Experts

Landmark Detection using TensorFlow Hub

Bhavesh Bhatt
@_bhaveshbhatt

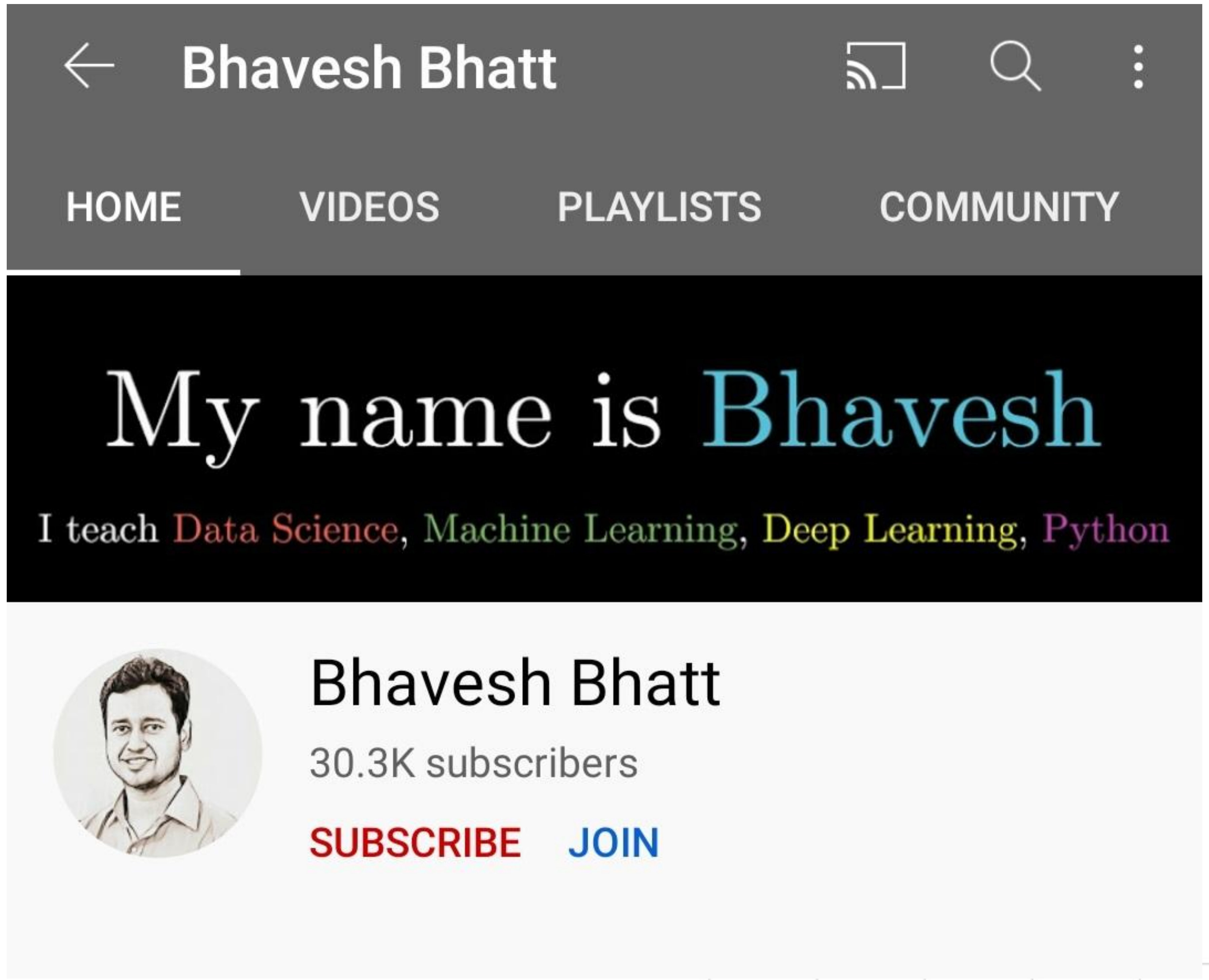
Google Developers

echo $(whoami)

- Google Developer Expert (Machine Learning)

Experts

- Awarded the prestigious 40 Under 40 Data Scientist award by Analytics India Magazine in January 2020.



