A project report on

A TRANSLATOR BETWEEN TEXT AND SIGN LANGUAGE

Submitted in partial fulfillment for the award of the degree of

B.Tech in Computer Science and Engineering

by

AYANABHA JANA (18BCE1044)

SCHOOL OF COMPUTER SCIENCE AND ENGINEERING (SCOPE)

May, 2022

VIT
Vellore Institute of Technology
(Deemed to be University under section 3 of UGC Act, 1956)

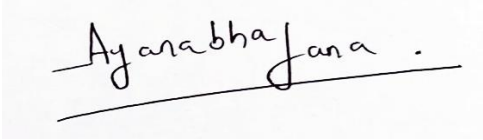
DECLARATION

I hereby declare that the thesis entitled "A TRANSLATOR BETWEEN TEXT AND SIGN LANGUAGE " submitted by me, for the award of the degree of Bachelor of Technology in Computer Science and Engineering, Vellore Institute of Technology, Chennai is a record of bonafide work carried out by me under the supervision of Dr. Shridevi S.

I further declare that the work reported in this thesis has not been submitted and will not be submitted, either in part or in full, for the award of any other degree or diploma in this institute or any other institute or university.

Place: Chennai

Date: May 2022



Signature of the Candidate

VIT
Vellore Institute of Technology
(Deemed to be University under section 3 of UGC Act, 1956)

School of Computer Science and Engineering

CERTIFICATE

This is to certify that the report entitled **"A Translator between text and sign language"** is prepared and submitted by **AYANABHA JANA**(**18BCE1044**) to Vellore Institute of Technology, Chennai, in partial fulfullment of the requirement for the award of the degree of **B.Tech. CSE** programme is a bonafide record carried out under my guidance. The project fulfills the requirements as per the regulations of this University and in my opinion meets the necessary standards for submission. The contents of this report have not been submitted and will not be submitted either in part or in full, for the award of any other degree or diploma and the same is certified.

Signature of the Guide: 

Name: Dr. Shridevi S.

Date: May 2022

Signature of the Internal Examiner	Signature of the External Examiner
Name:	Name:
Date:	Date:

Approved by the Head of Department, **B. Tech CSE**

Name: Dr. Nithyanandam P

Date:

(Seal of SCOPE)

ABSTRACT

The proposed research deals with constructing a sign gesture recognition system by engineering 6 types of features – hand coordinates, convolutional features, convolutional features with finger angles, CNN features on hand edges, L.C. of BRISK, CNN features on BRISK; trained on ensemble classifiers as well as artificial neural networks to accurately predict the label of the sign image provided as input. These features are utilized to handle a diverse variety of images that include labyrinthine backgrounds, user-specific distinctions, miniscule discrepancies between classes and image alterations. In addition to that, a hybrid ANN is also fabricated that takes 2 of the aforementioned features namely, convolutional features and CNN features on hand edges to precisely locate the hand region of the sign gesture under consideration in an attempt for classification. Experiments are also performed with CNN on the NUS I and II datasets which are not accurately classified by the previous 2 methods. When the above procedure is applied on the ASL alphabet, it gives the highest accuracy of 98.992% when training hand coordinates on an ANN and an accuracy of 95.368% on a hybrid ANN. Furthermore, the research also delves into the domain of sign image generation by testing various types of GAN on both simple and complex datasets as a means to comparatively analyse the quality and diversity of the hand gestures produced. As a result, it is able to show that utilizing autoencoder noise i.e. the latent representation of the image dataset itself, leads to a decrease in generative loss in GAN with less noisy images and better FID scores compared to a standard GAN using random noise.

ACKNOWLEDGEMENT

It is my pleasure to express with deep sense of gratitude to Dr. Shridevi S., Associate Professor, School of Computer Science and Engineering, Vellore Institute of Technology, Chennai, for her constant guidance, continual encouragement, understanding; more than all, she taught me patience in my endeavor. My association with her is not confined to academics only, but it is a great opportunity on my part of work with an intellectual and expert in the field of Artificial Intelligence.

It is with gratitude that I would like to extend thanks to our honorable Chancellor Dr. G. Viswanathan, all the vice presidents Mr. Sankar Viswanathan, Dr. Sekar Viswanathan and Mr. G V Selvam, Assistant Vice-President Ms. Kadhambari S. Viswanathan, Vice-Chancellor Dr. Rambabu Kodali and Pro-Vice Chancellor Dr. V. S. Kanchana Bhaaskaran for providing an exceptional working environment and inspiring all of us during the tenure of the course.

In jubilant mood I express ingeniously my whole-hearted thanks to Dr. Nithyanandam P, Head of the Department, Dr. Karmel A, Co Chair, and Project Coordinator Dr. Abdul Quadir Md, B. Tech. Computer Science and Engineering, SCOPE, Vellore Institute of Technology, Chennai , for their valuable support and encouragement to take up and complete the thesis.

Special mention to Dean, Dr. Ganesan R, Associate Dean, Dr. Geetha S, SCOPE, Vellore Institute of Technology, Chennai, for spending their valuable time and efforts in sharing their knowledge and for helping us in every aspect.

All teaching staff and members working as limbs of our university for which prompted the acquirement of the requisite knowledge to finalize my course study successfully. I would like to thank my parents for their support.

It is indeed a pleasure to thank my friends who encouraged me to take up and complete this task. At last but not least, I express my gratitude to all those who have helped me directly or indirectly toward the successful completion of this project.

Place: Chennai

Date: May 2022

Ayanabha Jana

CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF APPENDICES

LIST OF ABBREVIATIONS

CNN	Convolutional Neural Network
RGB	Red Green Blue
ReLU	Rectified Linear Unit
XGBoost	Extreme Gradient Boosting
PCA	Principal Component Analysis
SVM	Support Vector Machine
ASL	American Sign Language
SIFT	Scale Invariant Feature Transform
SURF	Speed Up Robust Features
ORB	Oriented FAST and Rotated BRIEF
FAST	Features from Accelerated Segment Test
BRIEF	Binary Robust Independent Elementary Features
KNN	K-Nearest Neighbors
NUS	National University of Singapore
ISL	Indian Sign Language
HSV	Hue Saturation Value
YCbCr	Luminance(Y) Chroma blue Chroma red
RVM	Relevance Vector Machine
SimpSVM	Simplification of Support Vector Machine
ASLID	American Sign Language Image Dataset
HGRS	Hand Gesture Recognition System
EOH	Edge Orientation Histogram
ROI	Region of Interest
COHST	Combined Orientaion Histogram and Statistical Parameters
LS-HAN	Hierarchical Attention Network with Latent Space
DTW	Dynamic Time Warping
LSTM	Long Short Term Memory
CSL	Chinese Sign Language

RNN	Recurrent Neural Network
CTC	Connectionist Temporal Classification
SLR	Sign Language Recognition
SLP	Sign Language Production
LC	Linear Combination
BRISK	Binary Robust Invariant Scalable Keypoints
csv	comma separated values
ML	Machine Learning
ANN	Artificial Neural Network
VGG	Visual Geometry Group
DCGAN	Deep Convolutional Generative Adversarial Network
WGAN	Wasserstein Generative Adversarial Network
GP	Gradient Penalty
RNN	Recurrent Neural Network
FID	Fretchet Inception Distance

Chapter 1

Introduction

1.1 INTRODUCTION

Sign Language is the mode of communication employed by specially-abled people who are unable to hear or speak or both. It comprises of a series of hand gestures mainly concentrated around the upper portion of the body, including the face occasionally to convey sentences and emotions. There are various sign language systems based on the diverse alphabet systems as well as regional variances. For instance, Indian Sign Language addresses both the Hindi alphabet as well as the English alphabet, varying with respect to the number of hands utilized as compared to the American Sign Language. Despite the contrasts exhibited by such systems, their main goal is to facilitate human interaction. However, not everyone is well versed with sign language which can be real hassle when a normal speaker is trying to understand a sign user and vice versa. Hence, it is necessary to devise a system capable of interconverting between normal text and sign language to foster better communication with maximum possible accuracy. With regards to this, the proposed system aims to:

- Tackle the problem of sign image classification by extracting features most representative of the input image.
- Generate interpretable sign images by training on input image features.

1.2 STATE-OF-THE-ART METHODS

Considering the first objective in the previous section, a standard approach in practise is to apply a convolutional neural network (CNN) to segregate the distinct image classes. To this end, [1] attempts to devise a hand-gesture recognition model without involving any hybrid processes such as image pre-processing, segmentation and classification. The CNN comprises of an input layer that takes a RGB image of 256x300 pixels and then passes it on to a convolutional layer of 16 filters of size 4x4. The feature maps thus generated is activated via a Rectified Linear Unit (ReLU) layer followed by cross-channel normalization with a window size of 2. A Max pooling layer is then applied to decrease spatial resolution of the input in order to avoid overfitting. After passing through this convolutional block, the pooled output is fed to a fully connected layer to identify further patterns in the input image with a rate and bias factor of 20. Lastly, a softmax layer is trained for classification purposes. The resultant CNN when trained on the Cambridge hand gesture dataset using stochastic gradient descent with momentum for 100 epochs with a learning rate of 0.00001 gives an accuracy of 96.66%. In comparison to this methodology which requires extensive hyperparameter tuning, the proposed system makes use of ensemble methods like Random Forest and XGBoost to augment overall classification accuracy at the expense of an initial feature extraction phase which reduces the input of the system to numerical features in favour of a classification mechanism with less complexity.

A thorough analysis of sign image classification also highlights the fact that hand segmentation is an important pre-processing step since very often the sign images are fraught with background complexity. [2] mitigates this issue by exploiting depth information of RGB images captured using a Microsoft Kinect sensor. In the first step, a threshold is set by assuming the hand to be the closest object to the sensor. This is followed by pixel normalization to brighten the hand region and

blacken the background. Then a median filter on the hand region removes noise followed by detecting the wrist line i.e. removing the forearm region. In the second step, the scaled hand segmented images are fed to a Principal Component Analysis network (PCANet) to learn features of the depth images. It comprises of two convolutional layers – the first one comprises of L_1 filters to learn low-level features which are divided into B blocks then passed to the second one with L_2 filters to learn high-level features. The histogram of each block in binary form is appended to obtain the feature vector. The filters are selected based on the largest eigenvalues obtained from PCA. Lastly, the features are trained on a Support Vector Machine (SVM) classifier to learn the sign labels via generalization. When trained on the ASL fingerspelling dataset using two architectural styles i.e. single PCANet model and user-specific PCANet model with score fusion, the resultant accuracies with leave-one-out-strategy are 88.70% and 84.50% respectively. Contrary to this, the proposed system trains hand coordinates of the sign gestures, which accurately captures the location of the hand region in 3D coordinate space all the while differentiating between similar image classes in terms of relative distance between the coordinates without the need for any extra hardware. In addition to that, ensemble methods like Random Forest overcome the scalability problem of SVM beyond a certain no. of data instances.

Another major feature extraction process for images is the detection of keypoints which describe spatial regions of interest in an image and are invariant to scale and rotation. With the assistance of these keypoints, [3] creates a system to identify Indian sign language gestures using depth images in a cluttered environment. In the first step, the hand region segmented from the depth image is converted to 8-bit grayscale followed by a Gaussian blur to remove noise. The rest of the image is removed using adaptive thresholding which dynamically blackens the background in blocks of fixed size across the image using a threshold. In the next step, these keypoint-related features such as SIFT (Scale Invariant Feature Transform), SURF (Speed Up Robust Features), ORB (Oriented FAST and Rotated BRIEF) are extracted which are then clustered using K-means algorithm to form a bag of visual words that gives a new representative feature vector. Finally, the vectors are learned using SVM and K-nearest neighbors (KNN) classifier. Training the aforementioned system on self-generated ISL digits gives a maximum accuracy (when using KNN) of 93.26%. It also gives maximum accuracy of 80.6% and 85.6% on NUS I and NUS II dataset respectively, when employing KNN. With depth images however, there is a possibility that certain objects are at same depth of the hand, hence they are segmented together. The proposed system attempts to alleviate this shortcoming by taking a skin mask which only retains the hand in a complex backdrop. Extracting finger angles with respect to each other as a feature takes into account the similarity between the various sign images to better separate the class labels, something which has been overlooked in [3]. For the NUS I and NUS II dataset, owing to the complicated backgrounds as well as negligible differences between disparate classes, CNN classifiers are also experimented with.

Moving on to the second objective of the research, the de facto standard of generating images is by tapping the power of GAN (Generative Adversarial Network). [4] puts this power to use in 3 steps:

- With the help of an autoencoder using Luong attention, it is able to decode hidden representations of sentences to generate a probability distribution of glosses from input text.
- A Motion graph then produces skeletal poses that translate between different glosses.

- The GAN generator conditioned on a skeletal pose map follows an encoder-decoder architecture to build the current image based on the previous image, whereas the GAN discriminator checks the correctness of the generated images against the skeletal pose map.

It obtains the skeletal poses using the RWTH-PHOENIX14T dataset and generates image sequences on the SMILE dataset. In contrast, the methodology described in section 3 aims to produce interpretable sign images by training variations of GANs with different parameters to inspect image quality and percentage of generated classes.

1.3 THESIS OUTLINE

Having discussed the need for a robust system for sign users along with state-of-the-art works in this domain, the next chapter covers a review of noteworthy literature for the same. Chapter 3 then outlines the proposed methodology along with the datasets used whereas Chapter 4 of this thesis presents the results of the experiments performed along with notable inferences and their explanations. Last but not the least, a conclusion is drawn on the overall research and any scope for future improvements is highlighted.

1.4 RESEARCH CONTRIBUTIONS

The experiments performed for this thesis has led to the following research contributions:

- The usage of numerical image features obtainable from a CNN, with classifiers other than ANN for acquiring stellar accuracy values, in this case, ensemble models such as Random Forest and XGBoost.
- A neoteric method to get representative features from an image with BRISK keypoints on hand edges.
- The inception of a hybrid ANN architecture that exploits both original image data and edge image data to augment classification performance on sign gestures compared to a single ANN.
- Quantitative and qualitative proof that when autoencoder noise is employed for training a GAN, the generator loss dwindles giving better quality sign images.

Chapter 2

Literature Review

2.1 HAND GESTURE RECOGNITION

[5] attempts to recognise the ASL alphabets and numbers by using image processing techniques laid out in 2 steps. In the first step, the HSV colour model is utilized to pre-process the images of the hand gestures, mainly to detect the skin area or more specifically the pixels pertaining to the skin colour in an input image. This is achieved by constructing a gray colour model using YCbCr colour space from the RGB image and then converting it into a black and white image, to which binary erosion and dilation is applied followed by a median filter to diminish salt and pepper noise. Now, in the second step the *Roundness* of the image is obtained from the following formula:

$$RND = \frac{(4 * \pi * ARE)}{PER * PER} \quad (1)$$

where,
ARE = Area (actual no. of pixels around the centroid of the image)
PER = Perimeter (the distance around the boundary of the centroid)
After that, eq. 2 is used to calculate the peak offset with respect to the centroid:

$$pkoffset = CEN(:,2) + 0.9 * (CEN(:,2)) \quad (2)$$

where,
CEN = centroid of the image
With this peak offset, the number of peaks is found which can be used along with the value of Roundness to identify the hand signs. The process is tested on 6 ASL alphabet sets and 6 ASL number sets with success rate or classification accuracy as the metric. For alphabets, the first 4 sets have a success rate of 100% while the last two sets have success rates of 70.83% and 87.5% respectively because of occluded images. Similarly, for numbers, sets 1-4 have 100% success rate whereas set 5 and set 6 have success rates of 80% and 90% respectively due to occlusion.

[6] classifies hand gestures dynamically from a real-time video using feature vectors trained on a SVM classifier. First and foremost, the input frame of the video is converted to HSV colour space and in order to determine the shade of skin, the hue value of 0.1 is used as the threshold to get the resultant binary image. Noise removal and smoothening is performed, and the depth information is segmented based on the hand location. Next, in order to recognize the hand in the frame, a feature vector is constructed for testing the system on 4 hand signs – “A”, “B”, “C” and “Hello”. The empty feature vector at the initialization stage has a length of 30 frames. The features include:

- Hu-moments which are location, angle and shape invariant
- Fingertip detection which is performed using spatial domain method that takes the centre of a circular filter, calculates the radius from various hand images, crops hand image from circular filter and then finds the extreme point of each finger
- Trajectory of the hand tracked using Kalman filter for avoiding occlusion, storing the difference between centroid coordinates of two adjacent frames

Now, that the feature vector is obtained, multiclass SVM is applied on it to classify the 4 input gestures. This system had a recognition rate of 97.5% for a total of 80 testing frames out of which 78 are classified accurately.

[7] aims to compare the performance of SimpSVM and RVM algorithms for sign language recognition on two datasets. SimpSVM or Simplified SVM approximates the normal vector of the hyperplane whereas RVM is built on a Bayesian model where the support vectors follow a Gaussian distribution. In the first case, the Auslan dataset is used which comprises of 22 features – x, y, z, roll, pitch, yaw, thumb bend, forefinger bend, middle finger bend, ring finger bend and little finger bend – for both hands. Feature extraction is performed on these 22 channels of information to get two types of features:

- Global (features of stream as a whole) – mean, maxima and minima
- Metafeatures (cluster of synthetic event attributes) – increasing, decreasing, flat, localmax, localmin

After this, parameter selection is done for both the models using grid search and sign classification is carried out using 5-fold cross validation. SimpSVM acquired an accuracy of 98.09% whereas RVM had an accuracy of 93.37%. However, RVM took less training time because it only had 521 basis functions compared to 561 support vectors for SimpSVM. In the second case, for the ASLID dataset which comprised of annotations for various signs, a grid search was conducted for parameter selection followed by 5-fold cross validation on both the algorithms. SimpSVM outperformed RVM in this case with even the training time of RVM being higher.

[8] evaluates the performance of two deep learning methods on the ASLID dataset, pre-annotated seven upper body joint locations namely left hand(LH), left elbow(LE), left shoulder(LS), head(H), right hand(RH), right shoulder(RS) and right elbow(RE). The methods are as follows:

- A deep neural network which uses the context of body joints by solving a regression equation and increasing accuracy via a cascade of pose regressors
- A neural network utilising heatmap regression for pose estimation

For both the models, transfer learning is used i.e. the weights of the neural network trained for pose estimation is utilised on a dataset for a similar application area – sign gesture recognition. The evaluation protocol is an estimation of correctness given the difference between the prediction and the actual joint is less than a threshold. The overall accuracy is a mean of the left and right portions of the body. Transfer learning provides a better performance for sign language estimation in this regard.

[9] utilizes the UoM-ISL sign language dataset comprising of sign videos to interpret the sentences being conveyed by the same. The proposed method aims at capturing spatial features by the following steps:

- Video frames are converted to HSV colour space to segregate the skin and non-skin regions. Further morphological operations on the data accurately generates the components pertaining to only the hands and face.
- Spatial relationships between the face, manual hand and non-manual hand are represented by the centroids of all the three components (C_1, C_2, C_3) and the global centroid (GC). The feature vector thus obtained is of the form:

$$F = \{d_1, d_2, d_3, d_4, d_5, d_6, \theta\} \quad (3)$$

where,
d_1, d_2, d_3 – distance between (GC, C_1), (GC, C_2) and (GC, C_3) respectively
d_4, d_5, d_6 – distance between (C_1, C_2), (C_2, C_3) and (C_3, C_1) respectively
θ – angle between C_1C_2 and C_2C_3
During instances when the manual and non-manual hand overlaps, the centroid of the current frame is obtained by taking the mean of the centroid in the previous frame and the resultant centroid due to overlapping in the current frame. For all the frames, the face component is considered to be static.

- To keep the no. of frames for the same sign in different instances constant i.e. set the key-frames, k-means clustering is applied where each sign is represented by k number of key frames. Also, to account for intra-class variations of a single sign, all the instances undergo hierarchical clustering. Once clustering has been performed, the interval valued type symbolic feature vector is derived which is the j^{th} key frame of reference feature vector for the p^{th} cluster of a sign S_i
- To identify a test sign, it is represented as a feature vector and a similarity measure is performed with all the signs in the database for L number of frames and the text corresponding to the one with the maximum similarity is returned using nearest neighbours classification.

The overall recognition rate for all the testing samples of the dataset in terms of F-measure is 58.82 and the method is found to be robust in terms of variations in the sign language gestures.

[10] designs a Hand Gesture Recognition System (HGRS) for identifying the 26 alphabets of ASL using a vision-based approach called Edge Orientation Histogram (EOH). First and foremost, the image of the sign is captured using a webcam at 1m distance with a frame size 160x120. In the image pre-processing stage, after applying colour conversion, noise removal and morphological operations, the skin pixels are extracted and a median filter and a Gaussian filter is applied on the gray image to preserve the edges and smooth the image respectively. Next, a blob of size 80x60 is used to extract the Region of Interest (ROI) of the hand. EOH technique is then applied on the noise-free image, in which the image is labelled using eight- pixel connectivity and the largest area set that is estimated from the labelled region is set to white and the other areas are set to black. This results in a blob which represents the hand in the video frame. In the following stage, a feature vector is created from EOH coefficients calculated using the CompareHist descriptor which computes the distance between running image R and training image X_j for ASL database α as follows:

$$C_j = \underset{j \in \alpha}{max} CompareHist(R, X_j) \quad (4)$$

The feature vector is then matched using the Sim-EOH algorithm and pattern recognition based on maximum similarity returns the appropriate ASL alphabet. This methodology described in [10] obtains a recognition rate of 88.26% within a time interval of 0.5 second.

[11] compares four different features – Orientation Histogram, Statistical Parameters, a combination of the previous two (COHST) and Wavelets – to recognize the ASL numerical digits (0-9). First of all, the RGB image of the hand gesture is captured with a plane background, varying in scale and rotation for a robust system. The hand is then segmented from the image by converting it to a binary image. Median filtering is used to remove noise and image thinning enhances the shape. A gray image is also formed by filling the white pixels with gray pixels from the original

image. The next stage involves feature extraction. For Orientation Histogram, the gradients dx and dy are obtained followed by taking an inverse tan of dy/dx. Converting the resultant pixel matrix to degrees, scanning returns the no. of degrees in each histogram bin which is used as a feature vector of length 18. The Statistical Parameters involves six components – mean, standard deviation, variance, coefficient of variance, range & root mean square of successive difference – which are extracted from the 1D version of the grayscale image. The COHST feature hence has a length of 24. In the case of wavelets, the binary image is used for extracting 2D DWT using Haar wavelet. This gives four sub-band regions: LL, LH, HL and HH, to which after applying 4 levels of decomposition, a feature vector of 64 elements are obtained. Lastly, the signs are identified by feeding the features individually to a neural network. Using the testing data, it is found that COHST outperforms their individual components obtaining a recognition rate of 87.94% and the Wavelet features has the highest recognition rate of 98.17%.

[12] trains a convolutional neural network on 10 ASL alphabets – A, B, C, D, H, K, N, O, T and Y – by taking 200 instances of each sign and a training to testing ratio of 80:20. In the first step, the RGB image is converted to HSV colour space for background removal and retaining only the hand gesture in the image, which undergoes dilation and erosion with an elliptical kernel. The image is then converted to grayscale where the non-black pixels are binarised. This retains the white hand against the black background with a frame size of 64 by 64 pixels. In the next stage, the binary pixels of the images are extracted as feature vectors. These vectors are fed to a 2D CNN with 32 3 by 3 feature maps that is compiled using category cross entropy loss function and the Adam optimizer. This proposed method achieves an overall accuracy of 98%.

[13] devises a system for continuous sign language recognition (SLR) called the Hierarchical Attention Network with Latent Space (LS-HAN) removing the need for temporal segmentation. It comprises of three components: a two-stream CNN for feature representation, a Latent Space (LS) for semantic bridging and a HAN for latent space based recognition. The proposed system considers both the error between video and sentence & also the recognition error. Each word in the sentence is annotated using a one-hot vector which are then mapped to semantic space along with the video features. Then the Dynamic Time Warping (DTW) algorithm finds the distance between both the feature sets. This data is then fed to a HAN that computes log-probability for the sentences, encoding them using bidirectional LSTM. Lastly, a softmax function over the probability distribution selects the most relevant word. This system is tested on the CSL dataset and the RWTH-PHOENIX-Weather dataset, with accuracy metric as follows:

$$Accuracy = 1 - \frac{S + I + D}{N} * 100\% \quad (5)$$

where,
S, I, D – minimum number of substitutions, insertions and deletions needed to transform a hypothesized sentence to ground truth
N – minimum number of words in ground truth
The system gives an accuracy of 82.7% and 61.7% respectively.

[14] maps video sequences of sign language usage to a limited training dataset annotated with gloss labels with the help of sequence learning by representing the input as a spatio-temporal feature via a recurrent neural network. The sequence learning model is trained using a bidirectional

LSTM which predicts the category of the input from the feature obtained from the RNN. Once the gloss labels have been predicted, a detection net employing stacked temporal convolution finds the detection score to select only those segments from the video that aligned maximally with the gloss labels. This is followed by a three-stage optimization which is as follows:

- Connectionist Temporal Classification (CTC) is used as an objective function on the gloss labels to give alignments between the input and target sequence.
- The alignments thus acquired is used to learn the features of the model by assigning these to the temporal segments to build further training samples for stronger supervision.
- Once stronger spatio-temporal features are obtained, the detection net sieves out unnecessary temporal segments which prevents overfitting in the RNN and improves its generalization power.

This proposed implementation is applied on the RWTH-PHOENIX-Weather dataset with the evaluation metric as word error rate (WER) defined as follows:

$$WER = \frac{\#sub + \#del + \#ins}{\#words\ in\ reference} \tag{6}$$

where,
#sub, #del, #ins – least no. of substitutions, deletions and insertions to transform the reference sequence into the target sequence
The error rate comes out to be 46.9%.

The proposed system attempts to surpass the accuracy of the following literature:

Table 1. Base papers for comparison - SLR

Paper	**Problem Statement**	**Dataset**	**ML/DL models used**	**Results**
[1] Sagayam, K. et al.	Devise a hand-gesture recognition model using a well-tuned deep CNN without using hybrid processes such as image pre-processing, segmentation and classification	Cambridge Hand Gesture Dataset	• Convolutional Neural Network • Stochastic Gradient Descent with momentum	Accuracy – 96.66% Sensitivity – 85% Specificity – 98.12%
[2] Aly, W. et al.	Devise a user-independent recognition system that exploits depth information of RGB images to learn features using PCA for classifying the sign gestures	ASL Finger Spelling dataset	• Depth thresholding and median filter • Principal Component Analysis (PCA) network • Support Vector Machine (SVM)	Accuracy • Single PCANet – 88.70% • User-specific PCANet model – 84.50%

[3] Gangrade, J. et al.	Devise a system to recognize signs in a cluttered environment invariant of scaling, rotation and lighting	Self-generated ISL digits 0-9, NUS Dataset I and II	• Adaptive Thresholding and Gaussian blur • 7Hu Moments, SIFT (Scale Invariant Feature Transform), SURF (Speed Up Robust Features), ORB (Oriented FAST and Rotated BRIEF) • K-means clustering • SVC and KNN classifier	Accuracy ISL Digits • ORB + Nu-SVC = 90.4% • ORB + KNN = 93.26% NUS Dataset I • ORB + Nu-SVC = 76.9% • ORB + KNN = 80.6% NUS Dataset II • ORB + Nu-SVC = 81.25% • ORB + KNN = 85.6%

2.2 IMAGE GENERATION

Despite GANs being the most popular neural network for generative purposes, another type of generative model is the autoencoder which is a neural network comprising of two parts – an encoder which represents an image as a 1D vector and a decoder which reconstructs the same image from the 1D vector. [15] discusses the various types of autoencoders with the regularized autoencoders for image compression, denoising autoencoders for noise removal and variational autoencoders for interpolation.

Even GANs have several types with the most widely used being the Deep Convolutional GAN or DCGAN[16] that employs convolutional layers to generate high quality images compared to a normal GAN. Another notable architecture is the SRGAN[17] built with a deep ResNet used for upscaling the generated images to super resolution by calculating the perceptual loss on feature maps of a VGGNet. Yet another interesting usage of GAN is detailed in [18] which generates images from textual descriptions by conditioning a DCGAN on text encodings obtained from a convolutional RNN. The generator is conditioned on the text description and random noise whereas the discriminator attempts to perform a feature matching of the input image given the text encoding.

The generated images in the GAN experiments outlined in Chapter 4 are compared qualitatively with the generated images of the following literature:

Table 2. Base paper for comparison – SLP

Paper	**Problem Statement**	**Dataset**	**ML/DL models used**
[4] Stoll, S. et al.	Devise a system to generate sign videos from text with minimum gloss and skeletal annotations	SMILE dataset	• AutoEncoder • Motion Graph • GAN

Chapter 3

Materials and Methodology

The proposed system is divided into two parts:

- *Sign Language Recognition (SLR)*
  It differentiates between various static hand gestures i.e. images, with the help of representative features with the core objective of maximizing the accuracy on the testing dataset.
- *Sign Language Production (SLP)*
  It generates interpretable sign images by training a generative model on the image dataset itself. The main aim is to generate as many image classes as possible all the while keeping a check on image quality.

