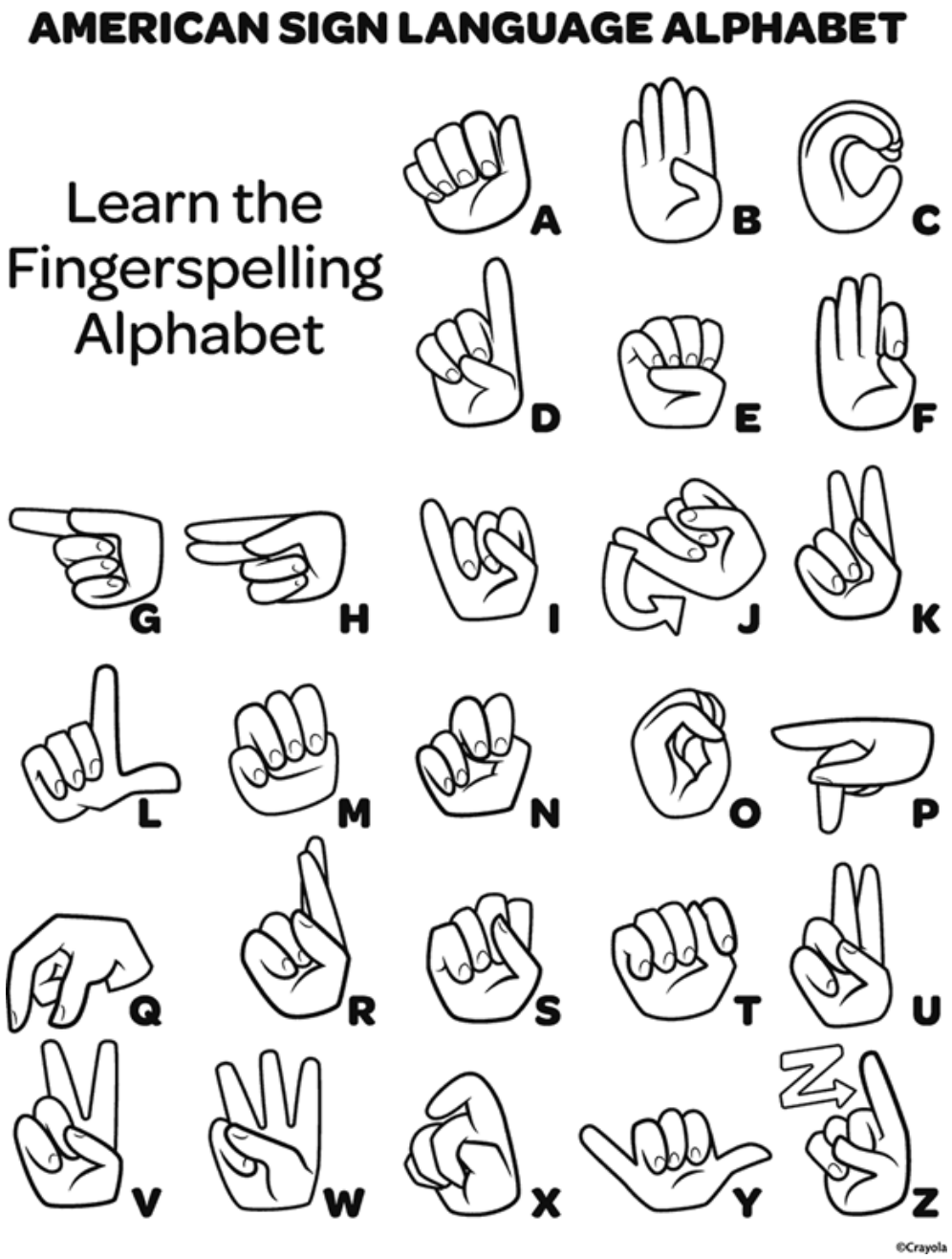
SignLanguage

Introduction:

SignLanguage is used for people who can not speak and hear properly. It helps them to communicate with others. Sign language is expressed through manual articulation in combination with non-manual markers.



AI technologies can play an important role in breaking down the communication barriers of deaf or hearing-impaired people in other communities, contributing significantly to their social inclusion. Artificial intelligence models make sign language easy in terms of recognition, representation, and capturing. Now days Ai algorithms made everything very easy to fulfill the needs of deaf people and hearing-impaired communities.

