This is a documentation of SIBI Translator.

1. Searching suitable dataset for this problem

We found that this dataset <https://www.kaggle.com/grassknoted/asl-alphabet> is the most suitable because it can translate 26 alphabet, space for add space, delete for delete the previous word, and nothing if there was noting in the screen.

1. Download dataset from notebooks using kaggle API

It was more faster to download it via kaggle API than download it manual from kaggle.

1. Unzip dataset that already downloaded via kaggle API
2. Load dataset into train datagen, split it into validation-set and train-set, then do some preprocessing

We split 20% data for validation-set and 80% data for train-set.

For preprocessing, we rescale image data with 1/255 so all data is in range 0-1.

1. Use transfer learning with MobileNet and VGG16

References: <https://www.kaggle.com/timmate/transfer-learning-with-vgg-16-and-mobilenet>

Change the top layer model shape into (200,200,3) like the image shape from dataset.

Set the trainable into false.

Modify the bottom layer model into 29 dense layer, 29 because there was 29 classes.

{'A': 0, 'B': 1, 'C': 2, 'D': 3, 'E': 4, 'F': 5, 'G': 6, 'H': 7, 'I': 8, 'J': 9, 'K': 10, 'L': 11, 'M': 12, 'N': 13, 'O': 14, 'P': 15, 'Q': 16, 'R': 17, 'S': 18, 'T': 19, 'U': 20, 'V': 21, 'W': 22, 'X': 23, 'Y': 24, 'Z': 25, 'del': 26, 'nothing': 27, 'space': 28}

1. Compile model with loss categorical\_cossentropy because it was more than 2 classes, adam optimizer with some learning rate, and set the metrics into accuracy.
2. Training the model and iterate it 5 times
3. Plot graph for the accuracy and validation accuracy to know if the model overfit or not
4. Evaluate the model
5. Save the model so we can load it later
6. Testing the model to make sure the model is really working
7. Pick the best model, is it VGG16 or MobileNet

Actually both are good model, but based on validation-set, MobileNet give a better accuracy.

1. Load model, set the optimizer, then convert it to TFLite model
2. Save the TFLite model so we can use it on android
3. ML Model was pushed to Github.
4. Create a bucket in Cloud Storage for ML model storage.
5. After that, the data on Github is downloaded and then trained again using Notebooks AI Platforms.
6. Start the Notebooks AI Platform API and create an instance.
7. Once created, open Jupyter Notebooks using the AI Platform.
8. train the data and then save the model in Cloud Storage.
9. Set the permissions of the bucket so that it can be fully accessed by the service account's compute engine.
10. Create a VM to deploy the Flask API. API used to connect Android with Storage containing ML models.
11. Connect the VM with the bucket using the Cloud API to allow the API to access Storage.
12. Client can receive and request data to Storage via API.
13. For database, we use Firebase to store data from Android. Unfortunately, we still develop our app. We already train the dataset with AI Platform and then store ML model to bucket but we haven't connected Android and ML model and we still struggle to connect Firebase to Android.