To build this system, the following libraries in Python are used:

- *Scikit-learn* – for train-test division, calculating machine-learning accuracies, using ensemble methods
- *Keras and tensorflow* – for building neural networks
- *Mediapipe* – for using hand detection API to get coordinates
- *OpenCV* – for image reading and processing, including keypoint finding
- *OS* – for handling image directories
- *Pickle* – for storing image features
- *Numpy* – for storing image data and getting random noise for GANs
- *Pandas* – for reading image features and managing csv files

3.1 SIGN LANGUAGE RECOGNITION

3.1.1 DATASETS

The proposed methodology is applied on the ASL alphabet dataset [19] with 80% training data and 20% testing data. In addition to that, the methodology is also demonstrated on datasets associated with state-of-the-art techniques to draw a comparative analysis on the results obtained and any significant improvement observed. These datasets include the Cambridge hand gesture dataset [20], ASL fingerspelling dataset[21], NUS dataset I and II [22] and ISL digits[23]. A glimpse of these datasets is shown in Table 3 whereas Table 4 outlines the details of the datasets used in this methodology.

Table 3. Datasets – ASL alphabet, ASL fingerspell dataset, Cambridge dataset, NUS I and II, ISL digits

ASL alphabet					

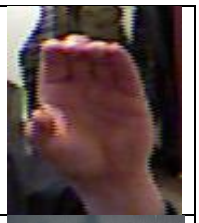
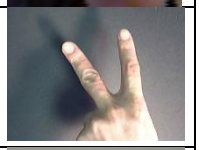
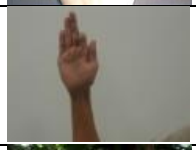
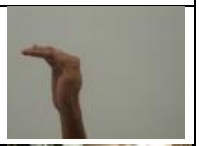
ASL fingerspell dataset					
Cambridge hand gesture dataset					
NUS I dataset					
NUS II dataset					
ISL digits					

Table 4. Dataset details - ASL alphabet, ASL fingerspell dataset, Cambridge dataset, NUS I and II, ISL digits

	Size	**No. of classes**	**Samples per class**	**Test size**
ASL alphabet	72000	24	3000	20%
ASL fingerspell dataset	65774	24	>2600	20%
Cambridge hand gesture dataset	63188	9	>6000	20%
NUS I dataset	240	10	24	25
NUS II dataset	2000	10	200	250
ISL digits	2000	10	200	250

3.1.2 SYSTEM ARCHITECTURE

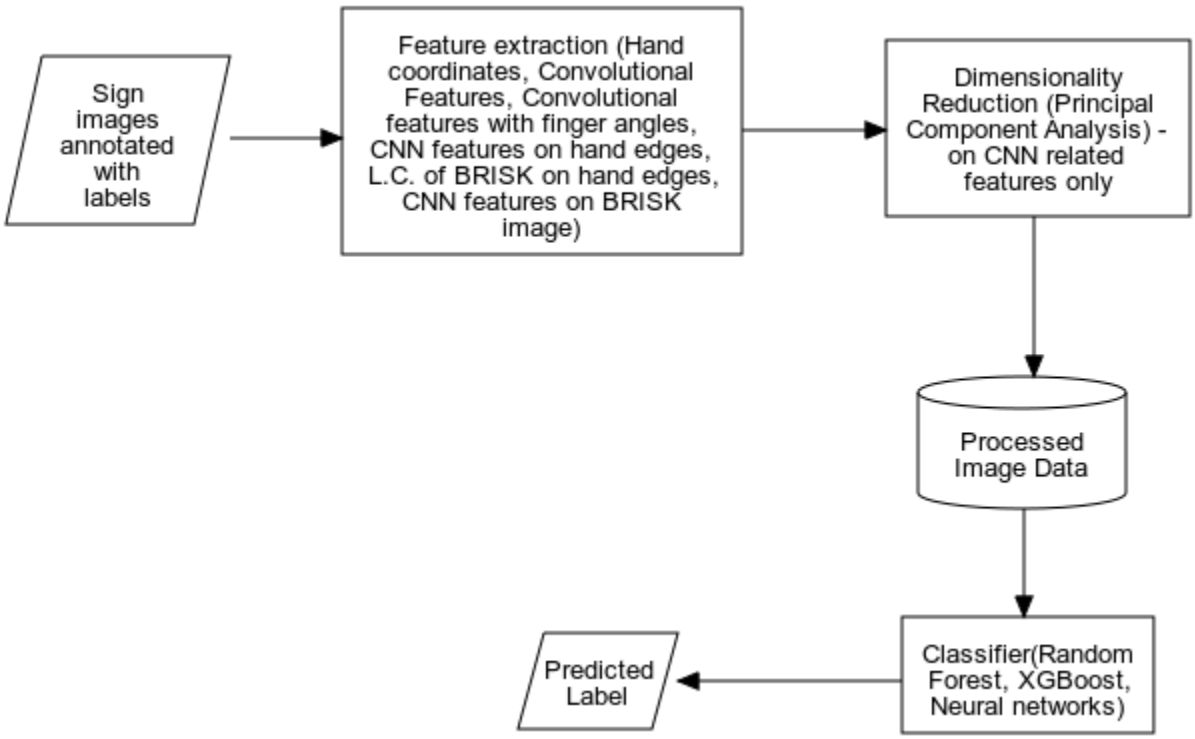


Figure 1. Architecture diagram for SLR

Figure 1 represents the operations involved in the SLR module. The input to the system is an annotated image dataset of sign images/hand gestures. Since image datasets are generally available in a directory structure, file operations are applied on the root directory as shown in Appendix 1 to extract the absolute paths and the image class directory name, and store them in a tabular csv format as illustrated in Figure 2.

	A	B	C
1		path	sign
2	0	D:\Capstone utilities\Datasets\asl_alphabet\A\A1.jpg	A
3	1	D:\Capstone utilities\Datasets\asl_alphabet\A\A10.jpg	A
4	2	D:\Capstone utilities\Datasets\asl_alphabet\A\A100.jpg	A
5	3	D:\Capstone utilities\Datasets\asl_alphabet\A\A1000.jpg	A
6	4	D:\Capstone utilities\Datasets\asl_alphabet\A\A1001.jpg	A
7	5	D:\Capstone utilities\Datasets\asl_alphabet\A\A1002.jpg	A
8	6	D:\Capstone utilities\Datasets\asl_alphabet\A\A1003.jpg	A
9	7	D:\Capstone utilities\Datasets\asl_alphabet\A\A1004.jpg	A
10	8	D:\Capstone utilities\Datasets\asl_alphabet\A\A1005.jpg	A
11	9	D:\Capstone utilities\Datasets\asl_alphabet\A\A1006.jpg	A
12	10	D:\Capstone utilities\Datasets\asl_alphabet\A\A1007.jpg	A
13	11	D:\Capstone utilities\Datasets\asl_alphabet\A\A1008.jpg	A
14	12	D:\Capstone utilities\Datasets\asl_alphabet\A\A1009.jpg	A
15	13	D:\Capstone utilities\Datasets\asl_alphabet\A\A101.jpg	A
16	14	D:\Capstone utilities\Datasets\asl_alphabet\A\A1010.jpg	A
17	15	D:\Capstone utilities\Datasets\asl_alphabet\A\A1011.jpg	A
18	16	D:\Capstone utilities\Datasets\asl_alphabet\A\A1012.jpg	A

Figure 2. Paths to various sign images

Such a format mitigates the complex navigation through image folders by reducing file access operations. After feature extraction and dimensionality reduction, the new image data is stored in the database, from where it can be utilized for training classification models that outputs the predicted label. The accuracy metric hence becomes crucial in evaluating the performance of the trained models.

3.1.3 METHODOLOGY

The task of sign recognition from a given hand gesture is essentially an image classification problem that requires features which are most representative of the input image. The proposed system tests six features extracted from the available dataset of images on supervised classifiers in an attempt to achieve maximum recognition accuracy.

3.1.3.1 FEATURE EXTRACTION

- **Hand Coordinates**

  It provides the location of 21 hand joints in 3D coordinate space exploiting the Mediapipe python library [24] as showed in Figure 3.

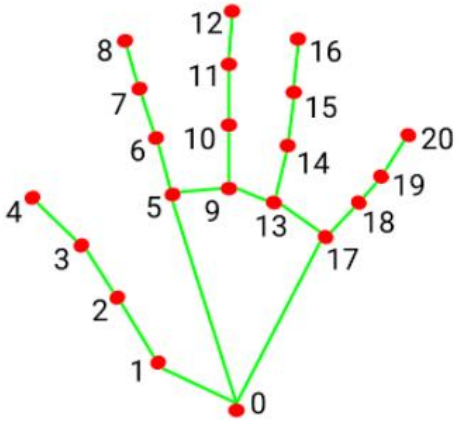


Figure 3. Mediapipe hands for finger tracking

As described in [24], Mediapipe makes use of 21 hand landmarks to track the hand region in real-time video as well as image/frames. These 21 points are obtained in the form of a 3-tuple represented as:

$$(x_i, y_i, z_i) \tag{7}$$

where, i=0 to 20

These coordinates are robust because even if the depth or orientation of the hand varies, the relative distance between the joints will be equal for the same sign. However, this procedure is constrained by the quality of the sign images and in certain scenarios may not detect the hand due to lighting conditions as well as image noise. Overall, it gives a feature vector of size 63.

- **Convolutional features**

  An artificial neural network cannot take images directly as input. A convolutional neural network (CNN) solves this problem by providing the neural network with convolutional layers that extract 2D feature maps using filters. Hence, when sign images are provided to

convolutional layers, these convolutional features can be obtained as a flattened 1D feature vector whose size depends on the CNN architecture. In the context of sign images, these features can indicate the position of fingers, hand orientation, etc. The architecture utilized for all convolutional-related features is described in Table 5.

Table 5. Architecture for convolutional-related features

Operation	Kernel	Stride	Filters	Pool	Activation
Conv2D	3x3	1x1	64		
Conv2D	3x3	1x1	64		
MaxPooling2D		1x1		2x2	
Conv2D	3x3	1x1	128		ReLU
Conv2D	3x3	1x1	128		ReLU
AveragePooling2D		1x1		19x19	
Flatten					

The two pooling layers in Table 5 extracts features in a summarised form from specific regions/windows of the image after convolution where the window size is specified as the pooling size. Pooling is also done in an attempt to reduce the dimensions of the feature maps.

- **Convolutional features + finger angles**

Different signs of the same language may look similar e.g. letter ‘a’ and ‘e’ in the American Sign Language as depicted in Figure 4.

Figure 4. Similarity between sign images: ‘a’ (left) and ‘e’(right)

However, such signs vary in terms of the relative position of the fingers with respect to each other. To address this issue, finger angles are proposed. There are four such finger angles which can be estimated from hand coordinates 0,3,5,8,12,16 and 20 extracted using Mediapipe[24] as shown in Figure 5 and Appendix 2.

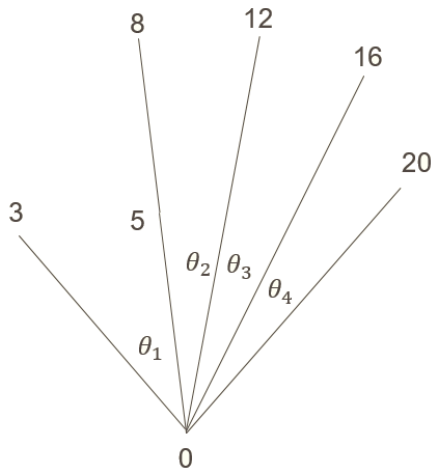


Figure 5. Finger angles

These finger angles can easily be calculated using the law of cosines for triangles which is as follows:

$$\cos\theta_1 = \frac{a_1^2 + b_1^2 - c_1^2}{2a_1b_1} \tag{8}$$

$$\cos\theta_2 = \frac{a_2^2 + b_2^2 - c_2^2}{2a_2b_2} \tag{9}$$

$$\cos\theta_3 = \frac{a_3^2 + b_3^2 - c_3^2}{2a_3b_3} \tag{10}$$

$$\cos\theta_4 = \frac{a_4^2 + b_4^2 - c_4^2}{2a_4b_4} \tag{11}$$

where,
a_1,b_1,c_1=dist(0,3),dist(0,5),dist(3,5)
a_2,b_2,c_3=dist(0,8),dist(0,12),dist(8,12)
a_3,b_3,c_3=dist(0,12),dist(0,16),dist(12,16)
a_4,b_4,c_4=dist(0,16),dist(0,20),dist(16,20)
dist(x,y)=Euclidean distance between points x and y

Algorithm 1 Calculating finger angles. Parameters used in this algorithm *static_image_mode* = False, *max_num_hands* = 1, *min_detection_confidence* = 0.5, *min_tracking_confidence* = 0.5, *vertices* = [0,3,5,8,12,16,20]

Require: *static_image_mode*, if set to True takes into account images in sequence. *max_num_hands*, number of hands to detect in image. *min_detection_confidence*, probability of hand detection. *min_tracking_confidence*, probability of hand tracking. *vertices*, list of coordinates for finding finger angles.

1: *hands* ← Mediapipe(*static_image_mode*, *max_num_hands*, *min_tracking_confidence*, *min_detection_confidence*)

2: Read *image*
3: *hand_landmarks* ← *hands*(*image*)
4: **for** *landmark* in *hand_landmarks* **do**
5: Store *landmark* (x,y,z) in *sign_hand_coordinate*
6: **end for**
7: **for** *vertex* in *vertices* **do**
8: *point_coordinates* ← *sign_hand_coordinate*[3**vertex*], *sign_hand_coordinate*[3**vertex*+1], *sign_hand_coordinate*[3**vertex*+2]
9: Store *point_coordinates* in *vertex_coordinates*
10: **end for**
11: **for** i = 1,…, length(*vertices*)-2 **do**
12: **if** i = 2 **skip**
13: *angle* ← *Cosine*(*vertex_coordinates*[0], *vertex_coordinates*[i], *vertex_coordinates*[i+1])
14: Store *angle* in *finger_angles*
15: **end for**
16: **function** *Cosine(A,B,C):*
$a \leftarrow \text{sqrt}(\text{sum}(A\text{-}B)^2)$
$b \leftarrow \text{sqrt}(\text{sum}(A\text{-}C)^2)$
$c \leftarrow \text{sqrt}(\text{sum}(B\text{-}C)^2)$
return $(a^2+b^2-c^2)/(2ab)$
end function

Similar to hand coordinates, finger angles also remain constant despite hand depth or orientation in the input image for the same class label. These finger angles combined with convolutional features outlined in Table 5 gives a hybrid feature.

- **CNN features on hand edges**
If a sign image is taken in a cluttered environment, the resulting convolution process may be unable to extract features specific to the sign. This is alleviated by segmenting the hand region with the help of an approximate skin mask so that the neighboring regions become black. Skin mask indicates a range of RGB values that encompasses the majority of skin tones that can be observed in sign images. A common RGB range is the real skin tone color scheme [25]. The proposed method however uses a modified version of the aforementioned scheme by decreasing the lower bound of the skin mask in an attempt to capture darker skin tones as well as skin tones under insufficient lighting. To make the hand region stand out even more, a Canny filter [26] is applied on the image after denoising using median blur to retain only the hand edges. Canny filter is preferred over Sobel filter [27] because the edges retained are smooth in nature. Figure 6 outlines the images obtained after every step of this method.

Figure 6. Original image(left);Masked image(middle);Edge image(right)

Algorithm 2 Finding hand edges from sign image. Parameters used in this algorithm *lb* = (112,54,28), *ub* = (255,219,172), *k_size* = 3, *lower* = 100, *upper* = 200

Require: *lb*, lower bound for skin mask. *ub*, upper bound for skin mask. *k_size*, kernel size for median blur filtering. *lower*, lower hysteresis threshold for Canny filter. *upper*, upper hysteresis threshold for Canny filter.

1: Read *image*
2: *mask* ← inRange(*lb*,*ub*)
3: *masked_image* ← *image* AND *mask*
4: *denoised_image* ← medianBlur(*masked_image*,*k_size*)
5: *edge_image* ← Canny(*denoised_image*,*lower*,*upper*)

Once the edge image has been obtained, the CNN described in Table 5 would be able to extract features corresponding to only the hand region without including any of the complex background since it has been removed when finding hand edges.

- **Linear combination of BRISK descriptors**

Keypoints describe regions of interest of an image where each keypoint is calculated by taking a neighborhood of pixels into consideration with specific intensities. Binary Robust Invariant Scalable Keypoints(BRISK) [28] returns these keypoints as a vector of size 64, representing the bins of an image histogram. When dealing with sign images however, BRISK may compute keypoints catering to objects in the background of the sign user if the images have not been taken in a clutter-free environment. So, in order to retain the keypoints of only the hand region, BRISK is applied on the edge image as shown in Figure 7.

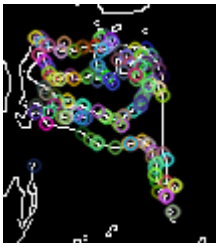


Figure 7. BRISK keypoints on hand edges

Each of the concentric circles in Figure 7 represents a keypoint and the radius of the circle indicates the span of pixels it considers. One thing to note while using these keypoints is the fact that their count can vary across images. Therefore, given a set of images, in order to classify them on the basis of a consistent number of features, a representative keypoint is required, which is estimated by taking a linear combination (Appendix 3) as follows:

$$K = \sum_{i=1}^{n} \alpha_i K_i \quad (12)$$

where,
α_i = weight of keypoint 'i'/total weight of keypoints
K_i = i^{th} keypoint

Algorithm 3 Linear combination of BRISK

Require: *edge_image*, final image obtained from Algorithm 2 after applying Canny filter.

```
brisk ← BRISK()
keypoints ← brisk.detectAndCompute(edge_image)
Initialize feature_vector of size 64 with 0s
total_weight ← sum(keypoints)
for keypoint in keypoints do
 Declare temporary variable temp
 proportion ← sum(keypoint)
 alpha ← proportion/total_weight
 for i=0,…,length(keypoint)-1 do
  temp ← feature_vector[i] + alpha*keypoint[i]
 end for
 feature_vector ← temp
end for
```

BRISK, similar to other keypoint algorithms, is invariant to scale and rotation i.e. any kind of affine transformation that preserves lines and features across diverse geometric variations of the same image. This is evident in Figure 8 where the same edge image when rotated clockwise by 90 degrees, retains the same keypoints.

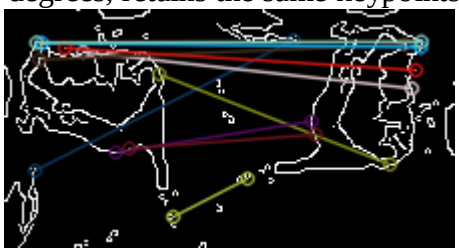


Figure 8. Keypoint matching

The colored lines in Figure 8 indicates the matched keypoints at the end of the lines in the original and the transformed image based on distance. Another added benefit of BRISK is that it is computationally less expensive than other keypoint algorithms such as SURF.

- **CNN features on BRISK image**

One major downside to Algorithm 3 is that finding a representative keypoint may not always suffice to segregate the various image classes in the dataset since the proposed method is essentially assigning larger weights to the prominent keypoints while smaller weights to keypoints whose bin values are less. But the keypoints with miniscule values

are not necessarily directly correlated to prominence in terms of constituting regions of interest. There is always a possibility that such keypoints are quintessential in defining a spatial region in the image that accurately distinguishes between two different image labels. Hence, in order to take all keypoints into consideration, the CNN described in Table 5 is applied on the BRISK image as shown in Figure 7 to get a feature vector.

3.1.2.2 DIMENSIONALITY REDUCTION

- **Principal Component Analysis**

  Since all the features of an image that are extracted using convolutional operations can be extensive in number, dimensionality reduction is applied on these convolutional-related features. This is done in order to retain only the most significant features so that the model does not learn from insignificant features that reduces accuracy. It also helps to lessen storage space and multicollinearity.

  In the proposed method, Principal Component Analysis or PCA is applied on the convolutional-related features. It utilizes orthogonal transformation to reduce a set of features specifically a covariance matrix of the features as shown in eq. 13 into a set of uncorrelated features, which correspond to the maximum eigen values and at the same time retaining trends and patterns such as patterns that are capable of identifying different hand gestures.

$$\begin{matrix} cov(if_1, if_1) & cov(if_1, if_2) & cov(if_1, if_3) \\ cov(if_2, if_1) & cov(if_2, if_2) & cov(if_2, if_3) \\ cov(if_3, if_1) & cov(if_3, if_2) & cov(if_3, if_3) \end{matrix} \quad (13)$$

  where,
  if_1, if_2, if_3 = image feature 1, image feature 2, image feature 3
  cov(x,y) = covariance between variables x and y

  The set of uncorrelated features obtained after applying PCA are referred to as principal components which are linear combinations of the original data containing as much compressed information as possible in decreasing order i.e. the first component has maximum information, the second component has second maximum information and so on. If this is the case, then the question that prevails is that how to decide on the number of principal components. This can be answered with the help of a scree graph [29] as shown in Figure 9 and Appendix 4.

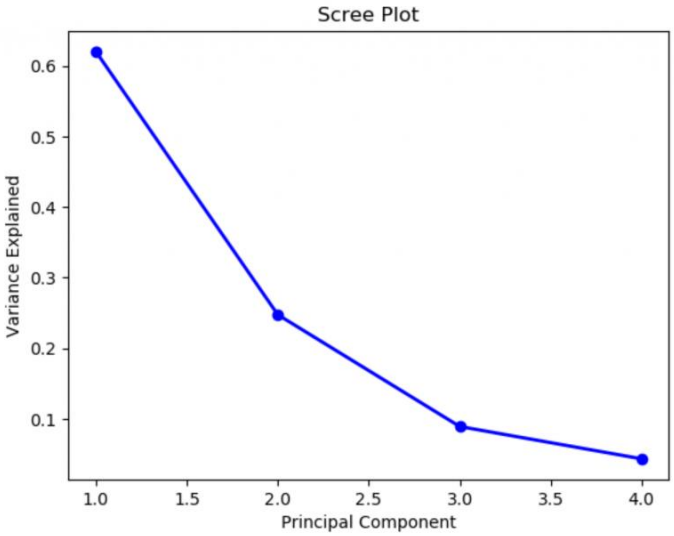


Figure 9. Scree graph to select number of principal components

A scree graph is a line graph between variance and number of components. Similar to the elbow curve used for diagnosis in k-means clustering for selecting the number of clusters in unsupervised data, by observing the scree graph, the elbow value corresponding to the x-axis beyond which there is negligible decrease in variance is chosen as the ideal number of components for PCA. For instance, in figure 9, 3 is chosen as the ideal PCA parameter. However, the scree graph is not always reliable due to presence of multiple elbows as can be seen in Figure 9.

3.1.3.3 CLASSIFICATION

3.1.3.3.1 ENSEMBLE METHODS

When the results from multiple machine-learning models are combined, an ensemble is obtained. An ensemble is capable of minimizing the variance of the predictions or the amount of error incurred. This ability is leveraged by the proposed system to augment classification accuracy for the sign images.

- **Random Forest**
  An ensemble method where all the individual models are decision trees is referred to as a Random Forest. A decision tree is a ML model that splits the input data at every node based on the objective of maximizing information gain until it reaches the leaves each of which comprises of data instances of a single class or a range of values. However, one decision tree is prone to overfitting. This is where bagging comes into picture. Bagging or bootstrap aggregating is used by Random Forest where it takes multiple decision trees, each trained on a random subset of features to obtain a final prediction which is a mean of results from all the trees. This ebbs away the variance which reduces overfitting. Another advantage of Random Forest is that it does appropriate feature selection at every node, so only the most significant predictors are retained. Figure 10 shows an intuitive decision tree that can be sampled by Random Forest for sign image recognition.

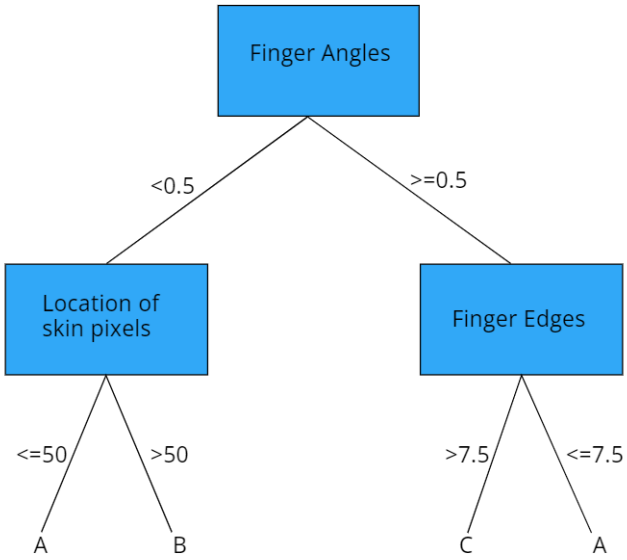


Figure 10. Decision tree for recognizing sign image

As can be observed in Figure 10, the decision tree splits the data based on the range of values for the given features at every node to classify three sign image alphabets – 'A', 'B' and 'C'. Many such decision trees are trained on the same or different subset of features to recognize other alphabets under consideration.

- **XGBoost**
  XGBoost or Extreme Gradient Boosting is an ensemble algorithm that uses regression trees as the base learners. Similar to decision trees, a single regression tree alone also suffers from the problem of overfitting. This is solved by XGBoost's boosting nature which gives more preference to misclassified instances during the next iteration of the algorithm and hence robust on unbalanced datasets. The model uses Gradient Boosting with each tree improving upon the errors of the previous tree in a sequential manner. After an initial prediction, a regression tree computes the gradient of the loss function and tries to minimize the same. Once the tree is optimized, it is added to the initial prediction, multiplied by a learning rate. This process continues iteratively until no further improvement in results is possible. What makes the algorithm 'Extreme' is the addition of L1 and L2 regularization which further reduces overfitting by shrinking the estimates of insignificant predictors. Figure 11 demonstrates the XGBoost process intuitively.

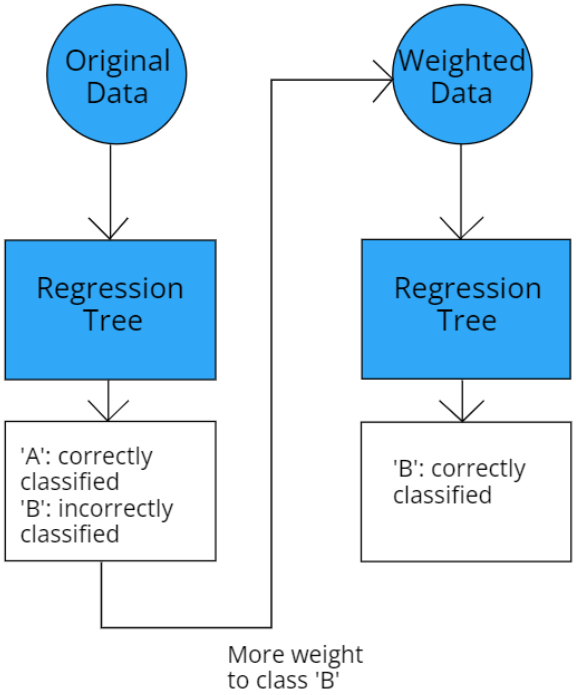


Figure 11. XGBoost algorithm for sign images

From Figure 11, it can be seen that when image data is fed initially to the first regression tree, it is able to correctly identify the sign images corresponding to the alphabet 'A' but the alphabet 'B' is misclassified. So, due to the algorithm's inherent boosting nature, during the next iteration, the sign images with class label 'B' are allocated more weight compared to class 'A'. A new regression tree is optimized on the gradients of the previous classification and as a result it is accurate in classifying the alphabet 'B'.

3.1.3.3.2. NEURAL NETWORKS

- **Artificial Neural Networks (ANN)**
  ANNs comprise of neurons or nodes arranged in layers starting from the input layer followed by some hidden layers and ending at an output layer, all of which are interconnected. At the beginning of the training phase, each node is assigned a weight and a threshold from which its output can be calculated as follows:

$$\sum_{i=1}^{n} w_i x_i + threshold \tag{14}$$

where,
w_i = weight of the node for input i
x_i = value of input i

The threshold determines the activation of the node wherein if the resultant output crosses this value, the corresponding node (or output) gets activated by the activation function[30] involved. After each step or epoch of the training phase, ANN is capable of learning hidden

patterns in the data (in this case, patterns distinguishing various sign gestures) and adjust its weights accordingly to generalize better on unseen data. Figure 12 provides an idea on how an ANN used for classifying sign images could look like.

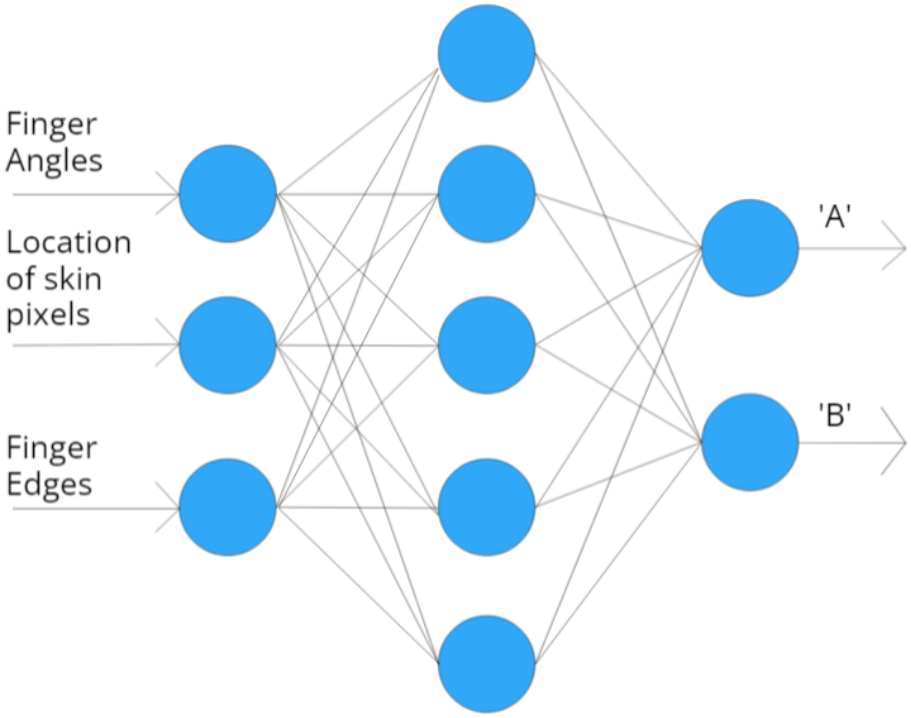


Figure 12. ANN for sign image recognition

From figure 12, it can be seen that a single hidden layer of nodes is used to learn the hand gesture patterns based on the numerical inputs and predict the English alphabet ‘A’ or ‘B’.

