Copyright 2024 The AI Edge Authors.

Licensed under the Apache License, Version 2.0 (the "License");

Show code

Image classification with TensorFlow Lite Model Maker

View on TensorFlow.org Run in Google Colab View source on GitHub Download notebook See TF Hub model

The TensorFlow Lite Model Maker library simplifies the process of adapting and converting a TensorFlow neural-network model to particular input data when deploying this model for on-device ML applications.

This notebook shows an end-to-end example that utilizes this Model Maker library to illustrate the adaption and conversion of a commonly-used image classification model to classify flowers on a mobile device.

Prerequisites

To run this example, we first need to install several required packages, including Model Maker package that in GitHub repo.

```
!sudo apt -y install libportaudio2
!pip install -q tflite-model-maker
```

Import the required packages.

```
import os

import numpy as np

import tensorflow as tf
assert tf.__version__.startswith('2')

from tflite_model_maker import model_spec
from tflite_model_maker import image_classifier
from tflite_model_maker.config import ExportFormat
from tflite_model_maker.config import QuantizationConfig
from tflite_model_maker.image_classifier import DataLoader

import matplotlib.pyplot as plt
```

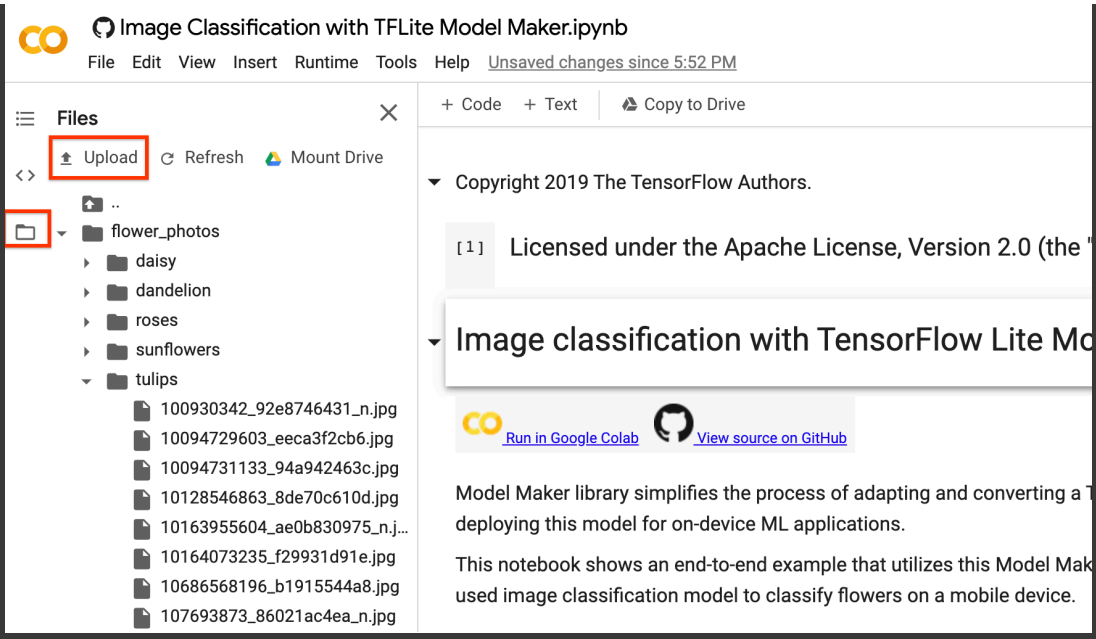
Simple End-to-End Example

Get the data path

Let's get some images to play with this simple end-to-end example. Hundreds of images is a good start for Model Maker while more data could achieve better accuracy.

```
image_path = tf.keras.utils.get_file(
      'flower_photos.tgz',
      'https://storage.googleapis.com/download.tensorflow.org/example_images/flower_photos.tgz',
      extract=True)
image_path = os.path.join(os.path.dirname(image_path), 'flower_photos')
```

You could replace `image_path` with your own image folders. As for uploading data to colab, you could find the upload button in the left sidebar shown in the image below with the red rectangle. Just have a try to upload a zip file and unzip it. The root file path is the current path.



