TABLE OF CONTENTS

LIST OF FIGURES

LIST OF ABBREVIATIONS

ABBREVIATIONS	EXPLANATION
AI	Artificial Intelligence
CNN	Convolutional Neural Network
COCO	Common Objects n Context
ISL	Indian Sign Language
LSTM	Long Short-Term Memory
SLRS	Sign Language Recognition System
SSD	Single Shot Multi-Box Detector
UUID	Universal Unique Identifier

CHAPTER 1

INTRODUCTION

1.1 OVERVIEW OF THE PROJECT

The Deaf community in India uses Indian Sign Language, a visual language, for communication. Its grammar, vocabulary, and syntax differ from spoken languages, so it is a rich and complicated language. The use of machine learning techniques to recognize and translate ISL into spoken languages has gained popularity in recent years. It can benefit Deaf individuals who rely on ISL as their primary mode of communication. It can allow them to interact better with the hearing population and access information and services.

There are numerous ways to construct a machine learning-based ISL recognition system. Convolutional neural networks are frequently used to identify images of hand motions, which can be recorded using a camera or another visual sensor. For example, the CNN can be trained on a large dataset of annotated images of ISL gestures and then used to classify new images and recognize the corresponding ISL words or phrases. Recurrent neural networks and natural language processing are two more methods for ISL recognition that use machine learning. These methods evaluate a hand movements sequence to find patterns representing words or sentences. Overall, the goal of ISL recognition using machine learning is to develop systems that can accurately and reliably translate ISL.

Recognition system using machine learning. One challenge is the limited availability of annotated datasets for training and evaluating such systems. Building a large and diverse dataset of annotated ISL gestures can be time-consuming and costly and may require collaboration with the Deaf community and experts in the field. Another challenge is the complexity of ISL itself, as it is a rich and expressive language with a wide range of hand gestures and facial expressions that can convey meaning. As a result, it can make it difficult

to accurately recognize and translate ISL gestures, especially for more subtle or nuanced expressions. To overcome these challenges, researchers and developers working on ISL recognition using machine learning may need to employ various techniques and approaches. In addition, they may need to work closely with experts in the field of linguistics and Deaf studies.

Additionally, it may be essential to ensure that any ISL recognition systems are developed and evaluated ethically and respectfully, considering the needs and concerns of the Deaf community. There are several potential applications for ISL recognition using machine learning. One possible application is developing assistive technology for Deaf individuals, such as sign language translation devices or apps that can convert ISL into spoken or written languages. Such technologies could help Deaf individuals communicate more easily with the hearing population and access information and services that may otherwise be unavailable to them. Another potential application is in the field of education, where ISL recognition technology could be used to create educational materials and resources that are more accessible to Deaf students. For example, teachers could use ISL recognition technology to develop sign language versions of lectures or presentations or to provide real-time translation of spoken language into ISL for Deaf students in the classroom. There are also potential applications for ISL recognition in healthcare settings, where Deaf patients may have difficulty communicating with hearing healthcare providers. ISL recognition technology could facilitate communication between Deaf patients and healthcare providers, ensuring that Deaf patients receive the same level of care and attention as hearing patients. Overall, the development of effective ISL recognition technology has the potential to significantly improve the lives of Deaf individuals and increase their participation in society.

In addition to the potential applications, I mentioned earlier, there are other possible uses for ISL recognition using machine learning. For example, ISL recognition technology could create more accurate and reliable automatic captioning for videos and other multimedia content. It could benefit educational and instructional materials, making such content more accessible to Deaf individuals. ISL recognition technology could also be used to develop sign language interpretation services for live events, such as conferences, meetings, and public lectures. It could help to make such events more inclusive and accessible for Deaf attendees.

Another potential use for ISL recognition technology is the research and development of new technologies for Deaf individuals. For example, researchers could use ISL recognition technology to analyze and study the use of ISL in various contexts and to identify areas where new technologies or services could be developed to support the Deaf community better. Overall, the development and use of ISL recognition technology can significantly benefit Deaf individuals and improve their access to information, communication, and participation in society.

1.2 COLABORATORY

Co-laboratory, or “Colab” for short, is a product from Google Research. Google is quite aggressive in AI research. Over many years, Google developed an AI framework called **TensorFlow** and a development tool called **co-laboratory**. Google made Co-laboratory free for public use. Another attractive feature that Google offers to developers is the use of GPU. Colab supports GPU and it is totally free. The reason for making it free for the public could be to make its software a standard in academics for teaching machine learning and data science. It may also have a long-term perspective of building a customer base for Google Cloud APIs which are sold per-use basis. Irrespective of the reasons, the introduction of Colab has eased the learning and development of machine learning applications.

Google Colab can be very helpful for deep learning projects because it provides a free, cloud-based platform for running and experimenting with machine learning models. Some of the ways that Colab can be helpful for deep learning include: Free access to GPUs and TPUs: Colab provides free access to GPUs and TPUs, which can significantly accelerate the training of deep learning models. This can be particularly useful for larger models or for training on larger datasets.

Colab provides a simple, intuitive interface for writing and running code. It also comes with a variety of pre-installed libraries and tools, including TensorFlow, Keras, and PyTorch, making it easy to get started with deep learning. Colab allows you to share your notebooks with others, making it easy to collaborate on deep learning projects. You can also use Colab

with a team by connecting to a shared runtime. Integration with Google Drive: Colab notebooks are stored in Google Drive, which makes it easy to access and manage your deep learning projects.

1.3 MACHINE LEARNING

Machine learning is a subfield of artificial intelligence that focuses on building algorithms that can learn from and make predictions or decisions based on data. There are several different types of machine learning. Supervised learning: This type of machine learning involves training a model on labeled data, where the correct output is provided for each example in the training set. The model makes predictions based on this input-output mapping. Unsupervised learning: This type of machine learning involves training a model on unlabeled data, allowing the model to discover patterns in the data on its own. Semi-supervised learning: This type of machine learning involves training a model on a mix of labeled and unlabelled data. Automating Indian Sign Language using deep learning would involve training a machine learning model to recognize and translate sign language gestures into text or spoken language. This could be done using a dataset of image examples of sign language gestures with corresponding text transcriptions. The model could then be trained to recognize the gestures in the video and output the corresponding text.

To train a machine learning model for this task, we would need a dataset of image examples of Indian Sign Language gestures, as well as the corresponding text transcriptions. Then use the data to train a machine learning model, such as TensorFlow object detection, to recognize the gestures in the video and output the corresponding text.

It is worth noting that automating sign language translation is a very challenging task, and it will likely require a significant amount of data and computational resources to achieve good results. Additionally, it is important to consider the ethical implications of automating sign language translation, as it could potentially be used to replace or diminish the role of human interpreters in some contexts.

SUMMARY

Sign language plays a vital role in the communication of deaf and dumb people in India. The proposed work is implemented to detect sign language and translate it to the regional languages in India. This Chapter is about an introduction to sign language and how to deep learning to implement the proposed system and the environment setup in the co-laboratory. The literature review of this project is discussed in the next chapter.