In the proposed system, experiments have been performed with diverse architectures of ANN differing in terms of the count of nodes and the number of layers. Starting with the ISL digits dataset[23], since it contains image classes which are not similar and free from complex backgrounds along with less number of data instances, a simple ANN structure when applied to the features extracted in section 3.1.3.1 suffices in classifying the digits with appreciable amount of accuracy. Table 6 outlines this ANN in detail.

Table 6. ANN architecture for ISL digits

Operation	Nodes	Activation
Dense	6	ReLU
Dense	6	ReLU
Dense	10	Softmax

For the Cambridge hand gesture dataset[20], the number of nodes and layers are changed across the different features. This dataset is also devoid of background complexity but the image classes involve sequential frames which is why hyperparameter tuning leads to disparate ANN architectures with the sole objective of maximizing classification accuracy on test data. Tables 7, 8, 9 & 10 describes the ANNs used in this scenario.

Table 7. ANN architecture for Cambridge hand gesture dataset – Hand coordinates

Operation	Nodes	Activation
Dense	32	ReLU
Dense	32	ReLU
Dense	9	Softmax

Table 8. ANN architecture for Cambridge hand gesture dataset – Convolutional features + PCA

Operation	Nodes	Activation
Dense	32	ReLU
Dense	32	ReLU
Dense	32	ReLU
Dense	16	ReLU
Dense	9	Softmax

Table 9. ANN architecture for Cambridge hand gesture dataset – Convolutional features + finger angles + PCA

Operation	Nodes	Activation
Dense	32	ReLU
Dense	32	ReLU
Dense	16	ReLU
Dense	16	ReLU
Dense	9	Softmax

Table 10. ANN architecture for Cambridge hand gesture dataset – CNN features on hand edges + PCA, L.C. of BRISK, CNN features on BRISK image

Operation	Nodes	Activation
Dense	64	ReLU
Dense	64	ReLU
Dense	64	ReLU
Dense	9	Softmax

As can be seen from Tables 7,8,9 & 10, not only the number of neurons has been increased but also there is a presence of more neural layers, since the Cambridge dataset[20] consists of a large number of instances and hence, the resultant ANNs require greater count of learnable parameters to account for both intra-class and inter-class variations.

In a similar manner, for the ASL alphabet dataset[19], two ANN architectures are utilized for the extracted features which are depicted in Tables 11 and 12.

Table 11. ANN architecture for ASL alphabet dataset – Hand coordinates, Convolutional features + PCA

Operation	Nodes	Activation
Dense	64	ReLU
Dense	64	ReLU
Dense	64	ReLU
Dense	24	Softmax

Table 12. ANN architecture for ASL alphabet dataset – Convolutional features + finger angles + PCA, CNN features on hand edges + PCA, L.C. of BRISK, CNN features on BRISK image; and ASL fingerspell dataset

Operation	Nodes	Activation
Dense	64	ReLU
Dense	64	ReLU
Dense	64	ReLU
Dense	64	ReLU
Dense	24	Softmax

The ANN in Table 12 is also used for the ASL fingerspell dataset[21]. Due to a cluttered backdrop and discrepancies in hand gestures performed by different users, this architecture has the highest number of nodes and layers among all ANNs described in this section to accurately segregate the class labels. Also, for both the ASL alphabet[19] and ASL fingerspell dataset[21], only 24 letters are taken into consideration because the letters 'j' and 'z' involve complicated movement and hence are outside the domain of static sign gesture recognition. With regards to the weight initialization technique, 'he_uniform'[31] is used as the preferable method for ReLU activation for all ANNs as shown in Appendix 5, except the one designed for the ISL digits dataset[23] which uses the default Glorot initialization[32].

- **Hybrid ANN**

The CNN features on hand edges as described in section 3.1.3.1 relies on an approximate masking operation to segment the hand edges. However, this technique is prone to retaining background edges as well, most specifically if the background encompasses hues within the skin mask used in this operation. To mitigate this issue, a hybrid ANN is proposed which is illustrated in Figure 13 and Appendix 6.

The hybrid ANN takes two set of inputs – convolutional features on the original image at the left of the neural network and CNN features on the edge image i.e. one containing hand edges at the right of the neural network, both of which are explained in detail in section 3.1.3.1. Both of these inputs are passed through fully connected blocks followed by a concatenation operation and a final fully connected block at the end of which the predicted class label is obtained.

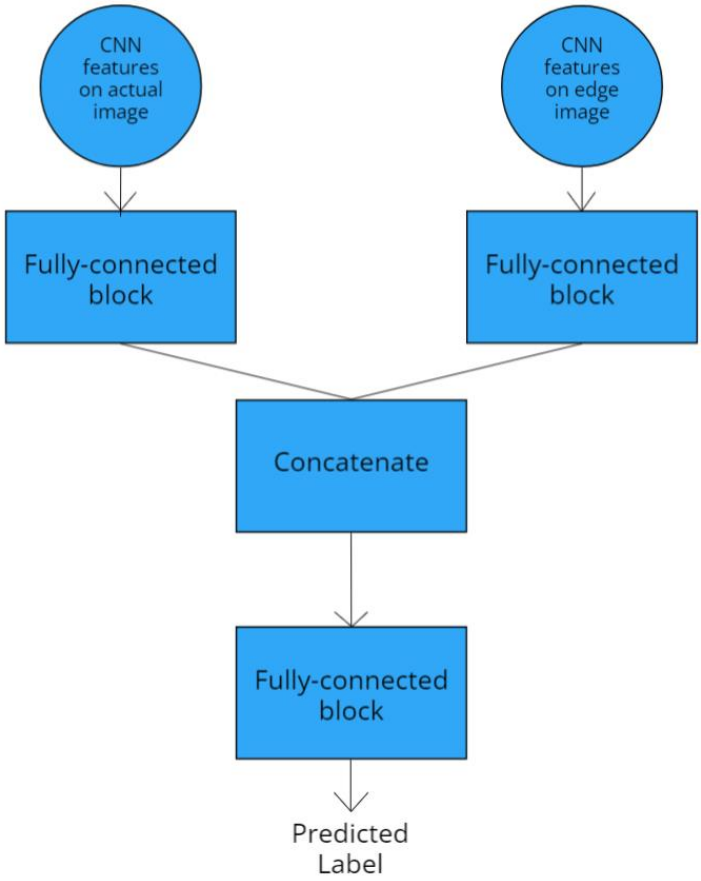


Figure 13. Illustration for hybrid ANN

The fully connected blocks correspond to a series of dense layers in a simple ANN whereas the Concatenate block gives a concatenated feature vector obtained by passing both the inputs through the fully-connected blocks. The inception of this hybrid ANN stems from the hypothesis that if the original image is passed in parallel along with the edge image, the neural network would be able to differentiate between the hand edges and the background edges, if any, because the original image features provide additional information to the ANN to capture the location of the both the background and the hand region.

Starting with the ISL digits[23], the structure of the upper and lower fully-connected blocks are showcased in Table 13.

Table 13. Hybrid ANN architecture – ISL digits

Upper fully-connected block		
Operation	**Nodes**	**Activation**
Dense	6	ReLU
Dense	6	ReLU
Dense	No. of CNN features after PCA	Linear
Lower fully-connected block		

Operation	Nodes	Activation
Dense	6	ReLU
Dense	6	ReLU
Dense	10	Softmax

For the Cambridge hand gesture dataset[20], ASL alphabet dataset[19] and ASL fingerspell dataset[21], the number of nodes are proliferated to detect the image variations in terms of intra-class differences and background intricacies & user specificity respectively with the description of the types of fully-connected blocks in Table 14.

Table 14. Hybrid ANN architecture – Cambridge hand gesture dataset, ASL alphabet and ASL fingerspell dataset

Upper fully-connected block		
Operation	**Nodes**	**Activation**
Dense	64	ReLU
Dense	64	ReLU
Dense	No. of CNN features after PCA	Linear
Lower fully-connected block		
Operation	**Nodes**	**Activation**
Dense	64	ReLU
Dense	64	ReLU
Dense	No. of classes	Softmax

Similar to the single ANN as described in the previous subsection of neural networks, the proposed hybrid ANNs also utilizes 'he_uniform' as an initialization scheme.

- **Transfer Learning**

It is the process of reusing a pre-trained model for the task under consideration, in this case, sign gesture recognition. In the context of neural networks, it involves making use of pre-trained weights and adding or removing extra layers for the problem statement. Transfer learning is utilized as a baseline model for the datasets to check how well the proposed method fares in comparison. To this end, VGGNet 16 architecture is used, which is shown in Figure 14.

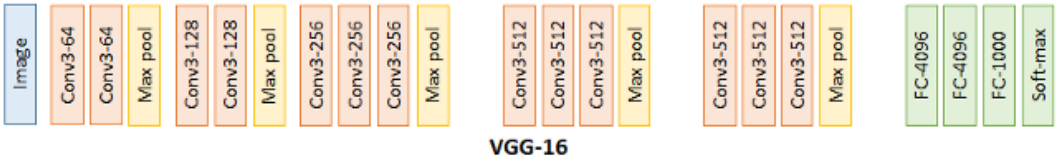


Figure 14. VGGNet 16 architecture

For the baseline experiment, the output of the fully-connected block is flattened followed by the addition of a Dense layer of 32 units and a Dense layer for softmax classification. The number of units in the classification layer is changed in accordance with the number of classes in the dataset.

- **Convolutional Neural Network (CNN)**
  One major drawback of transfer learning is that it is computationally intensive. The deep architecture although brings the objective of maximum accuracy to fruition with necessary tuning, it does extracts a heavy toll on the training system even for 15 epochs or so. Therefore, an attempt is made to obtain comparable classification results by using a standard CNN. As can be seen in Figure 14, VGGNet comprises of 5 convolutional blocks, each of which is made up of convolution operations and pooling operations. In the proposed method, CNNs are built for two datasets NUS I[22] and NUS II[22] which are complex in terms of background variations, image transformations such as zooming and minimal seperability between classes. These CNNs are designed with lesser convolutional blocks and their details are mentioned in Tables 15 and 16.

Table 15. CNN Architecture – NUS I dataset

Operation	Kernel	Stride	Filters/Nodes	Pool	Activation
Conv2D	3x3	1x1	32		ReLU
MaxPooling2D		2x2		2x2	
Conv2D	3x3	1x1	32		ReLU
MaxPooling2D		2x2		2x2	
Conv2D	3x3	1x1	32		ReLU
MaxPooling2D		2x2		2x2	
Flatten					
Dense			32		ReLU
Dense			10		Softmax

Table 16. CNN Architecture – NUS II dataset

Operation	Kernel	Stride	Filters/Nodes	Pool	Activation
Conv2D	3x3	1x1	32		ReLU
MaxPooling2D		2x2		2x2	
Conv2D	3x3	1x1	32		ReLU
MaxPooling2D		2x2		2x2	
Flatten					
Dense			128		ReLU
Dense			10		Softmax

In both tables 15 and 16, the ReLU layers are initialized using ‘he_uniform’ weight assignment scheme.

3.2 SIGN LANGUAGE PRODUCTION

3.2.1 DATASETS

For image generation, deep-convolutional GANs (DCGANs) are employed to produce sign gestures for 3 datasets, namely NUS I dataset[22], ISL digits[23] and a subset of RWTH_PHOENIX14T dataset[33]. Different variations of DCGAN are also tested with the aforementioned datasets to comparatively analyze the quality of the generated images as well as

the types of images generated. Table 17 provides a glimpse of the RWTH_PHOENIX14T dataset whereas Table 18 outlines the details of the subset of the aforementioned dataset used.

Table 17. Dataset – RWTH_PHOENIX14T

Table 18. Dataset details – RWTH_PHOENIX14T subset

Size	**No. of classes**	**Samples per class**
1107	10	>=40

3.2.2 SYSTEM ARCHITECTURE

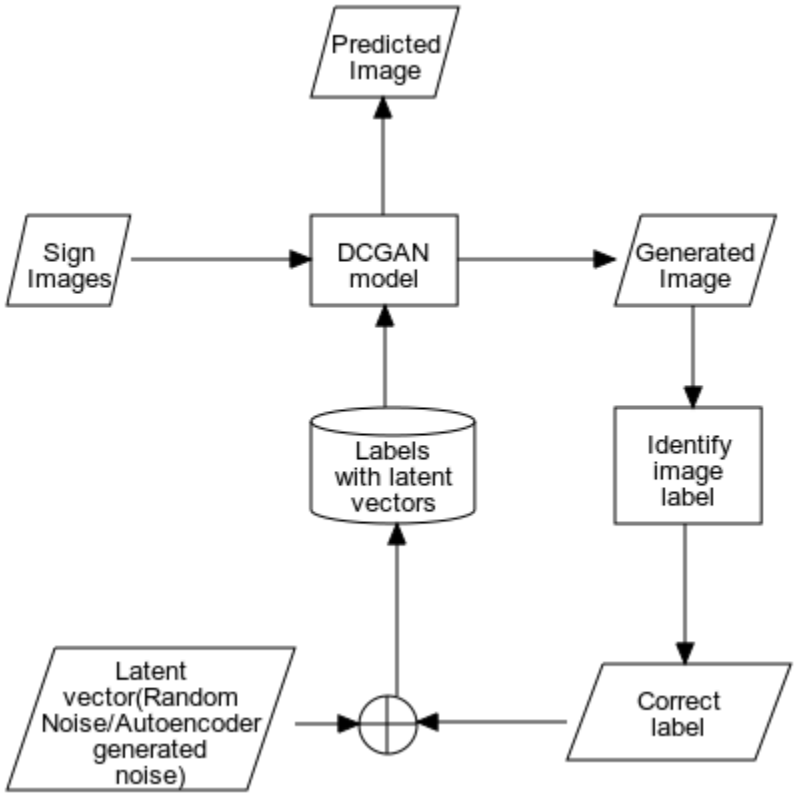


Figure 15. Architecture diagram for SLP

Figure 15 represents the operations of the SLP module. The input to the system are sign images – both real and generated, which means that at every training step the generated images are used by the DCGAN model to improve its performance. The sign images when fed to the system are in the form of a numpy array whose shape is as follows:

$$(n, h, w, c) \quad (15)$$

where,
n=no. of images
h=height of image
w=width of image
c=color channels of image
At the end of the training phase, one needs to identify the correct label of the images by visual inspection which demands a sufficient amount of domain knowledge. These labels are then stored alongside with the latent vectors that produced those labels. This stored data can be used later by the same trained DCGAN model to give interpretable sign images.

3.2.3 METHODOLOGY

3.2.3.1 LATENT VECTOR

It is the input upon which the generator portion of the DCGAN trains to produce images similar to the actual sign images in the dataset. In general, it is random noise with a Gaussian distribution which is mapped to different output images. These latent vectors, when trained over time represents spatial features of the generated images and hence are important in getting specific images as per the problem domain.

In the experiments performed on DCGANs, instead of providing random values, the generator is provided with latent vectors obtained from the dataset itself when passed through an autoencoder or more specifically, the encoder part of the autoencoder. The training loop for the same is shown in Appendix 8. This is similar to the method described in [34], but in this project, only one encoder is used since the goal is to produce more interpretable images with less noise, rather than conditionally editing the generated images as explained in [34]. The encoder used for obtaining the latent vectors is described in Table 19.

Table 19. Encoder for latent vectors

Operation	**Parameter**
InputLayer	Shape = (120,160,3)
Flatten	
Dense	Nodes = size of latent vector

3.2.3.2 GENERATIVE ADVERSARIAL NETWORK (GAN)

It is a neural network with two parts: a generator which generates images from a latent vector and a discriminator trained on both the dataset images as well as generated images trying to differentiate between them. The task of the generator in a GAN is to produce highly-realistic images that seem to be taken from the dataset itself such that the discriminator is unable to tell the difference whereas the discriminator needs to be smart enough to sieve out generated images. As a result of this condition, over time the generator is able to produce images with negligible disparity from the actual images. For SLP, in this methodology, the DCGAN architecture is described in Table 20.

Table 20. DCGAN Architecture

Generator						
Operation	**Kernel**	**Stride**	**Filter/ Nodes**	**Pool**	**Parameter**	**Activation**
Dense(noise_shape=100)			30x40x128			
Reshape					(30,40,128)	
LeakyReLU					alpha=0.2	
BatchNormalization					momentum=0.8	
Conv2DTranspose	3x3	2x2	64			
LeakyReLU					alpha=0.2	
BatchNormalization					momentum=0.8	
Conv2DTranspose	3x3	2x2	3			tanh
Discriminator						
Operation	**Kernel**	**Stride**	**Filter/ Nodes**	**Pool**	**Parameter**	**Activation**
Conv2D(image_shape=120,160,3)	3x3	2x2	64			
LeakyReLU					alpha=0.2	
Conv2D	3x3	2x2	128			
LeakyReLU					alpha=0.2	
Flatten						
Dense			100			
LeakyReLU					alpha=0.2	
Dense			1			sigmoid

As can be observed from Table 20, DCGANs use strided convolutions instead of pooling layers so that both the individual networks in the DCGAN learn to upsample or downsample by itself. This ensures better training of the DCGAN model. The use of LeakyReLU prevents the network from entering the dying state, where the vanishing gradient problem doesn't allow the generator to learn. Batch Normalization stabilizes the learning process of the generator. All these factors are in accordance to the design rules specified in [16]. The sigmoid activation function in the last layer of the discriminator allows to give a binary output – real image or fake image, whereas the tanh activation function in the last layer of the generator scales the output image in the range -1 to -1 which is the original scaling of the input images' pixels. When combined with the latent vectors from the encoder, the resultant DCGAN model can be visualized in Figure 16.

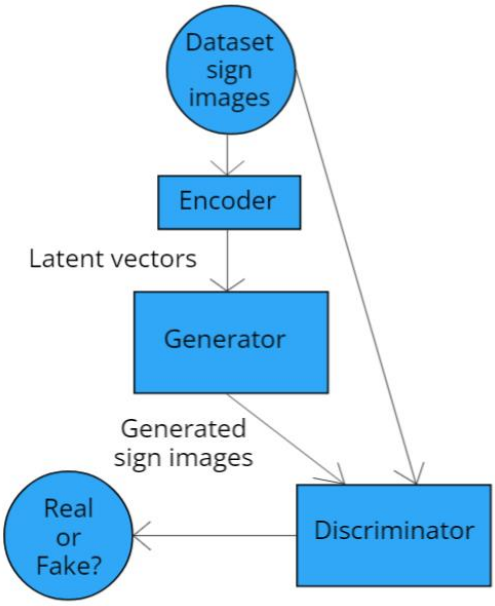


Figure 16. DCGAN with autoencoder noise

One major drawback of GANs however, is that it suffers from the problem of mode collapse in which the model is unable to generate all the different classes of images from the dataset because the generator aims at fooling the discriminator hence generating the same images again and again. This can be alleviated with the help of a Wasserstein GAN[35] or WGAN which uses the Wasserstein loss function described as follows:

$$Discriminator\ loss = D(x) - D(G(z)) \tag{16}$$

where,
D(x) = output on real images
D(G(z)) = output on fake images

The discriminator's task is to maximize eq. 16 so that the generator don't produce the same image over and over but try new images as well. To incorporate eq. 16 the following changes need to be made:

- Instead of sigmoid activation, make the last layer of the discriminator be a linear activation i.e. the default activation, in order to output image scores which quantify the realness (D(x)) which needs to be maximized or fakeness (D(G(z)) of the image which needs to be minimum.
- Assign label '1' for real images and label '-1' for generated images so that the network outputs larger scores for real images and smaller scores for generated/fake images in accordance to the following implementation of Wasserstein loss[36] in Keras:

```
def wasserstein_loss(y_actual,y_pred):
  return backend.mean(y_actual*y_pred)
```

In the above snippet, if *y_actual* i.e 1 for real images is multiplied by *y*_pred, its gives a

higher value whereas -1 for fake images when multipled by *y_pred* gives a lower value. The average of this product is used to train the WGAN model.

- Constrain or clip the discriminator weights in order to enforce Lipschitz continuity[37] i.e. maximize the Wasserstein loss within a particular limit so that it learns its constrained space till optimality which is why the discriminator is trained more than the generator.
- Use RMSProp instead of Adam optimizer to stabilize the training within this constrained space.

However, clipping weights diminishes the potential of the GAN model to learn complex functions necessary for image generation. This can be solved by applying gradient penalty(GP)[38] to WGAN. An optimal discriminator of a WGAN is meant to have a normalized gradient value of 1 or put simply a straight line between real and generated images with respect to any interpolated sample between these distributions. GP penalizes the gradient with a coefficient λ whenever the discriminator strays from optimality. This ensures a much better mapping between real and fake images along with the Wasserstein loss ensuring that the generator tries to match these distributions. In the experiments performed for SLP, the Keras implementation of GP is utilized as mentioned in [39] with the GP loss function as follows:

```
def gradient_penalty_loss(y_actual,y_pred,averaged_samples,gp_weight):
  gradients=backend.gradients(y_pred,averaged_samples)[0]
  gradients_sqr=backend.square(gradients)
  gradients_sqr_sum=backend.sum(gradients_sqr,axis=np.arange(1,len(gradients_sqr.shape)))
  gradients_l2_norm=backend.sqrt(gradients_sqr_sum)
  gradient_penalty=gp_weight*backend.square(1-gradients_l2_norm)
  return backend.mean(gradient_penalty)
```

In the above snippet, first the gradients are calculated for the predicted output with respect to the interpolated samples followed by normalizing said gradients which is a simple root of sum of squared values. GP is then calculated depending on how much the normalized gradients deviates from 1.

Note: All the parameter values for the GAN experiments are provided in Appendix 9

Chapter 4

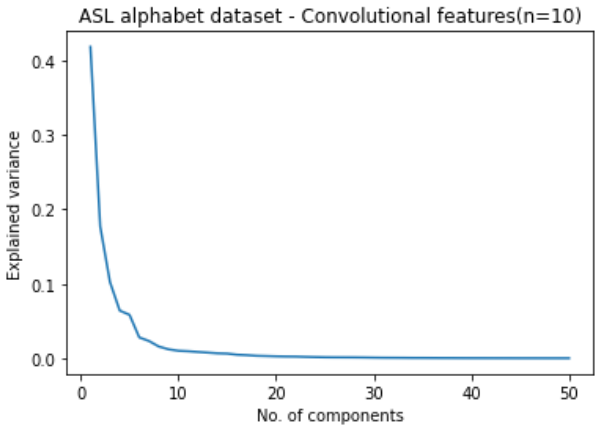
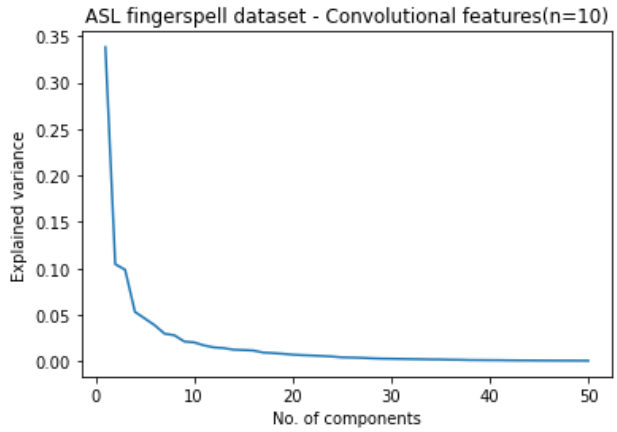
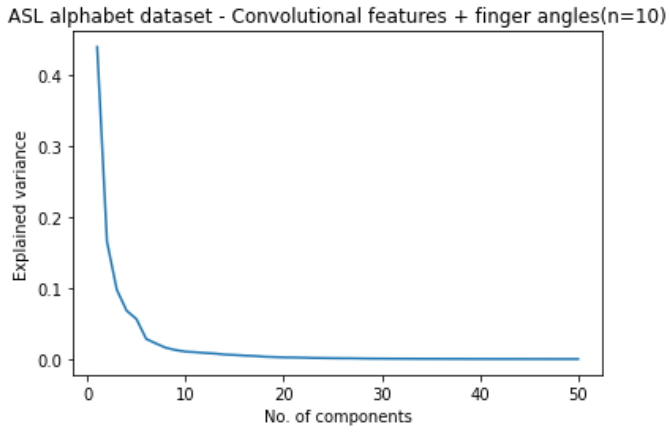
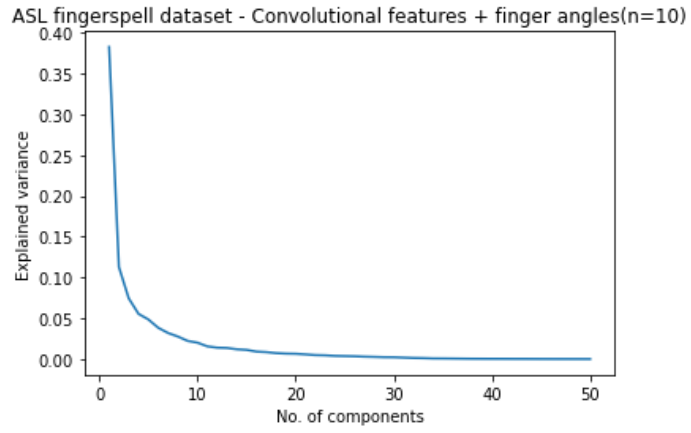
Results

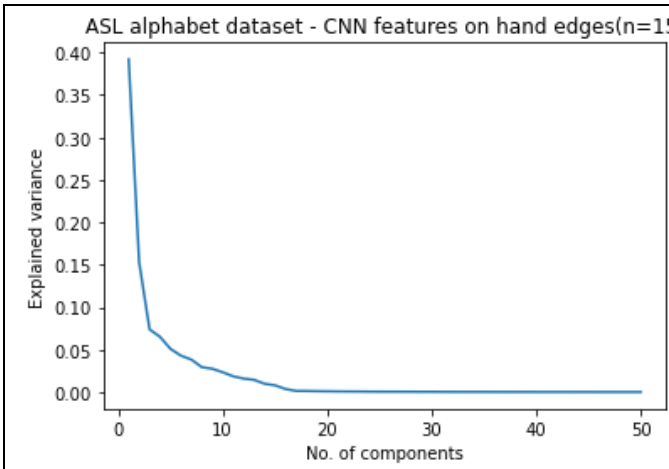
4.1 SIGN LANGUAGE RECOGNITION

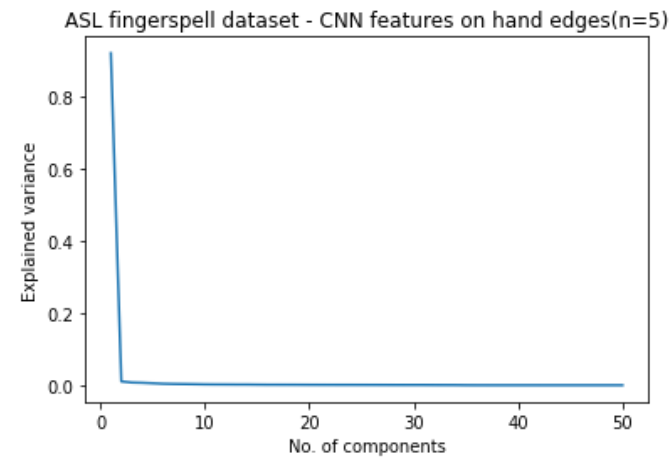
4.1.1 PRINCIPAL COMPONENT ANALYSIS (PCA)

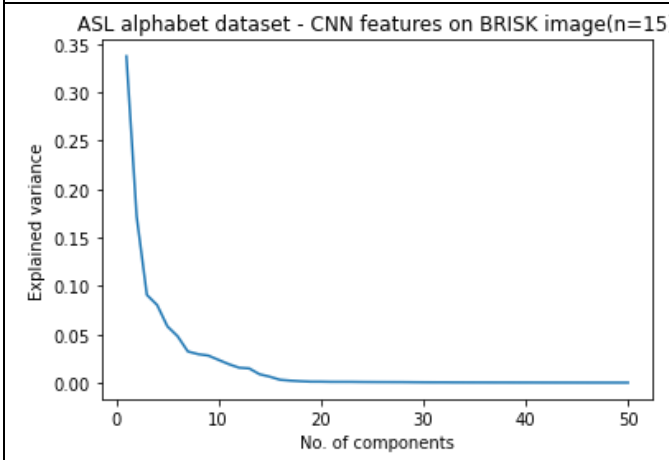
For a total of 6 datasets, PCA is applied on the 4 convolutional-related features to retain only the principal components of the extensive number of features. The ideal number of components is derived by observing the elbow of the scree graphs as shown in Table 21.