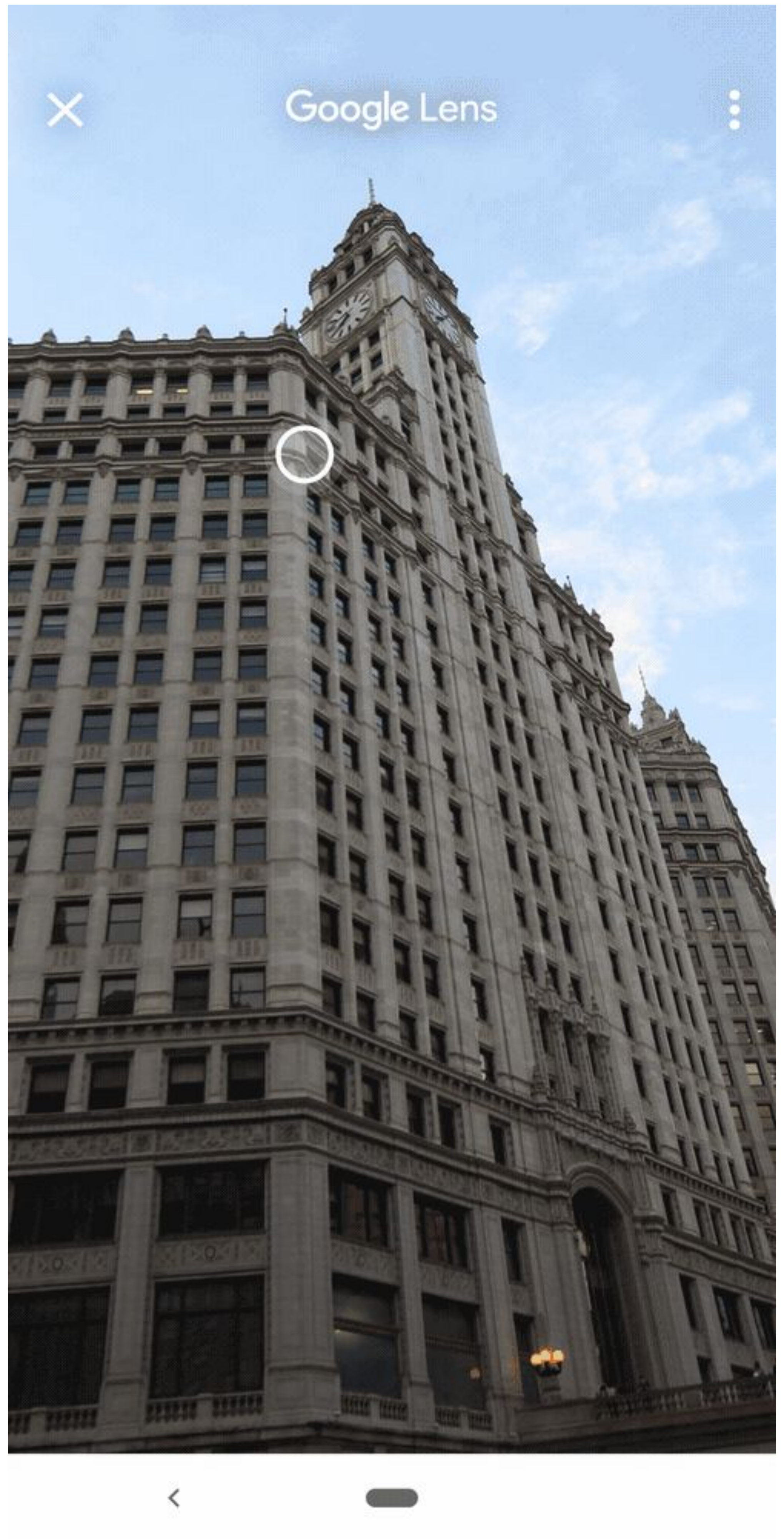


Image Courtesy : https://lens.google.com/

Experts

Challenges


arXiv:1810.04805v2 [cs.CL] 24 May 2019

BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding

Jacob Devlin Ming-Wei Chang Kenton Lee Kristina Toutanova
Google AI Language
{jacobdevlin,mingweichang,kentonl,kristout}@google.com

Abstract

We introduce a new language representation model called **BERT**, which stands for **B**idirectional **E**ncoder **R**epresentations from **T**ransformers. Unlike recent language representation models (Peters et al., 2018a; Radford et al., 2018), BERT is designed to pre-train deep bidirectional representations from unlabeled text by jointly conditioning on both left and right context in all layers. As a result, the pre-trained BERT model can be fine-tuned with just one additional output layer to create state-of-the-art models for a wide range of tasks, such as question answering and language inference, without substantial task-specific architecture modifications.

