DAYANANDA SAGAR UNIVERSITY

KUDLU GATE, BANGALORE – 560068

Bachelor of Technology
in
COMPUTER SCIENCE AND ENGINEERING

Major Project Phase-II Report

(SIGN LANGUAGE TRANSLATOR USING DEEP LEARNING)

By
B. Evans Nikith Royan – ENG18CS0054
Darryl Andrade – ENG18CS0079
Davin S Thomas - ENG18CS0082
Debjit Das – ENG18CS0083
Gyandwip Das – ENG18CS0109

Under the supervision of

Prof. Rashmi Mothkur
Assistant Professor, CSE

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING,
SCHOOL OF ENGINEERING
DAYANANDA SAGAR UNIVERSITY,
BANGALORE

(2021-2022)

DAYANANDA SAGAR UNIVERSITY

School of Engineering
Department of Computer Science & Engineering
Kudlu Gate, Bangalore – 560068
Karnataka, India.

CERTIFICATE

This is to certify that the Phase-II project work titled **"SIGN LANGUAGE TRANSLATOR USING DEEP LEARNING"** is carried out by **B. Evans Nikith Royan (ENG18CS0054), Darryl Andrade (ENG18CS0079), Davin S Thomas (ENG18CS0082), Debjit Das (ENG18CS0083), Gyandwip Das (ENG18CS0109),** bonafide students of Bachelor of Technology in Computer Science and Engineering at the School of Engineering, Dayananda Sagar University, Bangalore in partial fulfillment for the award of degree in Bachelor of Technology in Computer Science and Engineering, during the year **2021-2022**.

Prof. Rashmi Mothkur	**Dr. Girisha G S**	**Dr. A Srinivas**
Assistant Professor Dept. of CS&E, School of Engineering Dayananda Sagar University	Chairman CSE School of Engineering Dayananda Sagar University	Dean School of Engineering Dayananda Sagar University
Date:	Date:	Date:

Name of the Examiner **Signature of the Examiner**

1.

2.

DECLARATION

We, **B. Evans Nikith Royan (ENG18CS0054), Darryl Andrade (ENG18CS0079), Davin S Thomas (ENG18CS0082), Debjit Das (ENG18CS0083), Gyandwip Das (ENG18CS0109),** are students of the eighth semester B.Tech in **Computer Science and Engineering**, at School of Engineering, **Dayananda Sagar University**, hereby declare that the phase-II project titled **"Sign Language Translator Using Deep Learning"** has been carried out by us and submitted in partial fulfillment for the award of degree in **Bachelor of Technology in Computer Science and Engineering** during the academic year **2021-2022**

Student	**Signature**
Name: B. Evans Nikith Royan	
USN: ENG18CS0054	
Name: Darryl Andrade	
USN: ENG18CS0079	
Name: Davin S Thomas	
USN: ENG18CS0082	
Name: Debjit Das	
USN: ENG18CS0083	
Name: Gyandwip Das	
USN: ENG18CS0109	

Place : Bangalore

Date :

ACKNOWLEDGEMENT

It is a great pleasure for us to acknowledge the assistance and support of many individuals who have been responsible for the successful completion of this project work.

First, we take this opportunity to express our sincere gratitude to School of Engineering & Technology, Dayananda Sagar University for providing us with a great opportunity to pursue our Bachelor's degree in this institution.

We would like to thank **Dr. A Srinivas. Dean**, **School of Engineering & Technology**, **Dayananda Sagar University** for his constant encouragement and expert advice. It is a matter of immense pleasure to express our sincere thanks to **Dr. Girisha G S, Department Chairman**, **Computer Science, and Engineering**, **Dayananda Sagar University,** for providing the right academic guidance that made our task possible.

We would like to thank our guide **Prof. Rashmi Mothkur**, **Assistant Professor**, **Dept. of Computer Science and Engineering**, **Dayananda Sagar University**, for sparing her valuable time to extend help in every step of our project work, which paved the way for smooth progress and the fruitful culmination of the project.

We would like to thank our Project Coordinators **Dr. Meenakshi Malhotra** and **Dr. Bharanidharan N,** and all the staff members of Computer Science and Engineering for their support.

We are also grateful to our family and friends who provided us with every requirement throughout the course. We would like to thank one and all who directly or indirectly helped us in the Project work

TABLE OF CONTENTS

LIST OF ABBREVIATIONS

ASL	American Sign Language
BSL	British Sign Language
AuSLan	Australian Sign Language
ReLU	Rectified Linear Unit
CNN	Convolutional Neural Network
SVM	Support Vector Machine
SIFT	Scale Invariant Feature Transform
LSTM	Long Short Term Memory
RNN	Recurrent Neural Network
ROI	Region Of Interest
R-CNN	Region Based Convolutional Neural Network

LIST OF FIGURES

LIST OF TABLES

ABSTRACT

Human beings need to communicate with one another for a variety of reasons to coexist and survive. Unfortunately, millions of people worldwide lack the ability to speak or hear, either from birth or as a result of some accident. For people who suffer from these disabilities, it is difficult to communicate with the rest of the population. Hence, Sign language was created, which relies on physical gestures to represent letters, words or even sentences. A combination of these signs/gestures help a person to convey their message to another person. These signs/gestures, however, are not known by most of the general public, who will most definitely find it impossible to understand what another person is saying in sign language. The general public most often has no immediate need to learn sign language or has no motivation to do so. Hence there is a need for a Sign Language Translator, that would convert these gestures and movements into words and alphabets that are understandable to people who do not know sign language. 5 models, namely a basic convolutional network, ResNet50, ResNet50V2, Inceptionv3 and VGG16 were developed with CNN and Transfer Learning methodologies to translate sign language gestures in real time.

CHAPTER 1

INTRODUCTION

CHAPTER 1 INTRODUCTION

1.1. INTRODUCTION

A person who is incapable of speaking can communicate by writing, typing or sign language. Regular speech takes place spontaneously, in real time. Writing and typing require one person to complete their communication before the other can access it. Therefore, Sign language is the most spontaneous. Sign Language is how people who are hearing and speech impaired would express their feelings, contribute to a conversation and communicate in general. It is still a language of its own, and should be viewed as so. It is a form of communication, and is just as important as learning Spanish or French or any other language.

The problem here is that not many people know sign language. It is mainly confined to the affected and the people who interact with them on a daily basis. So, a translator is required. Who better to translate than a computer? The goal of this implementation is to allow a person to convey their signs into a camera and the computer will translate it to text for the recipient to understand in real time. In the recent past, research has found that deep learning techniques are very effective in image processing applications. The images of each sign can be translated using an appropriate model. This implementation develops various deep learning models and evaluates their performance in the same task.

1.2. SCOPE

The objective of the project is to translate sign language expressions in real-time with the help of a deep learning model. Sign language gestures are translated into text.

CHAPTER 2

PROBLEM DEFINITION

CHAPTER 2 PROBLEM DEFINITION

Sign language is the means of communication for people who are speech and hearing impaired. However, people who cannot understand sign language will most definitely find it hard and even impossible to understand what another person is saying to them in sign language. The proposed model will help to find a solution that would help non-sign- language speakers to identify gestures used in Sign Language.

CHAPTER 3

LITERATURE REVIEW

CHAPTER 3 LITERATURE REVIEW

Literature review of the problem statement shows that there have been many attempts at sign language translation and for different sign languages as well. In [1] the authors scaled all images to 50x50 and trained it on a multi layered CNN with 5 layers and leaky ReLU activation. This proposed method used both static (0-9 and A-Z) and dynamic (alone, afraid, anger, etc.) gestures in training, validation and blind testing to make the model more robust. The blind test accuracy obtained in this proposed methodology was found to be 99.89%.

Md. Jahangir Hossein et al [2] published a paper on Recognition of Bengali Sign Language by first developing a CNN architecture consisting of 22 layers. Then proposed a vision based method for classification, featuring deep learning methods and adopted data augmentation. Finally, the authors used SIFT based methods with binary SVM classifiers. The dataset used in this implementation covers 50 sets, 36 necessary Bengali signs and was obtained with the cooperation of deaf volunteers of multiple organizations. The multilayer model achieved a 94.88% of accuracy, the adopted data augmentation achieved an accuracy of 92% and the SIFT based model accomplished an accuracy of about 98%.

Transfer learning is beneficial and can drastically reduce training periods and does not need the manual creation of models from scratch. Manual Eugenio Morocho Cayamcela et al. [3] used transfer learning and fine tuning methodologies applied to pre-trained networks AlexNet and GoogleNet that were trained on ImageNet dataset. The discriminative filters from the pre-trained models were reused on a custom dataset. Aditya Das et al. [4] proposed a model that used InceptionV3 which stacks various layers of a CNN model parallely instead of one on top of the other. This model achieved an average accuracy of 90% on static sign language.

Sign Language Recognition using deep learning and computer vision carried out by Kshitij Bantupali et al. [5] used an ensemble of CNN and LSTM. The CNN model, Inception was used to extract spatial features and LSTM whereas a RNN model was used to extract temporal features from the video sequences. Gesture segments identified and processed by CNN and classified by LSTM (sequence data). The results for varying sample sizes 10, 50, 100, 150 the accuracy of softmax were 90%, 92%, 93% and 91% and for Pool layer, accuracy obtained were 55%, 58%, 58% and 55% respectively.

In [6] Murat Taskiran et al. developed a model on 80:20 split and used skin colour and convex hull algorithm to determine the position of the hand. The model made use of a sequential network. Real time algorithm used in this implementation consists of extracting hand bound convex hull points and classifying hand images with CNN. This model was made to operate in real time and each character, 36 in total (10 numbers and 26 alphabets), were tested 10 times giving an accuracy of 98.05%.

Saleh Ahmad Khan et at. [7] made use of Scikit learn label encoding on their custom handmade dataset which provided less Root Mean Square error than One Hot Encoding method. They also used customized ROI segmentation to determine the region of interest for the hand to be captured. By using Scikit learn encoding, their model was able to achieve an accuracy of 86.4% to 97.54%. The customized ROI segmentation provided lesser computation work for locating the hand.

A multilayer Recurrent Neural Network for sign detection was proposed by Mark Borg et al. [8]. Features from a two stream Convolutional Neural Network taking video image data and motion data was taken as input. This proposed system consisting of a two stream RNN compares against the baseline research and was found to give at least a 18% increase against baseline accuracy of 78%.