CHAPTER 2

LITERATURE REVIEW

A deep sign language recognition system for Indian sign language [1]

Soumen Das, Deaf people face major challenges during communication with normal people. Employing a human interpreter (a person who converts Sign language (SL) into a language that the normal/hearing community can understand) is not an effective solution to this problem due to the unavailability of professional interpreters. Thus, the Sign Language Recognition System (SLRS) is the most efficient and effective choice because it automatically converts SL into text/speech without an interpreter and reduces the communication barrier between deaf and normal people. This paper reports a work on Indian Sign Language (ISL) word recognition using a vision-based technique. The existing vision-based solutions for ISL word recognition are ineffective due to excessive pre-processing such as extracting features from a large sequence of frames. Therefore, a vision-based SLRS named Hybrid CNN-Bi, LSTM SLR (HCBSLR) is proposed, which overcomes the drawback of excessive pre-processing. The proposed model uses a Histogram Difference (HD) based key-frame extraction method to improve the accuracy and efficiency of the system by eliminating redundant or useless frames. The HCBSLR system uses VGG-19 for spatial feature extraction and Bidirectional Long Short Term Memory (Bi, LSTM) for temporal feature extraction. The proposed HCBSLR system has achieved an average accuracy of 87.67%, which is compared with some of the existing SLRS. The experimental results show that the proposed HCBSLR system is more accurate and efficient than the existing SLRS.

Real-time translation of Indian sign language to assist the hearing and speech impaired [2]

S.Rajarajeswari , The most predominant mode of communication among people with hearing and speech impairment is the sign language. Therefore, if those concerned individuals

are not equally skilled in sign language, there will be a communication barrier between them, thereby making the availability of an interpreter/translator immensely important for communication. Efficient recognition of gesture-based communication, hence, becomes essential to overcome the obstacle to communication among the people with hearing/speech impairment and people without such impediment. The ISL gesture motions are recorded by utilizing a mobile camera, and these pictures are then processed appropriately by our model. The user receives the output which is returned by the SoftMax layer as the most probable class after comparing confidence scores. The model created as part of this project availed us an accuracy of 91.84% under the adaptive thresholding filter. An accuracy of 92.78% was determined under the hybridized SIFT mode with adaptive thresholding filter.

Real-time sign language translator [3]

Khushbu Sinha, The main objective of this work is to build a real-time sign language translator to translate sign language to text using TensorFlow object detection and Python. It would provide an easy way for people to communicate with others using sign language further leading to the elimination of the middle person who generally acts as a medium of translation and would also provide an easy-to-use environment for the users by providing text output for the input sign gesture. Here, the sign language is to be fed on a real-time basis using a webcam and the input fed would be compared with the trained model to recognize the sign and those who do not know the sign language could use it to understand the sign language and get a text output for the same with accuracy. Currently, the model was trained for Indian Sign Language (ISL) consisting of alphabets from A to Z and digits from 1 to 9 counting to a total of 35 signs and we obtained a loss of 0.227 on the trained model and the model predicted the signs with nearly an accuracy of 50–80% in our case.

Recognition of isolated gestures for Indian sign language using transfer learning [4]

Kinjal Mistree , Sign language is a visual and complete natural language used by Deaf to communicate with each other and hearing people. The Deaf community finds it very difficult to express their feelings to the hearing people, because of which human interpreters

are needed to help them in emergency situations. But always human interpreting service is not available when needed, because in India only around 300 certified sign language interpreters are available. In order to fill this gap, an automated system can be designed that would facilitate recognition of Indian Sign Language (ISL) gestures. This paper describes such an approach that takes ISL image as an input and gives corresponding class as an output. We have shown that use of simple image manipulation techniques and transfer learning give promising result in ISL hand gesture recognition. For 11 categories of ISL words, we have achieved 99.07% accuracy that out performs accuracy reported by existing approach.

Benchmarking deep neural network approaches for Indian Sign Language recognition [5]

Ashish Sharma, A large portion of the hearing and speech impaired are deprived of any education and just around 1-2% of the hearing and speech impaired get schooling in Indian sign language. The large portion of them is incapable of using theoretical languages, have difficult issues expressing themselves or understanding printed texts, So , there is need to build an auto- mated translation system with innovations for helping hearing and speech impaired people in communicating among them. Sign language recognition is an evolving research area in computer vision and machine learning.

Continuous sign language recognition using isolated signs data and deep transfer learning [6]

S. Sharma, To facilitate communication between the signer and non-signer communities, an effective sign language recognition system (SLRS) can identify sign language motions. This research proposes a deep learning-based SLRS that is computer vision-based. The main three contributions of this work are: First, 65 distinct users were used in an uncontrolled situation to produce a sizable dataset of Indian sign language. Second, to boost the dataset's intra-class variance utilizing augmentation, the suggested work's generalizability was improved. This study creates three extra copies of each training image using three distinct affine modifications. Third, by applying convolutional neural networks, a distinct and trustworthy model for the feature extraction and classification of ISL has been

created (CNN). The performance of this technique is evaluated on a self-compiled ISL dataset and a publically available ASL dataset. Overall, three datasets were used, and the accuracy was 92.43, 88.01, and 99.52%. Accuracy, recall, f-score, and system time have all been used to evaluate the efficiency of this method. The outcomes reveal that, when compared to previous work, the suggested technique performs admirably.

Sign language recognition system using tensorflow object detection api [7]

Sharvani Srivastava, Communication is defined as the act of sharing or exchanging information, ideas or feelings. To establish communication between two people, both of them are required to have knowledge and understanding of a common language. But in the case of deaf and dumb people, the means of communication are different. Deaf is the inability to hear and dumb is the inability to speak. They communicate using sign language among themselves and with normal people but normal people do not take seriously the importance of sign language. Not everyone possesses the knowledge and understanding of sign language which makes communication difficult between a normal person and a deaf and dumb person. To overcome this barrier, one can build a model based on machine learning. A model can be trained to recognize different gestures of sign language and translate them into English. This will help a lot of people in communicating and conversing with deaf and dumb people. The existing Indian Sing Language Recognition systems are designed using machine learning algorithms with single and double-handed gestures but they are not real-time. In this paper, we propose a method to create an Indian Sign Language dataset using a webcam and then using transfer learning, train a TensorFlow model to create a real-time Sign Language Recognition system. The system achieves a good level of accuracy even with a limited size dataset.

Indian sign language recognition using convolutional neural networks [8]

Keerthi Reddy Velmula, The Communication plays an essential role in our daily life. People who are hearing and/or speech impaired find it difficult to communicate with others. To aid the deaf and dumb in communicating with others sign languages are used. There are many sign languages. In this paper we worked with Indian sign language. A convolutional

neural network is used to identify the images and classify them. Convolutional neural network comes under deep learning algorithms. Various actions are performed in each layer of the neural network to classify the image correctly. Employing convolutional neural networks increases the accuracy of the system. The data set consists of hand gestures which are already processed and the model is trained with the dataset. The model takes hand gestures as input and converts those signs into text. Real time video feed is used as input. The model can be further trained to detect more signs depending on the efficiency of the device.