BERT is conceptually simple and empirically powerful. It obtains new state-of-the-art results on eleven natural language processing tasks, including pushing the GLUE score to 80.5% (7.7% point absolute improvement), MultiNLI accuracy to 86.7% (4.6% absolute improvement), SQuAD v1.1 question answering Test F1 to 93.2 (1.5 point absolute improvement) and SQuAD v2.0 Test F1 to 83.1 (5.1 point absolute improvement).

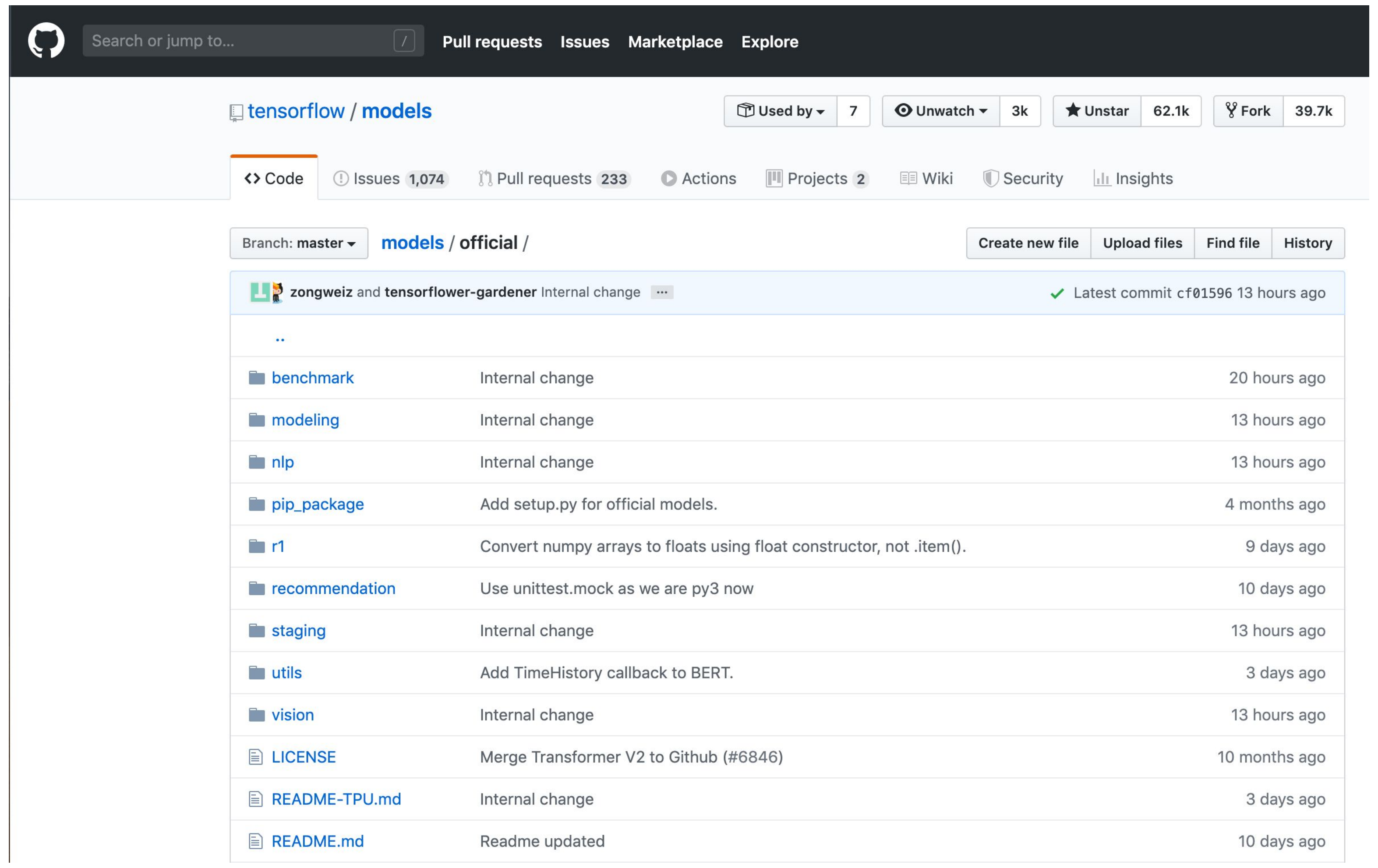
1 Introduction

Language model pre-training has been shown to be effective for improving many natural language processing tasks (Dai and Le, 2015; Peters et al., 2018a; Radford et al., 2018; Howard and Ruder, 2018). These include sentence-level tasks such as natural language inference (Bowman et al., 2015; Williams et al., 2018) and paraphrasing (Dolan and Brockett, 2005), which aim to predict the relationships between sentences by analyzing them holistically, as well as token-level tasks such as named entity recognition and question answering, where models are required to produce fine-grained output at the token level (Tjong Kim Sang and De Meulder, 2003; Rajpurkar et al., 2016).

There are two existing strategies for applying pre-trained language representations to downstream tasks: *feature-based* and *fine-tuning*. The feature-based approach, such as ELMo (Peters et al., 2018a), uses task-specific architectures that include the pre-trained representations as additional features. The fine-tuning approach, such as the Generative Pre-trained Transformer (OpenAI GPT) (Radford et al., 2018), introduces minimal task-specific parameters, and is trained on the downstream tasks by simply fine-tuning *all* pre-trained parameters. The two approaches share the same objective function during pre-training, where they use unidirectional language models to learn general language representations.