Artificial Intelligence model For SignLanguage Classification:

While making the model for sign language we need data for training and validation purpose so we have data that had 11391 files.
First I have zipped the file. So I unzipped it by using the command ***! unzip***

```
! unzip "/content/drive/MyDrive/SignLanguage.zip"
```

```
  inflating: SignLanguage/P/2021-07-25_22-22-21.jpg
  inflating: SignLanguage/P/2021-07-25_22-22-25.jpg
  inflating: SignLanguage/P/2021-07-25_22-22-26.jpg
```

After unzipping it I made an object file that I saved with an object***data_set***

```
data_set = "/content/SignLanguage"
```

Now I made an object of data set for visualizing data on screen we need Python library CV2 (computer vision)

```
import cv2
from google.colab.patches import cv2_imshow
img = cv2.imread('/content/SignLanguage/A/2021-07-24_19-09-13.jpg', cv2.IMREAD_UNCHANGED)
cv2_imshow(img)
```

Defining some parameters like batch size, height, and width.

```
batch_size = 32
img_height = 180
img_width = 180
```

In making the model we need some data for training purposes and some data for validation purposes. So with the help of tensor flow, I split data into training and validation, with 80% data in training and 20% data in validation.

```
import tensorflow as tf

train_ds = tf.keras.utils.image_dataset_from_directory(
  data_set,
  validation_split=0.2,
  subset="training",
  seed=123,
  image_size=(img_height, img_width),
  batch_size=batch_size)
```

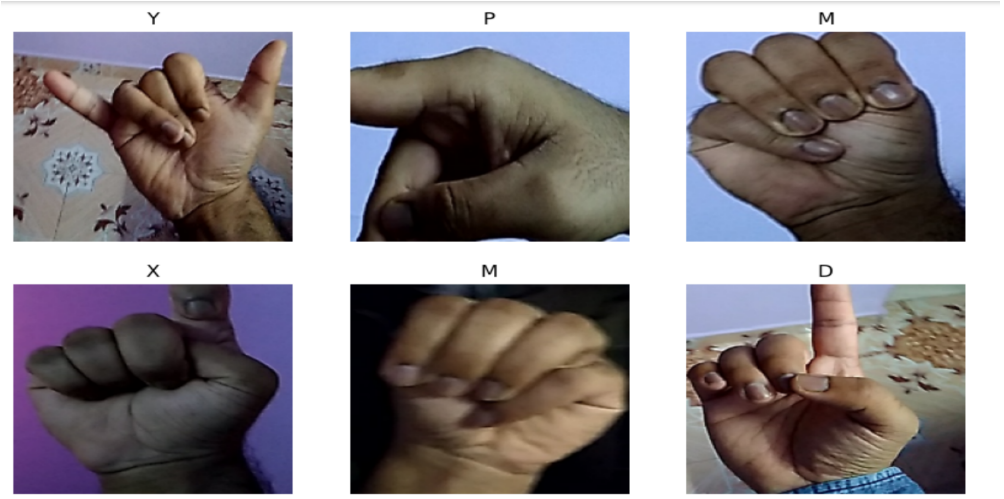
```
Found 11391 files belonging to 25 classes.
Using 9113 files for training.
```

As we can see in the above picture out of 11391,9113 files are split into training and remaining for validation purposes. All 11391 files are divided into 25 classes like “A”, “B”,” C” and so on. As shown in the below image.

```
class_names = train_ds.class_names
print(class_names)
```

```
['A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J', 'K', 'L', 'M', 'N', 'O', 'P', 'Q', 'R', 'S', 'T', 'U', 'V', 'W', 'X', 'Y']
```

Now i am visualizing first few line of training data set.



In the Actual picture, the pixel size of the image is higher and ranges between 0 t0 255. If the pixel size is high it takes too much time to calculate. If the data set is too big we need a high-configuration system. To make it easy we have to standardize it. In Standardization, we divided the data set by 255 to make calculation easy. As shown in the picture below

```
from tensorflow.keras import layers
normalizationa_layer = layers.Rescaling(1./255)
```

```
import numpy as np
normalized_ds = train_ds.map(lambda x, y:(normalizationa_layer(x),y))
image_batch, labels_batch = next(iter(normalized_ds))
first_image = image_batch[1]
print(np.min(first_image), np.max(first_image))
```

```
0.0 0.7114378
```

Hare we can see the maximum size of pixel is 0.7114 and minimum size of pixel is 0.0

Now with the help of tensor flow we have to apply layers. In this model we use three layers Conv2D, Maxpool, and Dense layer with Relu activation function

Keras Conv2D is a 2D Convolution Layer, this layer creates a convolution kernel that is wind with layers input which helps produce a tensor of outputs.