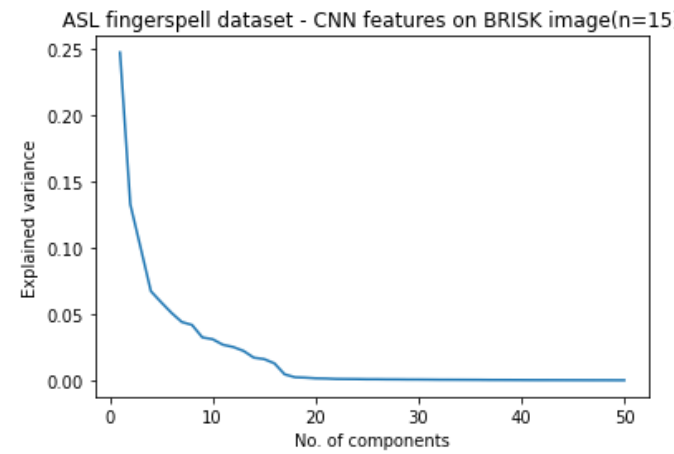
Table 21. Scree graphs for the datasets

ASL alphabet dataset	ASL fingerspell dataset
	
	

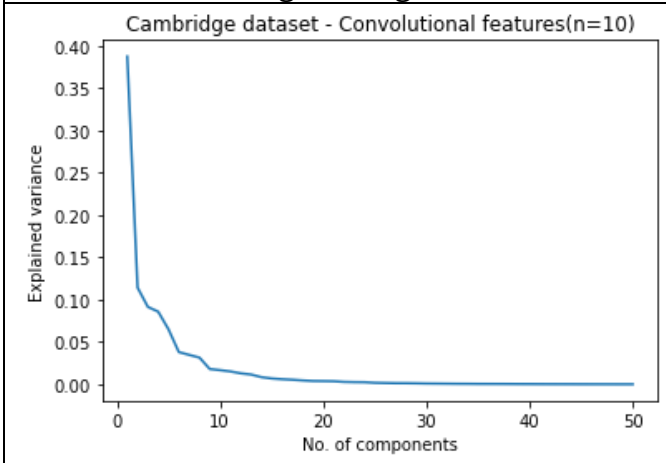




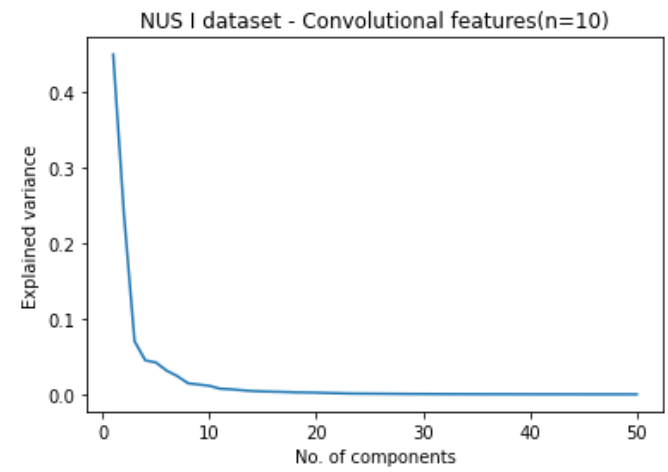


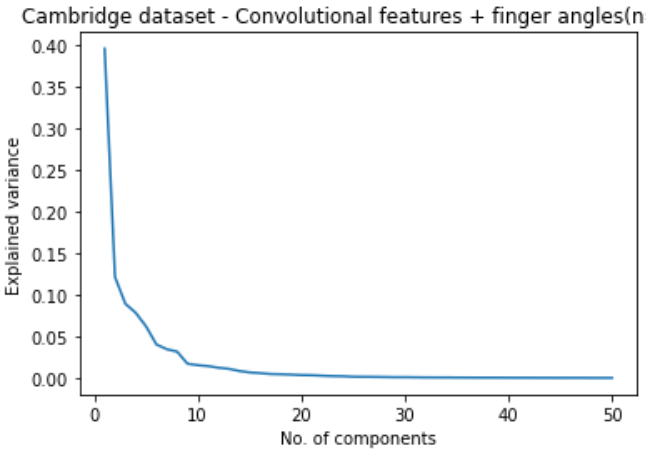


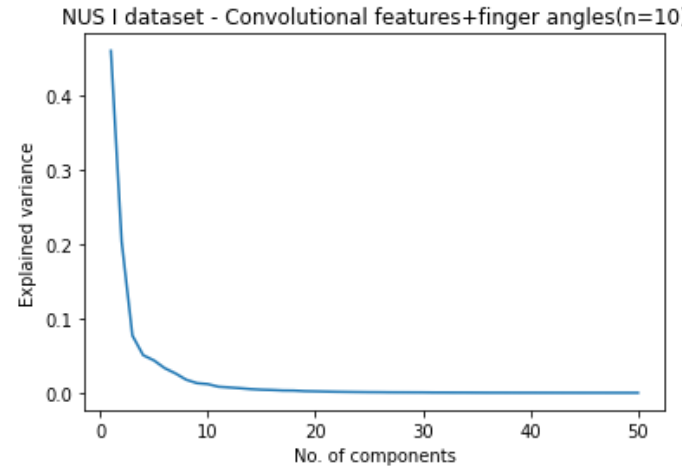
Cambridge hand gesture dataset

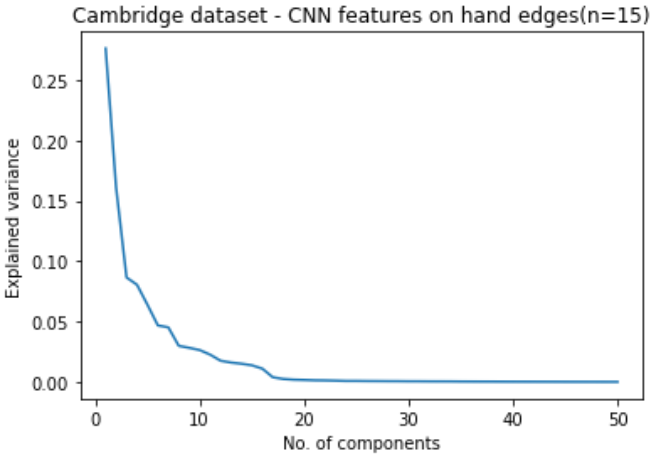


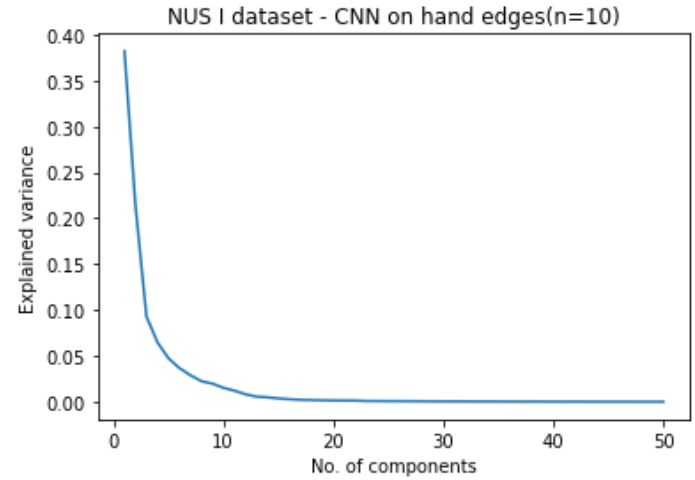
NUS I dataset

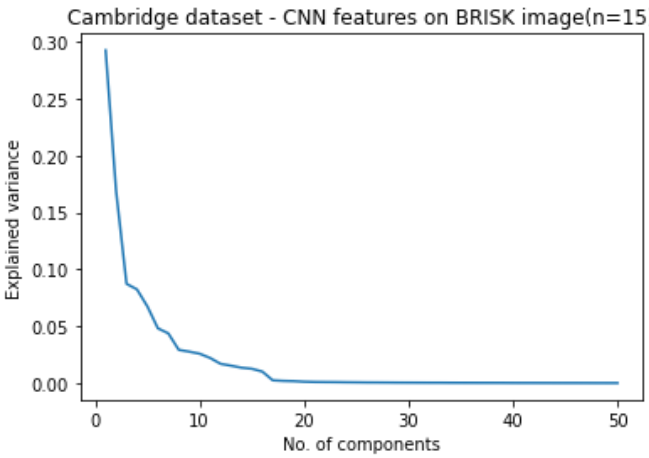


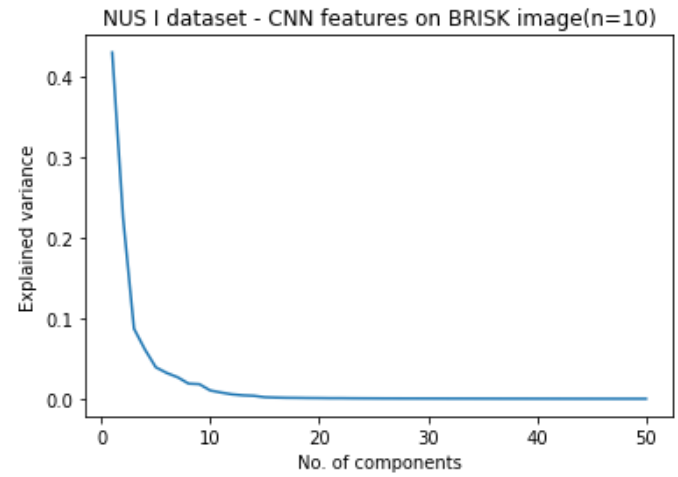




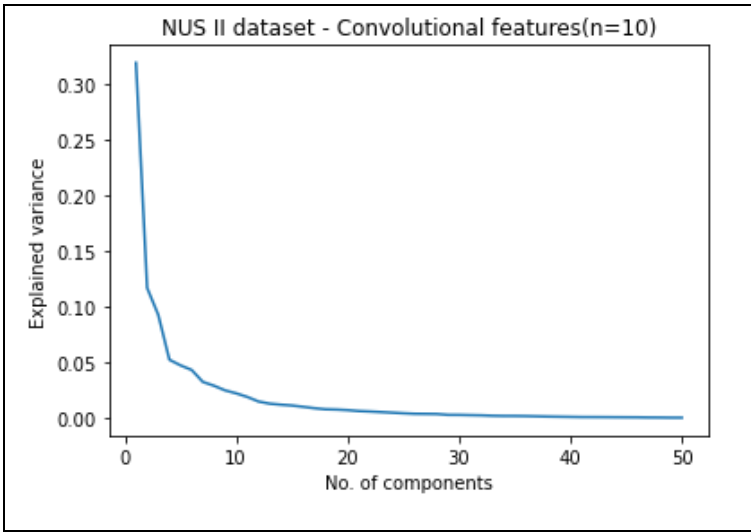


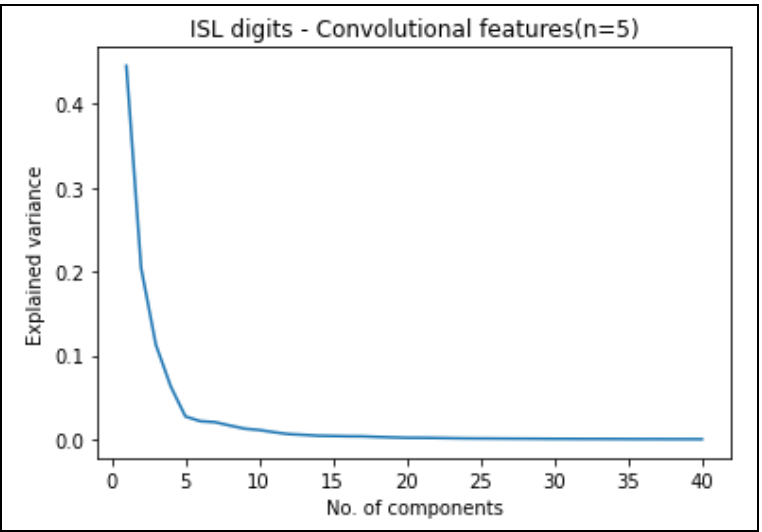


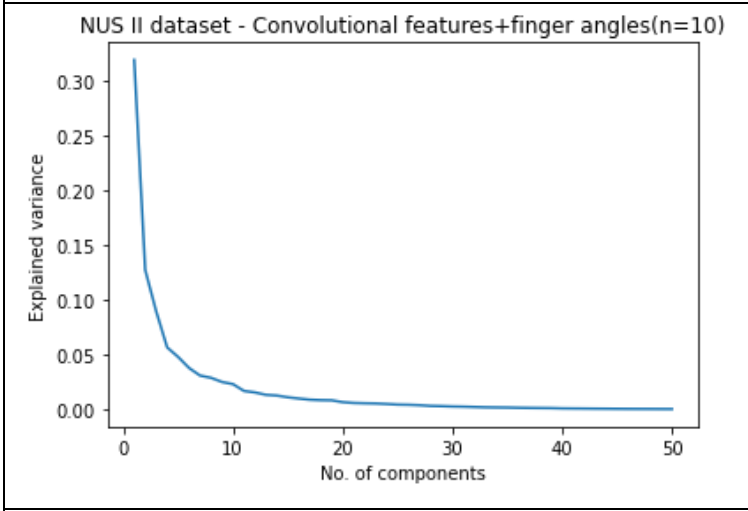


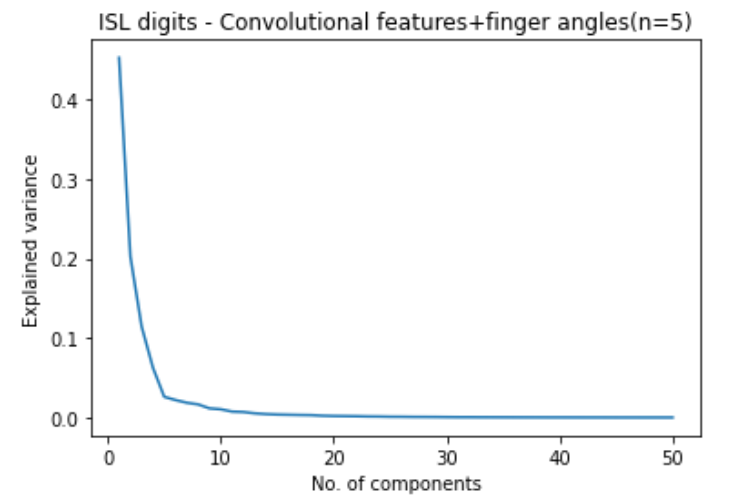


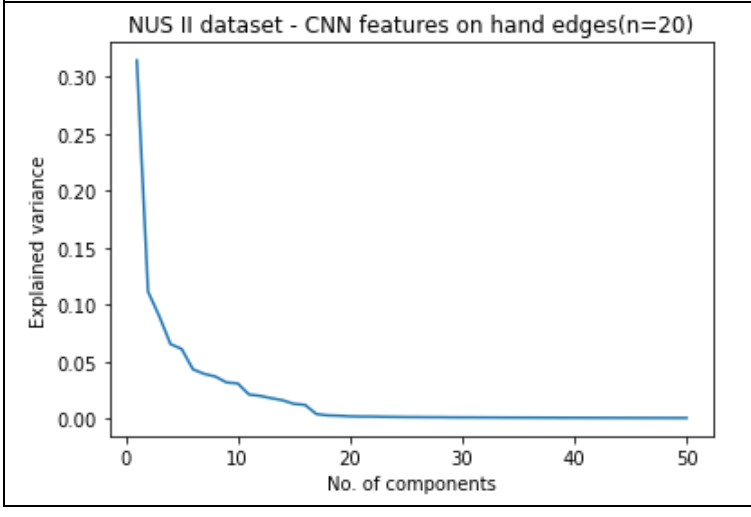
NUS II dataset	ISL digits

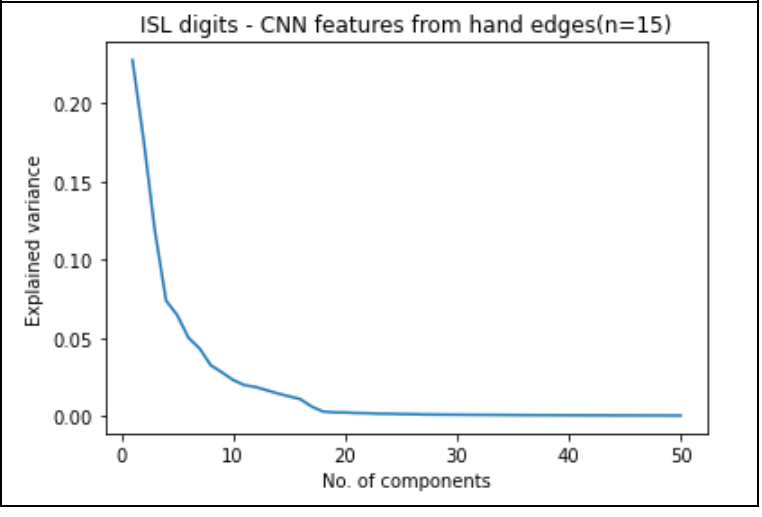

NUS II dataset - Convolutional features(n=10)
Explained variance
No. of components

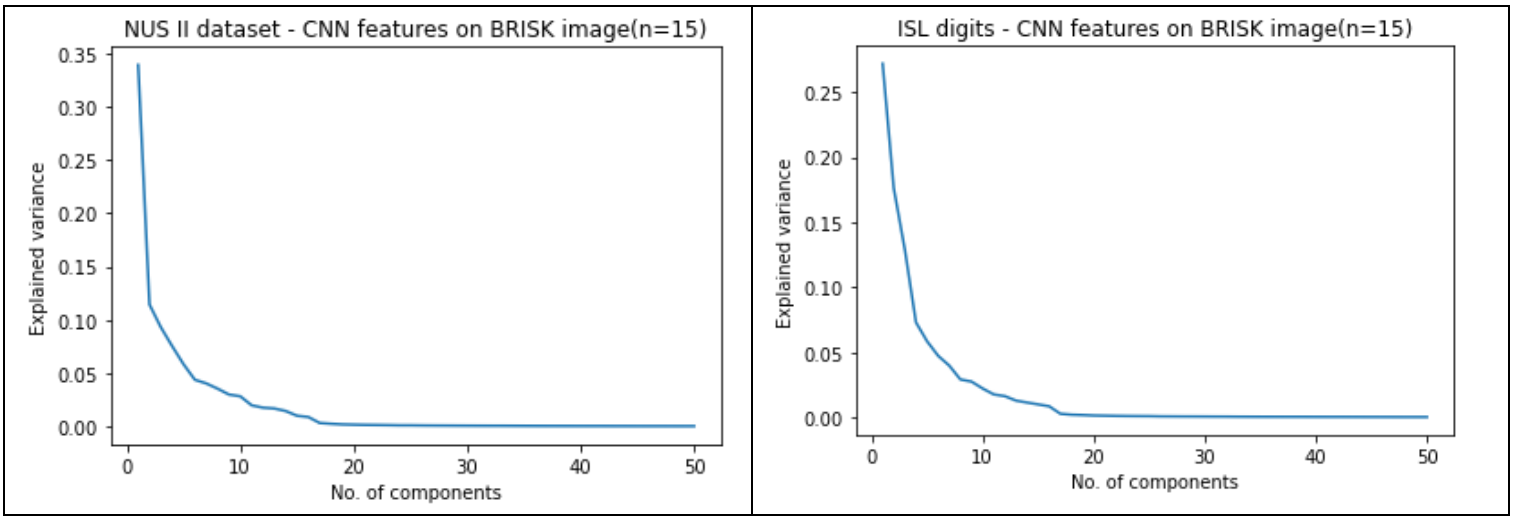

ISL digits - Convolutional features(n=5)
Explained variance
No. of components


NUS II dataset - Convolutional features+finger angles(n=10)
Explained variance
No. of components


ISL digits - Convolutional features+finger angles(n=5)
Explained variance
No. of components


NUS II dataset - CNN features on hand edges(n=20)
Explained variance
No. of components


ISL digits - CNN features from hand edges(n=15)
Explained variance
No. of components



It can be observed from Table 21 that the minimum no. of ideal principal components is 5 whereas the maximum is 20. In the majority of scree plots, multiple elbows are observed. In such a scenario, an approximate value is taken as the ideal number for experimentation with classification models.

4.1.2 SIGN RECOGNITION ACCURACIES

The metric that is used to quantify the performance of the SLR system is classification accuracy which is defined by the following equation:

$$accuracy = \frac{no.of\ sign\ gestures\ correctly\ identified}{total\ no.\ of\ sign\ gestures} \tag{17}$$

4.1.2.1 ENSEMBLE METHODS

Two ensemble techniques namely Random Forest and XGBoost is applied on the features extracted in section 3.1.3.1 with their intializations using sklearn as follows:

```
clf = RandomForestClassifier(n_estimators = 100,random_state=5)
clf = xgb.XGBClassifier(random_state=5)
```

Here, *n_estimators* indicate the number of trees used in Random Forest.

- **Ideal no. of PCA components**

  Table 22 highlights the accuracies obtained when the ensemble methods are used with the non-convolutional related features and convolutional related features using ideal no. of PCA components as described in Table 21.

  Table 22. Ensemble accuracies for ideal no. of PCA components in convolutional related features, and non-convolutional related features

	ASL alphabet	**ASL fingerspell dataset**	**Cambridge hand gesture dataset**	**NUS I dataset**	**NUS II dataset**	**ISL digits**

Hand coordinates + Random Forest	98.063%	98.967%	98.626%	80.000%	97.200%	100.000%
Hand coordinates + XGBoost	98.647%	99.136%	98.789%	80.000%	97.200%	99.602%
Convolutional Features + PCA + Random Forest	97.188%	86.811%	99.549%	52.000%	54.800%	98.000%
Convolutional Features + PCA + XGBoost	93.847%	79.818%	96.669%	52.000%	44.400%	96.000%
Convolutional Features + Finger Angles + PCA + Random Forest	96.806%	85.517%	99.243%	52.000%	52.400%	96.016%
Convolutional Features + Finger Angles + PCA + XGBoost	93.901%	77.913%	96.711%	32.000%	42.800%	95.219%
CNN features on hand edges + PCA + Random Forest	89.931%	39.111%	99.565%	48.000%	62.000%	98.800%
CNN features on hand edges + PCA + XGBoost	83.569%	37.157%	99.391%	48.000%	49.600%	98.800%
L.C. of BRISK descriptors on hand edges + Random Forest	36.056%	24.386%	52.105%	68.000%	20.400%	71.200%
L.C. of BRISK descriptors on hand edges + XGBoost	37.757%	27.761%	54.621%	60.000%	16.000%	70.000%
CNN features on BRISK + PCA + Random Forest	88.674%	79.506%	98.093%	56.000%	49.200%	96.400%

CNN features on BRISK + PCA + XGBoost	81.014%	74.170%	92.807%	60.000%	38.800%	96.000%

From Table 22, it is evident that 'Hand coordinates' outperform on all datasets when trained on ensemble models compared to other features since they capture specific positions of the hand region in 3D space such that the relative distance between each coordinate remains unchanged despite rotation or scaling. Hence, 2 signs of the same class in different conditions retain the same pattern in terms of coordinates. The only exception is the Cambridge hand gesture dataset because it comprises of sequential data which means that the coordinates change within the same class.

Secondly, for convolutional-related features, XGBoost performs less than or equal to Random Forest. This is because Random Forest uses the Bagging technique to train multiple subsets of the dataset on various decision trees to improve overall accuracy whereas, XGBoost uses the Boosting technique to iteratively improve prediction for misclassified instances but since PCA is performed on the convolutional-related features, there seems to be a loss of information for XGBoost to make any further improvement. This doesn't seem to be true for NUS I dataset using 'CNN features on BRISK' for XGBoost where the no. of principal components seem to be sufficient enough to rectify itself on the misclassified sign gestures and consequently surpass Random Forest.

Next, 'L.C. of BRISK descriptors' gives the least accuracy out of all features for all datasets which means taking a linear combination of all the keypoints seem to provide wrong weightage to the important keypoints which accurately represent the sign image. The NUS I dataset is an exception however because the rotation-invariance characteristic of BRISK is able to account for the movement in hand positions and the varying depths of the hand region prevalent in the dataset. 'CNN features on BRISK', on the other hand, improves upon the performance of a linear combination by considering all keypoints equally when performing convolution operations with the exception of NUS I dataset where applying PCA seems to be a leading to a loss of knowledge.

Moving on to the next inference, 'CNN features on hand edges' gives better accuracy compared to 'Convolutional features' alone since the system has segmented the hand region and retained only the outline in the form of edges. This has made the resultant image free from any background complexity and hence easier to learn by the ensemble models. However, it does has the following exceptions – ASL fingerspell dataset (due to retention of background edges), ASL alphabet dataset (due to deletion of hand edges falling outside of the skin mask) and NUS I dataset (due to deletion of information required for inter-class separation for similar images).

Last but not the least, with the exception of 'L.C. of BRISK' feature, the proposed features perform well on datasets where the hand gestures are uniform, the classes are easily distinguishable and there is no background complexity i.e. the ASL alphabet, Cambridge dataset and ISL digits. For the remaining datasets, the ensemble models are unable to learn

complex non-linear patterns in data and hence give lower accuracies.

- **Increased no. of PCA components**
  PCA is applied on the convolutional-related features that are giving less accuracies in Table 22 with increased no. of principal components so that more information is preserved for better inter-class segregation. Table 23 displays the accuracies thus obtained with the no. of components mentioned alongside the features.

  Table 23. Ensemble accuracies on convolutional-related features with increased no. of PCA components

ASL fingerspell dataset							
Convolutional features + PCA + Random Forest (n=60)	**Convolutional features + PCA + XGBoost (n=60)**	**Convolutional features + finger angles + PCA + Random Forest (n=60)**	**Convolutional features + finger angles + PCA + XGBoost (n=60)**	**CNN features on hand edges + PCA + Random Forest (n=60)**	**CNN features on hand edges + PCA + XGBoost (n=60)**	**CNN features on BRISK + PCA + Random Forest (n=60)**	**CNN features on BRISK + PCA + XGBoost (n=60)**
96.663%	96.556%	96.391%	96.121%	90.901%	90.422%	78.655%	78.449%
NUS I dataset							
Convolutional features + PCA + Random Forest (n=20)	**Convolutional features + PCA + XGBoost (n=20)**	**Convolutional features + finger angles + PCA + Random Forest (n=20)**	**Convolutional features + finger angles + PCA + XGBoost (n=20)**	**CNN features on hand edges + PCA + Random Forest (n=20)**	**CNN features on hand edges + PCA + XGBoost (n=20)**	**CNN features on BRISK + PCA + Random Forest (n=20)**	**CNN features on BRISK + PCA + XGBoost (n=20)**
56.000%	52.000%	60.000%	52.000%	52.000%	56.000%	64.000%	60.000%
NUS II dataset							
Convolutional features + PCA + Random Forest (n=50)	**Convolutional features + PCA + XGBoost (n=50)**	**Convolutional features + finger angles + PCA + Random Forest (n=40)**	**Convolutional features + finger angles + PCA + XGBoost (n=40)**	**CNN features on hand edges + PCA + Random Forest (n=50)**	**CNN features on hand edges + PCA + XGBoost (n=50)**	**CNN features on BRISK + PCA + Random Forest (n=50)**	**CNN features on BRISK + PCA + XGBoost (n=50)**
68.000%	68.800%	71.200%	64.000%	65.600%	66.400%	40.800%	40.400%
ASL alphabet							

CNN features on hand edges + PCA + Random Forest (n=60)	CNN features on hand edges + PCA + XGBoost (n=60)	CNN features on BRISK + PCA + Random Forest (n=60)	CNN features on BRISK + PCA + XGBoost (n=60)
95.306%	93.729%	88.993%	86.229%

It can be observed from Table 23 that increasing the number of PCA components increases accuracy due to an increase in information which help to differentiate better between image classes with complex backgrounds and transformations. However, this is not true for 'CNN features on BRISK' when applied on ASL fingerspell dataset and NUS II dataset due to the Hughes phenomenon pertaining to the curse of dimensionality, which states that beyond the optimal no. of features, increasing the no. of components deteriorates the performance of classifiers.

Also increasing the no. of PCA components gives greater accuracy for 'Convolutional features + finger angles' compared to 'Convolutional features' since finger angles are meant to capture the relative positioning of the fingers with respect to similar signs. With more no. of retained components i.e. more information, they are able to compensate for the loss of accuracy in case of 'Convolutional features' alone. ASL fingerspell dataset, however deviates from this behavior since the finger angles for the same sign change for different users.

4.1.2.2 NEURAL NETWORKS

This section applies all the neural network architectures discussed in section 3.1.3.3.2 on the datasets to see how much accuracy is obtained on the testing data. In addition to that, two types of diagnostic plots – the accuracy curve (which ideally should increase with time) and the loss curve (which ideally should decrease with time), are also visualized to study the behavior of the neural models over the training epochs. One thing to note is that wherever the neural models are trained on the PCA-applied features, the number of components used is the one that gave maximum accuracy on that particular dataset from the experiments performed in section 4.1.2.1. All the neural networks are compiled using categorical cross entropy loss function and Adam optimizer.

