Hand gesture recognition model customization guide

Run in Colab
(https://colab.research.google.com/github/googlesamples/mediapipe/blob/main/examples/customization/ge

License information

Toggle code

The MediaPipe Model Maker package is a low-code solution for customizing on-device machine learning (ML) Models.

This notebook shows the end-to-end process of customizing a gesture recognizer model for recognizing some common hand gestures in the HaGRID (https://www.kaggle.com/datasets/innominate817/hagrid-sample-30k-384p) dataset.

Prerequisites

Install the MediaPipe Model Maker package.

```
$ pip install --upgrade pip
$ pip install mediapipe-model-maker
```

Import the required libraries.

```
from google.colab import files
import os
import tensorflow as tf
assert tf.__version__.startswith('2')

from mediapipe_model_maker import gesture_recognizer

import matplotlib.pyplot as plt
```

Simple End-to-End Example

This end-to-end example uses Model Maker to customize a model for on-device gesture recognition.

Get the dataset

The dataset for gesture recognition in model maker requires the following format: `<dataset_path>/<label_name>/<img_name>.*`. In addition, one of the label names (`label_names`) must be `none`. The `none` label represents any gesture that isn't classified as one of the other gestures.

This example uses a rock paper scissors dataset sample which is downloaded from GCS.

```
!wget https://storage.googleapis.com/mediapipe-tasks/gesture_recognizer/rps_data_s
!unzip rps_data_sample.zip
dataset_path = "rps_data_sample"
```

Verify the rock paper scissors dataset by printing the labels. There should be 4 gesture labels, with one of them being the `none` gesture.

```
print(dataset_path)
labels = []
for i in os.listdir(dataset_path):
  if os.path.isdir(os.path.join(dataset_path, i)):
    labels.append(i)
print(labels)
```

To better understand the dataset, plot a couple of example images for each gesture.

```
NUM_EXAMPLES = 5

for label in labels:
  label_dir = os.path.join(dataset_path, label)
  example_filenames = os.listdir(label_dir)[:NUM_EXAMPLES]
  fig, axs = plt.subplots(1, NUM_EXAMPLES, figsize=(10,2))
  for i in range(NUM_EXAMPLES):
    axs[i].imshow(plt.imread(os.path.join(label_dir, example_filenames[i])))
    axs[i].get_xaxis().set_visible(False)
    axs[i].get_yaxis().set_visible(False)
  fig.suptitle(f'Showing {NUM_EXAMPLES} examples for {label}')

plt.show()
```

Run the example

The workflow consists of 4 steps which have been separated into their own code blocks.

Load the dataset

Load the dataset located at `dataset_path` by using the `Dataset.from_folder` method. When loading the dataset, run the pre-packaged hand detection model from MediaPipe Hands to detect the hand landmarks from the images. Any images without detected hands are ommitted from the dataset. The resulting dataset will contain the extracted hand landmark positions from each image, rather than images themselves.

The `HandDataPreprocessingParams` class contains two configurable options for the data loading process:

- `shuffle`: A boolean controlling whether to shuffle the dataset. Defaults to true.
- `min_detection_confidence`: A float between 0 and 1 controlling the confidence threshold for hand detection.

Split the dataset: 80% for training, 10% for validation, and 10% for testing.

```
data = gesture_recognizer.Dataset.from_folder(
    dirname=dataset_path,
    hparams=gesture_recognizer.HandDataPreprocessingParams()
)
train_data, rest_data = data.split(0.8)
validation_data, test_data = rest_data.split(0.5)
```

Train the model

Train the custom gesture recognizer by using the create method and passing in the training data, validation data, model options, and hyperparameters. For more information on model options and hyperparameters, see the Hyperparameters (#hyperparameters) section below.

```
hparams = gesture_recognizer.HParams(export_dir="exported_model")
options = gesture_recognizer.GestureRecognizerOptions(hparams=hparams)
model = gesture_recognizer.GestureRecognizer.create(
    train_data=train_data,
    validation_data=validation_data,
    options=options
)
```

Evaluate the model performance

After training the model, evaluate it on a test dataset and print the loss and accuracy metrics.

```
loss, acc = model.evaluate(test_data, batch_size=1)
print(f"Test loss:{loss}, Test accuracy:{acc}")
```

Export to Tensorflow Lite Model

After creating the model, convert and export it to a Tensorflow Lite model format for later use on an on-device application. The export also includes model metadata, which includes the label file.

```
model.export_model()
!ls exported_model
```

```
files.download('exported_model/gesture_recognizer.task')
```

Run the model on-device

To use the TFLite model for on-device usage through MediaPipe Tasks, refer to the Gesture Recognizer overview page (https://developers.google.com/mediapipe/solutions/vision/gesture_recognizer) .

Hyperparameters

You can further customize the model using the `GestureRecognizerOptions` class, which has two optional parameters for `ModelOptions` and `HParams`. Use the `ModelOptions` class to customize parameters related to the model itself, and the `HParams` class to customize other parameters related to training and saving the model.

`ModelOptions` has one customizable parameter that affects accuracy:

- `dropout_rate`: The fraction of the input units to drop. Used in dropout layer. Defaults to 0.05.
- `layer_widths`: A list of hidden layer widths for the gesture model. Each element in the list will create a new hidden layer with the specified width. The hidden layers are separated with BatchNorm, Dropout, and ReLU. Defaults to an empty list(no hidden layers).

`HParams` has the following list of customizable parameters which affect model accuracy:

- `learning_rate`: The learning rate to use for gradient descent training. Defaults to 0.001.
- `batch_size`: Batch size for training. Defaults to 2.
- `epochs`: Number of training iterations over the dataset. Defaults to 10.
- `steps_per_epoch`: An optional integer that indicates the number of training steps per epoch. If not set, the training pipeline calculates the default steps per epoch as the training dataset size divided by batch size.
- `shuffle`: True if the dataset is shuffled before training. Defaults to False.
- `lr_decay`: Learning rate decay to use for gradient descent training. Defaults to 0.99.
- `gamma`: Gamma parameter for focal loss. Defaults to 2

Additional `HParams` parameter that does not affect model accuracy:

- `export_dir`: The location of the model checkpoint files and exported model files.

For example, the following trains a new model with the dropout_rate of 0.2 and learning rate of 0.003.

```
hparams = gesture_recognizer.HParams(learning_rate=0.003, export_dir="exported_mod
model_options = gesture_recognizer.ModelOptions(dropout_rate=0.2)
options = gesture_recognizer.GestureRecognizerOptions(model_options=model_options,
model_2 = gesture_recognizer.GestureRecognizer.create(
    train_data=train_data,
    validation_data=validation_data,
    options=options
)
```

Evaluate the newly trained model.

```
loss, accuracy = model_2.evaluate(test_data)
print(f"Test loss:{loss}, Test accuracy:{accuracy}")
```