The **tf.layers.maxPooling2d()** function is used to apply max pooling operation on spatial data

The tf.layers.dense() is an inbuilt function of Tensorflow.js library. This function is used to create fully connected layers, in which every output depends on every input.

The ReLU function is the Rectified linear unit. It is the most widely used activation function.

```
num_classes = len(class_names)
model = Sequential([
  layers.Rescaling(1./255, input_shape=(img_height, img_width, 3)),
  layers.Conv2D(16, 3, padding='same', activation='relu'),
  layers.MaxPooling2D(),
  layers.Conv2D(32, 3, padding='same', activation='relu'),
  layers.MaxPooling2D(),
  layers.Conv2D(64, 3, padding='same', activation='relu'),
  layers.MaxPooling2D(),
  layers.Flatten(),
  layers.Dense(128, activation='relu'),
  layers.Dense(num_classes)
])
```

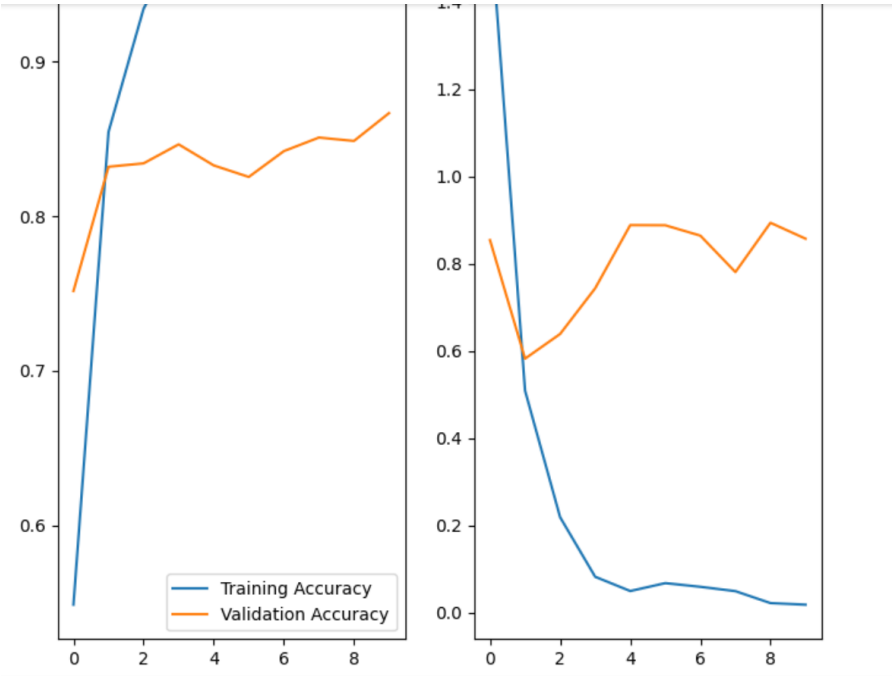
Now choose the Adam optimizer and Sparse Categorical Cross Entropy loss function. To view training and validation accuracy for each training epoch, pass the Metrics argument to compile the model. The compiler takes framework models as input and generates optimized codes for a variety of deep-learning hardware as output.

```
model.compile(optimizer='adam',loss = tf.keras.losses.SparseCategoricalCrossentropy(from_logits = True),
              metrics = ['accuracy'])
```

Now go for training for a model where I took 10 epochs. Training a model is a time-consuming process. A large number of epochs take more time.

```
epochs = 10
history = model.fit(train_ds,validation_data = val_ds,epochs = epochs)
```

After model fitting we have to visualize the mode loss and accuracy through a line graph.



On left figure is for accuracy between training and validation blue line denotes training and the orange line denotes validation On the right side blue line denotes training loss and the orange line denotes validation loss.
But The plots show that training and validation are off by large margins.

In the plots above, the training accuracy is increasing linearly over time, whereas validation accuracy stalls at around 80% in the training process. Also, the difference in accuracy between training and validation is noticeable—a sign of overfitting.

There are multiple ways to fight overfitting in the training process. Now we will use data augmentation and add dropout to your model.
Data augmentation is a technique of artificially increasing the training set by creating modified copies of a dataset using existing data.