- **Artificial Neural Networks (ANN)**

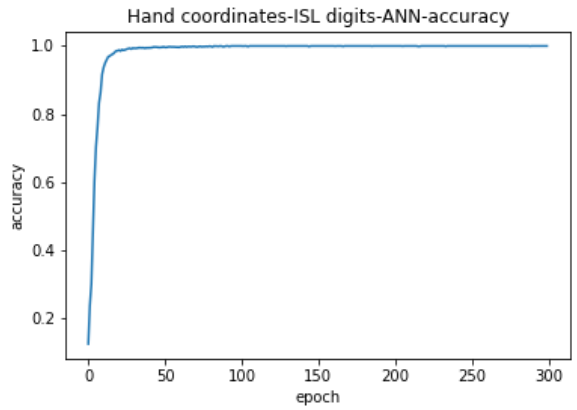
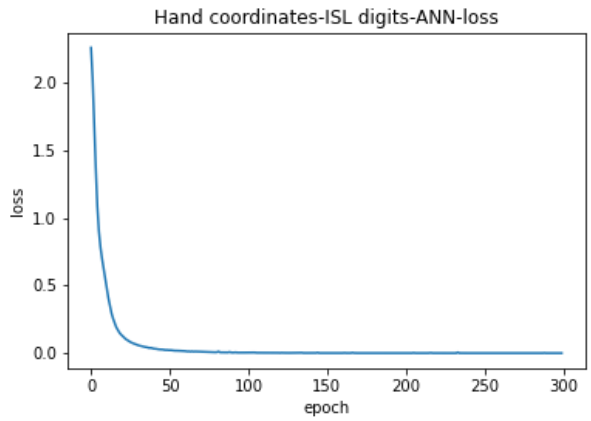
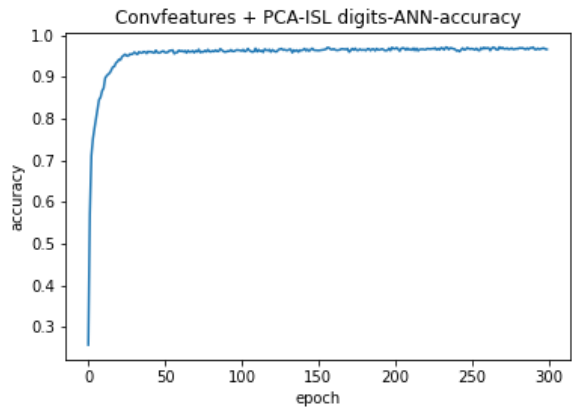
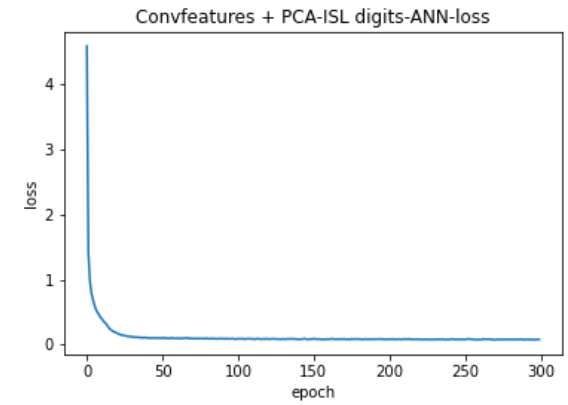
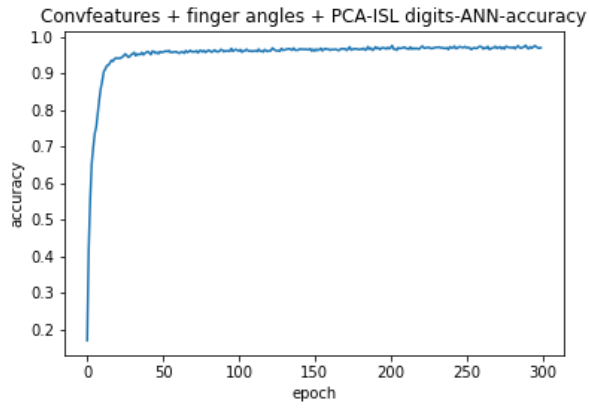
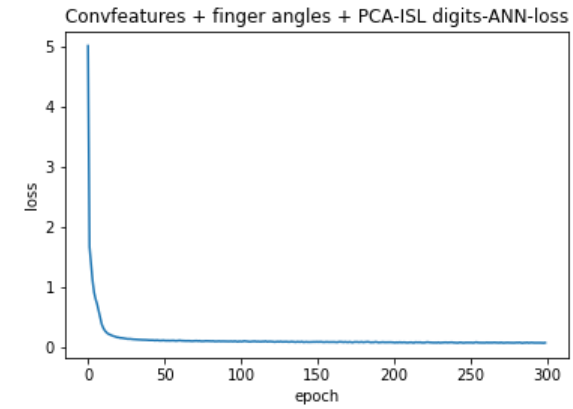
  The first experiment is performed with the ISL digits which is a relatively non-complex dataset for sign gestures devoid of any cluttered backdrop or image transformations. The ANN accuracies for this dataset with a batch size of 5 is shown in Table 24 and the corresponding diagnostic curves in Table 25.

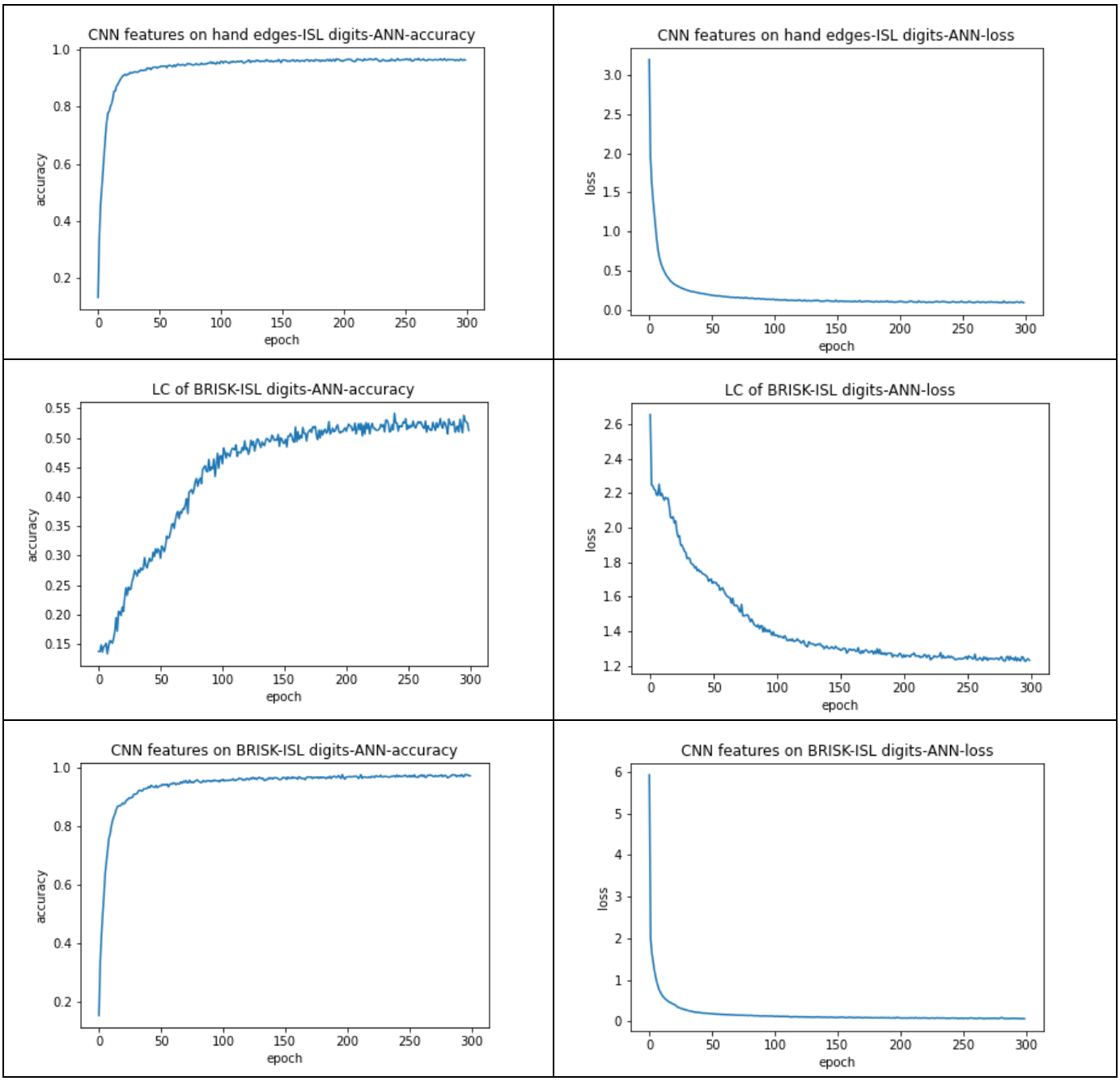
Table 24. ANN accuracies – ISL digits

Hand coordinates +ANN	Convolutional Features + PCA+ ANN	Convolutional Features + Finger angles + PCA + ANN	CNN features from hand edges + PCA + ANN	L.C. of BRISK descriptors on hand edges + ANN	CNN on BRISK image + PCA + ANN

100.000%	96.400%	94.024%	90.400%	46.800%	93.600%

Table 25. Diagnostic curves for ANN – ISL digits

Accuracy curve	Loss curve
	
	
	



It can be observed from Table 25 that after a certain number of epochs, both the accuracy and loss curves smooth out indicating that they have converged to an optima. The curves for 'L.C. of BRISK' however seems to be an anomaly since they exhibit regular spikes indicating the model is unable to reach an optimal value. In other words, 'L.C. of BRISK' is not a good representative feature.

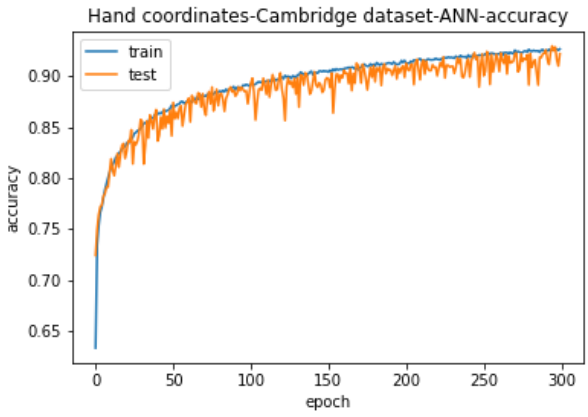
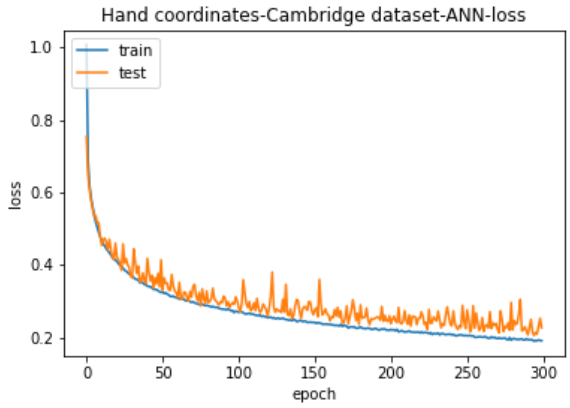
In the next experiment, ANN is applied on the Cambridge dataset, ASL alphabet and ASL fingerspell dataset with a batch size of 16. Since these datasets involve sequential images, different lighting conditions and complex backdrops respectively, the diagnostic curves for

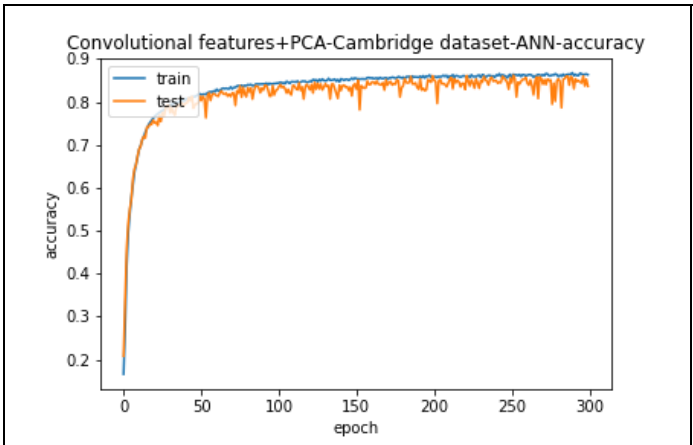
the test data is also plotted along with the training data as can be seen in Table 27 to see how well the ANN generalizes over time on these intricate datasets. When calculating the accuracies on the test data which are displayed in Table 26, only the best weights are considered over the entire training phase (Appendix 7).

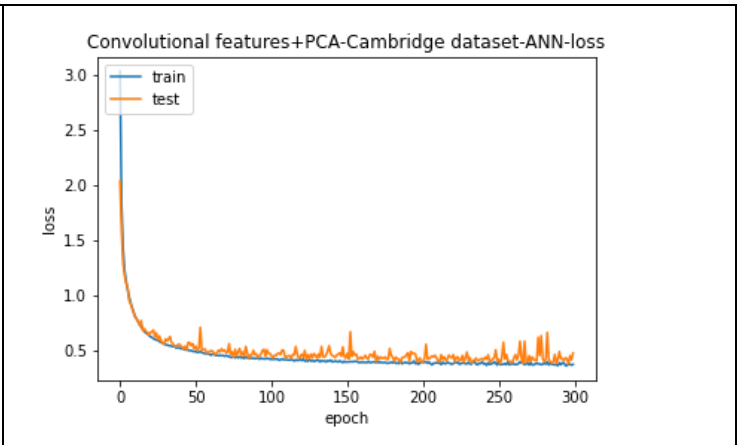
Table 26. ANN accuracies – Cambridge dataset, ASL alphabet, ASL fingerspell dataset

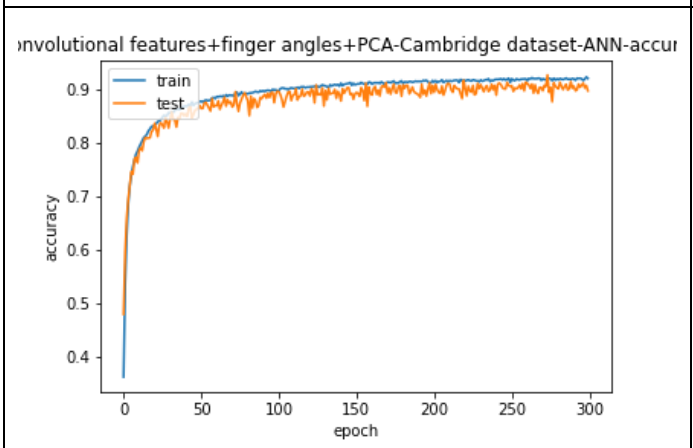
	Hand coordinates +ANN	**Convolutional Features + PCA+ ANN**	**Convolutional Features + Finger angles + PCA + ANN**	**CNN features from hand edges + PCA + ANN**	**L.C. of BRISK descriptors on hand edges + ANN**	**CNN on BRISK image + PCA + ANN**
Cambridge hand gesture dataset	92.178%	86.208%	92.838%	88.155%	42.467%	92.752%
ASL alphabet	98.992%	85.750%	86.233%	92.889%	25.965%	86.396%
	Convolutional features + PCA + ANN	**Convolutional Features + Finger angles + PCA + ANN**	**CNN features from hand edges + PCA + ANN**	**L.C. of BRISK descriptors on hand edges + ANN**	**CNN on BRISK image + PCA + ANN**	
ASL fingerspell dataset	96.192%	95.303%	88.712%	20.699%	78.586%	

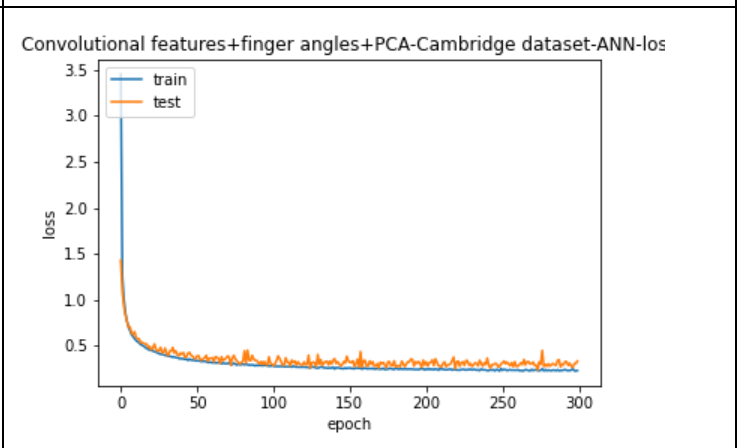
Table 27. Diagnostic curves for ANN – Cambridge dataset, ASL alphabet, ASL fingerspell dataset

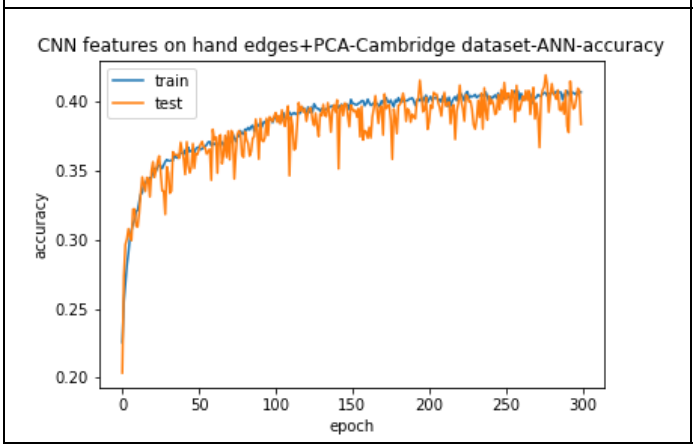
Accuracy curve	**Loss curve**
Cambridge hand gesture dataset	
	

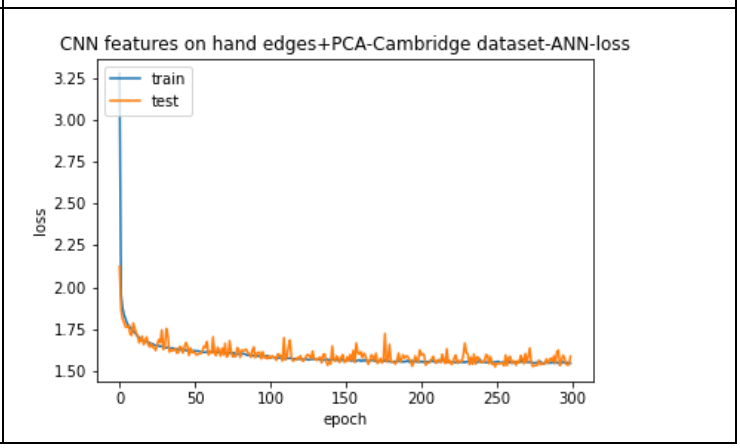

Convolutional features+PCA-Cambridge dataset-ANN-accuracy
train
test
accuracy
epoch

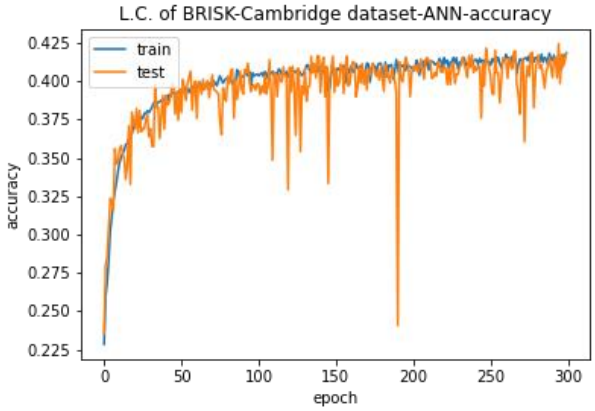

Convolutional features+PCA-Cambridge dataset-ANN-loss
train
test
loss
epoch

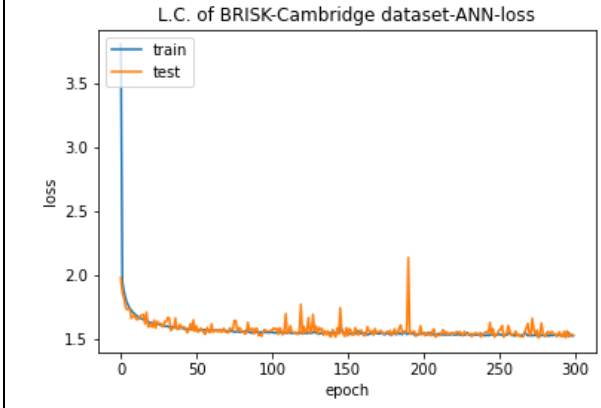

train
test
accuracy
epoch

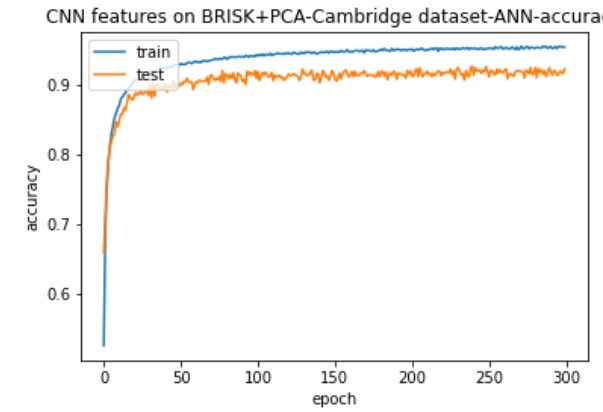

Convolutional features+finger angles+PCA-Cambridge dataset-ANN-los
train
test
loss
epoch

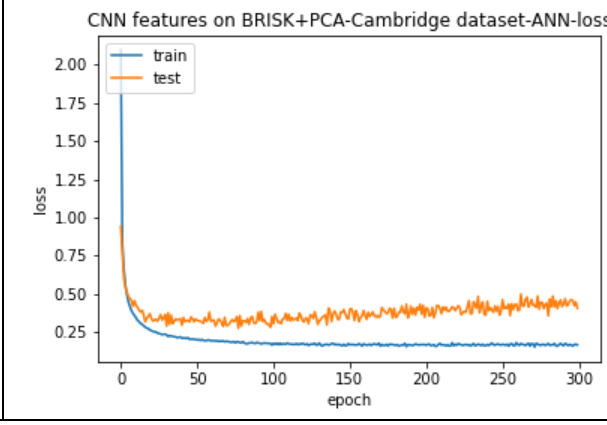

CNN features on hand edges+PCA-Cambridge dataset-ANN-accuracy
train
test
accuracy
epoch


CNN features on hand edges+PCA-Cambridge dataset-ANN-loss
train
test
loss
epoch

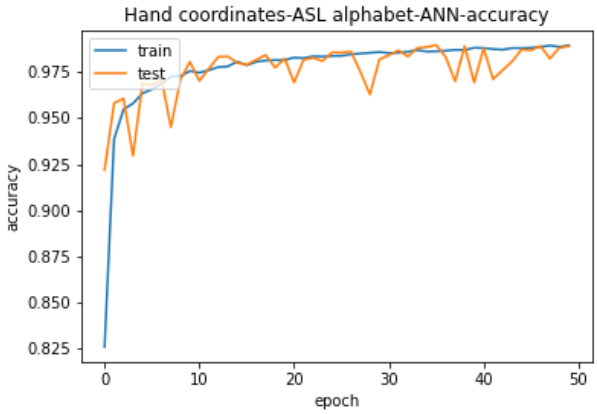


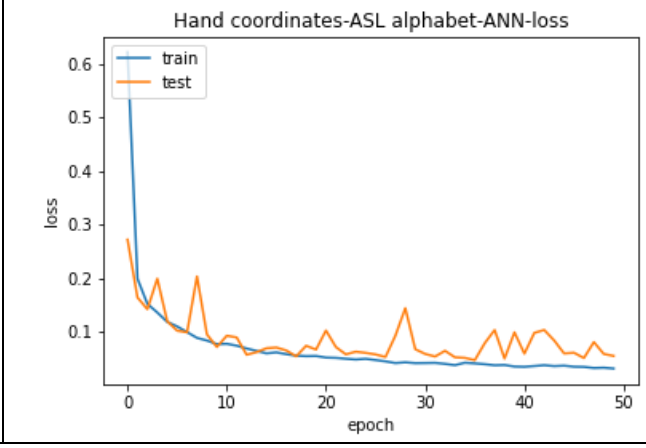


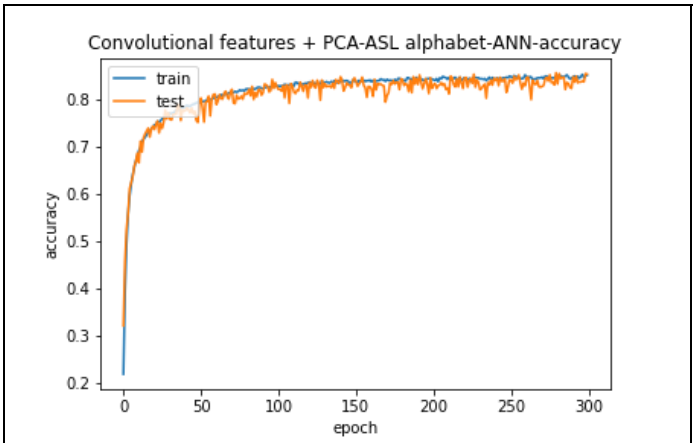


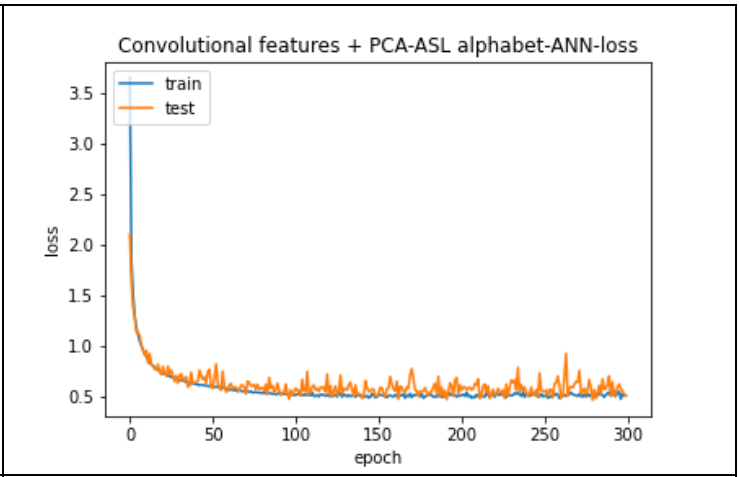


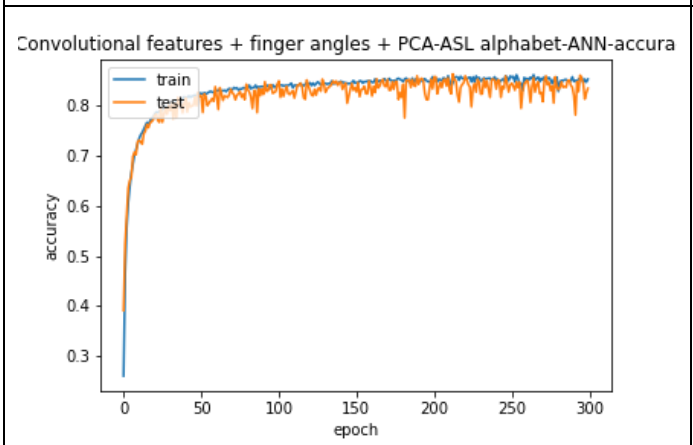
ASL alphabet

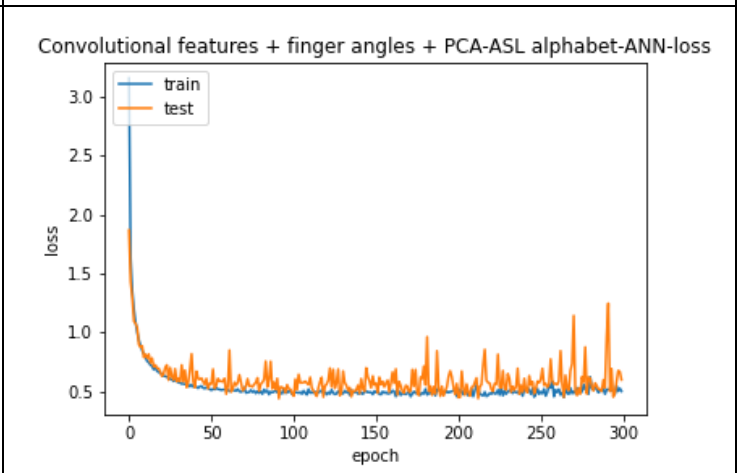


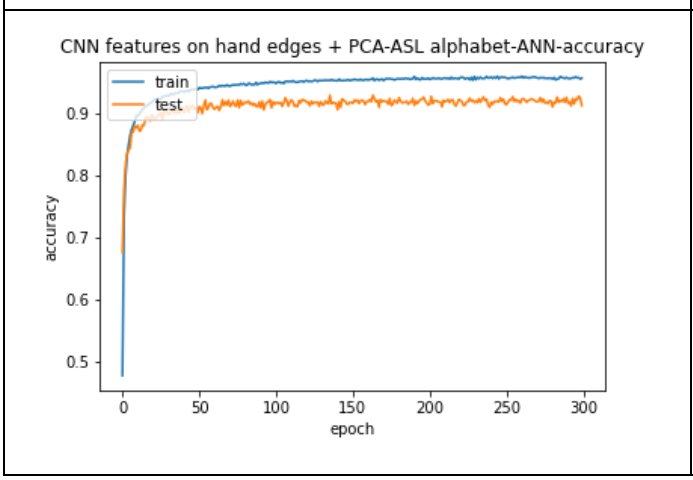


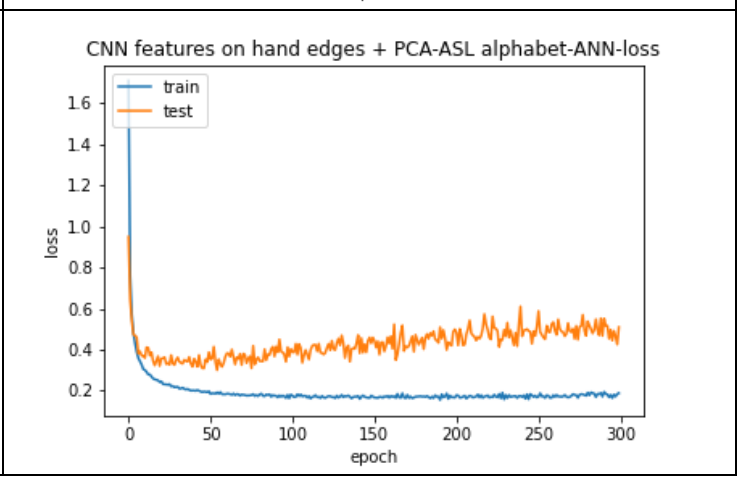

Convolutional features + PCA-ASL alphabet-ANN-accuracy
train
test
accuracy
epoch

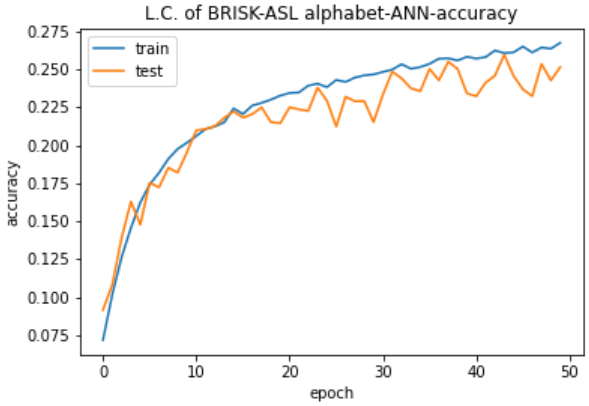

Convolutional features + PCA-ASL alphabet-ANN-loss
train
test
loss
epoch

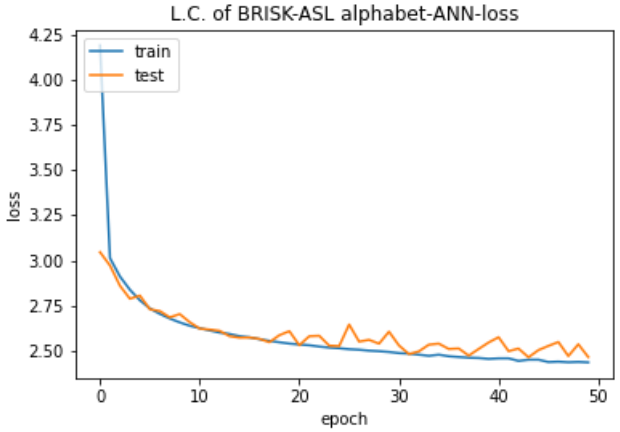

Convolutional features + finger angles + PCA-ASL alphabet-ANN-accura
train
test
accuracy
epoch