In [9] Jinalee Jayeshkumar Raval et al. proposed a two segmented approach. First, the image processing is done by applying Gaussian blur to the region of interest, this forms a skin colour mask. Then a series of dilation and erosion were applied to enhance and smoothen the required features. CNN was used to identify the vivid features of the hand and ReLu was used to remove the negatives, Softmax was used to provide an accurate position. After training this model on an 85:15 split the model was tested on different illuminations and backgrounds in real time and provided an 83% accuracy.

Siming He et al. [10] made use of methods like Faster R-CNN used to recognise and locate the hand from the video input and a 3D CNN feature extraction network and a LSTM network based on RNN were constructed to improve the accuracy by learning the context of sign language. To improve the colour tone (RGB signs) problem, all the methods were combined to build a recognition model. Upon testing and training this fusion method in a real time environment the recognition rate is a high 99%.

The paper [11] chose four different models with convolutional neural networks as their basis. A simple CNN and three transfer learning models (VGG16, VGG19, and

InceptionV3) pre-trained on the Imagenet dataset. This model obtained an accuracy of 97% and above on all four models, which were trained on 100 epochs.

A paper on Deep Residual Networks by Kaiming He et al. [12] adopts batch normalization after each convolution and before each activation which were done after scale augmentation, with per pixel mean subtracted and standard colour augmentation. Resnet reduced error rate by 3.57% in top-5 error on test set compared to VGG V5 (6.8%) and GoogleNet (6.66%).

A three network pipeline was adopted by Shruti Mohanty et al. [13] which utilizes a dataset annotated with ground truth 3D key points for the signs. The networks employed in this implementation are HandSegnet, Posnet and Poseprior. An error rate of 0.58 with SVM classifier was observed, on extracting additional features with SVM a reduced error rate of 0.39 was obtained. Transfer learning approach reduced error rate even further to 0.37 and on using InceptionV3 a final error rate of 0.36 was obtained.

Gautham Jayadeep et al. [14] implemented a vision based classification system having dynamic ISL signs for bank purposes. This implementation made use of a self-customized dataset from ISL dictionary. CNN was used for feature extraction and passed on to an LSTM model and finally output the signs to text form. The dataset resulted in higher accuracy compared to other categories. It was observed that the model gave accuracy of 100% during training.

An Inception V3 network model with Tensorflow to retrain the final layers of the Inception model was adopted by Xiaoling Xia et al. [15] to classify 17 species of flowers. Transfer learning was used to keep parameters of the previous layer and remove the last layer of the Inception V3 model, then retrain the last layer. This methodology achieved an accuracy of 95% on Oxford-17 flower dataset and 94% on Oxford-102 dataset

CHAPTER 4

PROJECT DESCRIPTION

CHAPTER 4 PROJECT DESCRIPTION

4.1. PROPOSED DESIGN

The High-Level Design of the Sign Language Translator is as follows:

- Image Capture
- Preprocessing of Image for Input
- Recognition of Sign
- Display as Text

4.1.1. Image Capture

The images will be captured through the web camera and be applied to the next step, where they will be processed.

4.1.2. Pre-processing of Captured Image

From the images, the region of the hand will be recognized and then cropped. Then suitable processing of the image will be done to make it prepared for recognition.

4.1.3. Recognition of Sign

The preprocessed images are given as input to the system which will recognize the sign and give it as output.

4.1.4. Display as Text

The recognized text will then be shown to the user via the screen/interface

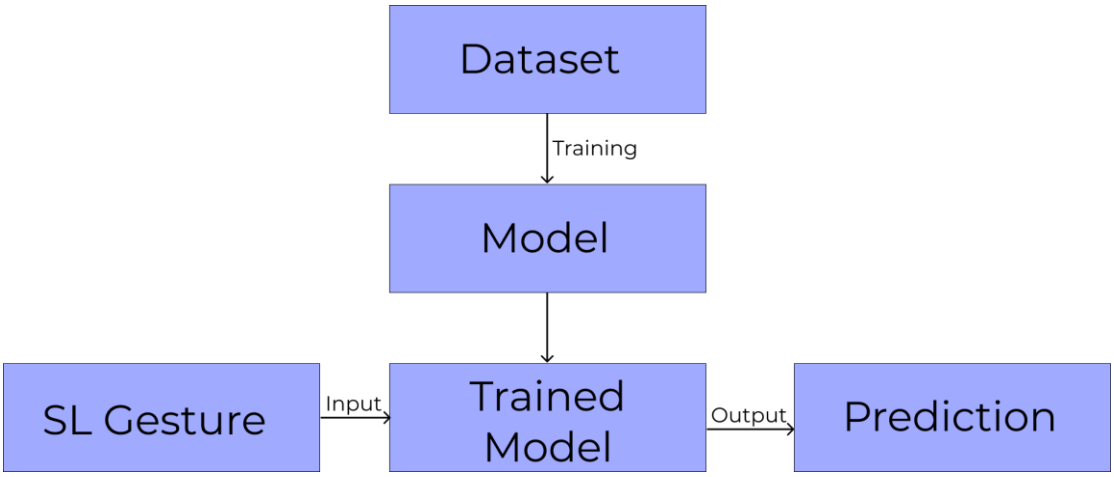

Figure 4.1 (a) High Level Design

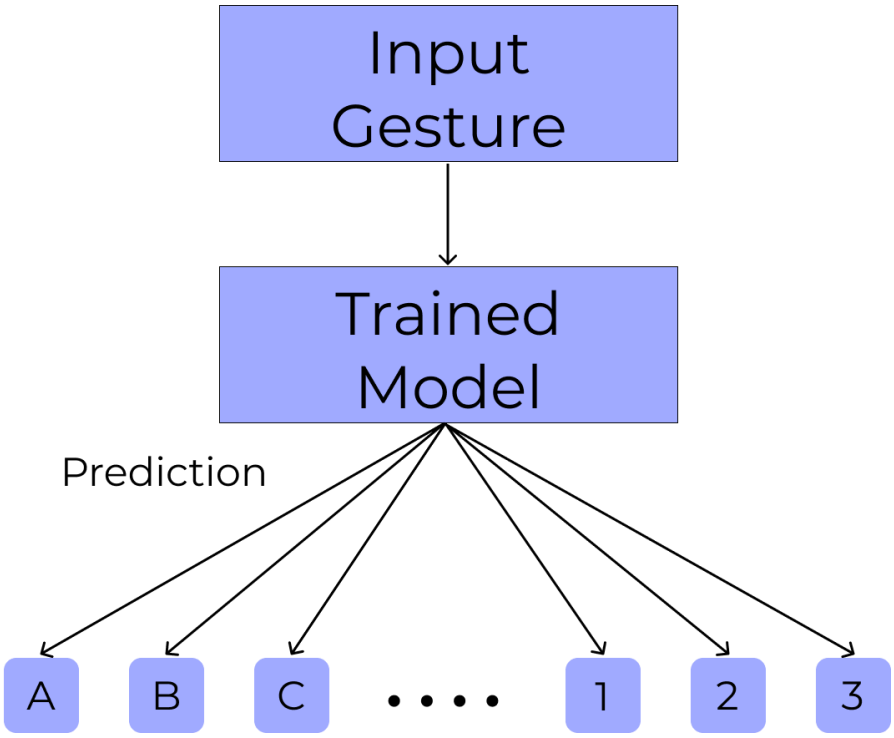

Figure 4.1 (b) Gesture Prediction

4.2. ASSUMPTIONS AND DEPENDENCIES

- The focus is mainly on fingerspelling where words are spelt out letter by letter.
- The focus is on American Sign Language (ASL).
- Data Set: The set contains 87,000 training images of size 200x200 pixels each. The images are segregated into 29 classes out of which 26 are for each letter of the alphabet. The remaining 3 classes are for ‘space’, ‘delete’ and ‘nothing’.

CHAPTER 5

REQUIREMENTS

CHAPTER 5 REQUIREMENTS

5.1. FUNCTIONAL REQUIREMENTS

- Making use of a suitable Deep Learning Architecture, which is effective and efficient in classifying sign language expressions to text.
- It must be able to recognize a set of signs made by hands and translate it into corresponding words or expressions.

5.2. NON-FUNCTIONAL REQUIREMENTS

- **User friendliness**

  The system should be easy to interact with and any user with no prior computer knowledge should be able to use it easily and effectively.

- **Performance and stability**

  The system should be stable and performance should be smooth with easy accessibility.

- **Reliability, availability, maintainability**

  Should be reliable so proper assessments are done with proper output, should be readily available and should be easy to maintain so future software handlers should also be able to continue to operate and update as and when required.

- **Robustness**

  The system should be able to recover if any problems occur.

- **User satisfaction**

  The system should meet the users' expectations.

5.3. SOFTWARE REQUIREMENTS

1. Operating System: Windows/Ubuntu/MacOS
2. Web Browser- Google Collab and Kaggle Notebooks
3. TensorFlow, Keras, Pandas and Scikit Learn

5.4. HARDWARE REQUIREMENTS

1. Processor: Intel Core i3 and above
2. RAM: 4GB and above
3. Internet Connection

CHAPTER 6

METHODOLOGY

CHAPTER 6 METHODOLOGY

6.1. DATASET

- The data set is a collection of images of alphabets from the American Sign Language, separated in 29 folders which represent the various classes.
- The training data set contains 87,000 images which are 200x200 pixels. There are 29 classes, of which 26 are for the letters A-Z and 3 classes for SPACE, DELETE and NOTHING.
- The 3 classes are very helpful in real-time applications and classification. Each class has 3000 image samples.
- The test data set contains a mere 29 images, to encourage the use of real-world test images.
- The dataset includes image samples with variations in lighting conditions and skin tones as well to help make the model more robust.

