (mean(scores)\*100)
2. print('Accuracy (KNN) : %.3f' % accuracy\_KNN, '%')

#Confusion Matrix

1. from sklearn.metrics import confusion\_matrix
2. import seaborn as ss
3. knn\_confusion\_Matrix = confusion\_matrix(y\_test, y\_predKNN)
4. xAxisLabels = ['0', '1', '2', '3', '4', '5', '6', '7', '8', '9' ‘10’]
5. yAxisLabels = ['0', '1', '2', '3', '4', '5', '6', '7', '8', '9' ‘10’]

#Printing the confusion matrix

1. ss.heatmap(knn\_confusion\_Matrix, annot = True, xticklabels=xAxisLabels, yticklabels=yAxisLabels)
2. #Preparing The Data
3. Y\_train\_image = train\_sign\_mnist['label'].values
4. X\_train\_image = train\_sign\_mnist.drop(columns = 'label')
5. #Normalization
6. X\_train\_image= X\_train\_image/255.0
7. #scaling the features
8. from sklearn.preprocessing import scale
9. X\_train\_image= scale(X\_train\_image)
10. Y\_test\_image=test\_sign\_mnist['label']
11. X\_test\_image=test\_sign\_mnist.drop(columns = 'label')
12. X\_test\_image= X\_test\_image/255.0
13. X\_test\_image= scale(X\_test\_image)
14. #SVM Model-Linear
15. svm\_model\_linear = SVC(kernel='linear')
16. svm\_model\_linear.fit(X\_train\_image, Y\_train\_image)

# Predicting the values

1. y\_prediction =svm\_model\_linear.predict(X\_test\_image)
2. from sklearn import metrics

# Accuracy

1. print("Accuracy:", metrics.accuracy\_score(Y\_test\_image,y\_prediction), "\n")
2. svm\_linear\_confusionMatrix = confusion\_matrix(Y\_test\_image, y\_prediction)
3. xAxisLabels = ['0', '1', '2', '3', '4', '5', '6', '7', '8', '9']
4. yAxisLabels = ['0', '1', '2', '3', '4', '5', '6', '7', '8', '9']
5. ss.heatmap(svm\_linear\_confusionMatrix, annot = True, xticklabels=xAxisLabels, yticklabels=yAxisLabels)
6. SVM non-linear model
7. svm\_non\_linear\_model = SVC(kernel='rbf')
8. # fit
9. svm\_non\_linear\_model.fit(X\_train\_image,Y\_train\_image)
10. # predicting the values
11. y\_predict = svm\_non\_linear\_model.predict(X\_test\_image)
12. #Printing the Accuracy
13. print("accuracy:", metrics.accuracy\_score(Y\_test\_image, y\_predict), "\n")
14. #SVM Ploy model
15. svm\_model\_Poly = SVC(kernel = 'poly', degree=3)
16. svm\_model\_Poly.fit(X\_train\_image, Y\_train\_image)
17. y\_pred\_svm\_model\_Poly =svm\_model\_Poly.predict(x\_test)
18. scores = cross\_val\_score(svm\_model\_Poly, X\_test\_image, Y\_test\_image, scoring='accuracy')
19. svm\_Poly\_Accuracy\_result = (mean(scores)\*100)
20. print('Accuracy (SVM - Polynomial) : %.2f' % svm\_Poly\_Accuracy\_result, '%')

#Random Forest Classifier

1. from sklearn.ensemble import RandomForestClassifier
2. rfc\_model\_classifier = RandomForestClassifier(n\_estimators = 10, criterion = 'gini')
3. rfc\_model\_classifier.fit(X\_train\_image,Y\_train\_image)
4. y\_predictionRFC=rfc\_model\_classifier.predict(x\_test)
5. scores = cross\_val\_score(rfc\_model\_classifier, X\_test\_image,Y\_test\_image, scoring='accuracy')
6. rfc\_model\_accuracy= (mean(scores)\*100)
7. print('Accuracy (RFC) : %.2f' % rfc\_model\_accuracy, '%')

#Confusion Marix

1. from sklearn.metrics import confusion\_matrix
2. import seaborn as ss
3. confusionMatrix\_RFC = confusion\_matrix(y\_test,y\_predictionRFC)
4. xAxisLabels = ['0', '1', '2', '3', '4', '5', '6', '7', '8', '9']
5. yAxisLabels = ['0', '1', '2', '3', '4', '5', '6', '7', '8', '9']
6. ss.heatmap(confusionMatrix\_RFC, annot = True, xticklabels=xAxisLabels, yticklabels=yAxisLabels)