If you prefer not to upload your images to the cloud, you could try to run the library locally following the guide in GitHub.

Run the example

The example just consists of 4 lines of code as shown below, each of which representing one step of the overall process.

Step 1. Load input data specific to an on-device ML app. Split it into training data and testing data.

```
data = DataLoader.from_folder(image_path)
train_data, test_data = data.split(0.9)
```

Step 2. Customize the TensorFlow model.

```
model = image_classifier.create(train_data)
```

Step 3. Evaluate the model.

```
loss, accuracy = model.evaluate(test_data)
```

Step 4. Export to TensorFlow Lite model.

Here, we export TensorFlow Lite model with metadata which provides a standard for model descriptions. The label file is embedded in metadata. The default post-training quantization technique is full integer quantization for the image classification task.

You could download it in the left sidebar same as the uploading part for your own use.

```
model.export(export_dir='.')
```

After these simple 4 steps, we could further use TensorFlow Lite model file in on-device applications like in image classification reference app.

Detailed Process

Currently, we support several models such as EfficientNet-Lite* models, MobileNetV2, ResNet50 as pre-trained models for image classification. But it is very flexible to add new pre-trained models to this library with just a few lines of code.

The following walks through this end-to-end example step by step to show more detail.

Step 1: Load Input Data Specific to an On-device ML App

The flower dataset contains 3670 images belonging to 5 classes. Download the archive version of the dataset and untar it.

The dataset has the following directory structure:

```
flower_photos
|__ daisy
    |______ 100080576_f52e8ee070_n.jpg
    |______ 14167534527_781ceb1b7a_n.jpg
    |______ ...
|__ dandelion
    |______ 10043234166_e6dd915111_n.jpg
    |______ 1426682852_e62169221f_m.jpg
    |______ ...
|__ roses
    |______ 102501987_3cdb8e5394_n.jpg
    |______ 14982802401_a3dfb22afb.jpg
    |______ ...
|__ sunflowers
    |______ 12471791574_bb1be83df4.jpg
    |______ 15122112402_cafa41934f.jpg
    |______ ...
|__ tulips
    |______ 13976522214_ccec508fe7.jpg
    |______ 14487943607_651e8062a1_m.jpg
    |______ ...
```

```
image_path = tf.keras.utils.get_file(
      'flower_photos.tgz',
      'https://storage.googleapis.com/download.tensorflow.org/example_images/flower_photos.tgz',
      extract=True)
image_path = os.path.join(os.path.dirname(image_path), 'flower_photos')
```

Use `DataLoader` class to load data.

As for `from_folder()` method, it could load data from the folder. It assumes that the image data of the same class are in the same subdirectory and the subfolder name is the class name. Currently, JPEG-encoded images and PNG-encoded images are supported.

```
data = DataLoader.from_folder(image_path)
```

Split it to training data (80%), validation data (10%, optional) and testing data (10%).

```
train_data, rest_data = data.split(0.8)
validation_data, test_data = rest_data.split(0.5)
```

Show 25 image examples with labels.

```
plt.figure(figsize=(10,10))
for i, (image, label) in enumerate(data.gen_dataset().unbatch().take(25)):
  plt.subplot(5,5,i+1)
  plt.xticks([])
  plt.yticks([])
  plt.grid(False)
  plt.imshow(image.numpy(), cmap=plt.cm.gray)
  plt.xlabel(data.index_to_label[label.numpy()])
plt.show()
```

Step 2: Customize the TensorFlow Model

Create a custom image classifier model based on the loaded data. The default model is EfficientNet-Lite0.

```
model = image_classifier.create(train_data, validation_data=validation_data)
```

Have a look at the detailed model structure.

```
model.summary()
```

Step 3: Evaluate the Customized Model

Evaluate the result of the model, get the loss and accuracy of the model.

```
loss, accuracy = model.evaluate(test_data)
```