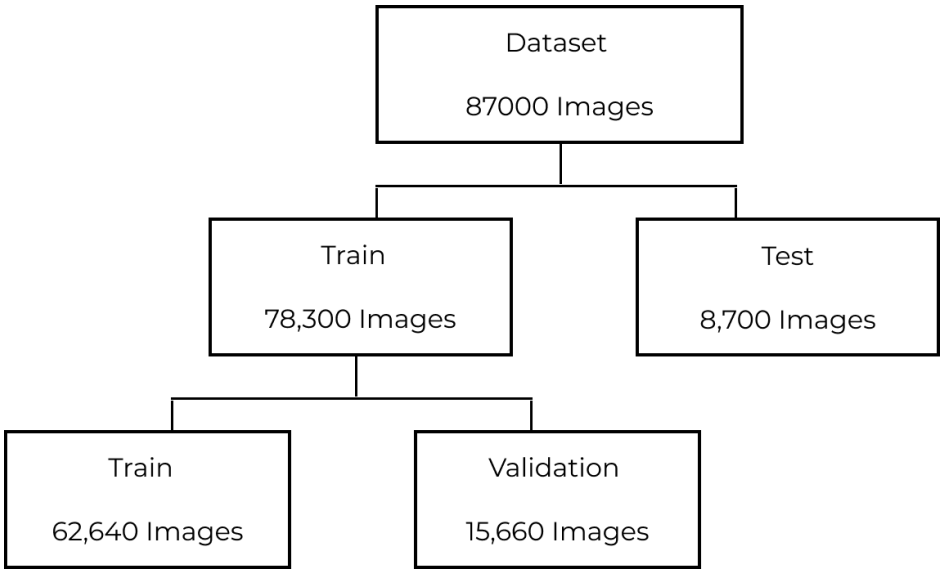

Figure 6.1 Dataset Split

6.2. DEEP LEARNING MODELS

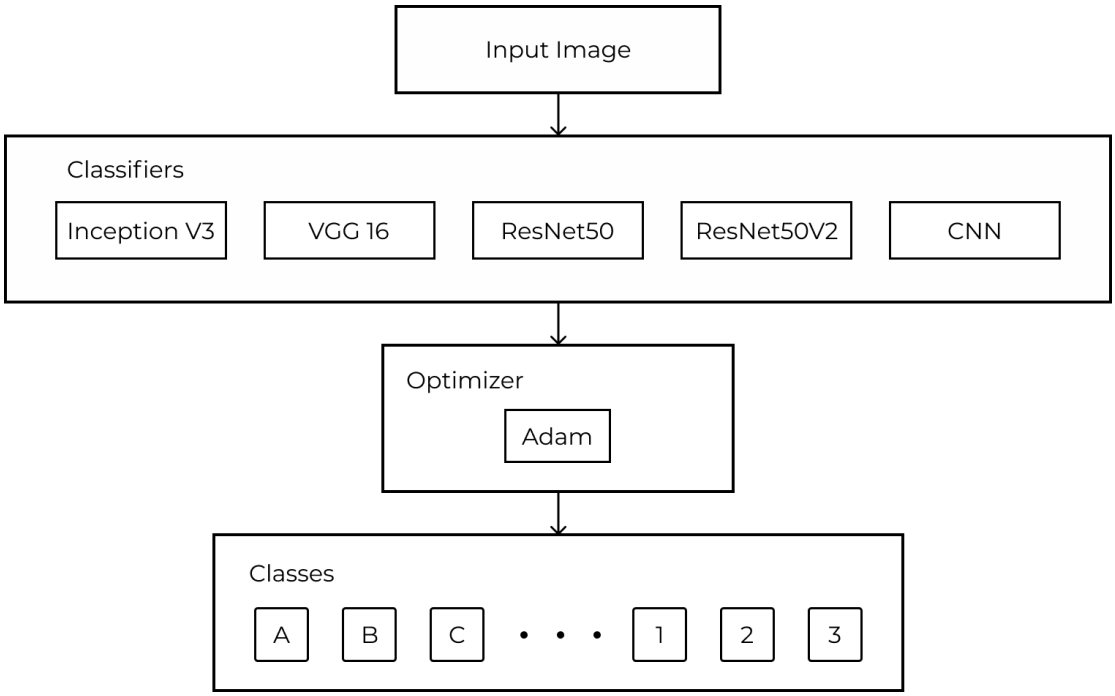

Figure 6.2 Model Architecture

6.2.1. Convolutional Neural Networks

- Convolutional Neural Networks popularly known as CNNs are one of the oldest deep neural networks that are widely used in computer vision tasks.
- They are composed of multiple building blocks such as, convolutional layers, pooling layers and fully connected layers.
- CNNs take in an input image, assign importance to different aspects and features within the image and are able to distinguish such images from others.
- CNNs form the basis of the various deep learning models that are implemented for this application.

6.2.2 Transfer Learning

- Transfer Learning is the reuse of a model on a problem that is different from the original one that it was previously developed and trained for.
- Transfer Learning allows us to use previously gained knowledge to newer problems without having to start from scratch.
- The benefits include shorter training times to develop models that work well in most cases.

6.2.3. Models Implemented

6.2.3.1 ResNet50

ResNet-50 is a deep residual network. The "50" refers to the number of layers it has. It's a subclass of convolutional neural networks, with ResNet most popularly used for image classification.

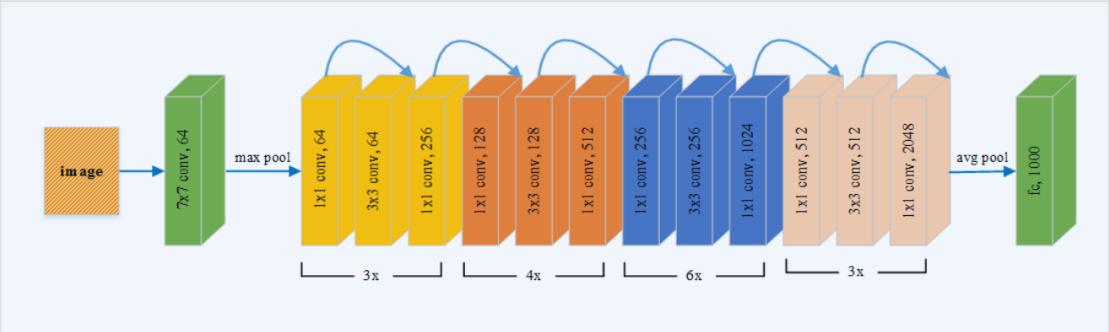

Figure 6.2.3 (a) ResNet50 architecture

6.2.3.2 ResNet50V2

ResNet50V2 is a modified version of ResNet50 that performs better than ResNet50 and ResNet101 on the ImageNet dataset. In ResNet50V2, a modification was made in the propagation formulation of the connections between blocks. ResNet50V2 also achieves a good result on the ImageNet dataset.

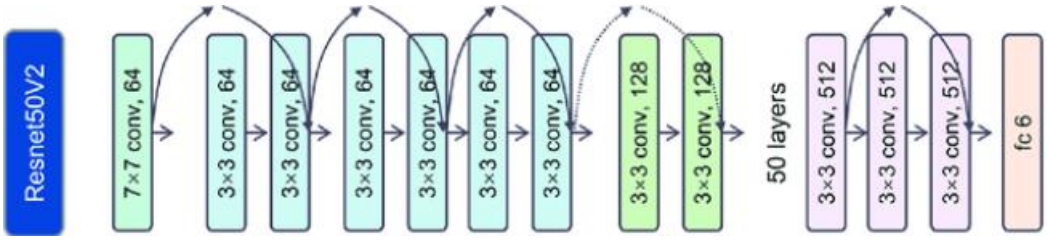

Figure 6.2.3 (b) ResNet50V2 architecture

6.2.3.3 Inception V3

The Inception V3 is a deep learning model based on Convolutional Neural Networks, which is used for image classification. The inception V3 is a superior version of the basic model Inception V1 which was introduced as GoogLeNet in 2014. As the name suggests it was developed by a team at Google.

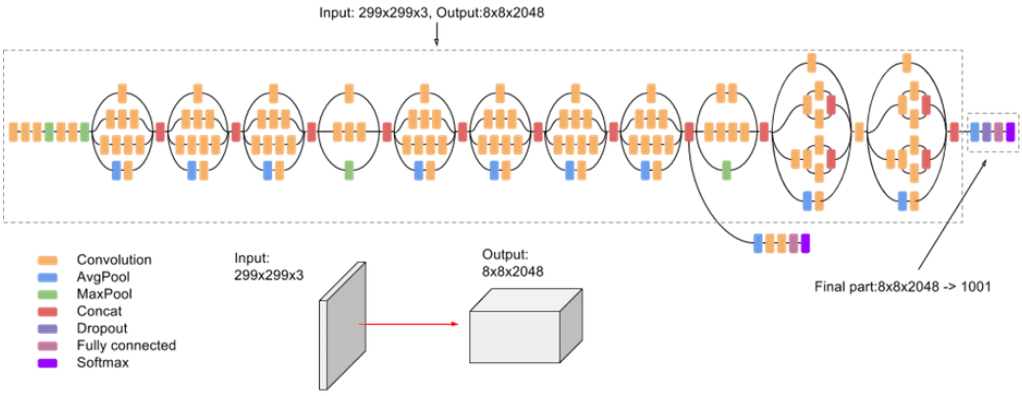

Figure 6.2.3 (c) InceptionV3 architecture

6.2.3.4 VGG16

VGG16 is a simple and widely used Convolutional Neural Network (CNN) Architecture used for ImageNet, a large visual database project used in visual object recognition software research. It won ILSVR(Imagenet) in 2014.

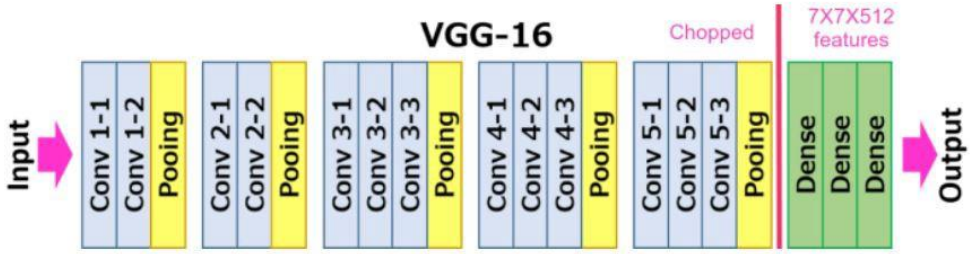

Figure 6.2.3 (d) VGG-16 architecture

The transfer learning models are developed using the above pre-trained models and the weights obtained on training them on the ImageNet dataset. The proposed approach involves removing the existing fully connected layers from the above pre-trained models and replacing them with another set that would cater to this specific application. The base models would be frozen and hence would not be able to update weights. The models are trained on images from the specified dataset.

Fine-tuning these pre-existing models in this fashion allows us to classify images for our specific application.