CNN and Stacked LSTM Model for Indian Sign Language Recognition [9]

C.Aparna *et al.*[9], In this paper, we propose deep learning for sign language recognition using a convolutional neural network (CNN) and long short-term memory (LSTM). The architecture used CNN as a pre-trained model for feature extraction and is passed to the LSTM for capturing spatiotemporal information. One more LSTM is stacked for increasing the accuracy. The deep learning model which captures temporal information is less. There is only less papers that deal with sign language recognition by using deep learning architectures such as CNN and LSTM. The algorithm was tested in the Indian sign language (ISL) dataset. We have presented the performance evaluation after testing with ISL dataset. Literature shows that deep learning models capturing temporal information is still an open research problem.

Sign Language Translator for alphabets using TensorFlow [10]

Raghuveera , Recognition of sign language by a system has become important to bridge the communication gap between the abled and the Hearing and Speech Impaired people. This paper introduces an efficient algorithm for translating the input hand gesture in Indian Sign Language (ISL) into meaningful English text and speech. The system captures hand gestures through Microsoft Kinect (preferred as the system performance is unaffected by the surrounding light conditions and object colour). The dataset used consists of depth and RGB images (taken using Kinect Xbox 360) with 140 unique gestures of the ISL taken from 21 subjects, which includes single-handed signs, double-handed signs and fingerspelling (signs for alphabets and numbers), totaling to 4600 images. To recognize the hand posture,

the hand region is accurately segmented and hand features are extracted using Speeded Up Robust Features, Histogram of Oriented Gradients and Local Binary Patterns. The system ensembles the three feature classifiers trained using Support Vector Machine to improve the average recognition accuracy up to 71.85%. The system then translates the sequence of hand gestures recognized into the best approximate meaningful English sentences. We achieved 100% accuracy for the signs representing 9, A, F, G, H, N and P.

SUMMARY

This chapter describes the literature reviews of 10 papers regarding the sign language recognition; in this most of the papers are implemented by using the various algorithms and approaches to perform the sign language recognition. Such papers are only focused on the detecting the alphabets of sign language. This proposed work focuses on sign language words. The proposed work of this project will be elaborated in chapter 3.

CHAPTER 3

PROPOSED WORK

3.1 PROBLEM STATEMENT

Sign language was used by the dumb and deaf to communicate. Deaf and dumb people feel difficult to communicate with normal people because normal people don't know the sign language that is used by them to communicate. This is becoming a barrier between normal people and deaf and dumb people.

The requirements are as follows:

(1) Detecting the Indian sign language in real-time with a webcam.

(2) The system must be very cheap; it must be available for all kinds of people.

(3) And the detected language must be changed to the Indian language.

3.2 PROPOSED METHODOLOGY

The Machine learning model is developed to detect the Indian sign language in real-time with the help of TensorFlow and the object detection API for deaf and dumb people who cannot communicate with normal people. This will remove a barrier between normal people and deaf and dumb people and the detected signs are translated into the given language to help the all-region people in India.

3.3 DATASET COLLECTION

Collecting data for the model plays a major role in getting good results after the whole

process. To collect the data here one can use a camera with help of OpenCV and python pictures are taken. For saving the images with different names UUID model in python is used. Real-time computer vision is the primary focus of OpenCV's functions. It facilitates the incorporation of artificial intelligence into commercial goods and offers a standard framework for computer vision-based applications. More than 2500 optimized algorithms are available in the collection, including a wide range of both traditional and cutting-edge computer vision and machine learning techniques These methods can be used for many different tasks, such as face detection and recognition, object identification, classifying human activities, following camera and object movements, and extracting 3D object models.

For each sign, a minimum of fifteen images can produce good results. So, we take fifteen images per sign. Seventy percent of the images per sign is used to train the model and thirty percent of the images per sign is used to test the model. Using a camera and python, OpenCV reduces the difficulties in collecting the data. Here augmentation or any other complex methods are not used to collect the data. Collecting data difficulties are reduced by this method as shown in figure 3.1.

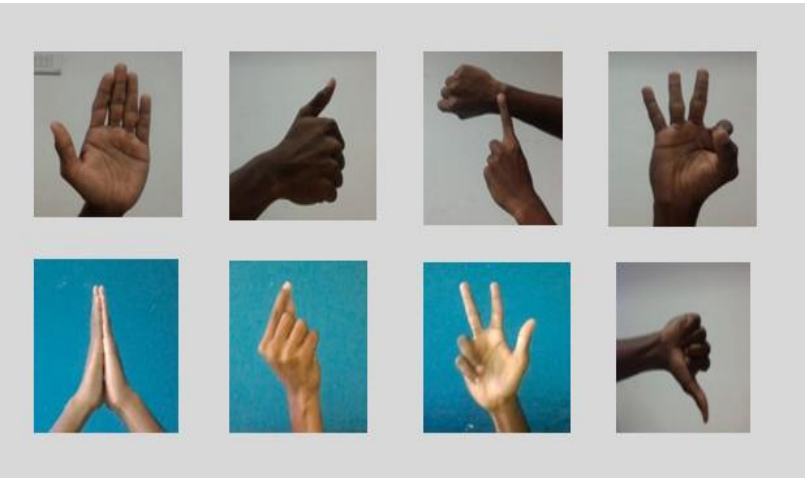

Figure 3.1. Dataset for the training purpose

The above figure is about the sign images captured using the webcam for the dataset collection. These images are used to train the proposed model.

3.4 DATA PREPROCESSING

Unclean data can be converted into clean data collection via a process known as data preparation. To put it another way, data is typically obtained from various sources in raw form, making analysis impractical. For this reason, pre-processing the data is necessary. In this case, to use TensorFlow object detection images must be labeled by their sign. It helps the object detection model recognize the sign in the image. After all the photographs have been taken, the Labelling package is used to name each one individually. A free open-source program called LabelImg is used to graphically label images. All the information about the photographs, including the labeled portion, is contained in the XML files. All the photos XML files are now accessible. TensorFlow records are created because of this. Following that, training data and validation data are split 80:20 amongst all the images and associated XML files. A training dataset was created and stored for (80%) of the sign images, while a validation dataset was created and stored for the remaining 20%.