We could plot the predicted results in 100 test images. Predicted labels with red color are the wrong predicted results while others are correct.

```
# A helper function that returns 'red'/'black' depending on if its two input
# parameter matches or not.
def get_label_color(val1, val2):
  if val1 == val2:
    return 'black'
  else:
    return 'red'

# Then plot 100 test images and their predicted labels.
# If a prediction result is different from the label provided label in "test"
# dataset, we will highlight it in red color.
plt.figure(figsize=(20, 20))
predicts = model.predict_top_k(test_data)
for i, (image, label) in enumerate(test_data.gen_dataset().unbatch().take(100)):
  ax = plt.subplot(10, 10, i+1)
  plt.xticks([])
  plt.yticks([])
  plt.grid(False)
  plt.imshow(image.numpy(), cmap=plt.cm.gray)

  predict_label = predicts[i][0][0]
  color = get_label_color(predict_label,
                          test_data.index_to_label[label.numpy()])
  ax.xaxis.label.set_color(color)
  plt.xlabel('Predicted: %s' % predict_label)
plt.show()
```

If the accuracy doesn't meet the app requirement, one could refer to Advanced Usage to explore alternatives such as changing to a larger model, adjusting re-training parameters etc.

Step 4: Export to TensorFlow Lite Model

Convert the trained model to TensorFlow Lite model format with metadata so that you can later use in an on-device ML application. The label file and the vocab file are embedded in metadata. The default TFLite filename is `model.tflite`.

In many on-device ML application, the model size is an important factor. Therefore, it is recommended that you apply quantize the model to make it smaller and potentially run faster. The default post-training quantization technique is full integer quantization for the image classification task.

```
model.export(export_dir='.')
```

See the image classification examples guide for more details about how to integrate the TensorFlow Lite model into mobile apps.

This model can be integrated into an Android or an iOS app using the ImageClassifier API of the TensorFlow Lite Task Library.

The allowed export formats can be one or a list of the following:

- `ExportFormat.TFLITE`
- `ExportFormat.LABEL`
- `ExportFormat.SAVED_MODEL`

By default, it just exports TensorFlow Lite model with metadata. You can also selectively export different files. For instance, exporting only the label file as follows:

```
model.export(export_dir='.', export_format=ExportFormat.LABEL)
```

You can also evaluate the tflite model with the `evaluate_tflite` method.

```
model.evaluate_tflite('model.tflite', test_data)
```

Advanced Usage

The `create` function is the critical part of this library. It uses transfer learning with a pretrained model similar to the tutorial.

The `create` function contains the following steps:

1. Split the data into training, validation, testing data according to parameter `validation_ratio` and `test_ratio`. The default value of `validation_ratio` and `test_ratio` are `0.1` and `0.1`.
2. Download a Image Feature Vector as the base model from TensorFlow Hub. The default pre-trained model is EfficientNet-Lite0.
3. Add a classifier head with a Dropout Layer with `dropout_rate` between head layer and pre-trained model. The default `dropout_rate` is the default `dropout_rate` value from make_image_classifier_lib by TensorFlow Hub.
4. Preprocess the raw input data. Currently, preprocessing steps including normalizing the value of each image pixel to model input scale and resizing it to model input size. EfficientNet-Lite0 have the input scale `[0, 1]` and the input image size `[224, 224, 3]`.
5. Feed the data into the classifier model. By default, the training parameters such as training epochs, batch size, learning rate, momentum are the default values from make_image_classifier_lib by TensorFlow Hub. Only the classifier head is trained.

In this section, we describe several advanced topics, including switching to a different image classification model, changing the training hyperparameters etc.

Customize Post-training quantization on the TensorFLow Lite model

Post-training quantization is a conversion technique that can reduce model size and inference latency, while also improving CPU and hardware accelerator inference speed, with a little degradation in model accuracy. Thus, it's widely used to optimize the model.

Model Maker library applies a default post-training quantization techique when exporting the model. If you want to customize post-training quantization, Model Maker supports multiple post-training quantization options using QuantizationConfig as well. Let's take float16 quantization as an instance. First, define the quantization config.

```
config = QuantizationConfig.for_float16()
```

Then we export the TensorFlow Lite model with such configuration.

```
model.export(export_dir='.', tflite_filename='model_fp16.tflite', quantization_config=config)
```

In Colab, you can download the model named `model_fp16.tflite` from the left sidebar, same as the uploading part mentioned above.