We argue that current techniques restrict the power of the pre-trained representations, especially for the fine-tuning approaches. The major limitation is that standard language models are unidirectional, and this limits the choice of architectures that can be used during pre-training. For example, in OpenAI GPT, the authors use a left-to-right architecture, where every token can only attend to previous tokens in the self-attention layers of the Transformer (Vaswani et al., 2017). Such restrictions are sub-optimal for sentence-level tasks, and could be very harmful when applying fine-tuning based approaches to token-level tasks such as question answering, where it is crucial to incorporate context from both directions.

In this paper, we improve the fine-tuning based approaches by proposing BERT: **B**idirectional **E**ncoder **R**epresentations from **T**ransformers. BERT alleviates the previously mentioned unidirectionality constraint by using a "masked language model" (MLM) pre-training objective, inspired by the Cloze task (Taylor, 1953). The masked language model randomly masks some of the tokens from the input, and the objective is to predict the original vocabulary id of the masked



How do I use it?

Is it safe?

Is it fair?

Is it the latest version?

Experts

TensorFlow Hub

A collection of SoTA* pre-trained models published by different teams as well community contributors.

TensorFlow Hub

A comprehensive collection of models

Image

Text

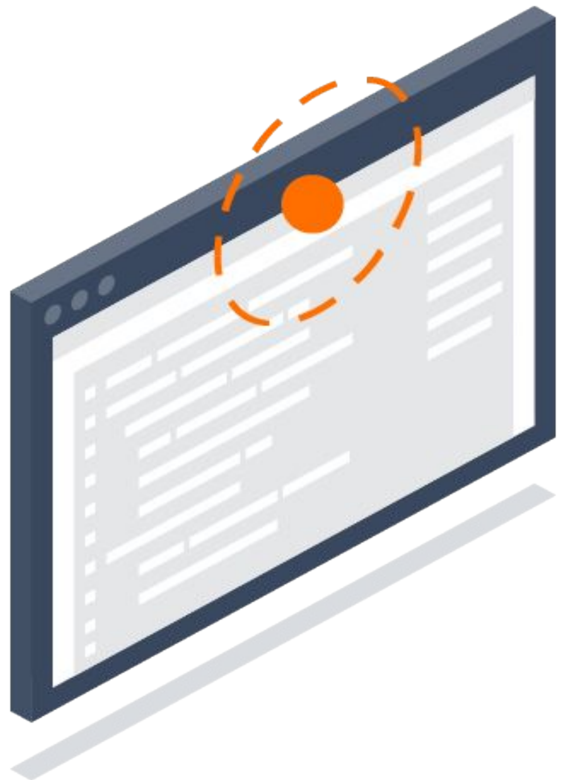
Video

Audio

Ready to Use

Pre-trained models ready for transfer learning on your own datasets and deployable anywhere you want