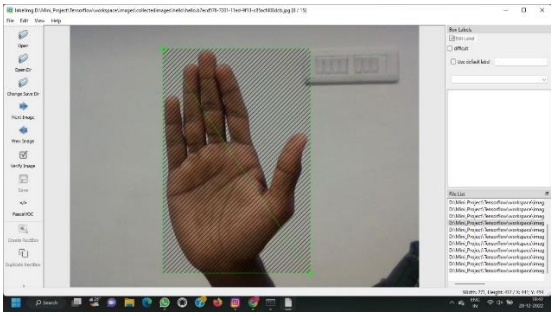

Figure 3.2. Collecting data

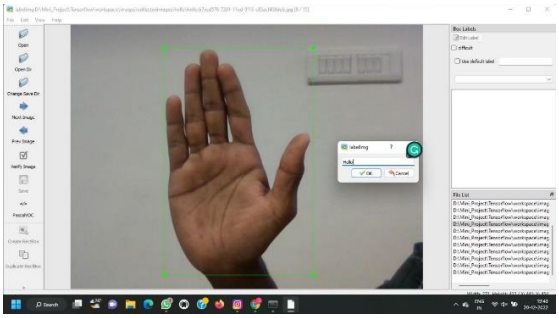

Figure 3.3. Labeling images with the python tool

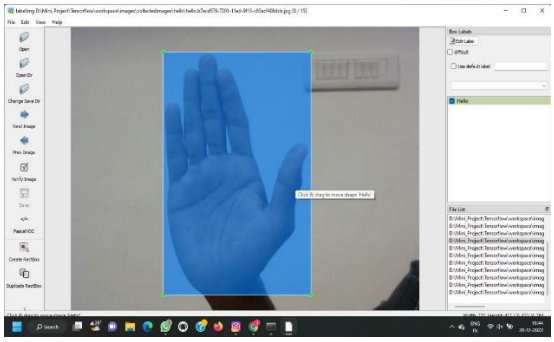

Figure 3.4. Marking the image with label

Figures 3.2, 3.3, 3.4 show the process of pre-processing the image data. Figure 2 shows the labelImg tool that is used for data pre-processing. Figure 3 shows the entering of the sign name for the image. Figure 4 shows the marking of the image with its sign.

3.5 OBJECT DETECTION TECHNIQUES

The TensorFlow object detection model helps to detect the object inside the images. In this system, TensorFlow is used the detect the hand signs inside the images to find the specific sign. Here we use the TensorFlow pre-trained zoo model because it reduces the training time of the system. This can save time and cost for the system. And we can get a good performance with the limited amount of the dataset and the training time will reduced significantly.

3.6 REAL-TIME DETECTION

In the colab webcam is used to capture the video, and the video frames are converted into images using JavaScript. After that images are given as input to the model that is trained using the set of sign language hand signs. The model will detect the sign with the help of TensorFlow object detection API. The signs with the accuracy score is drawn in the image and its shown in the colab using JavaScript. This is how the Indian signs or detected in real-time using TensorFlow object detection. This process takes less amount of time so the signs are detected very fast in normal hardware configuration.

3.7 SIGN TRANSLATION

Googletrans is an Open-Source python package that translates one language to another language. A python function is created to convert the English language text in to given language using that googletrans package. This reduces the work of manual translation for the proposed system. The detected sign language name is converted into a given Indian language using that python function. This helps various people in India who uses various languages to communicate with others.

SUMMARY

In this chapter, the overview of Indian sign language recognition and then approaches like TensorFlow object detection techniques, TensorFlow zoo model and Real-time detection of sign language and sign name translation are discussed briefly and how they were proposed for this project. Then the project implementation along with dataset descriptions are explained in the next chapter.

CHAPTER 4

PROJECT IMPLEMENTATION

4.1 METHODOLOGY

4.1.1 Object detection

The problem of object detection in computer vision is locating and identifying things in pictures and videos. It is an important and widely used application of machine learning, with a wide range of applications including self-driving cars, security systems, and robotics. In object detection, the goal is to identify and locate objects of interest within an image or video. This is typically done by training a machine learning model on a large dataset of images that contain the objects of interest, along with annotations that indicate the location of the objects within the images. The model is then able to use this training data to make predictions on new images, identifying and localizing the objects of interest within the image. This object detection will help full for finding the sign inside the images because it detects the object in the pictures. It can be used in sign language recognition to identify the hand and finger movements that make up different signs.

4.1.2 TensorFlow

The machine learning system TensorFlow performs at scale and under various conditions. TensorFlow uses dataflow graphs to explain computation, shared state, and the operations that change it. It distributes a dataflow graph's nodes across many computers in a cluster and a variety of computing devices within a single system, including multicore CPUs, general-purpose GPUs, and specifically designed ASICs known as Tensor Processing Units. This design gives the application developer flexibility because TensorFlow enables them to test out cutting-edge optimizations and training approaches, in contrast to older "parameter server" systems that managed shared states internally. A wide number of applications are

supported by TensorFlow, with training and inference on deep neural networks receiving especially strong support.

A mathematical entity known as a tensor is also represented by higher-dimensional arrays. The neural network receives these data arrays of various sizes and ranks as input. One-dimensional arrays, vectors, and matrices are all possible, as are two-dimensional arrays. Tensors, however, can have more dimensions than three, four, or five. Tensors, however, can have more dimensions than three, four, or five. As a result, it is helpful to have the data concentrated in one location and then centre all of the research on that.

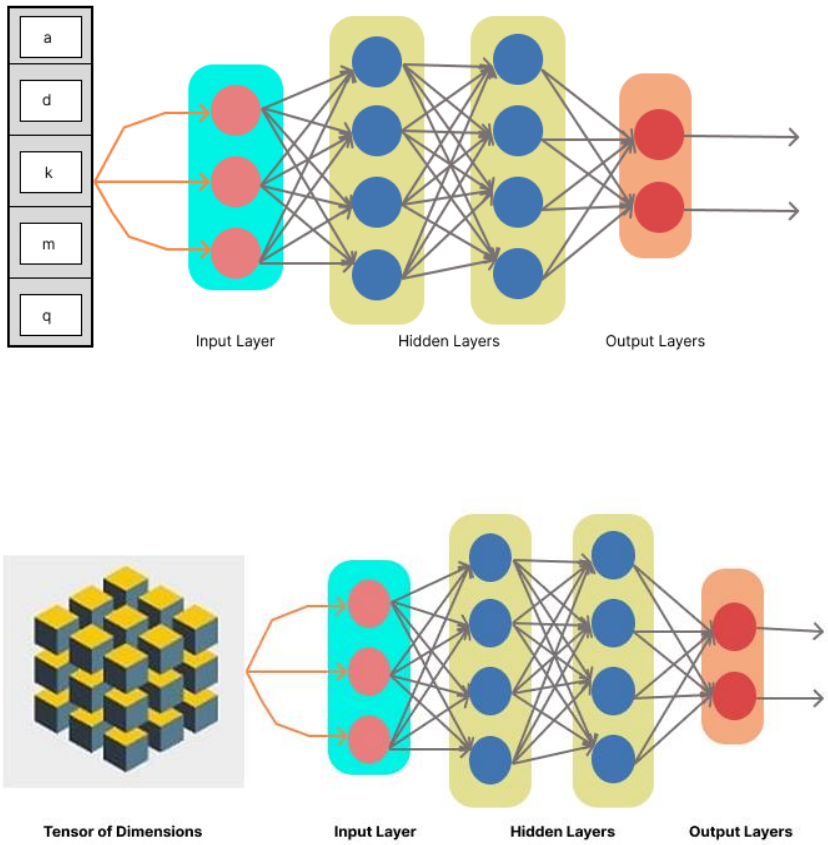

Figure 4.1. Formulation Of TensorFlow

Figure 4.1 shows the formulation of TensorFlow formulation and the different layers of TensorFlow. The TensorFlow has three layers (input layer, hidden layers, output layer are shown in the above image.