#Decision Tree Classifier

1. from sklearn.tree import DecisionTreeClassifier
2. modelDecisionTree\_classifier = DecisionTreeClassifier(criterion = 'gini')
3. modelDecisionTree\_classifier.fit(X\_train\_image,Y\_train\_image)
4. y\_predDecisionTree = modelDecisionTree\_classifier.predict(x\_test)
5. scores = cross\_val\_score(modelDecisionTree\_classifier, X\_test\_image, Y\_test\_image, scoring='accuracy')

#Printing accuracy

1. accuracy\_Decision\_tree = (mean(scores)\*100)
2. print('Accuracy (DTC) : %.2f' % accuracy\_Decision\_tree, '%')
3. from sklearn.model\_selection import train\_test\_split
4. x\_train, x\_test, y\_train, y\_test = train\_test\_split(X\_test\_image,Y\_test\_image, test\_size = 0.2, random\_state=0)

#Artificial Neural Network

1. import keras
2. from keras.models import Sequential
3. from keras.layers import Dense, Dropout
4. classifier = Sequential()
5. classifier.add(Dense(64, input\_shape = x\_train[0].shape))
6. classifier.add(Dense(128, activation='relu'))
7. classifier.add(Dropout(rate=0.3))
8. classifier.add(Dense(128, activation='relu'))
9. classifier.add(Dropout(rate=0.3))
10. classifier.add(Dense(128, activation='relu'))
11. classifier.add(Dropout(rate=0.3))
12. classifier.add(Dense(64, activation='relu'))
13. classifier.add(Dense(10, activation='softmax'))
14. classifier.compile(optimizer='adam',loss='sparse\_categorical\_crossentropy', metrics=['accuracy'] )
15. classifier.fit(mnist\_training\_images,Y\_train\_image, epochs=10, batch\_size=10)
16. print("Evaluation on test data : ")
17. results = classifier.evaluate(X\_test\_image,Y\_test\_image, batch\_size=100)

#printing the accuracy

1. accuracy\_ANN = results[1]\*100
2. print("Test Accuracy : %.2f" % accuracy\_ANN,'%')

#Autoencoder

1. import tensorflow as tf
2. input\_img = tf.keras.layers.Input(shape=(784,), name = "input")

# input encoded representation

1. encoded = Dense(1024, activation='relu', name="emb\_0")(input\_img)
2. encoded = Dense(512, activation='relu', name="emb\_1")(encoded)
3. encoded = Dense(256, activation='relu', name="emb\_2")(encoded)
4. encoded = Dense(128, activation='relu', name="emb\_3")(encoded)
5. encoded = Dense(64, activation='relu', name="emb\_4")(encoded)
6. encoded = Dense(16, activation='relu', name="emb\_5")(encoded)
7. latent\_vector = Dense(2, activation='relu', name="latent\_vector")(encoded)

#input loss reconstruction

1. decoded = Dense(16, activation='relu', name="dec\_1")(latent\_vector)
2. decoded = Dense(64, activation='relu', name="dec\_3")(decoded)
3. decoded = Dense(128, activation='relu', name="dec\_4")(decoded)
4. decoded = Dense(256, activation='relu', name="dec\_5")(decoded)
5. decoded = Dense(512, activation='relu', name="dec\_6")(decoded)
6. decoded = Dense(1024, activation='relu', name="dec\_7")(decoded)
7. output\_layer = Dense(784, activation = 'sigmoid', name="output")(decoded)
8. autoencoder = tf.keras.models.Model(input\_img, output\_layer)
9. autoencoder.summary()
10. encoder = tf.keras.models.Model(input\_img, latent\_vector)
11. encoder.summary()
12. autoencoder.compile(optimizer='adam', loss='mse')
13. auto\_history = autoencoder.fit(X\_train\_image, X\_train\_image, epochs=10, batch\_size=200,validation\_data=(X\_test\_image, X\_test\_image))
14. decoded\_imgs = autoencoder.predict(X\_test\_image)
15. test\_loss=autoencoder.evaluate(X\_test\_image)

#Printing the loss accuracy

1. print("Test loss : %.2f" % test\_loss,'%')