CHAPTER 7

EXPERIMENTATION

CHAPTER 7 EXPERIMENTATION

- The models used here are Resnet50, Resnet50V2, Inception V3, VGG16 and a simple sequential CNN. These models consist of an input layer followed by a combination of layers like 2D convolution, max pooling, batch normalization, activation, pooling, flatten, drop out and dense. These models were trained independently on a dataset of 87,000 images. Using Scikit-Learn the dataset was split into training, testing and validation sets. Initially, a 90:10 split was made, 10% of which was used for testing (8700). The remaining 90% (78300) was further split in the ratio 80:20. 80% of the split was used for training (62640) and the remaining 20% for testing (15660). We evaluate our models independently using the same training and test samples, this ensures that the model's accuracy can be verified on equal grounds.
- During initial implementation, the RAM provided by Google Colaboratory (Free Tier) was getting completely utilized, resulting in premature halting of model fitting. Hence, Google Colab was replaced with Kaggle notebooks (Free Tier). The specifications provided by Kaggle's free tier access includes an Intel Xeon CPU clocked at 2.2 or 2.3GHz with 16.4 GB of CPU RAM along with an NVIDIA Tesla P100 GPU with 17.1GB of GPU RAM, which drastically decreases the waiting periods for model fitting. Kaggle's environment was used due to its much faster runtime and high shared memory. Kaggle also has a much easier way to work with datasets and allows direct use of them if they are readily available on Kaggle.
- Low accuracy of models:- This was due to incorrect model fitting. This was easily fixed by making minor adjustments to how the model was using the dataset.
- Dataset Split:- Initially since the dataset split was not random. The last few image samples were used for validation (which were taken under darker lighting conditions compared to the rest of the set), which resulted in a very low validation accuracy. Compiling the dataset into a Pandas Dataframe and splitting it with the help of the Scikit Learn 'train_test_split' method helped overcome this issue.

CHAPTER 8

TESTING AND RESULTS

CHAPTER 8 TESTING AND RESULTS

The dataset consists of 87,000 images divided into 29 classes. The were 26 classes, one for each letter of the alphabet and 3 additional classes for “space”, “nothing” and “delete”. The models were trained on 62640 images. 15660 images were used for validation and 8700 images of the dataset were used for testing. Kaggle Notebooks were used for training, validating and testing the models. They were trained for 10 epochs each. Graphs depicting the validation accuracy against training accuracy and validation loss against training loss were obtained. Confusion matrices were also plotted by comparing the values obtained from the models with the ground truths.

8.1. RESNET50

```
Epoch 1/10
2022-04-07 15:24:40.627330: I tensorflow/stream_executor/cuda/cuda_dnn.cc:369] Loaded cuDNN version 8005
3915/3915 [==============================] - 411s 103ms/step - loss: 2.8156 - accuracy: 0.2100 - val_loss: 2.3302 - val_accuracy: 0.3562
Epoch 2/10
3915/3915 [==============================] - 140s 36ms/step - loss: 2.2132 - accuracy: 0.3544 - val_loss: 1.9428 - val_accuracy: 0.4676
Epoch 3/10
3915/3915 [==============================] - 140s 36ms/step - loss: 1.9429 - accuracy: 0.4202 - val_loss: 1.7362 - val_accuracy: 0.5077
Epoch 4/10
3915/3915 [==============================] - 141s 36ms/step - loss: 1.7736 - accuracy: 0.4617 - val_loss: 1.5825 - val_accuracy: 0.5453
Epoch 5/10
3915/3915 [==============================] - 140s 36ms/step - loss: 1.6440 - accuracy: 0.4991 - val_loss: 1.4724 - val_accuracy: 0.5713
Epoch 6/10
3915/3915 [==============================] - 140s 36ms/step - loss: 1.5428 - accuracy: 0.5271 - val_loss: 1.3988 - val_accuracy: 0.5884
Epoch 7/10
3915/3915 [==============================] - 141s 36ms/step - loss: 1.4646 - accuracy: 0.5479 - val_loss: 1.3088 - val_accuracy: 0.6144
Epoch 8/10
3915/3915 [==============================] - 149s 38ms/step - loss: 1.3994 - accuracy: 0.5637 - val_loss: 1.2583 - val_accuracy: 0.6242
Epoch 9/10
3915/3915 [==============================] - 143s 37ms/step - loss: 1.3409 - accuracy: 0.5838 - val_loss: 1.2019 - val_accuracy: 0.6377
Epoch 10/10
3915/3915 [==============================] - 142s 36ms/step - loss: 1.2905 - accuracy: 0.5935 - val_loss: 1.1584 - val_accuracy: 0.6495
```

Figure 8.1 (a) ResNet50 Model Fitting

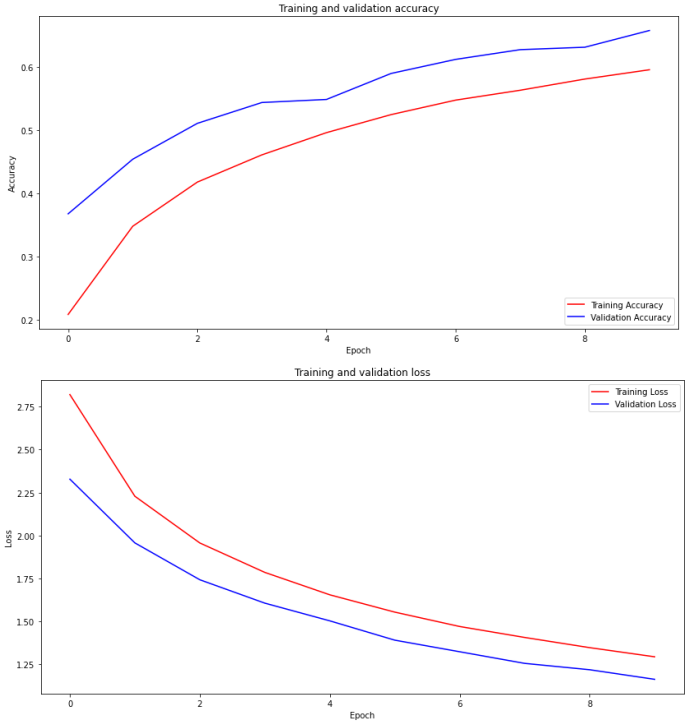

Figure 8.1 (b) ResNet50 Training and Validation Accuracy and Loss

	precision	recall	f1-score	support
A	0.68	0.79	0.73	300
B	0.71	0.75	0.73	300
C	0.92	0.88	0.90	300
D	0.90	0.75	0.82	300
E	0.83	0.57	0.68	300
F	0.75	0.80	0.78	300
G	0.86	0.69	0.77	300
H	0.74	0.94	0.83	300
I	0.73	0.67	0.70	300
J	0.68	0.85	0.76	300
K	0.64	0.77	0.70	300
L	0.85	0.79	0.82	300
M	0.66	0.43	0.52	300
N	0.59	0.66	0.62	300
O	0.93	0.49	0.64	300
P	0.74	0.78	0.76	300
Q	0.84	0.69	0.76	300
R	0.75	0.30	0.43	300
S	0.70	0.41	0.52	300
T	0.70	0.42	0.52	300
U	0.67	0.35	0.46	300
V	0.46	0.42	0.44	300
W	0.29	0.64	0.40	300
X	0.56	0.60	0.58	300
Y	0.34	0.71	0.46	300
Z	0.68	0.51	0.58	300
del	0.81	0.65	0.72	300
nothing	0.88	0.97	0.92	300
space	0.38	0.64	0.47	300
accuracy			0.65	8700
macro avg	0.70	0.65	0.66	8700
weighted avg	0.70	0.65	0.66	8700

Figure 8.1 (c) ResNet50 Precision, Recall, F1-Score and Support

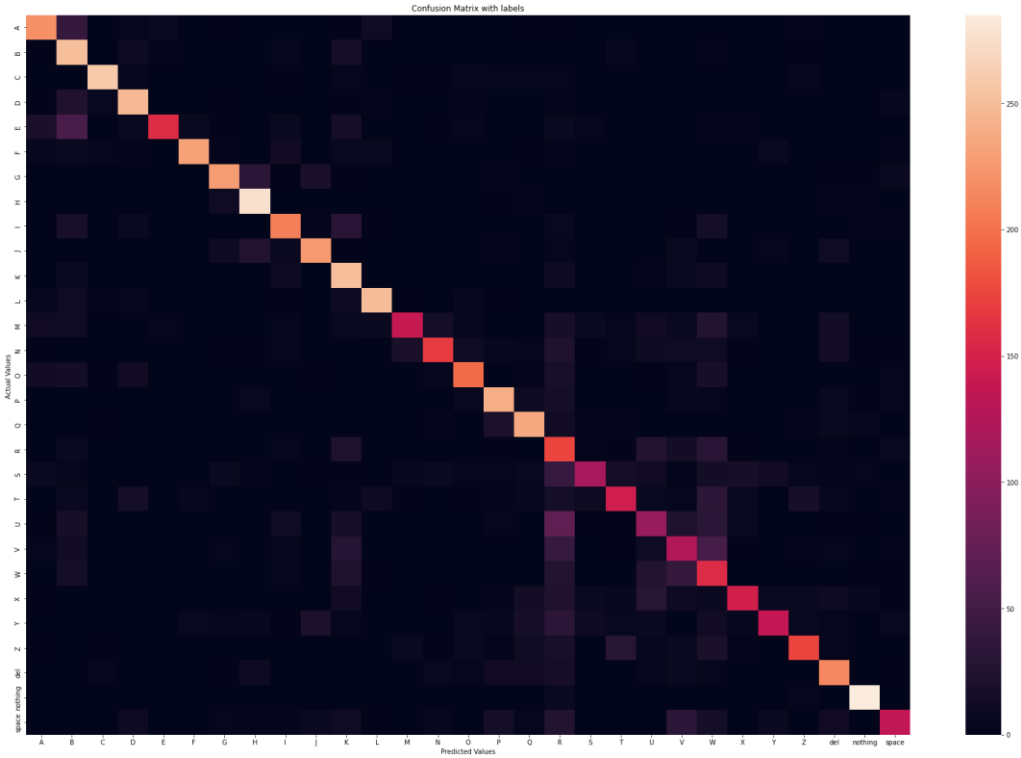

Figure 8.1 (d) ResNet50 Confusion Matrix

8.2. RESNET50V2

```
Epoch 1/10

2022-04-26 06:53:43.436137: I tensorflow/stream_executor/cuda/cuda_dnn.cc:369] Loaded cu
DNN version 8005

490/490 [==============================] - 437s 873ms/step - loss: 1.1537 - accuracy: 0.
6806 - val_loss: 0.3341 - val_accuracy: 0.9250
Epoch 2/10
490/490 [==============================] - 134s 274ms/step - loss: 0.3393 - accuracy: 0.
9031 - val_loss: 0.1564 - val_accuracy: 0.9676
Epoch 3/10
490/490 [==============================] - 136s 278ms/step - loss: 0.1940 - accuracy: 0.
9470 - val_loss: 0.0989 - val_accuracy: 0.9805
Epoch 4/10
490/490 [==============================] - 135s 275ms/step - loss: 0.1237 - accuracy: 0.
9680 - val_loss: 0.0676 - val_accuracy: 0.9859
Epoch 5/10
490/490 [==============================] - 136s 278ms/step - loss: 0.0883 - accuracy: 0.
9774 - val_loss: 0.0486 - val_accuracy: 0.9903
Epoch 6/10
490/490 [==============================] - 137s 279ms/step - loss: 0.0670 - accuracy: 0.
9841 - val_loss: 0.0387 - val_accuracy: 0.9927
Epoch 7/10
490/490 [==============================] - 136s 277ms/step - loss: 0.0522 - accuracy: 0.
9876 - val_loss: 0.0329 - val_accuracy: 0.9937
Epoch 8/10
490/490 [==============================] - 137s 280ms/step - loss: 0.0435 - accuracy: 0.
9894 - val_loss: 0.0261 - val_accuracy: 0.9944
Epoch 9/10
490/490 [==============================] - 139s 284ms/step - loss: 0.0354 - accuracy: 0.
9920 - val_loss: 0.0222 - val_accuracy: 0.9947
Epoch 10/10
490/490 [==============================] - 137s 279ms/step - loss: 0.0286 - accuracy: 0.
9937 - val_loss: 0.0211 - val_accuracy: 0.9954
```