Change the model

Change to the model that's supported in this library.

This library supports EfficientNet-Lite models, MobileNetV2, ResNet50 by now. EfficientNet-Lite are a family of image classification models that could achieve state-of-art accuracy and suitable for Edge devices. The default model is EfficientNet-Lite0.

We could switch model to MobileNetV2 by just setting parameter `model_spec` to the MobileNetV2 model specification in `create` method.

```
model = image_classifier.create(train_data, model_spec=model_spec.get('mobilenet_v2'), validation_data=validation_data)
```

Evaluate the newly retrained MobileNetV2 model to see the accuracy and loss in testing data.

```
loss, accuracy = model.evaluate(test_data)
```

Change to the model in TensorFlow Hub

Moreover, we could also switch to other new models that inputs an image and outputs a feature vector with TensorFlow Hub format.

As Inception V3 model as an example, we could define `inception_v3_spec` which is an object of image_classifier.ModelSpec and contains the specification of the Inception V3 model.

We need to specify the model name `name`, the url of the TensorFlow Hub model `uri`. Meanwhile, the default value of `input_image_shape` is `[224, 224]`. We need to change it to `[299, 299]` for Inception V3 model.

```
inception_v3_spec = image_classifier.ModelSpec(
    uri='https://tfhub.dev/google/imagenet/inception_v3/feature_vector/1')
inception_v3_spec.input_image_shape = [299, 299]
```

Then, by setting parameter `model_spec` to `inception_v3_spec` in `create` method, we could retrain the Inception V3 model.

The remaining steps are exactly same and we could get a customized InceptionV3 TensorFlow Lite model in the end.

Change your own custom model

If we'd like to use the custom model that's not in TensorFlow Hub, we should create and export ModelSpec in TensorFlow Hub.

Then start to define `ModelSpec` object like the process above.

Change the training hyperparameters

We could also change the training hyperparameters like `epochs`, `dropout_rate` and `batch_size` that could affect the model accuracy. The model parameters you can adjust are:

- `epochs`: more epochs could achieve better accuracy until it converges but training for too many epochs may lead to overfitting.
- `dropout_rate`: The rate for dropout, avoid overfitting. None by default.
- `batch_size`: number of samples to use in one training step. None by default.
- `validation_data`: Validation data. If None, skips validation process. None by default.
- `train_whole_model`: If true, the Hub module is trained together with the classification layer on top. Otherwise, only train the top classification layer. None by default.
- `learning_rate`: Base learning rate. None by default.
- `momentum`: a Python float forwarded to the optimizer. Only used when `use_hub_library` is True. None by default.
- `shuffle`: Boolean, whether the data should be shuffled. False by default.
- `use_augmentation`: Boolean, use data augmentation for preprocessing. False by default.
- `use_hub_library`: Boolean, use `make_image_classifier_lib` from tensorflow hub to retrain the model. This training pipeline could achieve better performance for complicated dataset with many categories. True by default.
- `warmup_steps`: Number of warmup steps for warmup schedule on learning rate. If None, the default warmup_steps is used which is the total training steps in two epochs. Only used when `use_hub_library` is False. None by default.
- `model_dir`: Optional, the location of the model checkpoint files. Only used when `use_hub_library` is False. None by default.

Parameters which are None by default like `epochs` will get the concrete default parameters in make_image_classifier_lib from TensorFlow Hub library or train_image_classifier_lib.

For example, we could train with more epochs.

```
model = image_classifier.create(train_data, validation_data=validation_data, epochs=10)
```

Evaluate the newly retrained model with 10 training epochs.

```
loss, accuracy = model.evaluate(test_data)
```

Read more

You can read our image classification example to learn technical details. For more information, please refer to:

- TensorFlow Lite Model Maker guide and API reference.
- Task Library: ImageClassifier for deployment.
- The end-to-end reference apps: Android, iOS, and Raspberry PI.