#Displaying the original images

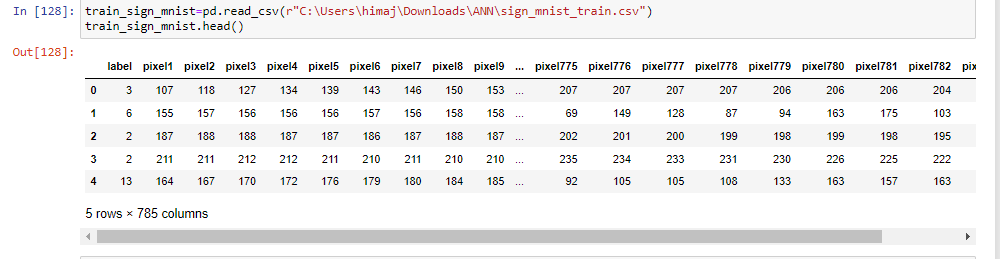
#below code taken from kaggle for image confirmation

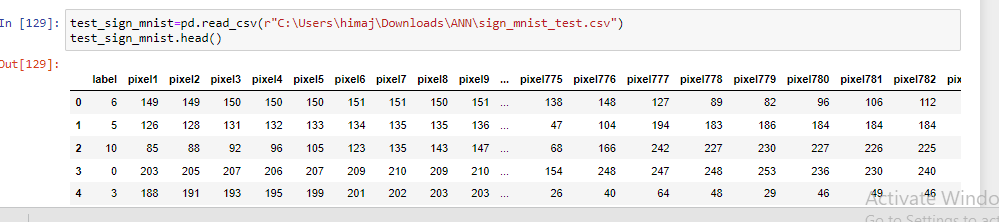
1. k= 10
2. plt.figure(figsize=(21, 5))
3. for i in range(k):
4. # display original
5. axis = plt.subplot(2, k, i + 1)
6. plt.imshow(X\_test\_image[i].reshape(28, 28))
7. plt.gray()
8. axis.get\_xaxis().set\_visible(False)
9. axis.get\_yaxis().set\_visible(False)
11. # display reconstruction
12. axis = plt.subplot(2, n, i + 1 + n)
13. plt.imshow(decoded\_imgs[i].reshape(28, 28))
14. plt.gray()
15. axis.get\_xaxis().set\_visible(False)
16. axis.get\_yaxis().set\_visible(False)
17. y\_pred = np.argmax(model.predict(X\_test\_image),axis = 1)