Figure 8.2 (a) ResNet50V2 Model Fitting

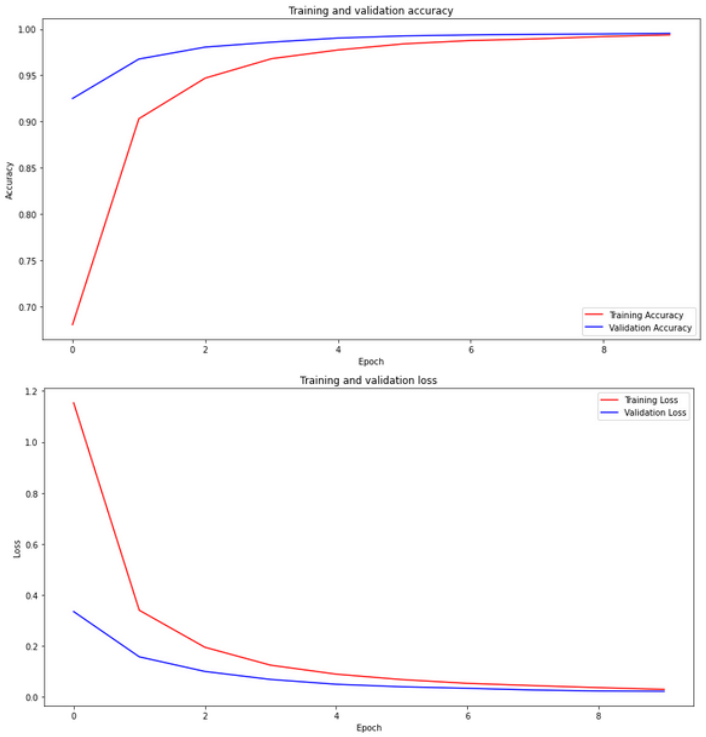

Figure 8.2 (b) ResNet50V2 Training and Validation Accuracy and Loss

	precision	recall	f1-score	support
A	1.00	1.00	1.00	300
B	0.99	1.00	1.00	300
C	1.00	1.00	1.00	300
D	1.00	1.00	1.00	300
E	1.00	0.99	0.99	300
F	1.00	1.00	1.00	300
G	1.00	0.99	0.99	300
H	0.99	1.00	1.00	300
I	1.00	1.00	1.00	300
J	1.00	1.00	1.00	300
K	1.00	0.99	0.99	300
L	1.00	1.00	1.00	300
M	1.00	1.00	1.00	300
N	0.99	0.99	0.99	300
O	1.00	0.99	0.99	300
P	1.00	0.99	0.99	300
Q	1.00	1.00	1.00	300
R	0.99	0.99	0.99	300
S	0.99	1.00	0.99	300
T	0.99	1.00	1.00	300
U	0.99	0.99	0.99	300
V	0.99	0.98	0.98	300
W	0.98	0.99	0.99	300
X	1.00	1.00	1.00	300
Y	1.00	0.99	1.00	300
Z	1.00	1.00	1.00	300
del	1.00	1.00	1.00	300
nothing	1.00	1.00	1.00	300
space	1.00	1.00	1.00	300
accuracy			1.00	8700
macro avg	1.00	1.00	1.00	8700
weighted avg	1.00	1.00	1.00	8700

Figure 8.2 (c) ResNet50V2 Precision, Recall, F1-Score and Support

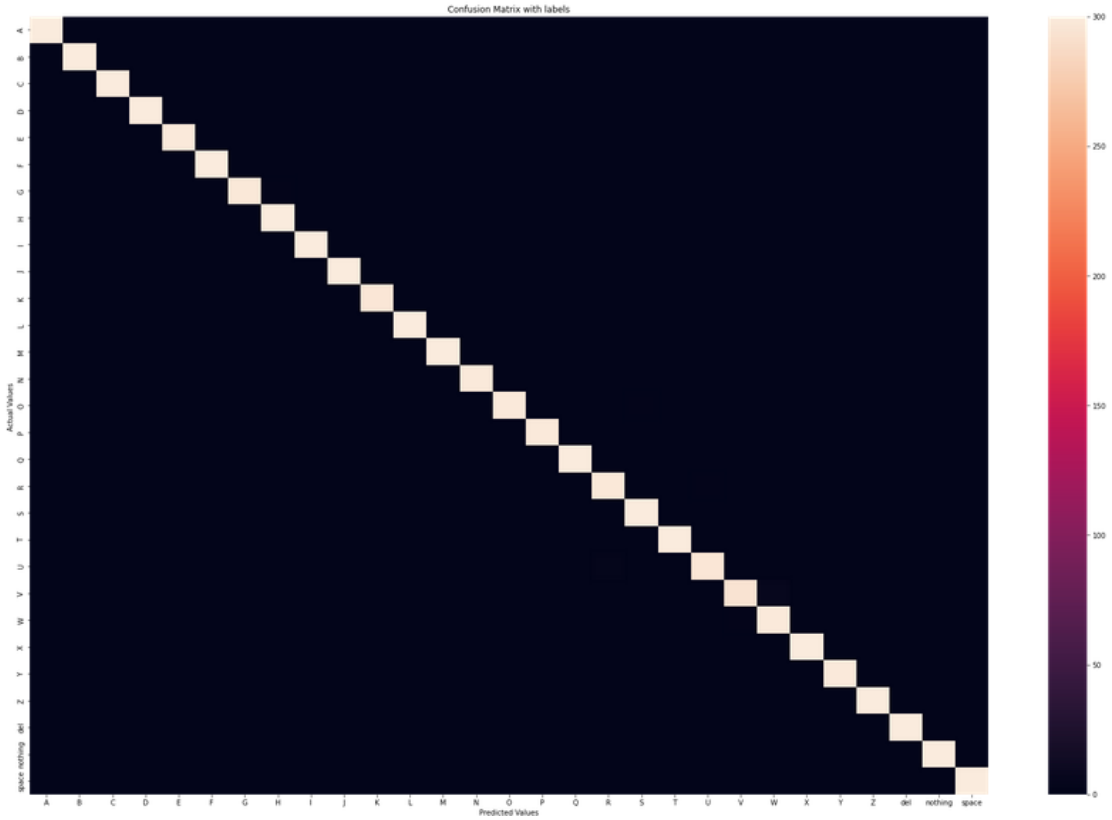

Figure 8.2 (d) ResNet50V2 Confusion Matrix

8.3. INCEPTION V3

```
Epoch 1/10
2022-04-07 14:35:36.015967: I tensorflow/stream_executor/cuda/cuda_dnn.cc:369] Loaded cuDNN version 8005
3915/3915 [==============================] - 489s 123ms/step - loss: 1.3821 - accuracy: 0.6161 - val_loss: 0.5539 - val_accuracy: 0.8673
Epoch 2/10
3915/3915 [==============================] - 158s 40ms/step - loss: 0.5349 - accuracy: 0.8481 - val_loss: 0.3014 - val_accuracy: 0.9255
Epoch 3/10
3915/3915 [==============================] - 153s 39ms/step - loss: 0.3350 - accuracy: 0.9052 - val_loss: 0.2048 - val_accuracy: 0.9495
Epoch 4/10
3915/3915 [==============================] - 153s 39ms/step - loss: 0.2366 - accuracy: 0.9333 - val_loss: 0.1505 - val_accuracy: 0.9651
Epoch 5/10
3915/3915 [==============================] - 153s 39ms/step - loss: 0.1788 - accuracy: 0.9512 - val_loss: 0.1174 - val_accuracy: 0.9720
Epoch 6/10
3915/3915 [==============================] - 152s 39ms/step - loss: 0.1409 - accuracy: 0.9612 - val_loss: 0.0974 - val_accuracy: 0.9753
Epoch 7/10
3915/3915 [==============================] - 152s 39ms/step - loss: 0.1139 - accuracy: 0.9687 - val_loss: 0.0824 - val_accuracy: 0.9792
Epoch 8/10
3915/3915 [==============================] - 154s 39ms/step - loss: 0.0945 - accuracy: 0.9747 - val_loss: 0.0705 - val_accuracy: 0.9823
Epoch 9/10
3915/3915 [==============================] - 154s 39ms/step - loss: 0.0803 - accuracy: 0.9785 - val_loss: 0.0632 - val_accuracy: 0.9833
Epoch 10/10
3915/3915 [==============================] - 155s 40ms/step - loss: 0.0688 - accuracy: 0.9821 - val_loss: 0.0543 - val_accuracy: 0.9861
```