4.1.3 TensorFlow's COCO

TensorFlow's COCO (Common Objects in Context) API is a dataset of images that have been annotated with labels for a variety of objects and scenes. The dataset includes many images that depict common objects in their everyday context and is often used to train and evaluate object detection and segmentation models. The COCO dataset is commonly used in computer vision research and is a good resource for people who are interested in developing and testing machine learning models that can recognize and classify objects in images.

The COCO dataset is organized into a series of categories, each of which corresponds to a particular type of object or scene. The categories in the COCO dataset include things like people, animals, vehicles, furniture, and various household objects. Each image in the dataset has been annotated with labels indicating the presence and location of objects from these categories. The dataset also includes several additional annotations, such as segmentation masks for each object and captions describing the objects and their context in the image.

In addition to the image data and annotations, the COCO dataset includes several tools and utilities for working with the data. These tools include a Python API for accessing the data and annotations, as well as several scripts for creating and evaluating object detection models. The COCO API is widely used in the computer vision community and is a useful resource for anyone who is interested in developing machine learning models that can recognize and classify objects in images.

4.1.4 OpenCV

OpenCV (Open-Source Computer Vision) is a free and open-source library of computer vision and machine learning algorithms. It is widely used for a variety of tasks in

the field of computer vision, such as image and video processing, object detection, and image recognition. In the context of object detection, OpenCV can be used to design and implement algorithms that can identify and locate objects in images and video streams. These algorithms can be based on a variety of techniques, including machine learning, template matching, and feature extraction. One of the strengths of OpenCV is its ability to process images and video in real time, which makes it well-suited for use in applications that require fast object detection. OpenCV also provides several pre-trained object detection models that can be used out-of-the-box, as well as tools for training custom object detection models. Overall, OpenCV is a powerful and widely used library for object detection and computer vision tasks. It is a good resource for anyone who is interested in developing machine-learning models or building applications that involve object detection. The OpenCV helps to manipulate the images. This helps to draw the name of the sign in the image. With that, we can know the detected sign name with its score of it.

4.1.5 Google trans package

Googletrans is a free and open-source library that provides translation services using the Google Translate API. It allows you to translate text between languages using the Google Translate service. To use googletrans, you will need to install it using pip, the Python package manager. After installing googletrans, you can use it in your Python code by importing it and then calling one of the library's functions. This package helps to translate one language into another one for the detected sign language name.

4.2 ALGORITHM STEP

Step 1: Download and extract the TensorFlow model zoo.

Step 2: Download and install and compile the protobuf.

Step 3: Install the Object detection API to detect the hand sign inside the images.

Step 4: Prepare the data to feed the TensorFlow zoo model.

Step 5: Transform the data to use as a dataset for the model.

Step 6: To connect labels to integer values, TensorFlow Object

Detection API.

Step 7: Select the suitable model architecture for the project.

Step 8: Train the model with the collected data set.

Step 9: Utilize an object detection tool to find the object in real-time.

Step10: The detected class is translated into the Indian language using the google trans module.

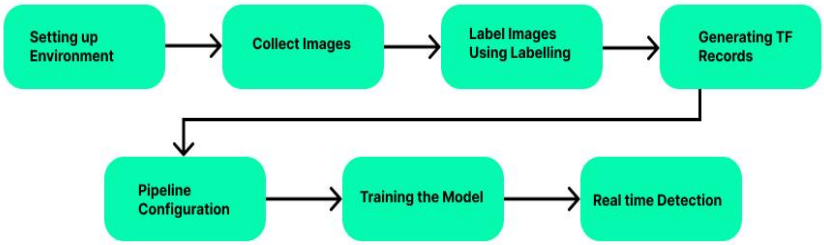

Figure 4.2. Execution flow of the system

The above flow diagram image shows the execution flow of the proposed system. The whole process of the proposed system is shown in the above diagram.

SUMMARY

This chapter tells about how the tools like co-laboratory and labelImg tools are used and the algorithm for the proposed system is discussed briefly. Based on this implementation the result of this project work is discussed in chapter 5.

CHAPTER 5

RESULTS AND DISCUSSION

Indian Sign Language poses are instantly recognized by the established system. The system was developed utilizing TensorFlow's object detection API. The pre-trained model, SSD Mobile Net v2 320x320, was acquired from the TensorFlow model zoo. The recently created dataset, which has been trained via transfer learning, comprises 15 photos for each symbol with labelled data. The loss rate of TensorFlow gives the accuracy rate of the trained model. How the lost rate is reduced more the good result will be achieved in TensorFlow. The proposed system gives an overall loss of 0.08 during the final 20,000 training steps; localization loss was 0.0002; classification loss was 0.016; and regularization loss was 0.06. As can be seen in the graphic below, the step 17600 loss was the least, coming in at 0.08.