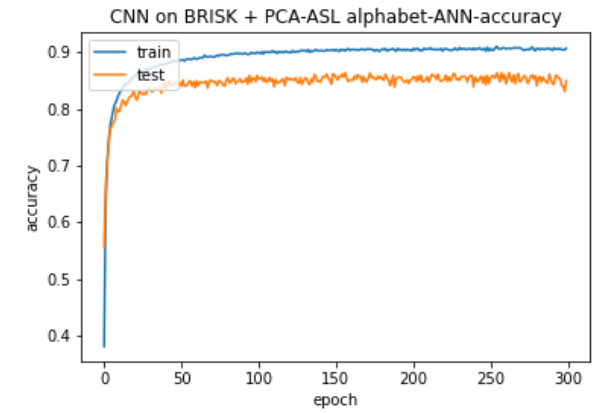

Convolutional features + finger angles + PCA-ASL alphabet-ANN-loss
train
test
loss
epoch

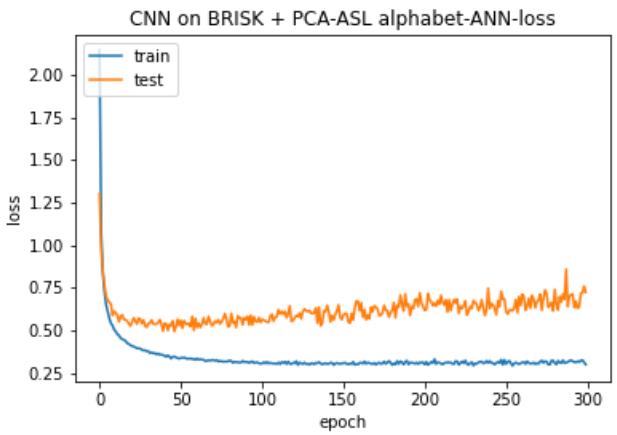

CNN features on hand edges + PCA-ASL alphabet-ANN-accuracy
train
test
accuracy
epoch


CNN features on hand edges + PCA-ASL alphabet-ANN-loss
train
test
loss
epoch

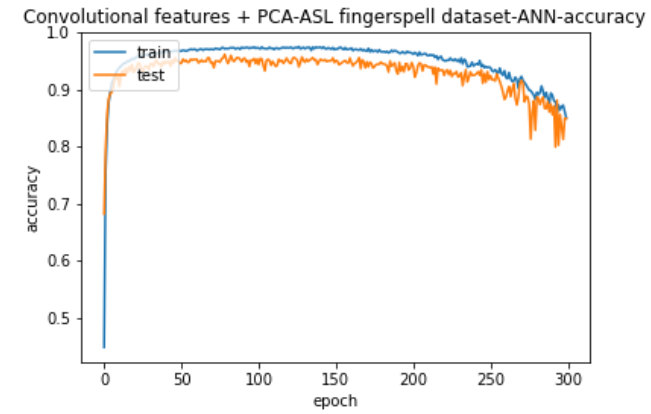


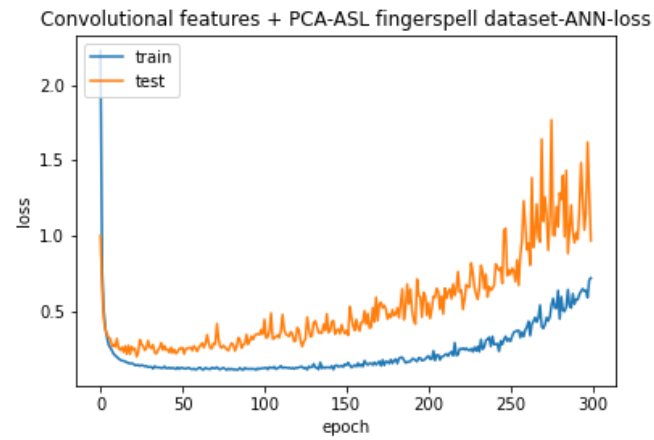


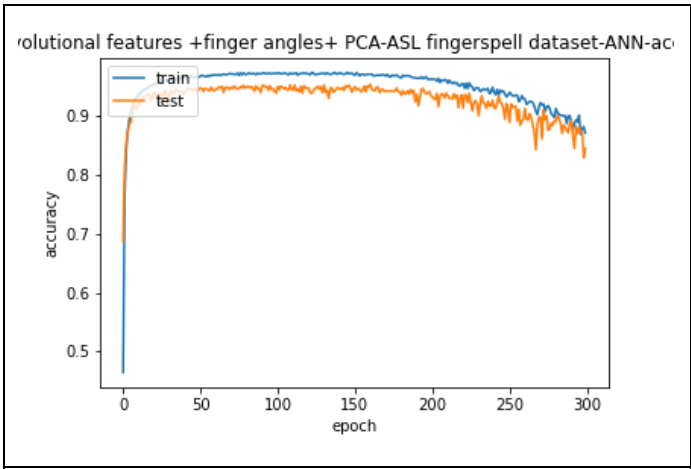


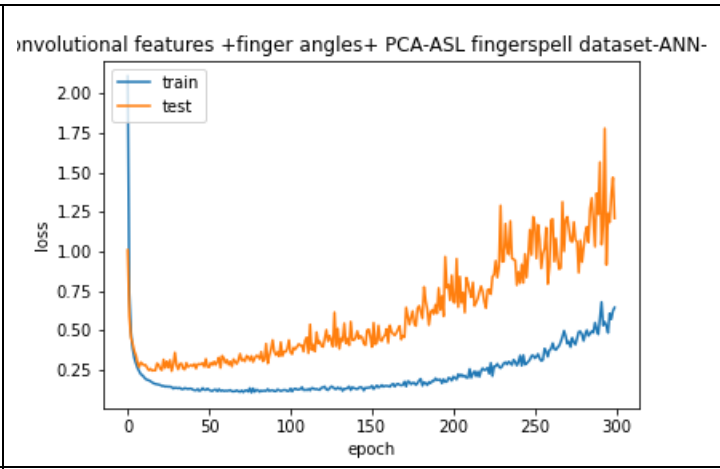


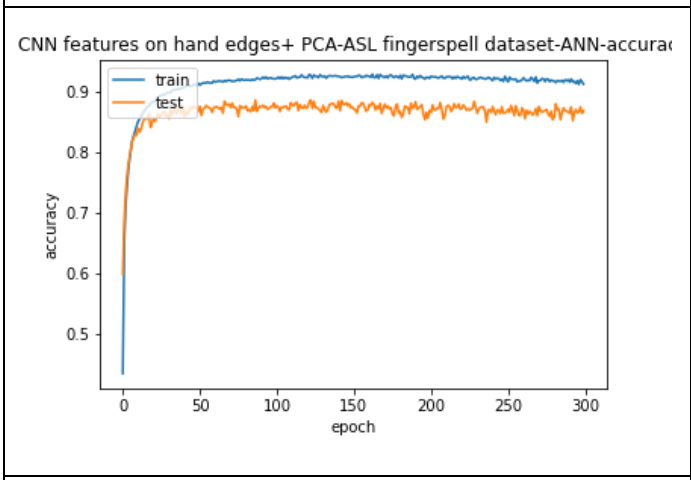
ASL fingerspell dataset

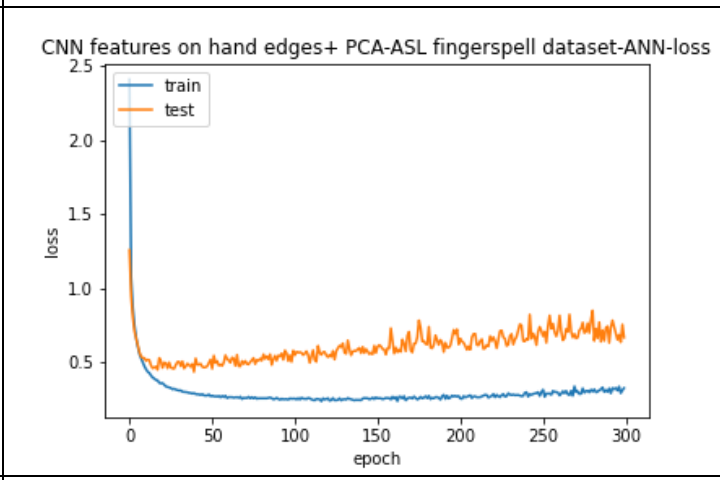


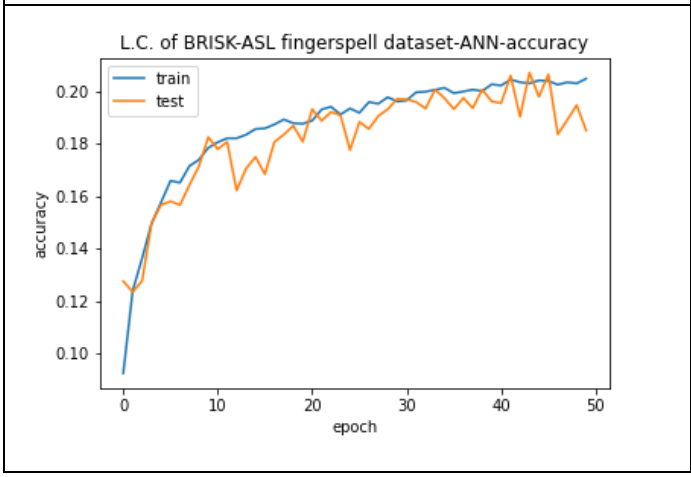


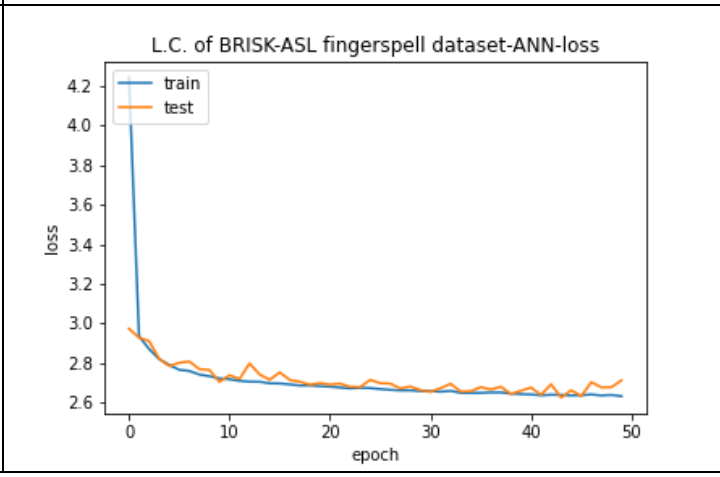

olutional features +finger angles+ PCA-ASL fingerspell dataset-ANN-ac
train
test
accuracy
epoch

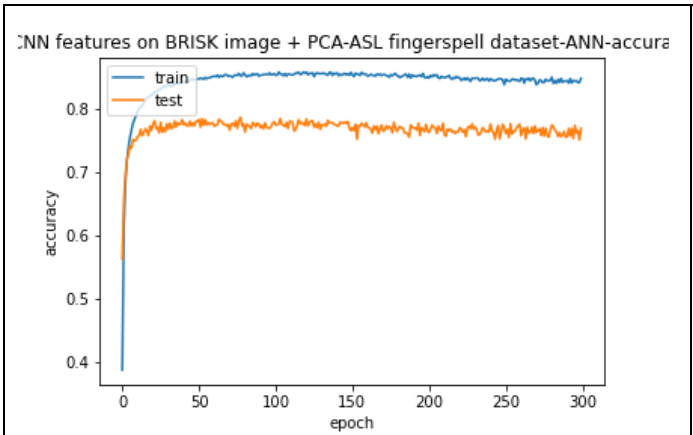

nvolutional features +finger angles+ PCA-ASL fingerspell dataset-ANN-
train
test
loss
epoch

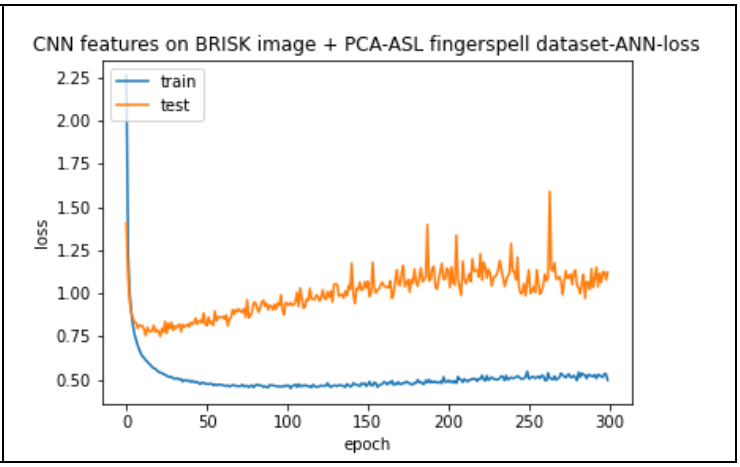

CNN features on hand edges+ PCA-ASL fingerspell dataset-ANN-accurac
train
test
accuracy
epoch


CNN features on hand edges+ PCA-ASL fingerspell dataset-ANN-loss
train
test
loss
epoch


L.C. of BRISK-ASL fingerspell dataset-ANN-accuracy
train
test
accuracy
epoch


L.C. of BRISK-ASL fingerspell dataset-ANN-loss
train
test
loss
epoch





From Table 27, it is evident that 'L.C. of BRISK' trained on the ANN represents an underfitting scenario which means that the model is unable to learn the training data properly as indicated by the high loss values in the corresponding loss curves of the datasets. The highly erratic movements or the spikes of the test data around the training curve in the accuracy and loss graphs also indicate that 'L.C. of BRISK' is an unrepresentative feature for sign image classification.

For the ASL fingerspell dataset, there is a clear gap between the training and testing curves for both types of diagnostic plots. This scenario where the testing accuracy is less than the training accuracy (or, the testing loss is greater than the training loss) is known as overfitting where the model's generalization potential on unseen data is less. This can be attributed to the variations between the same sign gestures performed by the 5 different users in the ASL fingerspell dataset. This problem of overfitting can also be observed in the case of ASL alphabet for 'CNN features on hand edges' and 'CNN features on BRISK', and Cambridge dataset for 'CNN features on BRISK' mainly because the approximate skin masking is leading to a feature representation not generalizable by the ANN model.

The remaining diagnostic curves in Table 27 are a good fit as can be inferred by the negligible gap between the training and testing data. This represents a point of stability for the model which means given a particular architecture and dataset, this is the optimal possible performance as highlighted in Table 26.

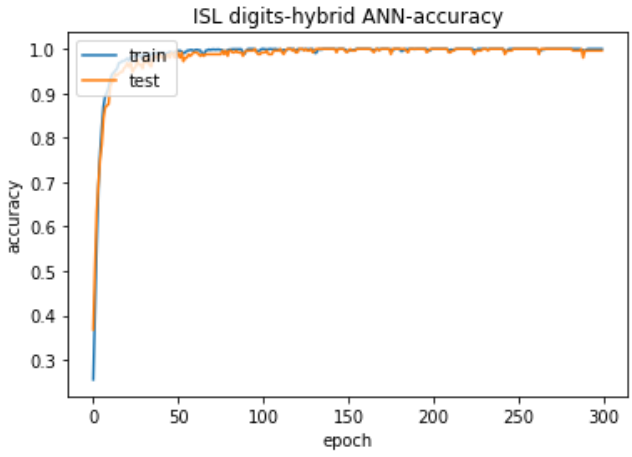
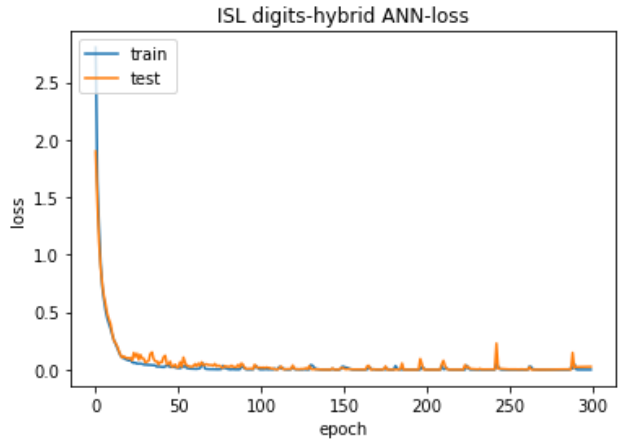
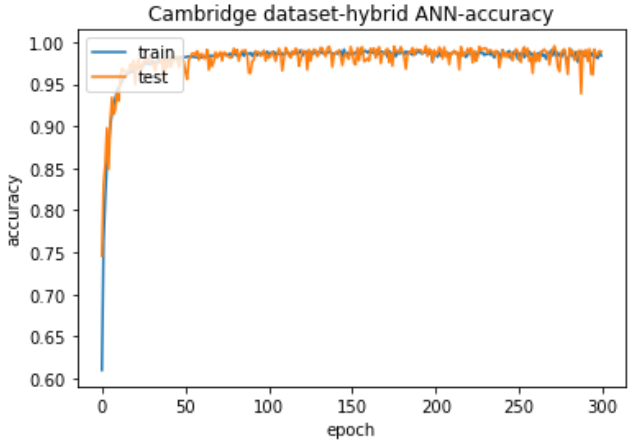
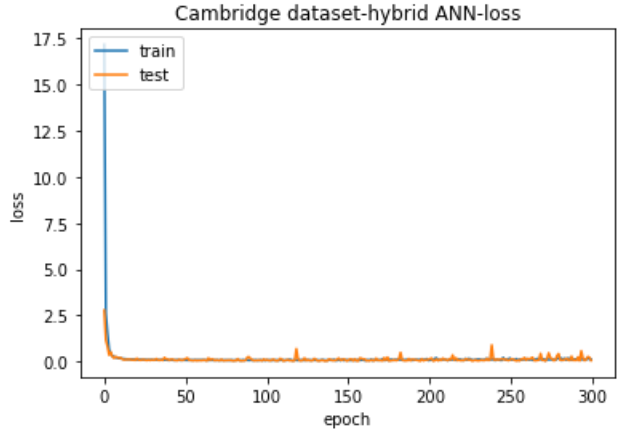
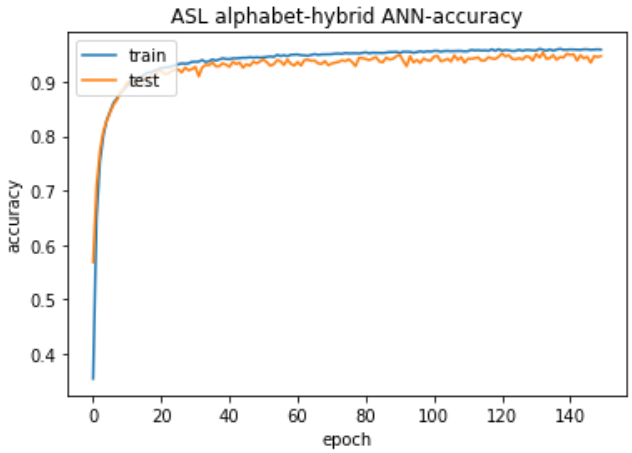
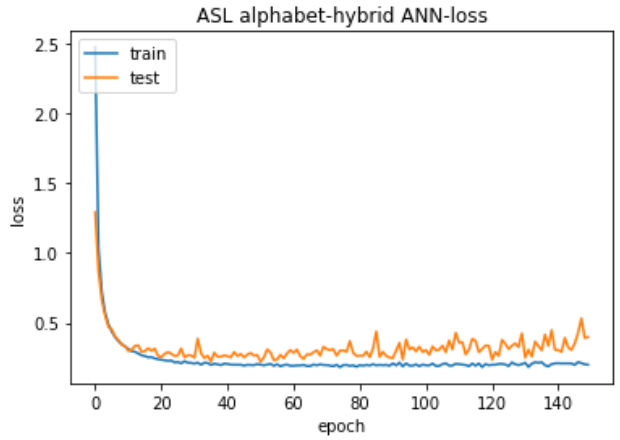
- **Hybrid ANN**

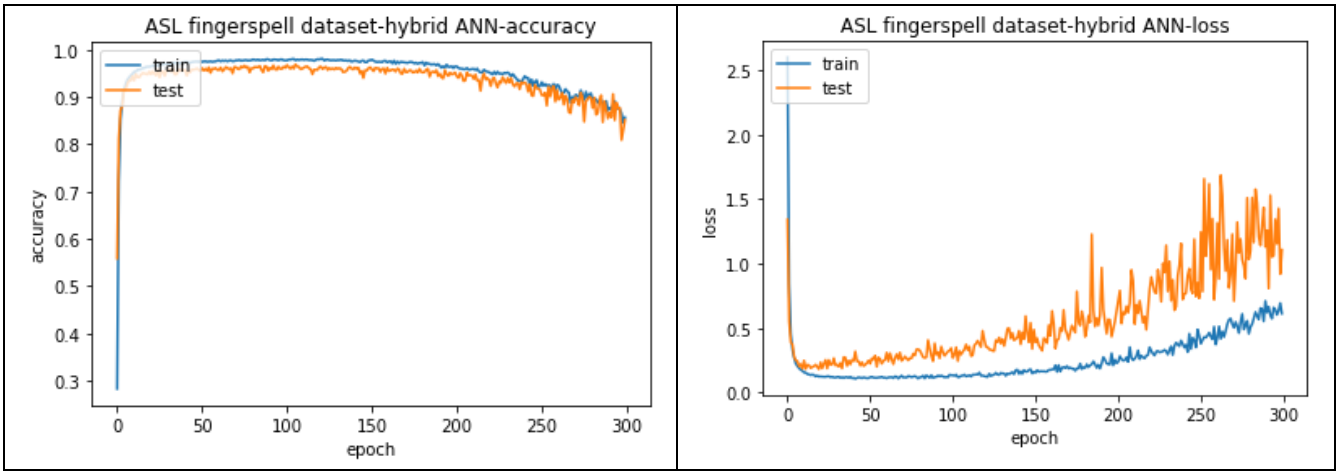
  Table 28 lists the accuracies obtained when using the hybrid ANN structure as described in section 3.1.3.3.2 with a batch size of 5 for ISL digits and a batch size of 16 for Cambridge dataset, ASL alphabet and ASL fingerspell dataset. Table 29 on the other hand displays the corresponding accuracy and loss graphs. All the accuracies are computed with the best weights over the training epochs (Appendix 7).

Table 28. Hybrid ANN accuracies

ISL digits	Cambridge hand gesture dataset	ASL alphabet	ASL fingerspell dataset
100.000%	99.533%	95.368%	96.883%

Table 29. Diagnostic curves for hybrid ANN

Accuracy curve	Loss curve
ISL digits	
	
Cambridge hand gesture dataset	
	
ASL alphabet	
	
ASL fingerspell dataset	



From Table 29, it can be seen that for ISL digits and Cambridge dataset, the diagnostic curves indicate a perfect fit with each of them attaining maximum possible accuracy. The curves for ASL alphabet however shows a little bit of overfitting judging from the small gap between the training and testing plots. Despite this, the small loss and a constant trend in the accuracy and loss curves are indicative of a representative feature set fed to the hybrid ANN as it is properly able to learn in the training stage. The curves for the ASL fingerspell dataset however shows degradation after a certain point in its training i.e. decrease in accuracy and increasing in loss. This can be attributed to the particular mini-batch being trained. The ASL fingerspell dataset contains variations among the same sign gestures owing to user-specific usage. There may be a possibility that for a given mini-batch samples, the current network weights aren't able to generalize properly due to these variations.

It is also clear from Table 28 that the hybrid ANN is able to overcome the accuracies of a single ANN in a majority of cases as discussed in the previous subsection. Hence, the hypothesis for the hybrid ANN as presented in section 3.1.3.3.2 that providing the edge image data along with the original image data assists in effectively identifying the hand region rather than a single ANN taking these features separately, stands out to be true.

- **Trasfer learning**

  Transfer learning with VGGNet 16 gives the following results with a batch size of 32 in Table 30.

Table 30. Transfer learning results

ASL fingerspell (epochs=15)	**Cambridge hand gesture dataset (epochs=15)**	**NUS I dataset (epochs=50)**	**NUS II dataset (epochs=50)**	**ISL digits (epochs=15)**
78.110%	84.310%	96.000%	65.600%	99.200%

Looking at Table 30, VGGNet performs better than ensemble methods when compared with convolutional-related features in Table 22 due to the deep architecture of said network

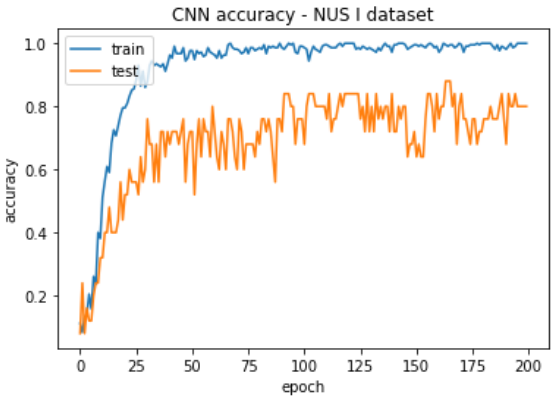
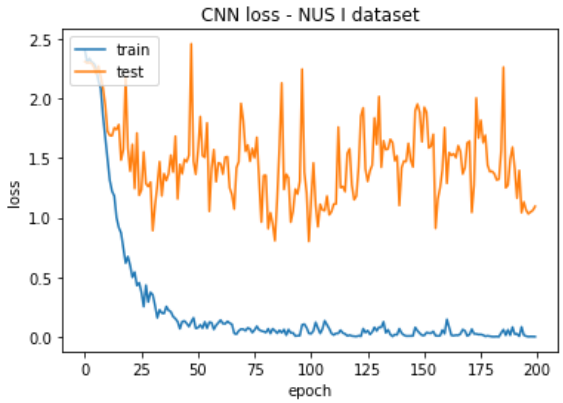
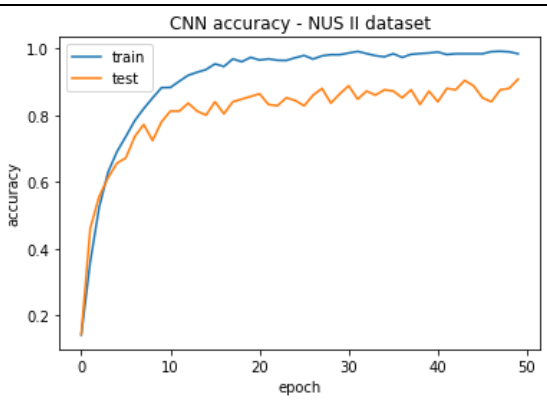
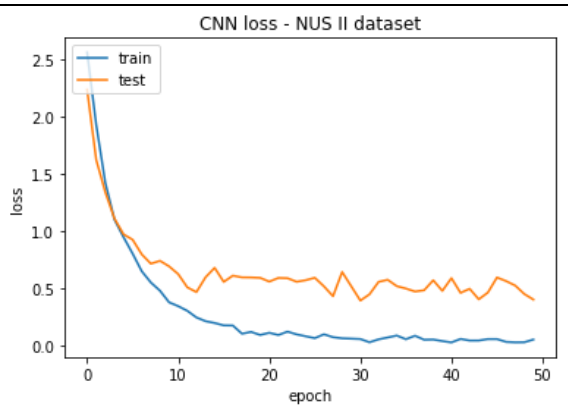
that seems to be ideal for extracting complex features from sign images and hence maximally separating the classes. With increased PCA components however, datasets such as the NUS II dataset is able to surpass VGGNet as shown in Table 23. Another thing to note are the exceptions to VGGNet's performance – ASL fingerspell dataset (due to user-specific image variations) and Cambridge hand gesture dataset (due to sequential images that can be better captured using a RNN than a CNN).