Figure 8.3 (a) InceptionV3 Model Fitting

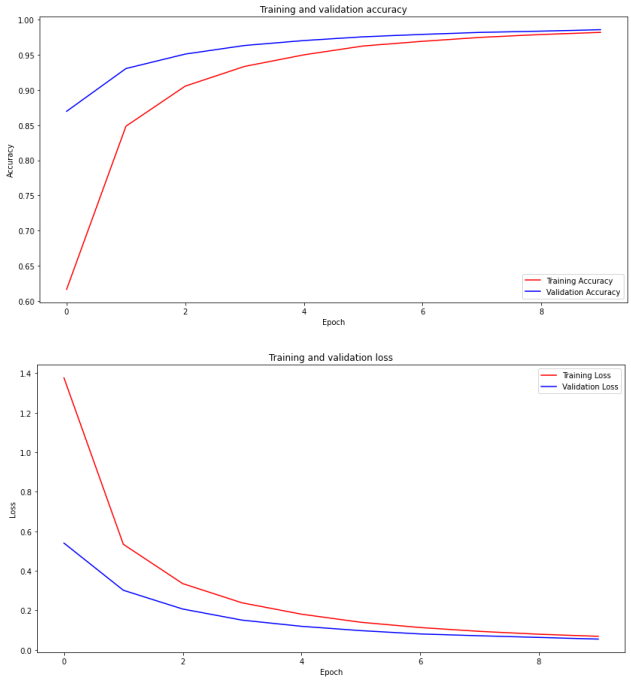

Figure 8.3 (b) InceptionV3 Training and Validation Accuracy and Loss

	precision	recall	f1-score	support
A	1.00	0.98	0.99	300
B	0.98	0.99	0.99	300
C	1.00	0.99	0.99	300
D	0.99	1.00	1.00	300
E	1.00	0.99	0.99	300
F	1.00	0.98	0.99	300
G	1.00	0.98	0.99	300
H	0.99	1.00	0.99	300
I	0.98	0.99	0.99	300
J	0.99	0.99	0.99	300
K	0.97	0.99	0.98	300
L	0.99	1.00	0.99	300
M	0.97	0.99	0.98	300
N	0.99	0.99	0.99	300
O	1.00	0.98	0.99	300
P	0.99	0.99	0.99	300
Q	1.00	1.00	1.00	300
R	0.96	0.96	0.96	300
S	1.00	0.98	0.99	300
T	0.99	0.98	0.98	300
U	0.95	0.96	0.96	300
V	0.97	0.96	0.97	300
W	0.98	0.98	0.98	300
X	0.97	0.99	0.98	300
Y	0.99	0.99	0.99	300
Z	1.00	0.98	0.99	300
del	0.99	1.00	0.99	300
nothing	1.00	1.00	1.00	300
space	1.00	1.00	1.00	300
accuracy			0.99	8700
macro avg	0.99	0.99	0.99	8700
weighted avg	0.99	0.99	0.99	8700

Figure 8.3 (c) InceptionV3 Precision, Recall, F1-Score and Support

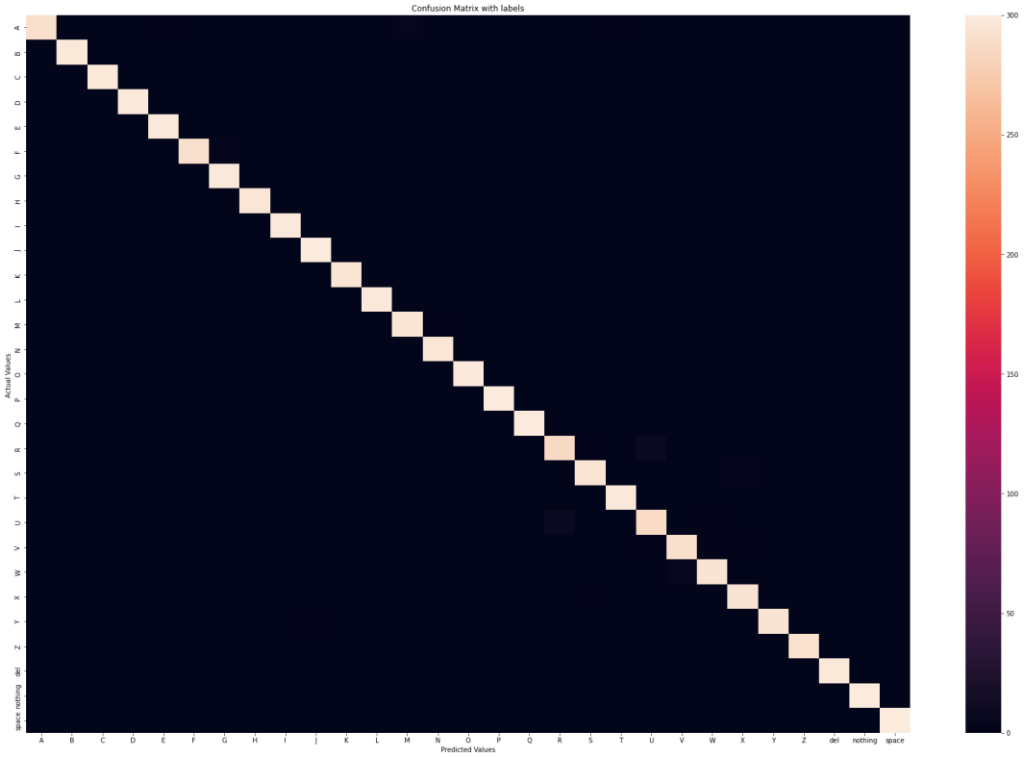

Figure 8.3 (d) InceptionV3 Confusion Matrix

8.4. VGG16

```
Epoch 1/10

2022-04-06 15:20:42.884305: I tensorflow/stream_executor/cuda/cuda_dnn.cc:369] Loaded cuDNN version 8005

490/490 [==============================] - 386s 774ms/step - loss: 2.1085 - accuracy: 0.4608 - val_loss: 1.1428 - val_accuracy: 0.
8015
Epoch 2/10
490/490 [==============================] - 114s 232ms/step - loss: 0.9893 - accuracy: 0.7614 - val_loss: 0.6529 - val_accuracy: 0.
8897
Epoch 3/10
490/490 [==============================] - 115s 234ms/step - loss: 0.6508 - accuracy: 0.8449 - val_loss: 0.4500 - val_accuracy: 0.
9237
Epoch 4/10
490/490 [==============================] - 114s 232ms/step - loss: 0.4731 - accuracy: 0.8895 - val_loss: 0.3331 - val_accuracy: 0.
9425
Epoch 5/10
490/490 [==============================] - 115s 236ms/step - loss: 0.3704 - accuracy: 0.9156 - val_loss: 0.2611 - val_accuracy: 0.
9577
Epoch 6/10
490/490 [==============================] - 113s 231ms/step - loss: 0.2974 - accuracy: 0.9338 - val_loss: 0.2063 - val_accuracy: 0.
9656
Epoch 7/10
490/490 [==============================] - 114s 233ms/step - loss: 0.2435 - accuracy: 0.9467 - val_loss: 0.1678 - val_accuracy: 0.
9734
Epoch 8/10
490/490 [==============================] - 114s 233ms/step - loss: 0.2039 - accuracy: 0.9558 - val_loss: 0.1371 - val_accuracy: 0.
9805
Epoch 9/10
490/490 [==============================] - 115s 234ms/step - loss: 0.1735 - accuracy: 0.9638 - val_loss: 0.1152 - val_accuracy: 0.
9826
Epoch 10/10
490/490 [==============================] - 115s 234ms/step - loss: 0.1477 - accuracy: 0.9691 - val_loss: 0.0971 - val_accuracy: 0.
9867
```

Figure 8.4 (a) VGG16 Model Fitting

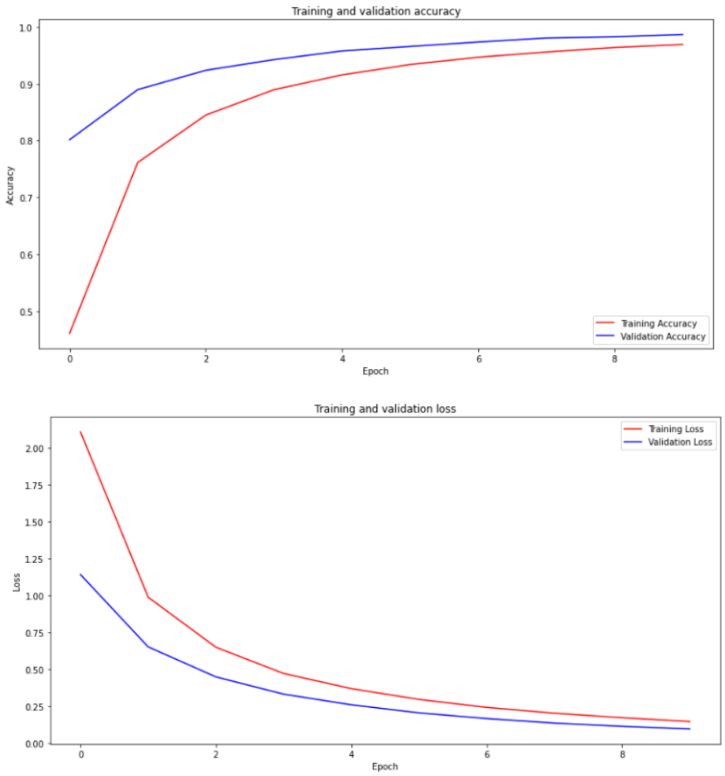

Figure 8.4 (b) VGG16 Training and Validation Accuracy and Loss

```
68/68 [==============================] - 35s 518ms/step
              precision    recall  f1-score   support

           A       0.99      0.99      0.99       300
           B       0.99      0.99      0.99       300
           C       0.99      1.00      1.00       300
           D       0.99      1.00      1.00       300
           E       0.99      0.94      0.97       300
           F       1.00      0.99      1.00       300
           G       1.00      0.99      0.99       300
           H       0.99      1.00      1.00       300
           I       0.98      0.99      0.99       300
           J       1.00      1.00      1.00       300
           K       1.00      0.99      0.99       300
           L       1.00      1.00      1.00       300
           M       0.98      0.97      0.97       300
           N       0.97      0.98      0.98       300
           O       1.00      0.99      1.00       300
           P       0.99      1.00      1.00       300
           Q       1.00      1.00      1.00       300
           R       0.96      0.97      0.97       300
           S       0.96      0.95      0.95       300
           T       0.99      0.99      0.99       300
           U       0.94      0.98      0.96       300
           V       0.96      0.96      0.96       300
           W       0.99      0.98      0.98       300
           X       0.95      0.98      0.97       300
           Y       1.00      0.99      0.99       300
           Z       0.99      0.99      0.99       300
         del       1.00      1.00      1.00       300
     nothing       1.00      1.00      1.00       300
       space       1.00      1.00      1.00       300

    accuracy                           0.99      8700
   macro avg       0.99      0.99      0.99      8700
weighted avg       0.99      0.99      0.99      8700
```