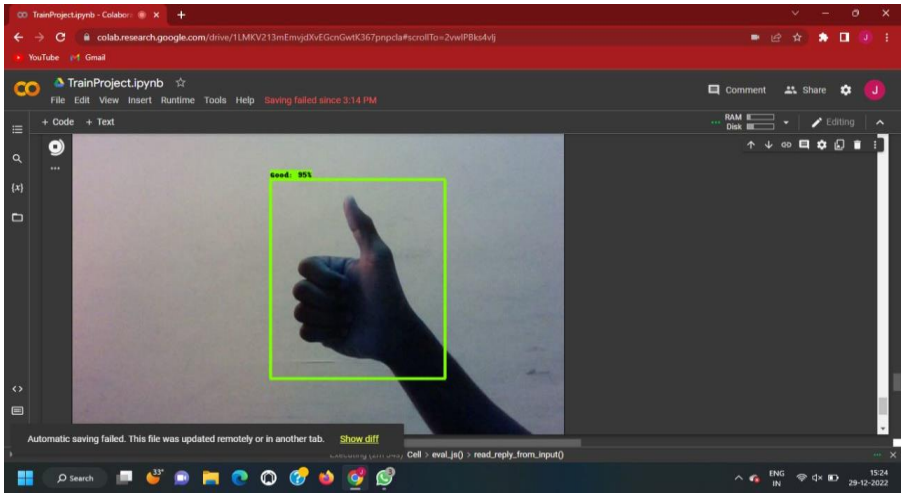

Figure 5.1. Real-time detection for good

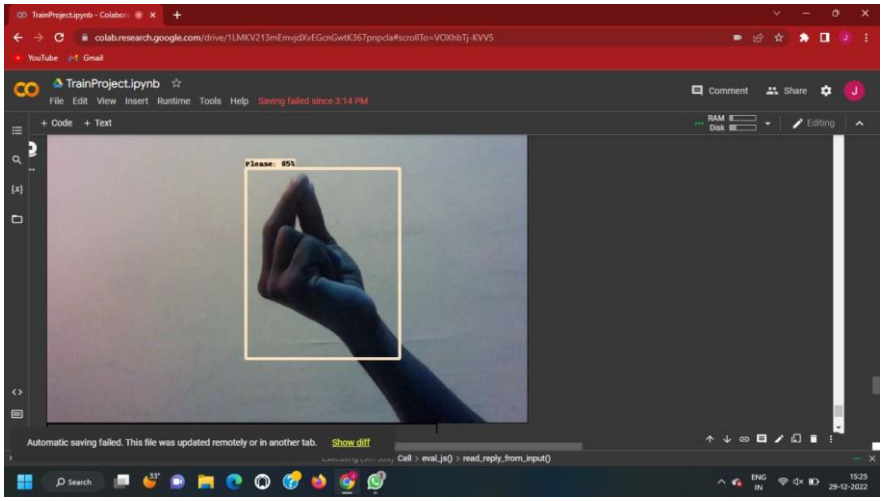

Figure 5.2. Real-time detection for please

The above images 7 and 8 show the real-time detection of the two signs (good, please), both signs have more than 95 % of the detection result. This show the efficiency of the proposed system.

```
INFO:tensorflow:Step 17400 per-step time 1.873s
I1225 19:56:30.591351 140699338602368 model_lib_v2.py:705] Step 17400 per-step time 1.873s
INFO:tensorflow:{'Loss/classification_loss': 0.019253742,
 'Loss/localization_loss': 0.0026312857,
 'Loss/regularization_loss': 0.06875377,
 'Loss/total_loss': 0.0906388,
 'learning_rate': 0.05985175}
I1225 19:56:30.591747 140699338602368 model_lib_v2.py:708] {'Loss/classification_loss': 0.019253742,
 'Loss/localization_loss': 0.0026312857,
 'Loss/regularization_loss': 0.06875377,
 'Loss/total_loss': 0.0906388,
 'learning_rate': 0.05985175}
INFO:tensorflow:Step 17500 per-step time 1.829s
I1225 19:59:33.451373 140699338602368 model_lib_v2.py:705] Step 17500 per-step time 1.829s
INFO:tensorflow:{'Loss/classification_loss': 0.015910752,
 'Loss/localization_loss': 0.0051227845,
 'Loss/regularization_loss': 0.06847169,
 'Loss/total_loss': 0.08950523,
 'learning_rate': 0.0596287}
I1225 19:59:33.451754 140699338602368 model_lib_v2.py:708] {'Loss/classification_loss': 0.015910752,
 'Loss/localization_loss': 0.0051227845,
 'Loss/regularization_loss': 0.06847169,
 'Loss/total_loss': 0.08950523,
 'learning_rate': 0.0596287}
INFO:tensorflow:Step 17600 per-step time 1.841s
I1225 20:02:37.573128 140699338602368 model_lib_v2.py:705] Step 17600 per-step time 1.841s
INFO:tensorflow:{'Loss/classification_loss': 0.01615333,
 'Loss/localization_loss': 0.002677649,
 'Loss/regularization_loss': 0.068178706,
 'Loss/total_loss': 0.08700968,
 'learning_rate': 0.05940484}
I1225 20:02:37.573523 140699338602368 model_lib_v2.py:708] {'Loss/classification_loss': 0.01615333,
 'Loss/localization_loss': 0.002677649,
 'Loss/regularization_loss': 0.068178706,
 'Loss/total_loss': 0.08700968,
 'learning_rate': 0.05940484}
```

Figure 5.3. Loss rate for the developed model

The figure 5.3 shows the loss rate of the trained model in TensorFlow. The system's output depends on the confidence level, which is now 95% on average. The confidence rate

for each sign is noted and summarized in the outcome by expanding the dataset, the system's identification capabilities. Consequently, the system's performance is enhanced and improved. In addition, the recognized classes of signs are translated into a given Indian language, which helps various regions of Indian utilize the Indian sign language recognition system.

SUMMARY

The current chapter is about the efficiency of the proposed system in Indian sign language recognition with generated datasets in normal hardware and tools like a webcam and the co-laboratory. This shows a good accuracy rate with less cost. This helps the system proposed will reach normal people.

CHAPTER 6

CONCLUSION AND FUTURE WORK

6.1 CONCLUSION

In India, deaf and dumb people have many problems because of the communication gap with normal people. They use sign language and hand signs to communicate with others, but normal people don't have knowledge about that. Because of that, they can't share their thoughts, feelings, and expressions with normal people. An automated Indian sign language detection system can solve this problem. It helps translate to sign language Signs of the normal Indian language. By using TensorFlow Object detection API we achieve this solution for this problem. This model is trained with the signs of deaf and dumb people, so it can detect the signs in real time with a good accuracy rate. To create the data collection, webcam Python and OpenCV are used to reduce the cost. The created model gives 88.20% of the average score in detection in real-time. With a smaller number of datasets, this model has achieved good accuracy.

6.2 FUTURE WORK

In the future, more signs can be detected by giving more datasets with a good amount of accuracy. This system can be used in various sign languages with the appropriate dataset. To incorporate context and semantics into the sign language detection system. For example the system could to designed to consider the context in which a sign is being used (e.g., a conversation between two people) and use this information to better understand the meaning of the sign.

APPENDIX

```
from google.colab import drive
drive.mount('/content/drive')

!pip install googletrans==3.1.0a0

!pip install wget
import wget

%cd /content/drive/MyDrive/RealTimeObjectDetection-main/

!ls

%cd /content/drive/MyDrive/RealTimeObjectDetection-main/
!rm -rf models
!git clone https://github.com/tensorflow/models.git
%cd models/research
# Compile protos.
!protoc object_detection/protos/*.proto --python_out=.
# Install TensorFlow Object Detection API.
%cp object_detection/packages/tf2/setup.py .
!python -m pip install .
# Test the installation
!pip install tf-nightly==2.12.0.dev20221226

%cd /content/drive/MyDrive/RealTimeObjectDetection-main

!ls
```

SETUP PATHS :

```
WORKSPACE_PATH = 'Tensorflow/workspace'
```

```
SCRIPTS_PATH = 'Tensorflow/scripts'
APIMODEL_PATH = 'Tensorflow/models'
ANNOTATION_PATH = WORKSPACE_PATH+'/annotations'
IMAGE_PATH = WORKSPACE_PATH+'/images'
MODEL_PATH = WORKSPACE_PATH+'/models'
PRETRAINED_MODEL_PATH = WORKSPACE_PATH+'/pre-trained-models'
CONFIG_PATH = MODEL_PATH+'/my_ssd_mobnet/pipeline.config'
CHECKPOINT_PATH = MODEL_PATH+'/my_ssd_mobnet/'
CUSTOM_MODEL_NAME = 'my_ssd_mobnet'
```

CREATE LABEL MAP :

```
labels = [{'name':'Good', 'id':1}, {'name':'Bad', 'id':2},{'name':'Church', 'id':3},
        {'name':'Expensive', 'id':4},{'name':'Little', 'id':5},{'name':'Please', 'id':6},
        {'name':'Request', 'id':7},{'name':'Time', 'id':8},{'name':'Today', 'id':9},
        {'name':'Welcome', 'id':10},{'name':'ILoveYou', 'id':11},{'name':'Nice',
'id':12},{'name':'hello', 'id':13}]

 with open(ANNOTATION_PATH + '/label_map.pbtxt', 'w') as f:
    for label in labels:
        f.write('item { \n')
        f.write('\tname:\'{}\'\n'.format(label['name']))
        f.write('\tid:{}\n'.format(label['id']))
        f.write('}\n')

import tensorflow as tf
print(tf.version.VERSION)
```