TensorFlow Extended



TensorFlow .JS

TensorFlow Lite

Coral


INDIA

TensorFlow Hub

Search for models, collections & publishers

Send feedback

Back

Model formats

TF | TFLite (on_device_vision/classifier/landmarks_classifier_asia_V1)

Hub module (v1)

Usage data: 132 Downloads

Fine tunable: No License: Apache-2.0 Last updated: 05/15/2021 Format: Hub module

...dmarks_classifier_asia_V1/1 Copy URL

Download 37.74MB

Landmark recognition model. Tuned for Asia.

Overview

This model is trained to recognize more than 17,771 popular landmarks from Asia. The model is mobile-friendly and can run on-device.

Input

This model takes images as input.

- Inputs are expected to be 3-channel RGB color images of size 321 x 321, scaled to [0, 1].

Output

This model outputs to `prediction:logits` .

- `prediction:logits` : A vector of 98960 similarity scores, corresponding to items in the labelmap (2021-03-18: we fixed label map inconsistency). Note that landmark names are in English, for example, "Fushimi Inari Taisha". Each landmark may appear in the labelmap multiple times; there are 17771 unique labels. Because the output vectors contains duplicate labels, **users of this model should postprocess its output**. To do this, group each dimension of the output vector by the corresponding labels in the label map, and take the highest-scoring logit from each group. For example, for a output vector containing `[0.3, 0.5, 0.1]` and a label map of `['label_1', 'label_2', 'label_1']` , generate `{"label_1": 0.3, "label_2": 0.5}` as your output.

Example use

```
# TF1 version
import tensorflow.compat.v1 as tf
import tensorflow_hub as hub

m = hub.Module('https://tfhub.dev/google/on_device_vision/classifier/landmarks_classifier_asia_V1/1')
...

# TF2 version
import tensorflow.compat.v2 as tf
import tensorflow_hub as hub

m = hub.KerasLayer('https://tfhub.dev/google/on_device_vision/classifier/landmarks_classifier_asia_V1/1')
...
```

```python
import numpy as np
import pandas as pd
import PIL.Image as Image
import tensorflow as tf
import tensorflow_hub as hub

IMAGE_SHAPE = (321, 321)
TF_MODEL_URL = "https://tfhub.dev/google/on_device_vision/classifier/landmarks_classifier_asia_V1/1"
LABEL_MAP_URL = "https://www.gstatic.com/aihub/tfhub/labelmaps/landmarks_classifier_asia_V1_label_map.csv"
```

```
classifier = tf.keras.Sequential([hub.KerasLayer(TF_MODEL_URL,
                                                 input_shape=IMAGE_SHAPE+(3,),
                                                 output_key="predictions:logits")])
```

```
df = pd.read_csv(LABEL_MAP_URL)
label_map = dict(zip(df.id, df.name))

# Preprocess the image
img = np.array(img)/255.0
img = img[np.newaxis, ...]
```

```
result = classifier.predict(img)
print(label_map[np.argmax(result)])
```


```
[13] result

array([[ 0.13966966,  0.21523541,  0.14669707, ..., -0.00261727,
        -0.06463932,  0.06689671]], dtype=float32)
```

```
print(label_map[np.argmax(result)])

'Gateway Of India Mumbai'
```


```
[19] result

     array([[ 0.10243182,  0.10270512,  0.05167606, ...,  0.01087124,
              0.02359215, -0.01156662]], dtype=float32)

[20] print(label_map[np.argmax(result)])

     India Gate
```

Q&A

Experts

Google Developers

Slides & Code available here -

Experts

Experts

Thank You!

Bhavesh Bhatt
@_bhaveshbhatt