Figure 8.4 (c) VGG16 Precision, Recall, F1-Score and Support

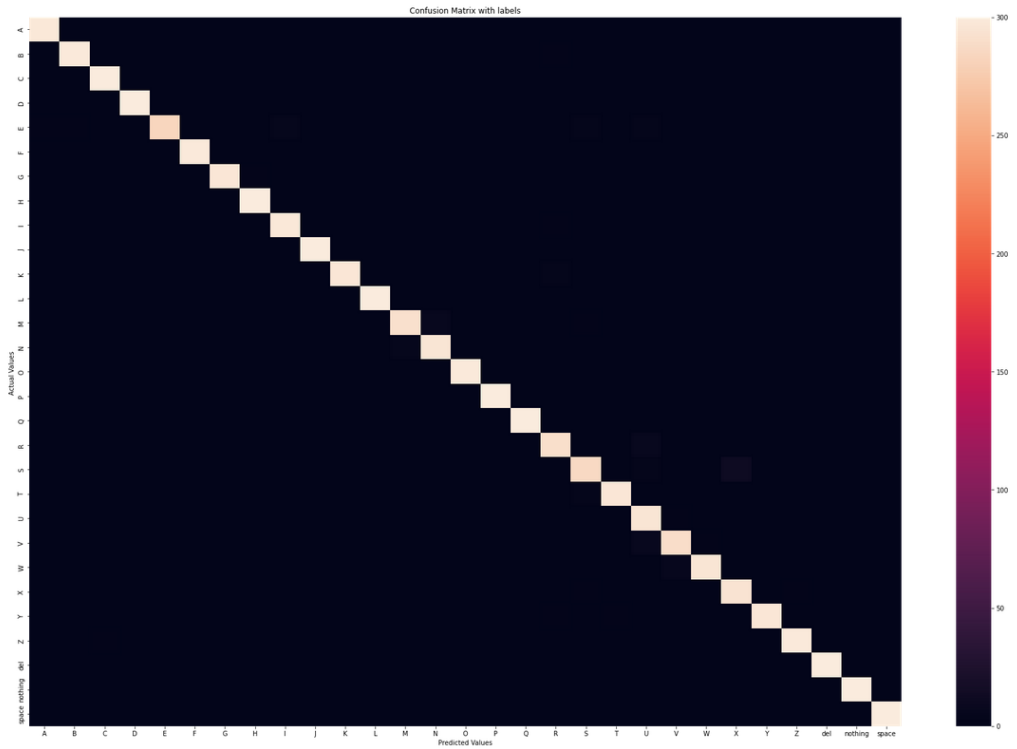

Figure 8.4 (d) VGG16 Confusion Matrix

8.5. SIMPLE CNN

```
Epoch 1/10

2022-04-26 13:37:09.458458: I tensorflow/stream_executor/cuda/cuda_dnn.cc:369] Loaded cu
DNN version 8005

490/490 [==============================] - 417s 838ms/step - loss: 2.7570 - accuracy: 0.
2275 - val_loss: 2.0572 - val_accuracy: 0.3990
Epoch 2/10
490/490 [==============================] - 121s 248ms/step - loss: 1.6379 - accuracy: 0.
5248 - val_loss: 1.3132 - val_accuracy: 0.6191
Epoch 3/10
490/490 [==============================] - 118s 242ms/step - loss: 1.0958 - accuracy: 0.
6817 - val_loss: 0.9552 - val_accuracy: 0.7211
Epoch 4/10
490/490 [==============================] - 119s 242ms/step - loss: 0.8016 - accuracy: 0.
7699 - val_loss: 0.7400 - val_accuracy: 0.7793
Epoch 5/10
490/490 [==============================] - 122s 249ms/step - loss: 0.6123 - accuracy: 0.
8263 - val_loss: 0.5738 - val_accuracy: 0.8312
Epoch 6/10
490/490 [==============================] - 119s 243ms/step - loss: 0.4827 - accuracy: 0.
8628 - val_loss: 0.4491 - val_accuracy: 0.8731
Epoch 7/10
490/490 [==============================] - 118s 242ms/step - loss: 0.3908 - accuracy: 0.
8898 - val_loss: 0.3676 - val_accuracy: 0.8954
Epoch 8/10
490/490 [==============================] - 120s 245ms/step - loss: 0.3215 - accuracy: 0.
9097 - val_loss: 0.3394 - val_accuracy: 0.8981
Epoch 9/10
490/490 [==============================] - 120s 245ms/step - loss: 0.2616 - accuracy: 0.
9285 - val_loss: 0.2528 - val_accuracy: 0.9281
Epoch 10/10
490/490 [==============================] - 120s 245ms/step - loss: 0.2261 - accuracy: 0.
9371 - val_loss: 0.2096 - val_accuracy: 0.9439
```

Figure 8.5 (a) CNN Model Fitting

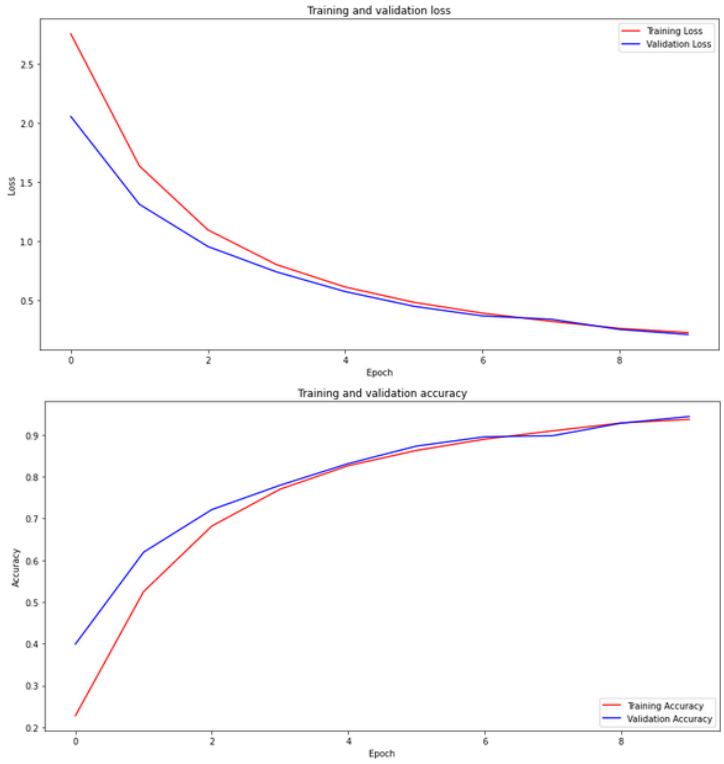

Figure 8.5 (b) CNN Training and Validation Accuracy and Loss

	precision	recall	f1-score	support
A	0.90	0.96	0.93	300
B	0.95	0.96	0.96	300
C	0.99	0.98	0.98	300
D	0.96	0.96	0.96	300
E	0.95	0.87	0.91	300
F	0.97	0.98	0.98	300
G	0.99	0.99	0.99	300
H	0.98	0.99	0.98	300
I	0.94	0.97	0.95	300
J	0.99	0.95	0.97	300
K	0.97	0.93	0.95	300
L	0.96	0.99	0.98	300
M	0.84	0.93	0.88	300
N	0.96	0.87	0.91	300
O	0.94	0.96	0.95	300
P	0.99	0.95	0.97	300
Q	0.94	0.99	0.97	300
R	0.94	0.84	0.89	300
S	0.92	0.92	0.92	300
T	0.95	0.89	0.92	300
U	0.82	0.90	0.86	300
V	0.85	0.87	0.86	300
W	0.90	0.90	0.90	300
X	0.91	0.91	0.91	300
Y	0.92	0.95	0.93	300
Z	0.98	0.97	0.98	300
del	0.98	0.97	0.97	300
nothing	0.99	1.00	1.00	300
space	0.97	0.97	0.97	300
accuracy			0.94	8700
macro avg	0.94	0.94	0.94	8700
weighted avg	0.94	0.94	0.94	8700

Figure 8.5 (c) CNN Precision, Recall, F1-Score and Support

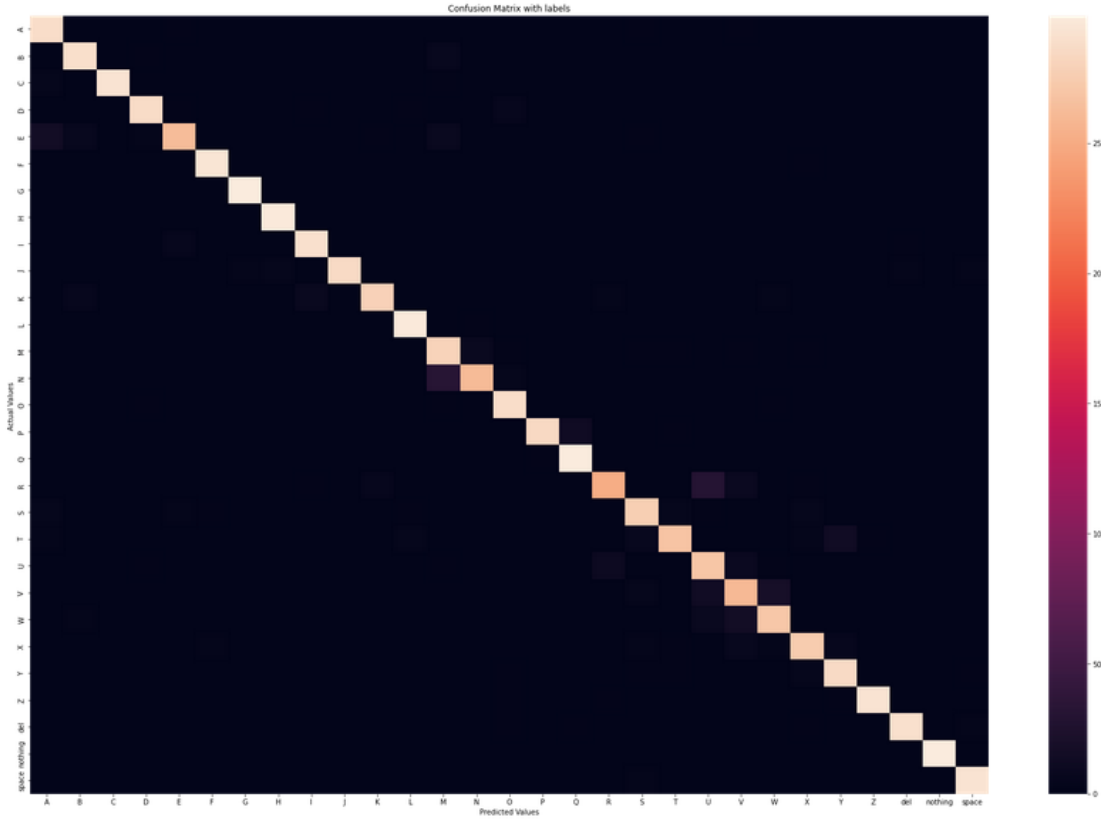

Figure 8.5 (d) CNN Confusion Matrix

The results obtained were compared as shown in Table 1.