CREATE TF RECORDS :

```
 !python {SCRIPTS_PATH + '/generate_tfrecord.py'} -x {IMAGE_PATH + '/train'} -l
{ANNOTATION_PATH + '/label_map.pbtxt'} -o {ANNOTATION_PATH + '/train.record'}
 !python {SCRIPTS_PATH + '/generate_tfrecord.py'} -x{IMAGE_PATH + '/test'} -l
{ANNOTATION_PATH + '/label_map.pbtxt'} -o {ANNOTATION_PATH + '/test.record'}
```

COPY MODEL CONFIG TO TRAINING FOLDER :

```
import os
 PIPELINE_PATH = “Tensorflow/workspace/models/”+CUSTOM_MODEL_NAME
 PIPELINE_PATH
 try:
    os.mkdir(PIPELINE_PATH)
 except FileExistsError:
     # directory already exists
     pass
 !cp {PRETRAINED_MODEL_PATH+’/ssd_mobilenet_v2_fpnlite_320x320_coco17_tpu-8/pipeline.config’} {MODEL_PATH+’/’+CUSTOM_MODEL_NAME}
```

UPDATE CONFIG FOR TRANSFER LEARNING :

```
import tensorflow as tf
from object_detection.utils import config_util
from object_detection.protos import pipeline_pb2
from google.protobuf import text_format

config = config_util.get_configs_from_pipeline_file(CONFIG_PATH)

pipeline_config = pipeline_pb2.TrainEvalPipelineConfig()
with tf.io.gfile.GFile(CONFIG_PATH, "r") as f:
    proto_str = f.read()
text_format.Merge(proto_str, pipeline_config)

pipeline_config.model.ssd.num_classes = 13
pipeline_config.train_config.batch_size = 4
pipeline_config.train_config.fine_tune_checkpoint =
PRETRAINED_MODEL_PATH+'/ssd_mobilenet_v2_fpnlite_320x320_coco17_tpu-8/checkpoint/ckpt-0'
pipeline_config.train_config.fine_tune_checkpoint_type = "detection"
```

```
pipeline_config.train_input_reader.label_map_path= ANNOTATION_PATH +
'/label_map.pbtxt'
pipeline_config.train_input_reader.tf_record_input_reader.input_path[:] =
[ANNOTATION_PATH + '/train.record']
pipeline_config.eval_input_reader[0].label_map_path = ANNOTATION_PATH +
'/label_map.pbtxt'
pipeline_config.eval_input_reader[0].tf_record_input_reader.input_path[:] =
[ANNOTATION_PATH + '/test.record']



config_text =
text_format.MessageToString(pipeline_config)
with tf.io.gfile.GFile(CONFIG_PATH, "wb") as f:
    f.write(config_text)
```

LOAF TRAIN MODEL FROM CHECKPOINT :

```
import os
from object_detection.utils import label_map_util
from object_detection.utils import visualization_utils as viz_utils
from object_detection.builders import model_builder

%cd /content/drive/MyDrive/RealTimeObjectDetection-main

# Load pipeline config and build a detection model
configs = config_util.get_configs_from_pipeline_file(CONFIG_PATH)
detection_model = model_builder.build(model_config=configs[‘model’], is_training=False)

# Restore checkpoint
ckpt = tf.compat.v2.train.Checkpoint(model=detection_model)
ckpt.restore(os.path.join(CHECKPOINT_PATH, ‘ckpt-35’)).expect_partial()
```

```
@tf.function
def detect_fn(image):
    image, shapes = detection_model.preprocess(image)
    prediction_dict = detection_model.predict(image, shapes)
    detections = detection_model.postprocess(prediction_dict, shapes)
    return detections
```