**Appendix B**

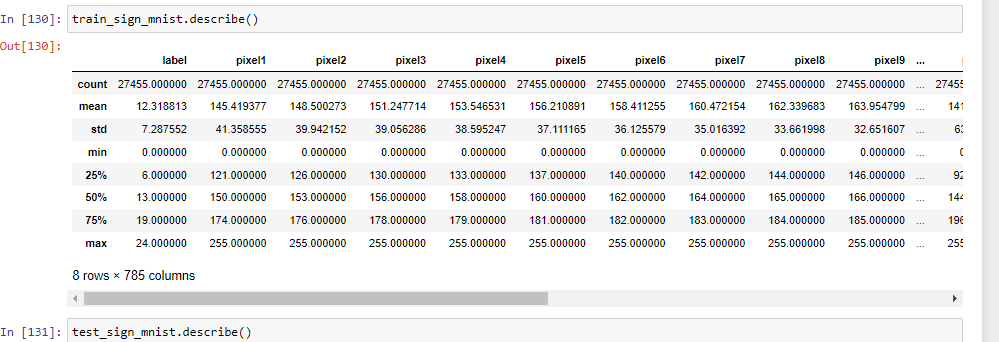
**Result Screenshots**

* 1. Train data image

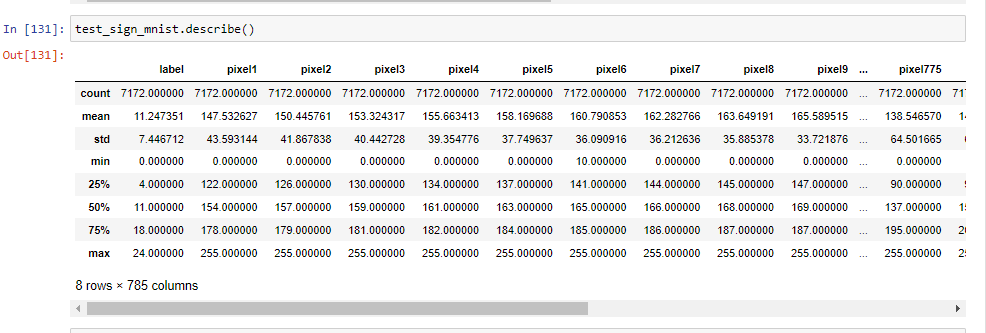
2. Test data image



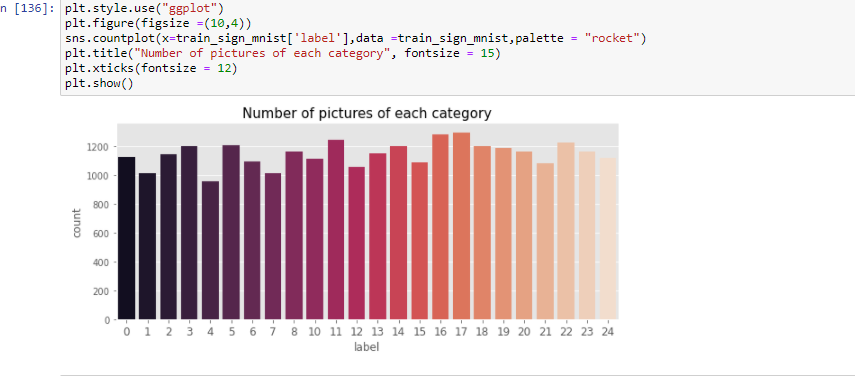
3.Descibing the train data



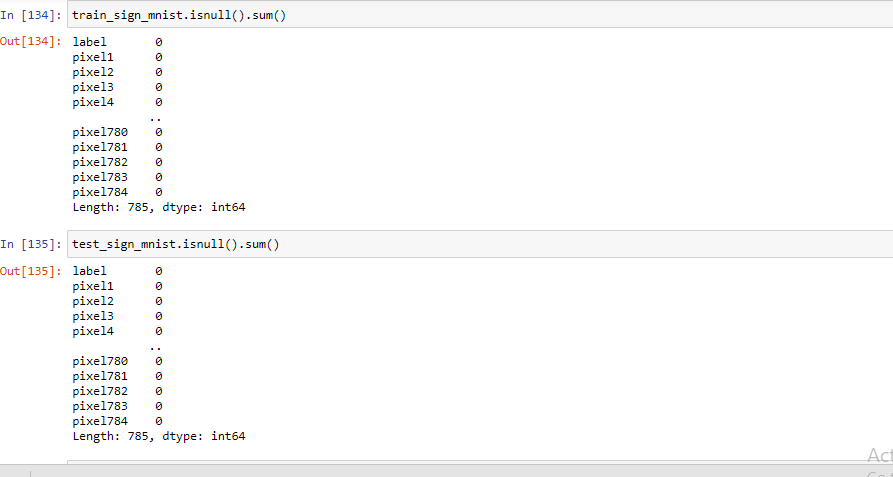
4.Describing the test data



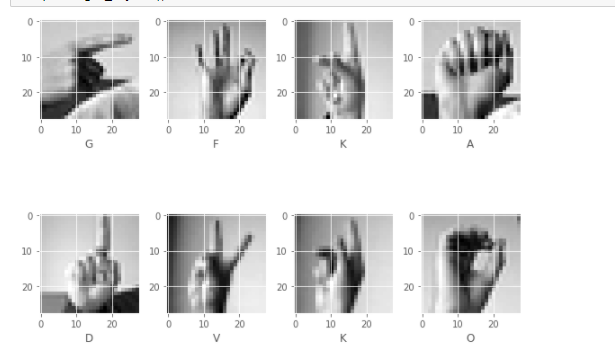
5. Each category Image



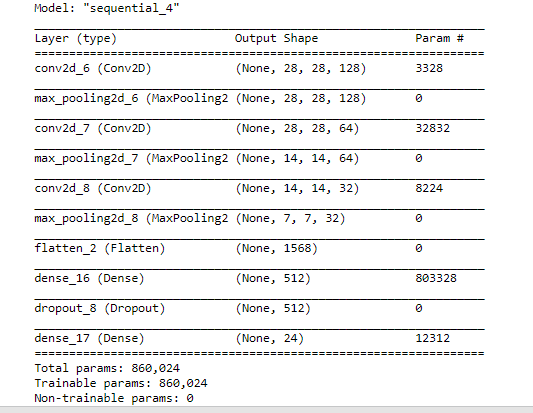
6.check the duplicate values

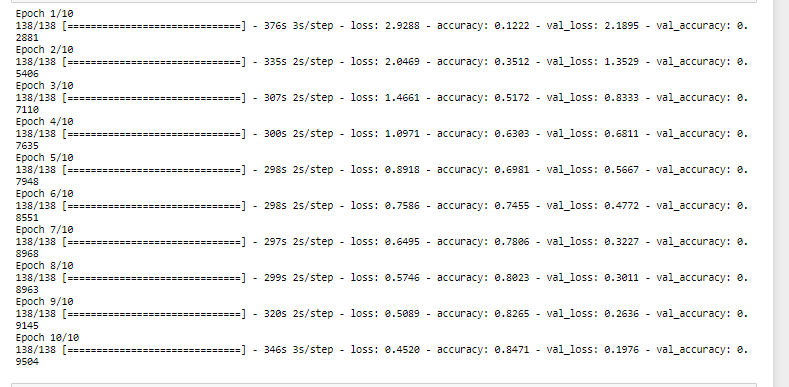


1. ASL Sample images

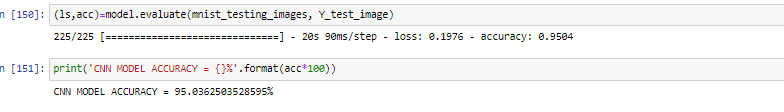


1. CNN



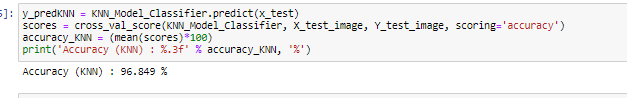


1. CNN Accuracy

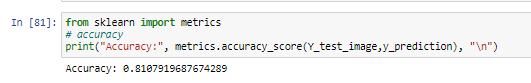


Machine learning algorithms

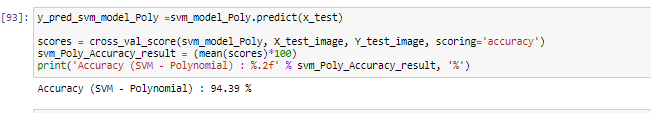
KNN



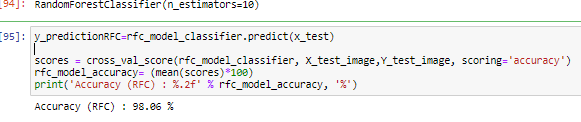
SVM-Linear



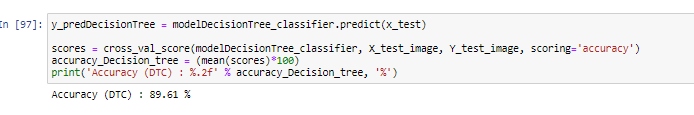
SVM-ploy



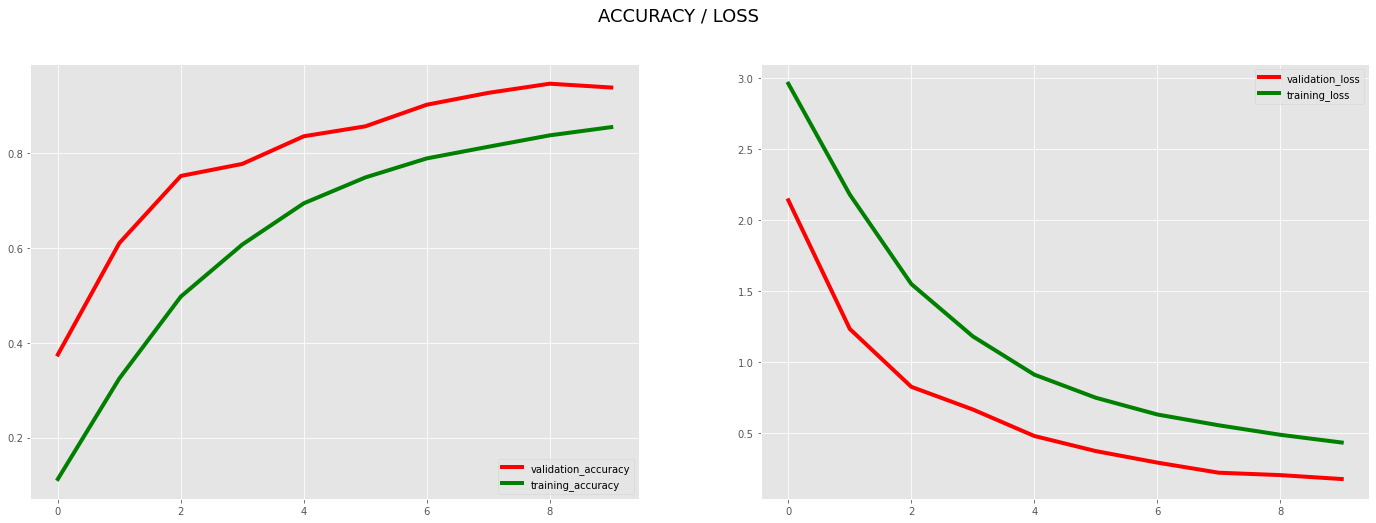
RFC



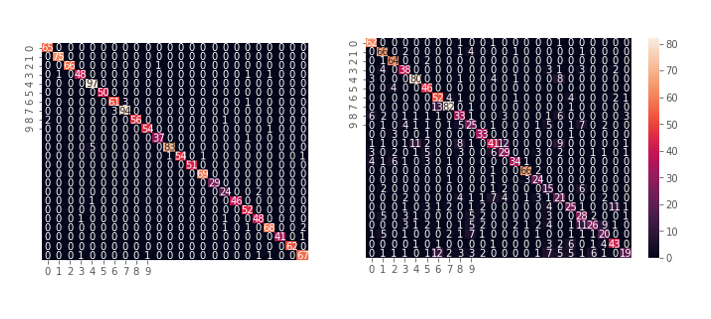
Decision Tree



Test Accuracy and Loss:



KNN and SVM Confusion Matrix.



**Project Proposal**

**Student Name: Jyothi Yendamuri**

**Student ID: 11467683**

**Project Proposal**

**Title:** Sign Language Recognition Using Machine Learning and Deep Learning Algorithms

**Project Description:**

Communication is a vital sign used in our daily