- **Convolutional Neural Networks (CNN)**
  The NUS I and NUS II datasets are among the most complex datasets used in the experiments. Although transfer learning gives appreciable accuracy on NUS I dataset as seen in Table 30, NUS II dataset is still behind in terms of accuracy. In this subsection, the CNNs described in section 3.1.3.3.2 are applied on these datasets to see if it can obtain comparable or even better accuracy than transfer learning. The results are highlighted in Table 31 and the diagnostic curves in Table 32.

Table 31. CNN accuracies

NUS I dataset (batch size=5)	NUS II dataset (batch size=16)
80.000%	90.800%

Table 32. Diagnostic curves for CNN

Accuracy curve	Loss curve
NUS I dataset	
	
NUS II dataset	
	

From Table 32, it is evident that the curves for NUS I dataset indicate overfitting due to the large gap between training and testing plots. Therefore, although the CNN is complex enough to capture the variations in the images of this dataset in terms of scaling which is why it is able to obtain a decent accuracy of 80% in Table 31, the dataset is too small to learn all the parameters of the model. This is not the case for transfer learning where the majority of parameters are already pre-trained and hence the accuracy is higher.

Moving on to the NUS II dataset, the gap between the training and testing curves shrinks indicating less overfitting. So, despite having an intricate architecture to capture background complexity in this dataset, the dataset is large enough to learn the model parameters and alleviate the overfitting problem compared to NUS I dataset. This alleviation is also the reason why the NUS II dataset acquires a greater accuracy of 90.8% in Table 31 compared to transfer learning.

Overall, for these two datasets, CNN outperforms the ensemble methods when compared with convolutional-related features since all the convolutional features are being utilized to learn the complex patterns in the aforementioned datasets rather than applying PCA on said features which leads to information loss.

4.1.2.3 LEAVE-ONE-OUT-ACCURACY

This is applied only to the ASL fingerspell dataset which comprises of user-specific implementations of sign gestures. It works in the following manner:

- Consider User A. To find the leave-one-out-accuracy for this user, the model under consideration is trained on the remaining users and tested on User A.
- This process is repeated for all users. The mean accuracy for all the users gives the leave-one-out-accuracy for the entire dataset.

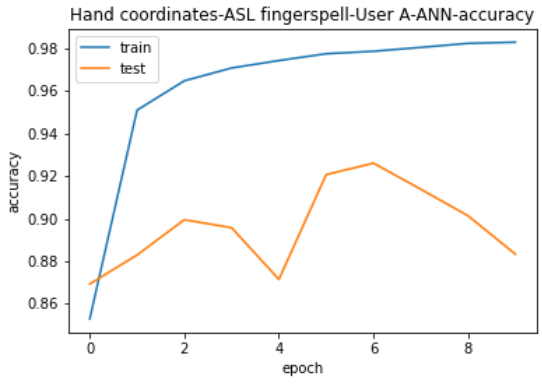
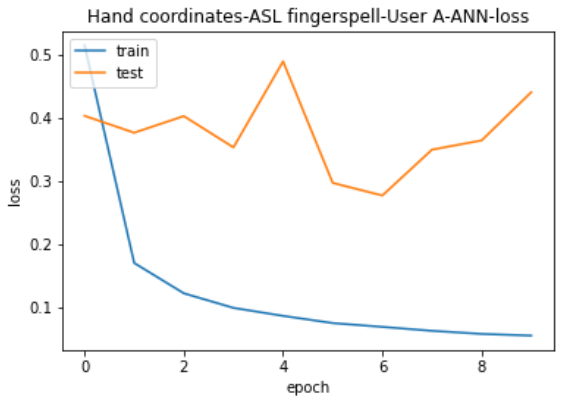
Table 33 outlines the leave-one-out accuracies whereas Table 34 exhibits the diagnostic curves for ANN when applied on 'Hand coordinates' for every user with a batch size of 16 and 10 epochs.

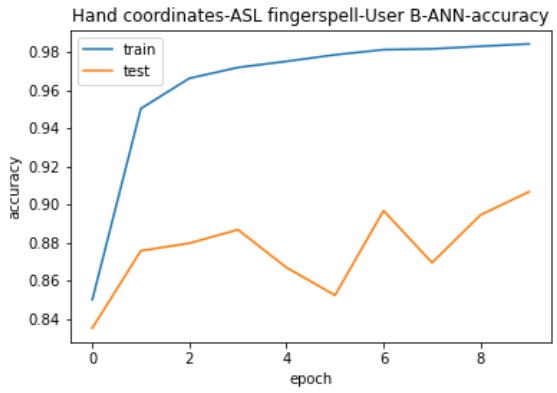
Table 33. Leave-one-out-accuracies

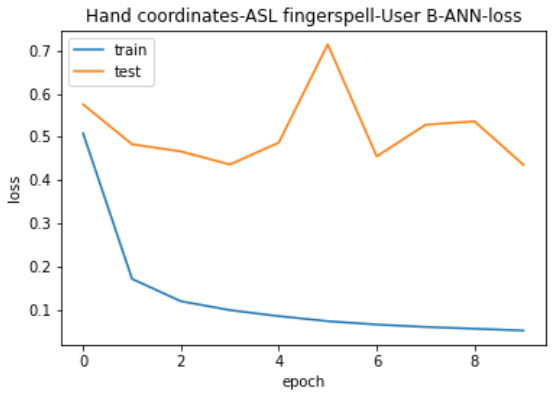
	User A	**User B**	**User C**	**User D**	**User E**	**Average accuracy**
Hand coordinates + Random Forest	85.472%	85.695%	93.436%	85.588%	92.222%	88.483%
Hand coordinates + XGBoost	87.916%	88.780%	94.789%	87.276%	92.387%	90.230%
Hand coordinates + ANN	92.611%	90.687%	95.028%	91.689%	92.676%	92.538%
Convolutional Features + PCA + Random Forest(n=60)	34.431%	40.092%	38.968%	25.650%	43.976%	36.623%
Convolutional Features + PCA + XGBoost(n=60)	35.929%	42.697%	42.037%	28.250%	48.615%	39.506%

Convolutional Features + Finger Angles + PCA + Random Forest(n=60)	35.073%	39.087%	36.887%	25.721%	40.571%	35.468%
Convolutional Features + Finger Angles + PCA + XGBoost(n=60)	38.469%	43.487%	41.246%	25.729%	45.704%	38.927%
CNN features from hand edges + PCA + Random Forest(n=60)	30.701%	23.082%	27.447%	26.806%	39.423%	29.492%
CNN features from hand edges + PCA + XGBoost(n=60)	30.693%	23.183%	27.529%	25.772%	42.255%	29.886%
L.C. of BRISK descriptors on hand edges + Random Forest	12.911%	15.736%	13.477%	9.807%	15.882%	13.563%
L.C. of BRISK descriptors on hand edges + XGBoost	13.756%	15.348%	14.187%	9.784%	15.444%	13.704%
CNN on BRISK + PCA + Random Forest(n=60)	34.144%	36.351%	35.026%	24.130%	35.847%	33.099%
CNN on BRISK + PCA + XGBoost(n=60)	37.101%	37.344%	36.825%	23.787%	39.157%	34.843%

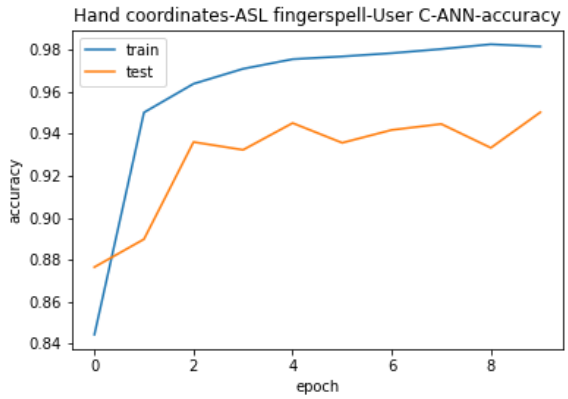
Table 34. Diagnostic curves for every user – Hand coordinates on ASL fingerspell dataset

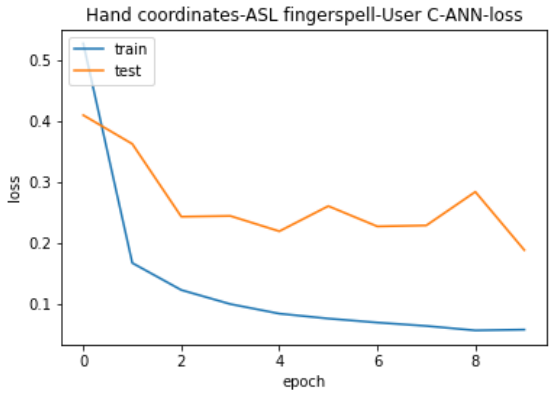
Accuracy curve	**Loss curve**
User A	
	
User B	



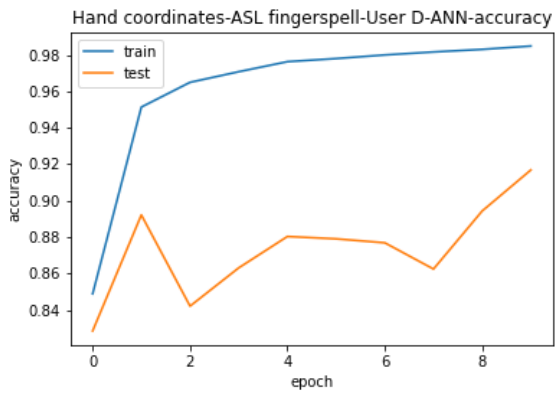


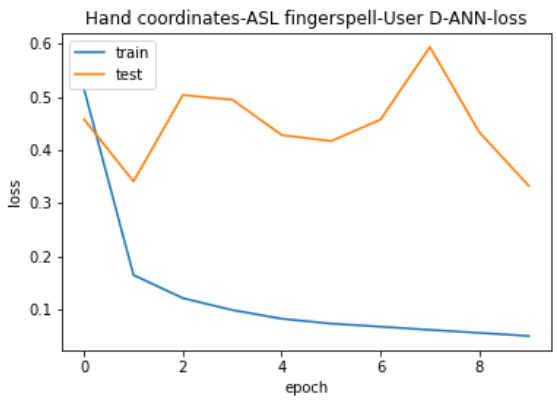
User C





User D





User E

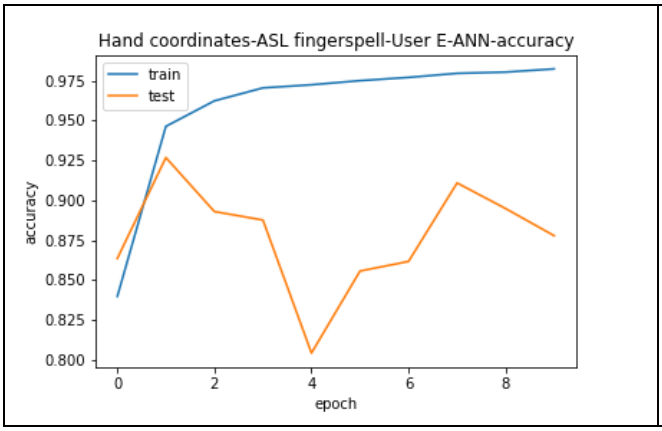


Table 33 indicates that 'Hand coordinates' is best suited for leave-one-out-accuracy since it is able to capture the similar patterns in relative positioning of the hand region joints despite user-specific variations. These variations are effectively captured when we apply the ANN described in Table 12 to the dataset and hence, despite the erratic movement of the testing curve around the training curve in the diagnostic plots in Table 34, which indicates an unrepresentative test data, the mean accuracy is greater than the mean accuracies of the ensemble models.

4.1.2.4 COMPARISON WITH BASE PAPER RESULTS

Table 35. Results Comparison - SLR

Paper	**Dataset**	**Test size**	**Paper Technique**	**Proposed Technique**
[1] Sagayam, K. et al.	Cambridge hand gesture dataset	20%	96.66% (CNN classification)	• 98.626% (Hand coordinates + Random Forest) • 98.789% (Hand coordinates + XGBoost) • 99.549% (Conv features + PCA + Random Forest) • 96.669% (Conv features + PCA + XGBoost) • 99.243% (Conv features + finger angles + PCA + Random Forest) • 96.711% (Conv features + finger angles + PCA + XGBoost) • 99.565% (CNN features on hand edges + PCA + Random Forest) • 99.391% (CNN features on hand edges + PCA + XGBoost) • 98.093% (CNN features on BRISK + PCA + Random

				Forest) • 99.533% (Hybrid ANN)
[2] Aly, W. et al.	ASL fingerspell dataset	Leave-one-out-accuracy	• 88.70% (Single PCANet model + SVM) • 84.50% (User-specific PCANet model + SVM)	• 88.483% (Hand coordinates + Random Forest) • 90.230% (Hand coordinates + XGBoost) • 92.538% (Hand coordinates + ANN)
[3] Gangrade, J. et al.	Self-generated ISL digits	250	• 90.4% (ORB + Nu-SVC) • 93.26% (ORB + KNN)	• 100.000% (Hand coordinates + Random Forest) • 99.602% (Hand coordinates + XGBoost) • 98.000% (Conv features + PCA + Random Forest) • 96.000% (Conv features + PCA + XGBoost) • 96.016% (Conv features + finger angles + PCA + Random Forest) • 95.219% (Conv features + finger angles + PCA + XGBoost) • 98.800% (CNN features on hand edges + PCA + Random Forest) • 98.800% (CNN features on hand edges + PCA + XGBoost) • 96.400% (CNN features on BRISK + PCA + Random Forest) • 96.000% (CNN features on BRISK + PCA + XGBoost) • 100.000% (Hand coordinates + ANN) • 96.400% (Conv features + PCA + ANN) • 94.024% (Conv features + finger angles + PCA + ANN) • 90.400% (CNN features on hand edges + PCA + ANN) • 93.600% (CNN features on BRISK + PCA + ANN) • 100.000% (Hybrid ANN)

	NUS I dataset	25	• 76.9% (ORB + Nu-SVC) • 80.6% (ORB + KNN)	• 80.000% (Hand coordinates + Random Forest) • 80.000% (Hand coordinates + XGBoost) • 80.000% (CNN)
	NUS II dataset	250	• 81.25% (ORB + Nu-SVC) • 85.6% (ORB + KNN)	• 97.200% (Hand coordinates + Random Forest) • 97.200% (Hand coordinates + XGBoost) • 90.800% (CNN)

Note: In the proposed technique, the following ISL digits[23] have been used instead of a self-generated dataset

It is evident from Table 35 that in case of the Cambridge dataset, whose hand gestures are devoid of any cluttered backdrop, using the convolutional-related features with ensemble classifiers give increased accuracy compared to a CNN with softmax classification as described in [1]. This is because by nature, ensembles bolster the predictive performance by decreasing variance of the predictive process with the aid of multiple classifiers each of which add bias. A single CNN, on the other hand, is limited by its specific architecture and count of epochs to achieve only a certain level of accuracy. [1] could have obtained better results with an ensemble of CNNs. Secondly, the usage of hand coordinates pinpoint exact locations of the regions of interest of the hand gesture for better mapping between input features and target label, compared to a CNN which require meticulous hyperparameter tuning to capture the exact same features. It can also be seen that the hybrid ANN outperforms the base paper CNN since it takes into consideration both the actual image and the edge information of the gestures to categorize the sign images.

Table 35 also highlights the fact that for the ASL fingerspell dataset, to test leave-one-out-accuracy, hand coordinates transcend the convolutional features from a PCANet model as specified in [2]. This is because despite user-specificity, when using hand coordinates, the relative distance between the hand joints for the same sign remain consistent across all users whereas in [2], some of this information is lost since the PCANet model uses PCA to select its convolutional filters. Moreover, in [2], the PCANet seems to be generating features greater than the number of instances in the dataset. This causes the SVM to get stuck in a particular hyperplane without advancing any further which could have given better separation between image classes. Ensemble classifiers and ANN don't suffer from this problem.

For the ISL digits in Table 35, the usage of depth images as mentioned in [3] is causing the hand gesture to be a white region amidst a black environment. This is leading to some information loss in terms of sign-specific hand features for the ORB algorithm to detect valuable keypoints. The proposed technique, however, gets rid of depth images in favour of hand coordinates (exact location of hand region), convolutional features on original image as well as edge image (to get exact hand outline), finger angles (to account for sign-specific positioning) and convolution operation on BRISK (to take into consideration all important keypoints). These factors assist in better separation between the digits.

Since NUS I dataset (having minimal segregation between individual classes) and NUS II dataset

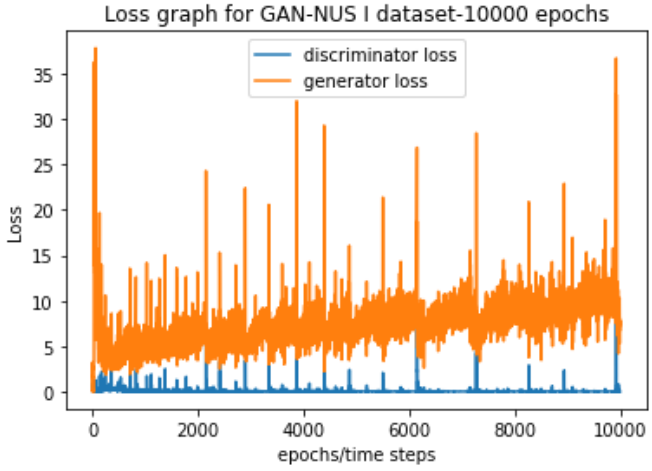
(having highly intricate backgrounds) are complex in type, it is clear from Table 35 that hand coordinates is the ideal feature to classify the images contained within it as they deal with hand region location in 3D space. Also the utilization of a CNN is ideal in this scenario compared to ORB as specified in [3] because with the help of convolutional filters, a CNN is able to identify both low-level and high-level features necessary for distinguishing labels of such complicated datasets whereas a representative ORB keypoint of the same features can be same for multiple classes.

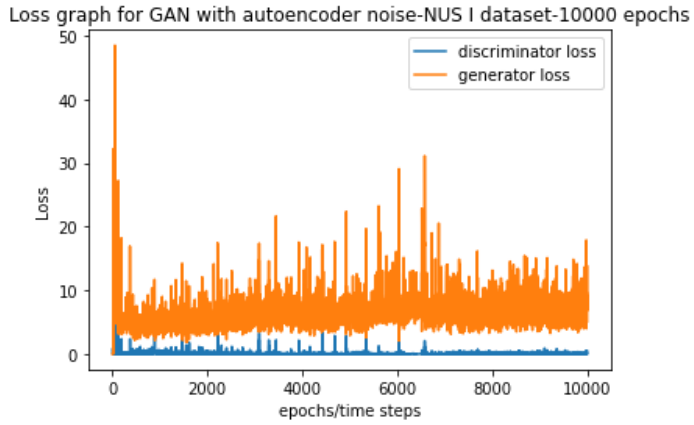
4.2 SIGN LANGUAGE PRODUCTION

4.2.1 QUALITY OF GENERATED SIGN IMAGES

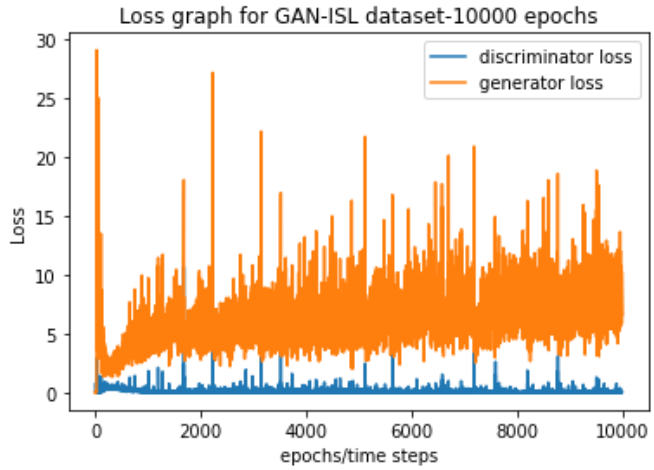
In this subsection, the DCGAN architecture described in Table 20 is applied on the NUS I dataset, ISL digits and a small subset of the RWTH-PHOENIX 14T dataset with a batch size of 16 to visually inspect the quality of the generated images and assess its interpretability when compared with the original images. Table 36 showcases the generated images and the loss curves for both the generator and discriminator using the variations of the DCGAN architecture. On the other hand, Table 37 compares the generated images with the ground truth images.

Table 36. Generated images and loss curves

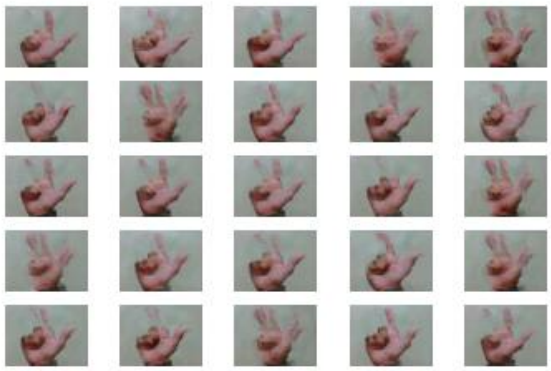
Generated images	**Loss curve**
DCGAN – NUS I dataset(noise size=100)	
	
DCGAN with autoencoder noise – NUS I dataset(noise size=100)	

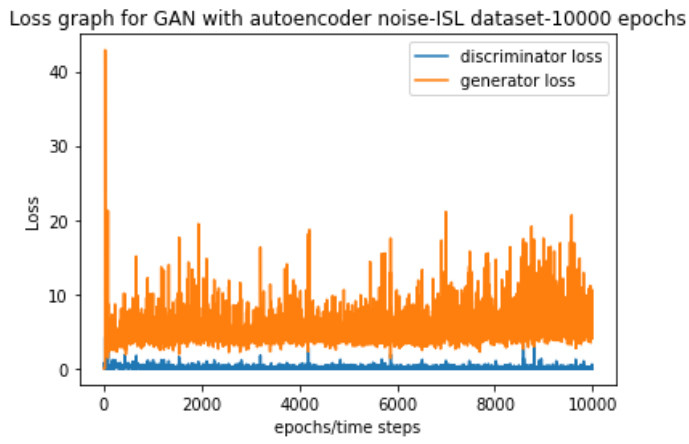


DCGAN – ISL digits(noise size=100)

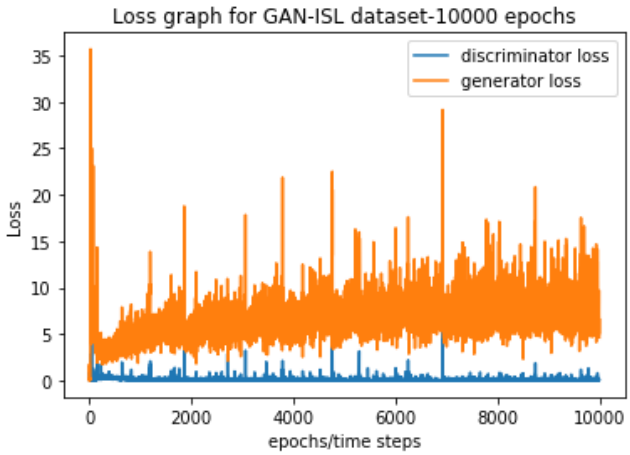


DCGAN with autoencoder noise – ISL digits(noise size=100)

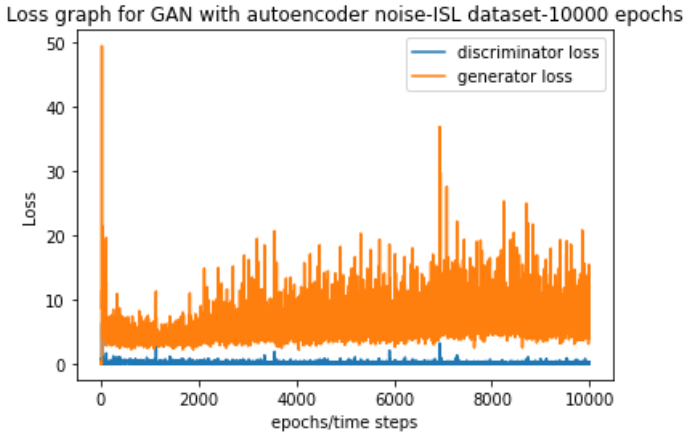




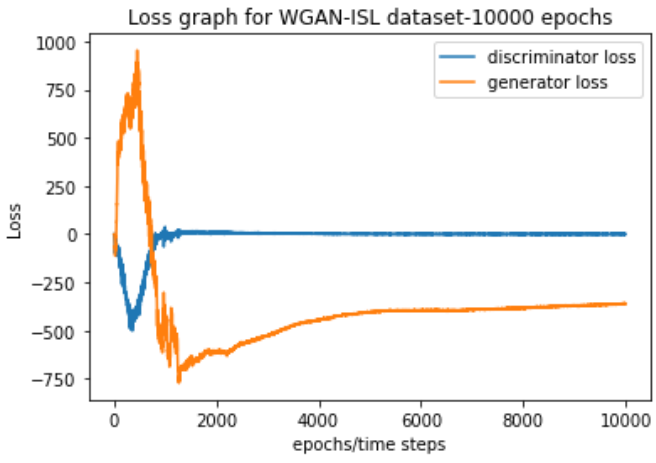
DCGAN – ISL digits(increased noise size=256)



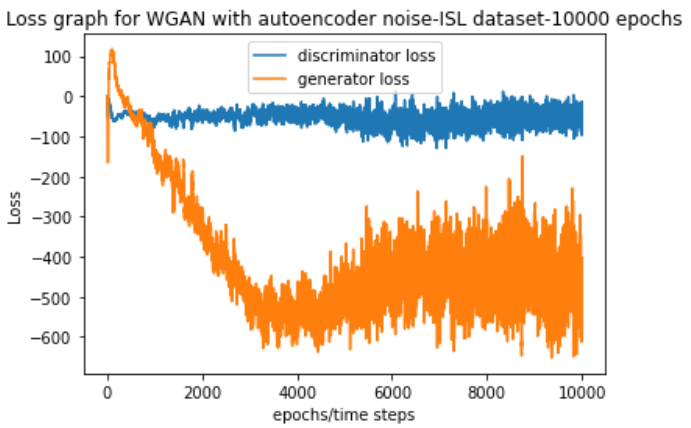
DCGAN with autoencoder noise – ISL digits(increased noise size=256)



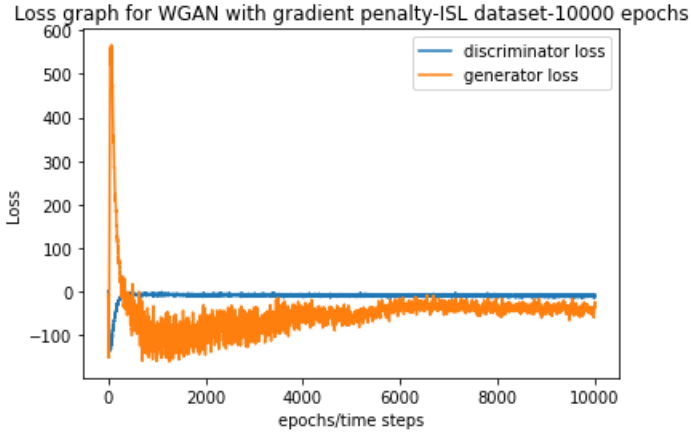
WGAN – ISL digits(noise size=100)



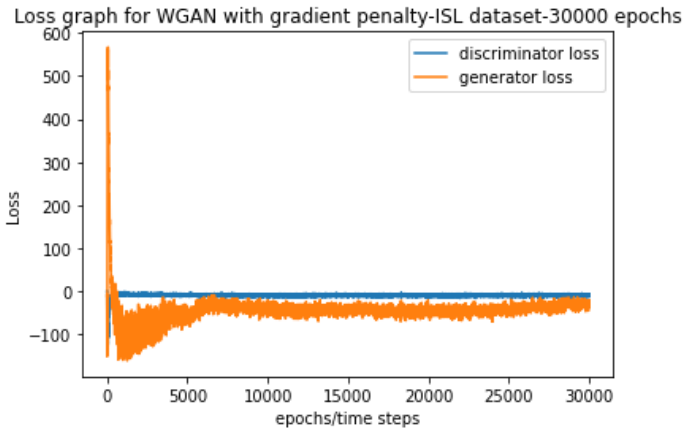
WGAN with autoencoder noise – ISL digits(noise size=100)