	Training Accuracy	Validation Accuracy	Test Accuracy
Resnet50	59.35%	64.95%	65.0%
Resnet50V2	99.37%	99.54%	100%
InceptionV3	98.21%	98.61%	99.0%
VGG16	96.91%	98.67%	99.0%
CNN	93.71%	94.39%	94%

Table 1: Results Summary of all Models

CHAPTER 9

CONCLUSION

CHAPTER 9 CONCLUSION

The deep learning models proposed in this study for the translation of sign language have significant results. ResNet50V2, InceptionV3 and VGG16 performed the best, the Simple CNN model performed decently and ResNet50 performed poorly. The deep learning models were able to efficiently extract features from the images of various sign language gestures and thus can be utilized to develop a translator that can output a fully understandable translation on being fed sign language gestures. As a result, it would help in the development of a sign language translator that would be able to assist Non-Sign Language Speakers to be able to understand what is being said by Sign Language Speakers. In-turn helping the population of Sign Language speakers to communicate and be understood easily.

CHAPTER 10

FUTURE WORK

CHAPTER 10 FUTURE WORK

Future research and work should be devoted to:

- Integrating Word-level translation of Sign Language.
- Development of a smartphone application.
- Translation of spoken language to sign language.

REFERENCES

[1] Rajarshi Bhadra and Subhajit Kar in 2021, “Sign Language Detection from Hand Gesture Images using Deep Multi-layered Convolution Neural Network” in IEEE Second International Conference on Control, Measurement and Instrumentation (CMI).

[2] Md. Jahangir Hossein and Md. Sabbir Ejaz - “Recognition of Bengali Sign Language using Novel Deep CNN.” in 2020 2nd International Conference on Sustainable Technologies for Industry 4.0.

[3] Manuel Eugenio Morocho Cayamcela and Wansu Lim in 2019, “Fine-tuning a pre-trained Convolutional Neural Network Model to translate American Sign Language in Real-time” in International Conference on Computing, Networking and Communications (ICNC).

[4] Aditya Das, Shantanu Gawde, Khyati Suratwala1 and Dr. Dhananjay Kalbande in 2018, “Sign Language Recognition Using Deep Learning on Custom Processed Static Gesture Images” in International Conference on Smart City and Emerging Technology (ICSCET).

[5] Kshitij Bantupalli and Ying Xie in 2018, “American Sign Language Recognition using Deep Learning and Computer Vision” in IEEE International Conference on Big Data (Big Data).

[6] Murat Taskiran, Mehmet Killioglu and Nihan Kahraman in 2018, “A Real-Time System For Recognition Of American Sign Language By Using Deep Learning” in 41st International Conference on Telecommunications and Signal Processing (TSP).

[7] Saleh Ahmad Khan, S. M. Asaduzzaman, Amit Debnath Joy, and Morsalin Hossain in 2019, “An Efficient Sign Language Translator Device Using Convolutional Neural Network and Customized ROI Segmentation” in 2nd International Conference on Communication Engineering and Technology (ICCET).

[8] Mark Borg and Kenneth P. Camilleri in 2019, “Sign Language Detection “In The Wild” with Recurrent Neural Networks” in ICASSP IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP).

[9] Jinalee Jayeshkumar Raval and Ruchi Gajjar in 2021, “Real-time Sign Language Recognition using Computer Vision” in 3rd International Conference on Signal Processing and Communication (ICPSC).

[10] Siming He in 2019, “Research of a Sign Language Translation System Based on Deep Learning” in International Conference on Artificial Intelligence and Advanced Manufacturing (AIAM).

[11] Gaurav Labhane, Rutuja Pansare, Saumil Maheshwari, Ritu Tiwari and Anupam Shukla in 2020, “Detection of Pediatric Pneumonia from Chest X-Ray Images using CNN and Transfer Learning” in 3rd International Conference on Emerging Technologies in Computer Engineering: Machine Learning and Internet of Things (ICETCE-2020)

[12] Kaiming He, Xiangyu Zhang, Shaoqing Ren and Jian Sun in 2016,"Deep Residual Learning for Image Recognition" in 2016 IEEE Conference on Computer Vision and Pattern Recognition

[13] Shruti Mohanty, Supriya Prasad, Tanvi Sinha and B. Niranjana Krupa in 2020, “German Sign Language Translation using 3D Hand Pose Estimation and Deep Learning” in 2020 IEEE REGION 10 CONFERENCE (TENCON)

[14] Gautham Jayadeep, N.V. Vishnupriya, Vyshnavi Venugopal, S. Vishnu, M. Geetha in 2020, “Mudra: Convolutional Neural Network based Indian Sign Language Translator for Banks” in 2020 4th International Conference on Intelligent Computing and Control Systems (ICICCS)

[15] Xiaoling Xia, Cui Xu & Bing Nan. In 2017 “Inception-v3 for flower classification” in 2017 2nd International Conference on Image, Vision and Computing (ICIVC).

APPENDIX A

CODE

Defining training folder, file paths and Data Frame

```
train_folder = '/kaggle/input/asl-alphabet/asl_alphabet_train/asl_alpha
bet_train'
all_data = []
for folder in os.listdir(train_folder):

    label_folder = os.path.join(train_folder, folder)
    onlyfiles = [{'label':folder,'path':os.path.join(label_folder, f)}
for f in os.listdir(label_folder) if os.path.isfile(os.path.join(label_
folder, f))]
    #print(onlyfiles)
    all_data += onlyfiles
data_df = pd.DataFrame(all_data)
data_df
```

Defining Train, Test and Holdout (Validation) splits

```
x_train,x_holdout = train_test_split(data_df, test_size= 0.10, random_s
tate=42,stratify=data_df[['label']])
x_train,x_test = train_test_split(x_train, test_size= 0.20, random_stat
e=42,stratify=x_train[['label']])
```

Creating Image Data generators for the splits

```
img_width, img_height = 64, 64
batch_size = 16
y_col = 'label'
x_col = 'path'
no_of_classes = len(data_df[y_col].unique())


train_datagen = ImageDataGenerator(rescale = 1/255.0)

train_generator = train_datagen.flow_from_dataframe(
    dataframe=x_train,x_col=x_col, y_col=y_col,
    target_size=(img_width, img_height),class_mode='categorical', batch
_size=batch_size,
    shuffle=False,
)

validation_datagen = ImageDataGenerator(rescale = 1/255.0)
validation_generator = validation_datagen.flow_from_dataframe(
    dataframe=x_test, x_col=x_col, y_col=y_col,
    target_size=(img_width, img_height), class_mode='categorical', batc
h_size=batch_size,
```

```
    shuffle=False
)
holdout_datagen = ImageDataGenerator(rescale = 1/255.0)
holdout_generator = holdout_datagen.flow_from_dataframe(
    dataframe=x_holdout, x_col=x_col, y_col=y_col,
    target_size=(img_width, img_height), class_mode='categorical', batc
h_size=batch_size,
    shuffle=False
)
```

Importing base models

```
base_model = ResNet50(input_shape=(64,64,3), weights='imagenet', include_to
p=False)
base_model.trainable = False

base_model = ResNet50V2(input_shape=(64, 64, 3), include_top=False, wei
ghts="imagenet")
base_model.trainable = False

base_model = VGG16(weights = "imagenet", include_top = False, input_sha
pe = (64, 64, 3))
base_model.trainable = False

base_model = InceptionV3(input_shape=(75,75,3), weights='imagenet', inc
lude_top=False)
base_model.trainable = False
```

Modelling a sequential CNN

```
model = Sequential([
    layers.Conv2D(32, (3, 3), activation='relu', input_shape=(64, 64, 3
)),
    layers.MaxPool2D((2, 2)),
    layers.Conv2D(64, (3, 3), activation='relu'),
    layers.MaxPool2D((2, 2)),
    layers.Conv2D(64, (3, 3), activation='relu'),
    layers.MaxPool2D((2, 2)),
    layers.Flatten(),
    layers.Dense(128, activation='relu'),
    layers.Dense(29, activation='softmax')
])
```

Fine tuning base model

```
flatten_layer = layers.Flatten()
dense_layer_1 = layers.Dense(512, activation='relu')
dropout_layer_1 = layers.Dropout(0.5)
#dense_layer_2 = layers.Dense(512, activation='relu')
#dropout_layer_2 = layers.Dropout(0.5)
```

```
prediction_layer = layers.Dense(29, activation='softmax')


model = models.Sequential([
    base_model,
    flatten_layer,
    dense_layer_1,
    dropout_layer_1,
    prediction_layer
])
```

Model fitting

```
classes = 29
epochs = 10
learning_rate = 0.0001

adam = Adam(lr=learning_rate)
model.compile(optimizer= adam, loss='categorical_crossentropy', metrics
=['accuracy'])
history = model.fit(train_generator,
                    epochs=epochs,
                    verbose=1,
                    validation_data=validation_generator, shuffle=True)
```

Github Link:

https://github.com/Davin07/Team31--Sign-Language-Translator-Using-Deep-Learning

FUNDING AND PAPER PUBLICATION DETAILS

Paper Title	Sign Language Translator using Deep Learning
Journal	International Journal of Innovative Research in Computer and Communication Engineering. Volume 10, Issue 5, May 2022
Year of Publishing	2022
Abstract	Human beings need to communicate with one another for a variety of reasons to coexist and survive. Unfortunately, millions of people worldwide lack the ability to speak or hear, either from birth or as a result of some accident. For people who suffer from these disabilities, it is difficult to communicate with the rest of the population. Hence, Sign language was created, which relies on physical gestures to represent letters, words or even sentences. A combination of these signs/gestures help a person to convey their message to another person. These signs/gestures, however, are not known by most of the general public, who will most definitely find it impossible to understand what another person is saying in sign language. The general public most often has no immediate need to learn sign language or has no motivation to do so. Hence there is a need for a Sign Language Translator, that would convert these gestures and movements into words and alphabets that are understandable to people who do not know sign language. 5 models, namely a basic convolutional network, ResNet50, ResNet50V2, Inceptionv3 and VGG16 were developed with CNN and Transfer Learning methodologies to translate sign language gestures in real time.
Authors	B.Evans Nikith Royan Darryl Andrade Davin S Thomas Gyandwip Das Debjit Das Prof. Rashmi Mothkur