lives. People with hearing disabilities confront a lot of hurdles while communicating with normal people. Many developed countries made systems for overcoming a problem in communication with deaf people. Sign Language Recognition (SLR) is an advanced platform in contemporary computer research. The development of a real-time sign language translator is a major breakthrough that bridges the gap between hearing-impaired and normal people. This paper proposes the recognition of American Sign Language (ASL) gestures using powerful artificial intelligence tools convolutional Neural Networks (CNN), Autoencoders to predict the sign letters. Sign Language has its own syntax and vocabulary which is entirely different from other spoken and written languages. The development of sign language applications will help those who do not understand sign language and also, they can easily communicate with unhearing and dumb people.

**Dataset:**

American sign Language data set is a primary source of this project taken from “Kaggle’s website”.

Dataset Name: American Sign Language

Link: <https://www.kaggle.com/datamunge/sign-language-mnist>

The University of Surrey’s Center for Vision created the American Sign Language hand gestures dataset. The dataset pattern is matched closely with the MNIST (Modified National Institute Of Standards and Technology). Each training and test data represent a label (0-25) for each alphabetic letter(A-Z). The training data contains 27,455 images and 785 columns, while the remaining data contains 7172 images and 785 columns.

The first column of the image contains the label of the image while the rest of the 784 columns represent the flattened 28 by 28-pixel images with grayscale values between 0-255. Data only have values for 24 alphabets, and as per the problem overview letters, J(9) and Z(25) cannot come in image data as it requires multiple gestures.

**Objective/Aim of the study**

The main aim of this project is to build Deep learning techniques and Machine learning algorithms to classify which letter of the American Sign Language alphabet is being signed, given an image of a hand gesture. The objective is to understand which model is the best prediction model for predicting the signs.

**Methodology:**

ASL letter prediction will be done using the Machine Learning Classification techniques, KNN, SVM and Deep learning Algorithms Convolutional Neural Networks (CNN -using pooling layer, fully connected layer), Backpropagation, Autoencoders.

**Program Language**: Python code will be implemented to predict the sign language letter.