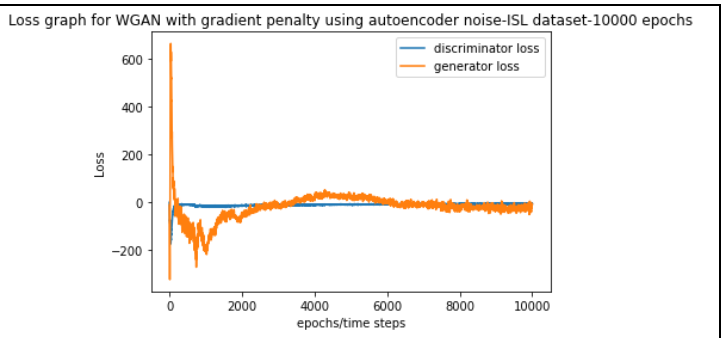
WGAN using GP – ISL digits(noise size=100)



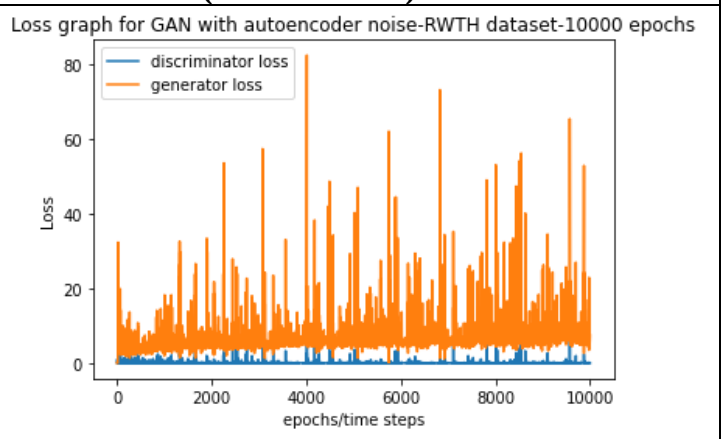
WGAN using GP – ISL digits(noise size=100, increased epochs)



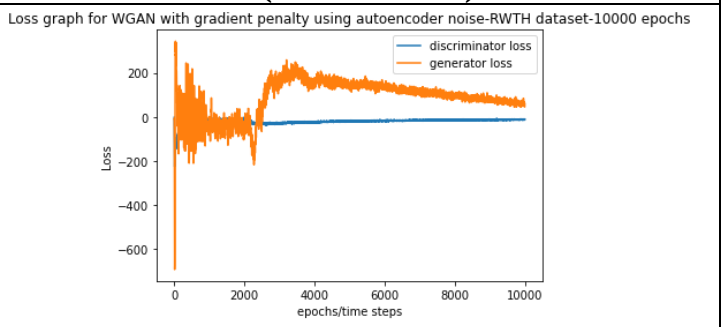
WGAN using GP with autoencoder noise – ISL digits(noise size=100)



DCGAN with autoencoder noise – RWTH subset(noise size=100)



WGAN using GP with autoencoder noise – RWTH subset(noise size=100)



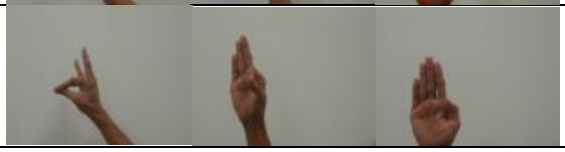
From Table 36, it is clear that in a regular DCGAN, whenever autoencoder noise is used, the spikes in generator loss are significantly reduced compared to the usage of random noise. This is because, when using autoencoder noise, the no. of options for the latent vector is limited to the no. of instances in the dataset which is why the GAN is coerced to learn patterns within this range. This decrease in randomness reduces generator loss. In a WGAN however, using autoencoder noise is

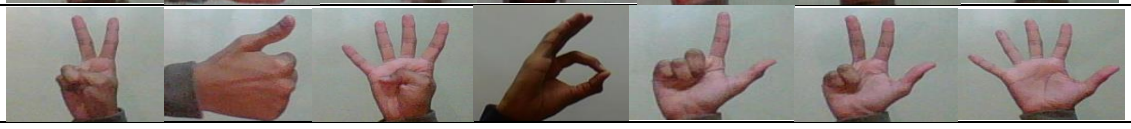
detrimental because the generator don't have a lot of choices to replicate the dataset images within its constrained space due to weight clipping which is why the loss strays significantly away from the convergence point of 0. Even in the case of a normal WGAN where the generator loss steadily moves towards the zero point, the images are too noisy to be interpretable since the generator is always trying to fool the discriminator by trying out diverse images. As a result, it takes longer to converge.

Moving on to the second inference, a WGAN using GP converges very quickly but this is only true for the ISL digits which is a simplistic image dataset to learn compared to the complex RWTH dataset comprising of sequential frames which is why it takes a bit longer to move towards zero. Even then, the convergence is faster than a WGAN not using GP since in this case the WGAN don't have constrained space and the GP formulation directly tries to map between real and fake image distributions.

In Table 36, the results of regular DCGANs also provide an insight into the characteristics of the datasets used. For instance, when using two datasets of same count of image classes, namely NUS I dataset and ISL digits, the smaller sized one i.e. NUS I dataset have higher surges in generator loss because DCGANs by nature require large no. of samples to produce diverse types of high quality images. On the other hand, images with intricate features such as the RWTH dataset have higher peaks of generator loss as it requires even more samples than what is provided to diminish the gap between real and generated images in terms of features. Due to this, the images are non-interpretable.

Table 37. Image comparison for GAN

DCGAN – NUS I dataset(noise size=100) – 7000 epochs	
Generated image	
Ground Truth	
DCGAN with autoencoder noise – NUS I dataset(noise size=100) – 9000 epochs	
Generated image	
Ground Truth	
DCGAN – ISL digits(noise size=100) – 9500 epochs	
Generated image	

Ground Truth	
DCGAN with autoencoder noise – ISL digits(noise size=100) – 8000 epochs	
Generated image	
Ground Truth	
DCGAN – ISL digits(increased noise size=256) – 9000 epochs	
Generated image	
Ground Truth	
DCGAN with autoencoder noise – ISL digits(increased noise size=256) – 10000 epochs	
Generated image	
Ground Truth	
WGAN using GP – ISL digits(noise size=100) – 10000 epochs	
Generated image	
Ground Truth	
WGAN using GP – ISL digits(noise size=100, increased epochs) – 30000 epochs	
Generated image	
Ground Truth	
WGAN using GP with autoencoder noise – ISL digits(noise size=100) – 10000 epochs	

Generated image	
Ground Truth	

Since in the DCGAN experiments, the generator model is saved every 500 epochs or time steps, in Table 37, the generated images are obtained from the best generator model in terms of both image fidelity and image diversity.

A close look in Table 37 highlights the fact that WGAN alleviates the problem of mode collapse by generating more image classes than a standard DCGAN. This is because the discriminator in WGAN attempts to maximize the difference between real and fake images, so the generator tries out different images to pass the discriminator test, wheras in a normal DCGAN, the generator focusses on a small subset of high-quality generated images to be passed as 'real'.

Also, the application of autoencoder noise to the variations of GAN give images with less noise compared to their normal counterparts since the autoencoder noise is nothing but the representation of the actual image in latent space, and hence easier for the generator to comprehend. This usage of latent space image representation sometimes generates more classes than a GAN using random noise like the DCGAN for ISL digits with a noise vector of 100 and the WGAN for ISL digits using GP with a noise vector of 100, both of which uses autoencoder noise.

4.2.2 FID SCORES

Fretchet Inception Distance or FID[40] is a quantitative measure of performance of a GAN in terms of quality of generated images. To put simply, it is the difference between the distributions of real and generated images. Hence, a lower FID score is always favoured for a generative model. To obtain the aforementioned 'difference', it makes use of the Inception network[41] to get numerical feature vector representations of real and fake images. The FID score is calculated using the following formula:

$$FID = \left|\left|\mu_{real} - \mu_{fake}\right|\right|^2 + tr(cov_{real} + cov_{fake} - 2 * \sqrt{cov_{real}.cov_{fake}}) \qquad (18)$$

where,

μ_{real}, μ_{fake} = mean of real and fake images

cov_{real}, cov_{fake} = covariance matrix of real and fake images

tr() = trace of a matrix

Table 38 displays the FID scores for the GAN experiments performed in this section.

Table 38. FID Scores for GAN

GAN experiment	FID score
DCGAN – NUS I dataset(noise size=100)	4387.317700251981
DCGAN with autoencoder noise – NUS I dataset(noise size=100)	4343.731198839419
DCGAN – ISL digits(noise size=100)	3406.5373038612433
DCGAN with autoencoder noise – ISL digits(noise size=100)	3268.7425839872662

DCGAN – ISL digits(increased noise size=256)	3416.8589442437033
DCGAN with autoencoder noise – ISL digits(increased noise size=256)	3301.329565459356
WGAN – ISL digits(noise size=100)	3390.11603730464
WGAN with autoencoder noise – ISL digits(noise size=100)	3234.3503771907863
WGAN using GP – ISL digits(noise size=100)	3398.5829057702476
WGAN using GP – ISL digits(noise size=100, increased epochs)	3394.6415922436418
WGAN using GP with autoencoder noise – ISL digits(noise size=100)	3254.371914932548
DCGAN with autoencoder noise – RWTH subset(noise size=100)	4108.398806145122
WGAN using GP with autoencoder noise – RWTH subset(noise size=100)	4032.99692905415

As is evident from Table 38, the GAN experiments using autoencoder noise has lesser FID scores compared to their non-autoencoder noise types. This indicates better quality images since autoencoder noise is the latent representation of the actual image dataset. Moreover, the use of WGAN on a dataset results in better FID scores in comparison to its standard DCGAN equivalent due to the usage of Wasserstein loss which produces diverse images to minimize the 'difference' between the real and fake ones.

4.2.3 COMPARISON WITH BASE PAPER RESULTS

Table 39 exhibits the generated images of [4] from the SMILE dataset and the generated images obtained from Table 37 on the RWTH subset when using WGAN-GP with autoencoder noise. Both of these datasets are similar in terms of encompassing the entire sign user.

Table 39. Results comparison – SLP

Base paper technique


WGAN-GP with autoencoder noise	
Generated	
GT	

It is evident from Table 39 that [4] is able to produce better quality images which are much closer to the ground truth compared to the usage of the WGAN-GP with autoencoder noise. This is because [4] uses a GAN conditioned on sequential skeletal poses and hence the generator is able to give images closer to the skeletal poses under consideration despite the image complexity. This however, comes at the cost of a large training time of 90000 epochs and an additional 10000 epochs after hyperparameter tuning. On the other hand, in this project, WGAN-GP with autoencoder noise is able to produce comparable images in just 10000 epochs/time steps. The only shortcoming of this method is that the images are noisy around the intricate face portions of the sign users and the minute details of hand regions and hence, at large are non-interpretable. This is because the use of autoencoder noise tries to replicate the generated images as close as possible to the image dataset but due to the absence of a large no. of instances as well as the limiting size of the autoencoder noise, the highly-intricate details are not reproduced properly. Training the WGAN-GP for more no. of iterations along with larger data sizes and larger autoencoder noise dimensions may overcome this problem.

Chapter 5

Conclusion

5.1 CONCLUSION

In conclusion, the proposed features give astounding results with ensemble classifiers and ANNs on datasets free from any background complexity and ones containing distinguishable and uniform hand gestures. The two datasets which deviate from getting appreciable accuracies from the majority of these features are the NUS I and II datasets. In this case, only the hand coordinates feature is able to maximally represent the sign images. The aforementioned anomalous datasets also perform well when trained with a standard CNN. In addition to that, L.C. of BRISK is the one consistent feature that performs inadequately on all datasets owing to the misrepresentative weightage to significant keypoints. However, with a modification that extracts convolutional features from a BRISK –infused image i.e. the CNN features on BRISK, the new BRISK feature is able to surpass its predecessor by a significant amount. Furthermore, the hybrid ANN has proved to be highly effective for sign gesture recognition by taking a combination of 2 convolutional-related features. Last but not the least, with regards to sign image generation, the usage of autoencoder noise has proved to be beneficial in reducing generator loss with the only exception being the RWTH dataset whose sheer complexity in terms of image features encompassing the entire sign user cannot be handled by the GAN architecture used and hence require a more deeper architecture or more training epochs.

As far as the research objectives are concerned, the proposed methodology has been successful in constructing a hand gesture recognition system with the constraint that only with the help of hand coordinates alone can user-specificity be enforced which refers to the application of this system beyond the scope of training data provided. This can be improved by training a multitude of different sign image datasets, captured under different conditions, on a deep-tuned CNN. Moreover, although the sign images generated are interpretable in nature, the problem of mode collapse still exists and has only been partially alleviated with the help of Wasserstein loss. Improvements can be made in this regard with the usage of large amounts of data for training GANs.

5.2 FUTURE WORK

With the core system being completed, the following future improvements can be made:

- Extending the sign gesture recognition system to capture real-time sign language usage in diverse environements.
- Addition of a sentiment detection module to the recognition system to capture the resultant emotions of the sign users for resolving ambiguous sign phrases.
- Capturing sign words and sentences without idiosyncracy using Natural Language rules.
- Building a conditional WGAN using GP to produce more classes than a normal WGAN using GP and hence further remove the problem of mode collapse.

REFERENCES

[1] Sagayam, K. M., Andrushia, A. D., Ghosh, A. and Deperlioglu, O. (2021). *Recognition of Hand Gesture Image Using Deep Convolutional Neural Network*, International Journal of Image and Graphics, Vol. 21, No. 1, pp. 2140008-2140022

[2] Aly, W., Aly, S. and Almotairi, S. (2019). *User-Independent American Sign Language Alphabet Recognition Based on Depth Image and PCANet Features*, IEEE Access, Vol. 7, pp. 123138-123150

[3] Gangrade, J., Bharti, J. and Mulye, A. (2020). *Recognition of Indian Sign Language Using ORB with Bag of Visual Words by Kinect Sensor*, IETE Journal of Research, pp. 1-15

[4] Stoll, S., Camgoz, N. C., Hadfield, S. and Bowden, R. (2020). *Text2Sign: Towards Sign Language Production Using Neural Machine Translation and Generative Adversarial Networks*, International Journal of Computer Vision, Vol. 128, No. 4, pp. 891-908

[5] Shivashankara, S. and Srinath, S. (2018). *American Sign Language Recognition System: An Optimal Approach*, International Journal of Image, Graphics and Signal Processing, Vol. 10, No. 8, pp. 18-30

[6] Raheja, J. L., Mishra, A. and Chaudhary, A. (2016). *Indian sign language recognition using SVM*, Pattern Recognition and Image Analysis, Vol. 26, No. 2, pp. 434-441

[7] Thang, P. Q., Dung, N. D. and Thuy, N. T. (2017). *A Comparison of SimpSVM and RVM for Sign Language Recognition*, Proc. of the 2017 International Conference on Machine Learning and Soft Computing, Ho Chi Minh City, pp. 98-104, Vietnam

[8] Gattupalli, S., Ghaderi, A. and Athitsos, V. (2016). *Evaluation of Deep Learning based Pose Estimation for Sign Language Recognition*, Proc. of the 9th ACM International Conference on Pervasive Technologies Related to Assistive Environments, Corfu Island, Greece

[9] Kumara, B. M., Nagendraswamy, H. S. and Chinmayi, R. L. (2016). *Spatial Relationship Based Features for Indian Sign Language Recognition*, International Journal of Computing, Communication and Instrumentation Engineering, Vol. 3, No. 2

[10] Pansare, J. R. and Ingle M. (2016). *Vision-based approach for American Sign Language recognition using Edge Orientation Histogram*, 2016 International Conference on Image, Vision and Computing, Portsmouth City, pp. 86-90, UK

[11] Thalange, A. and Dixit, S. K. (2016). *COHST and Wavelet Features Based Static ASL Numbers Recognition*, Procedia Computer Science, Vol. 92, pp. 455-460

[12] Hurroo, M. and Walizad, Md. E. (2020). *Sign Language Recognition System using Convolutional Neural Network and Computer Vision*, International Journal of Engineering Research and Technology, Vol. 9, No. 12

[13] Huang, J., Zhou, W., Zhang, Q., Li, H. and Li, W. (2018). *Video-based Sign Language Recognition without Temporal Segmentation*, Proc. of the 32nd AAAI Conference on Artificial Intelligence, New Orleans City, pp. 2257-2264, USA

[14] Cui, R., Liu, H. and Zhang, C. (2017). *Recurrent Convolutional Neural Networks for Continuous Sign Language Recognition by Staged Optimization*, 2017 IEEE Conference on Computer Vision and Pattern Recognition, Honolulu City, pp. 1610-1618, USA

[15] Bank, D., Koenigstein, N. and Giryes, R. (2020). *Autoencoders*

[16] Radford, A., Metz, L. and Chintala, S. (2016). *Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks*, 4th International Conference on Learning Representations, San Juan City, Puerto Rico

[17] Ledig, C., Theis, L., Huszár, F., Caballero, J., Aitken, A.P., Tejani, A., Totz, J., Wang, Z. and Shi, W. (2017). *Photo-Realistic Single Image Super-Resolution Using a Generative Adversarial Network*, 2017 IEEE Conference on Computer Vision and Pattern Recognition, Honolulu City, pp. 105-114, USA

[18] Reed, S. E., Akata, Z., Yan, X., Logeswaran, L., Schiele, B. and Lee, H. (2016). *Generative Adversarial Text to Image Synthesis*, Proc. of the 33rd International Conference on Machine Learning, New York City, USA

[19] Akash. *ASL alphabet*, https://www.kaggle.com/datasets/grassknoted/asl-alphabet

[20] Kim, T. K. *Cambridge hand gesture dataset*, https://labicvl.github.io/ges_db.htm

[21] Geislinger, V. *ASL fingerspell dataset*, https://www.kaggle.com/datasets/mrgeislinger/asl-rgb-depth-fingerspelling-spelling-it-out or https://empslocal.ex.ac.uk/people/staff/np331/index.php?section=FingerSpellingDataset

[22] Kumar, P., Vadakkepat, P. and Poh, L. A. *NUS Dataset I and II*, https://scholarbank.nus.edu.sg/handle/10635/137241 & https://scholarbank.nus.edu.sg/handle/10635/137242

[23] Shenoy, K. *ISL digits*, https://www.kaggle.com/datasets/kartik2112/indian-sign-language-translation-letters-n-digits

[24] *Mediapipe hands*, https://google.github.io/mediapipe/solutions/hands.html

[25] *Skin color code*, https://huebliss.com/skin-color-code/#:~:text=A%20typical%20natural%20skin%20color,be%20used%20for%20general%20purposes

[26] *Canny edge detection*, https://docs.opencv.org/4.x/da/d22/tutorial_py_canny.html

[27] Haidar, L. *Canny over Sobel filter*, https://medium.com/codex/sobel-vs-canny-edge-detection-techniques-step-by-step-implementation-11ae6103a56a#:~:text=The%20main%20advantages%20of%20the,Non%2Dmaxima%20suppression%20and%20thresholding.

[28] Leutenegger, S., Chli, M. and Siegwart, R. Y. (2011). *BRISK: Binary Robust invariant scalable keypoints*, 2011 International Conference on Computer Vision, Barcelona City, pp. 2548-2555, Spain

[29] Mangale, S. *Scree plot*, https://sanchitamangale12.medium.com/scree-plot-733ed72c8608

[30] Tiwari, S. *Activation functions*, https://www.geeksforgeeks.org/activation-functions-neural-networks/

[31] He, K., Zhang, X., Ren, S. and Sun, J. (2015). *Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification*, 2015 IEEE International Conference on Computer Vision, Santiago City, pp. 1026-1034, Chile

[32] Glorot, X. and Bengio, Y. (2010). *Understanding the difficulty of training deep feedforward neural networks*

[33] *RWTH_PHOENIX2014T Dataset*, https://www.kaggle.com/datasets/mariusschmidtmengin/phoenixweather2014t-3rd-attempt

[34] Perarnau, G., Weijer, J. V., Raducanu, B. and Álvarez, J. M. (2016). *Invertible Conditional GANs for image editing*

[35] Arjovsky, M., Chintala, S. and Bottou, L. (2017). *Wasserstein GAN*

[36] Brownlee, J. *Implement Wasserstein loss in keras*, https://machinelearningmastery.com/how-to-implement-wasserstein-loss-for-generative-adversarial-networks/

[37] *Lipschitz continuity*, https://en.wikipedia.org/wiki/Lipschitz_continuity

[38] Gulrajani, I., Ahmed, F., Arjovsky, M., Dumoulin, V. and Courville, A. C. (2017). *Improved Training of Wasserstein GANs*

[39] *Keras implementation of GP in WGAN*, https://github.com/keras-team/keras-contrib/blob/master/examples/improved_wgan.py

[40] *Fretchet Inception Distance*, https://en.wikipedia.org/wiki/Fr%C3%A9chet_inception_distance

[41] Vaishnav, D. *InceptionNet*, https://www.geeksforgeeks.org/ml-inception-network-v1/

Appendix 1

STORING IMAGE PATHS AND LABELS

```
#initializing root directory for sign images
image_path='D:\\Capstone utilities\\Datasets\\asl_alphabet'
#initializing lists to store absolute paths and image classes
x_label=[]
y_label=[]
arr=os.listdir(image_path)
for image_directory in arr:
    path = image_path+'\\'+image_directory
    images=os.listdir(path)
    for image in images:
        img_path = path+'\\'+image
        img=cv2.imread(img_path,cv2.IMREAD_COLOR)
        if img is not None: #checking for corrupted images
            x_label.append(img_path) #storing absolute paths
            y_label.append(image_directory) #storing image classes
#dumping data as pickle files
pickle.dump(x_label,open('C:\\Users\\Ayanabha\\Capstone\\Pickles\\x_asl.pkl','wb'))
pickle.dump(y_label,open('C:\\Users\\Ayanabha\\Capstone\\Pickles\\y_asl.pkl','wb'))
```

Appendix 2

FINGER ANGLES

```
#gathering coordinates for angle calculation
vertices=[0,3,5,8,12,16,20]
vertex_coords=[]
for vertex in vertices:
    point_coord=[sign_hand_coordinate[3*vertex],sign_hand_coordinate[3*vertex+1],
                 sign_hand_coordinate[3*vertex+2]]
    vertex_coords.append(point_coord) #storing (x,y,z) for every coordinate
#passing points for angle calculation
for j in range(1,len(vertices)-1):
    if j==2:
      continue
    angle=Cosine(vertex_coords[0],vertex_coords[j],vertex_coords[j+1])
    finger_angles.append(angle) #appending angles to empty list

def Cosine(pointA,pointB,pointC):
  #law of cosines for triangles
  a=math.sqrt(sum([pow(i-j,2) for (i,j) in zip(pointA,pointB)]))
  b=math.sqrt(sum([pow(i-j,2) for (i,j) in zip(pointA,pointC)]))
  c=math.sqrt(sum([pow(i-j,2) for (i,j) in zip(pointB,pointC)]))
  res=(a*a+b*b-c*c)/(2*a*b)
  return res
```

Appendix 3

LINEAR COMBINATION OF BRISK

```
#creating a color mask
mask=cv2.inRange(imag,lb,ub)
result=cv2.bitwise_and(imag,imag,mask=mask)
#denoising
filtered_result=cv2.medianBlur(result,3)
#edge filtering
edge_result_canny=cv2.Canny(filtered_result,100,200)
#BRISK
brisk=cv2.BRISK_create()
kp,des=brisk.detectAndCompute(edge_result_canny,None)
#initializing feature vector
feature_vector=[0 for j in des[0]]
#getting total weight of all keypoints
total_weight=sum([sum(descriptor) for descriptor in des])
for descriptor in des:
    temp=[]
    proportion=sum(descriptor)
    alpha=proportion/total_weight # alpha value for current keypoint
    for k in range(len(descriptor)):
      temp.append(feature_vector[k]+alpha*descriptor[k])
    feature_vector=temp
```

Appendix 4

VISUALIZATION OF SCREE GRAPH

```
#finding right no. of principal components for PCA
k=50
X=np.array(data)
x_axis=[i for i in range(1,k+1)]
pca=PCA(n_components=k)
x_pca=pca.fit_transform(X)

plt.plot(x_axis,pca.explained_variance_ratio_)
plt.title('ASL alphabet dataset - CNN features on BRISK image(n=15)')
plt.ylabel('Explained variance')
plt.xlabel('No. of components')
plt.show()
```

Appendix 5

ANN WITH 'HE_UNIFORM' INITIALIZATION

```
init=tf.keras.initializers.he_uniform(seed=5)
model=tf.keras.models.Sequential()
model.add(tf.keras.layers.Dense(units=64,kernel_initializer=init,activation='relu'))
model.add(tf.keras.layers.Dense(units=64,kernel_initializer=init,activation='relu'))
model.add(tf.keras.layers.Dense(units=64,kernel_initializer=init,activation='relu'))
model.add(tf.keras.layers.Dense(units=64,kernel_initializer=init,activation='relu'))
model.add(tf.keras.layers.Dense(units=len(np.unique(data.iloc[:,-1])), activation='softmax'))
```

Appendix 6

HYBRID ANN USING KERAS 'MODEL' API

```
'''individual fully-connected blocks for original image and edge image'''
def model_ind(input_shape,n_classes):
  inp=keras.Input(shape=input_shape)
  x=tf.keras.layers.Dense(units=6,kernel_initializer=init,activation='relu')(inp)
  x=tf.keras.layers.Dense(units=6,kernel_initializer=init,activation='relu')(x)
  x=tf.keras.layers.Dense(units=input_shape[0])(x)
  model=keras.Model(inp,x)
  return model
'''creating separate models for original image (df1) and edge image (df2)'''
#no. of classes
n_classes_1=len(np.unique(df1.iloc[:,-1]))
n_classes_2=len(np.unique(df2.iloc[:,-1]))
#input shape
shape_1=(df1.shape[1]-1,)
shape_2=(df2.shape[1]-1,)
#getting the models
m_1=model_ind(shape_1,n_classes_1)
m_2=model_ind(shape_2,n_classes_2)
#providing input
input_1=keras.Input(shape=shape_1)
input_2=keras.Input(shape=shape_2)
#getting the output
out_1=m_1(input_1)
out_2=m_2(input_2)
'''combining both fully-connected blocks'''
#concatenating the outputs
concat=tf.keras.layers.Concatenate()([out_1,out_2])
#final fully-connected block
x=tf.keras.layers.Dense(units=6,kernel_initializer=init,activation='relu')(concat)
x=tf.keras.layers.Dense(units=6,kernel_initializer=init,activation='relu')(concat)
x=tf.keras.layers.Dense(units=n_classes_1,activation='softmax')(x)
model=keras.Model([input_1,input_2],x)
```

Appendix 7

USING BEST WEIGHTS IN NEURAL NETWORKS

```
#defining save spot for best weights
checkpoint_filepath='Weights\\best.h5'
#using Keras callbacks to save weights
callbacks = [tf.keras.callbacks.ModelCheckpoint(filepath=checkpoint_filepath,save_weights_only=True,
                                                monitor='val_accuracy',mode='max',save_best_only=True)]
model.fit(X_train,y_train,validation_data=(X_test,y_test),batch_size=16,epochs=300,callbacks=callbacks)
#loading saved weights
model.load_weights(checkpoint_filepath)
results=model.evaluate(X_test,y_test)
```

Appendix 8

TRAINING LOOP OF GAN WITH AUTOENCODER NOISE

```
for epoch in range(epochs+1):
  '''train discriminator'''
  #select random batch of real images from dataset - X_train
  id=np.random.randint(0,X_train.shape[0],batch_size)
  imgs=X_train[id]
  #generate batch of fake images from autoencoder noise - X_train_noise
  idn=np.random.randint(0,X_train_noise.shape[0],batch_size)
  gen_imgs=generator.predict(X_train_noise[idn])
  #discriminator loss
  d_loss_real=discriminator.train_on_batch(imgs,np.ones((batch_size,1))) #loss on real images
  d_loss_fake=discriminator.train_on_batch(gen_imgs,np.zeros((batch_size,1))) #loss on fake images
  d_loss=0.5*np.add(d_loss_real,d_loss_fake)
  #storing discriminator loss
  disc_loss.append(d_loss[0])
  '''train generator'''
  #generate batch of fake images and pass to discriminator for validation
  idn=np.random.randint(0,X_train_noise.shape[0],batch_size)
  noise=X_train_noise[idn]
  valid_y=np.array([1]*batch_size)
  #generator loss - loss of fake images after passing through discriminator
  g_loss=combined.train_on_batch(noise,valid_y)
  #storing generator loss
  gen_loss.append(g_loss)
```

Appendix 9

GAN PARAMETER VALUES

Parameter	Value
GAN LeakyReLU ‘alpha’	0.2
GAN BatchNormalization ‘momentum’	0.8
DCGAN Adam ‘learning_rate’	0.0002
DCGAN Adam ‘beta_1’	0.5
WGAN ‘weight-clip value’	0.01
WGAN RMSProp ‘learning_rate’	0.00005
WGAN-GP Adam ‘learning_rate’	0.0001
WGAN-GP Adam ‘beta_1’	0.9
WGAN-GP Adam ‘beta_2’	0.9
WGAN-GP ‘penalty coefficient’	10

Note: All the above values are in accordance to the design principles of [16], [35] and [38]