DETECT IN REAL-TIME :

```
import cv2
import numpy as np

category_index =
label_map_util.create_category_index_from_labelmap(ANNOTATION_PATH+'/label_map
.pbtxt')

# JavaScript to properly create our live video stream using our webcam as input
from IPython.display import display, Javascript

from google.colab.output import eval_js
from base64 import b64decode
def video_stream():
  js = Javascript('''
    var video;
    var div = null;
    var stream;
    var captureCanvas;
    var imgElement;
    var labelElement;

    var pendingResolve = null;
    var shutdown = false;
```

```
function removeDom() {
   stream.getVideoTracks()[0].stop();
   video.remove();
   div.remove();
   video = null;
   div = null;
   stream = null;
   imgElement = null;
   captureCanvas = null;
   labelElement = null;
}

function onAnimationFrame() {
  if (!shutdown) {
    window.requestAnimationFrame(onAnimationFrame);
  }
  if (pendingResolve) {
    var result = "";
    if (!shutdown) {
      captureCanvas.getContext('2d').drawImage(video, 0, 0, 640, 480);
      result = captureCanvas.toDataURL('image/jpeg', 0.8)
    }
    var lp = pendingResolve;
    pendingResolve = null;
    lp(result);
  }
}

async function createDom() {
  if (div !== null) {
    return stream;
  }
```

```
div = document.createElement('div');
div.style.border = '2px solid black';
div.style.padding = '3px';
div.style.width = '100%';
div.style.maxWidth = '600px';
document.body.appendChild(div);

const modelOut = document.createElement('div');
modelOut.innerHTML = "<span>Status:</span>";
labelElement = document.createElement('span');
labelElement.innerText = 'No data';
labelElement.style.fontWeight = 'bold';
modelOut.appendChild(labelElement);
div.appendChild(modelOut);

video = document.createElement('video');
video.style.display = 'block';
video.width = div.clientWidth - 6;
video.setAttribute('playsinline', '');
video.onclick = () => { shutdown = true; };
stream = await navigator.mediaDevices.getUserMedia(
   {video: { facingMode: "environment"}});
div.appendChild(video);

imgElement = document.createElement('img');
imgElement.style.position = 'absolute';
imgElement.style.zIndex = 1;
imgElement.onclick = () => { shutdown = true; };
div.appendChild(imgElement);

const instruction = document.createElement('div');
instruction.innerHTML =
   '<span style="color: red; font-weight: bold;">' +
```

```
    'When finished, click here or on the video to stop this demo</span>';
  div.appendChild(instruction);
  instruction.onclick = () => { shutdown = true; };

  video.srcObject = stream;
  await video.play();

  captureCanvas = document.createElement('canvas');
  captureCanvas.width = 640; //video.videoWidth;
  captureCanvas.height = 480; //video.videoHeight;
  window.requestAnimationFrame(onAnimationFrame);

  return stream;
}
async function stream_frame(label, imgData) {
  if (shutdown) {
    removeDom();
    shutdown = false;
    return '';
  }

  var preCreate = Date.now();
  stream = await createDom();

  var preShow = Date.now();
  if (label != "") {
    labelElement.innerHTML = label;
  }

  if (imgData != "") {
    var videoRect = video.getClientRects()[0];
    imgElement.style.top = videoRect.top + "px";
    imgElement.style.left = videoRect.left + "px";
```

```
      imgElement.style.width = videoRect.width + "px";
      imgElement.style.height = videoRect.height + "px";
      imgElement.src = imgData;
    }

    var preCapture = Date.now();
    var result = await new Promise(function(resolve, reject) {
      pendingResolve = resolve;
    });
    shutdown = false;
    return {'create': preShow - preCreate,
            'show': preCapture - preShow,
            'capture': Date.now() - preCapture,
            'img': result};
  }
  '')
  display(js)
def video_frame(label, bbox):
  data = eval_js('stream_frame("{}", "{}")'.format(label, bbox))
  return data
# function to convert the JavaScript object into an OpenCV image
def js_to_image(js_reply):
  """
  Params:
        js_reply: JavaScript object containing image from webcam
  Returns:
        img: OpenCV BGR image
  """
  # decode base64 image
  image_bytes = b64decode(js_reply.split(',')[1])
  # convert bytes to numpy array
  jpg_as_np = np.frombuffer(image_bytes, dtype=np.uint8)
  # decode numpy array into OpenCV BGR image
```

```
  img = cv2.imdecode(jpg_as_np, flags=1)
  return img
# function to convert OpenCV Rectangle bounding box image into base64 byte string to be
overlayed on video stream
def bbox_to_bytes(bbox_array):
  """
  Params:
        bbox_array: Numpy array (pixels) containing rectangle to overlay on video stream.
  Returns:
       bytes: Base64 image byte string
  """
  # convert array into PIL image
  bbox_PIL = PIL.Image.fromarray(bbox_array, 'RGBA')
  iobuf = io.BytesIO()
  # format bbox into png for return
  bbox_PIL.save(iobuf, format='png')
  # format return string
  bbox_bytes = '{}'.format((str(b64encode(iobuf.getvalue()), 'utf-8')))
  return bbox_bytes
import cv2
import base64
def ndarray_to_b64(ndarray):
    """
    converts a np ndarray to a b64 string readable by html-img tags
    """
    img = cv2.cvtColor(ndarray, cv2.COLOR_RGB2BGR)
    _, buffer = cv2.imencode('.png', img)
    return base64.b64encode(buffer).decode('utf-8')

from googletrans import Translator

translator = Translator()
```

```
# Now you can use the `translate` method to translate your text
def translate(text):
  if(len(text)==0):
    return ""
  else:
    translation = translator.translate(text , src='en',dest='ta')
    return translation.text



from google.colab.patches import cv2_imshow as imshow
from matplotlib import pyplot as plt
from IPython.display import display, Javascript
from google.colab.output import eval_js

# start streaming video from webcam
video_stream()
# label for video
label_html = 'Capturing...'
# initialze bounding box to empty
bbox = ''
count = 0
js = Javascript('''
  const image = document.createElement("img");
  image.style.width = "60vw"
  image.style.backgroundColor= "blue";
  image.id = 'img'
  image.style.border= "1px solid black";
  image.style.display="block"
  document.body.appendChild(image);
    ''');
display(js)
while True:
```

```
js_reply = video_frame(label_html, bbox)
if not js_reply:
   break
img = js_to_image(js_reply["img"])
frame = img
image_np = np.array(frame)
input_tensor = tf.convert_to_tensor(np.expand_dims(image_np, 0), dtype=tf.float32)
detections = detect_fn(input_tensor)

num_detections = int(detections.pop('num_detections'))
detections = {key: value[0, :num_detections].numpy()
            for key, value in detections.items()}
detections['num_detections'] = num_detections

# detection_classes should be ints.
detections['detection_classes'] = detections['detection_classes'].astype(np.int64)

label_id_offset = 1
image_np_with_detections = image_np.copy()



viz_utils.visualize_boxes_and_labels_on_image_array(
          image_np_with_detections,
          detections['detection_boxes'],
          detections['detection_classes']+label_id_offset,
          detections['detection_scores'],
          category_index,
          use_normalized_coordinates=True,
          max_boxes_to_draw=1,
          min_score_thresh=.7,
          agnostic_mode=False)
js = Javascript('''
  function showImg(src)
```

```
  {
  const image = document.getElementById("img");
  console.log('src',src)
  image.src=""+src;

  }
  ")
display(js)
eval_js('showImg("{0}")'.format(ndarray_to_b64(image_np_with_detections)))
```

REFERENCE

[1]. Das, S., Biswas, S. K., & Purkayastha, B. (2022). A deep sign language recognition system for Indian sign language. *Neural Computing and Applications*, 1-13.

[2]. Rajarajeswari, S., Renji, N. M., Kumari, P., Keshavamurthy, M., & Kruthika, K. (2022). Real-Time Translation of Indian Sign Language to Assist the Hearing and Speech Impaired. In Innovations in Computational Intelligence and Computer Vision (pp. 303-322). Springer, Singapore.

[3]. Sinha, K., Miranda, A. O., & Mishra, S. (2022). Real-Time Sign Language Translator. In *Cognitive Informatics and Soft Computing* (pp. 477-489). Springer, Singapore.

[4]. Mistree, K., Thakor, D., & Bhatt, B. (2021, April). Recognition of Isolated Gestures for Indian Sign Language Using Transfer Learning. In *International Conference on Advances in Computing and Data Sciences* (pp. 107-115). Springer, Cham.

[5]. Sharma, A., Sharma, N., Saxena, Y., Singh, A., & Sadhya, D. (2021). Benchmarking deep neural network approaches for Indian Sign Language recognition. *Neural Computing and Applications*, *33*(12), 6685-6696.

[6]. Sharma, S., Gupta, R., & Kumar, A. (2021). Continuous sign language recognition using isolated signs data and deep transfer learning. *Journal of Ambient Intelligence and Humanized Computing*, 1-12.

[7]. Srivastava, S., Gangwar, A., Mishra, R., & Singh, S. (2021, December). Sign Language Recognition System using TensorFlow Object Detection API. In International Conference on Advanced Network Technologies and Intelligent Computing (pp. 634-646). Springer, Cham.

[8]. Velmula, K. R., Linginani, I., Reddy, K. B., Meghana, P., & Aruna, A. (2021). Indian Sign Language Recognition Using Convolutional Neural Networks. In *Proceedings of International Conference on Advances in Computer Engineering and Communication*

Systems (pp. 393-400). Springer, Singapore.

[9]. Aparna, C., & Geetha, M. (2020). CNN and stacked LSTM model for Indian Sign Language recognition. In *Symposium on Machine Learning and Metaheuristics Algorithms, and Applications* (pp. 126-134). Springer, Singapore.

[10]. Raghuveera, T., Deepthi, R., Mangalashri, R., & Akshaya, R. (2020). A depth-based Indian sign language recognition using microsoft kinect. *Sādhanā*, *45*(1), 1-13.