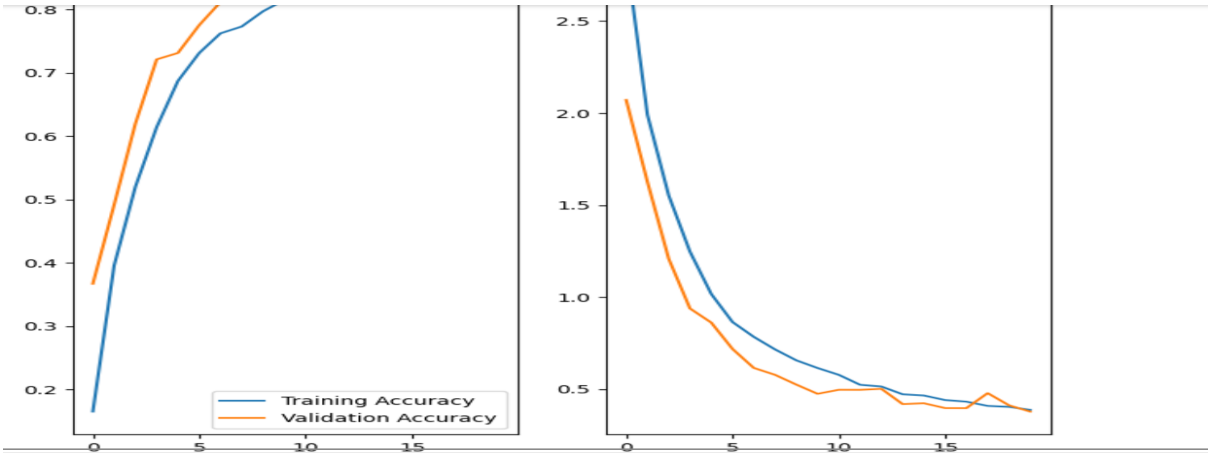
```
from tensorflow import keras
```

```
data_augmentation = keras.Sequential([layers.RandomFlip("horizontal",input_shape=(img_height,img_width,3)),layers.RandomRotation(0.1),layer
```

Now create New Neural Network and use Augmentated data.

```
model = Sequential([
  data_augmentation,
  layers.Rescaling(1./255),
  layers.Conv2D(16, 3, padding='same', activation='relu'),
  layers.MaxPooling2D(),
  layers.Conv2D(32, 3, padding='same', activation='relu'),
  layers.MaxPooling2D(),
  layers.Conv2D(64, 3, padding='same', activation='relu'),
  layers.MaxPooling2D(),
  layers.Dropout(0.2),
  layers.Flatten(),
  layers.Dense(128, activation='relu'),
  layers.Dense(num_classes, name="outputs")
])
```

Now compile data and fit the data with fit model with 20 epochs. And visualize the accuracy and loss.



Now Upload the image and predict it

```
img = cv2.imread('/content/drive/MyDrive/2021-07-25_22-17-38.jpg', cv2.IMREAD_UNCHANGED)
img_array = tf.keras.utils.img_to_array(img)
img_array = tf.expand_dims(img_array, 0)
```

```
predictions = model.predict(img_array)
score = tf.nn.softmax(predictions)
```

```
1/1 [==============================] - 1s 706ms/step
```

```
print("This image most likely belongs to {} with a {:.2f} percent confidence".format(class_names[np.argmax(score)], 100 * np.max(score)))
```

```
This image most likely belongs to C with a 94.72 percent confidence
```

As we can see in the image the accuracy is 94.72%.

The next step is to use Python library to capture the image and by the action of hand sign detect letters
We use the cv2 library for video capture. and a box on the video for recognizing the image. and also use the crop function for crop video images.I use 1 as a waitKey for stopping the framing

```
import cv2
from cvzone.HandTrackingModule import HandDetector
import numpy as np

cap = cv2.VideoCapture(0)
detector = HandDetector(maxHands=1)
while True:
    success, img = cap.read()
    hands, img = detector.findHands(img)
    if hands:
        hand = hands[0]
        x, y, w, h = hand["bbox"]
        imgWhite =  np.ones((imgSize, imgSize ,3),uint8)*255
        imgCrop = img[y-offset : y+h+offset, x-offset: x+w+offset]

        imgCropShape = imgCrop.shape
        imgWhite[0:imgCropShape[0],0:imgCropShape[1]] = imgCrop

        cv2.imshow("imageCrop",imgCrop)
        cv2.imshow("WhiteImage",imgWhite)
    cv2.imshow("Image",img)
    cv2.waitKey(1)
```