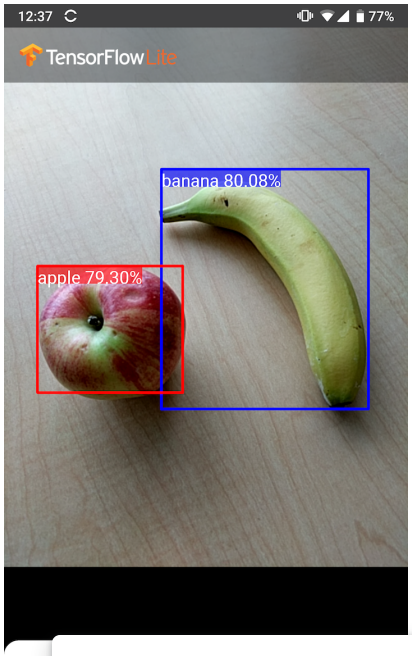
Objekterkennung

Bei einem gegebenen Bild oder einem Videostream kann ein Objekterkennungsmodell identifizieren, welches von einem bekannten Satz von Objekten vorhanden sein könnte, und Informationen über ihre Positionen innerhalb des Bildes bereitstellen.

Dieser Screenshot der Beispielanwendung (#get_started) zeigt beispielsweise (#get_started) , wie zwei Objekte erkannt und ihre Positionen mit Anmerkungen versehen wurden:



Lo

Wenn Sie TensorFlow Lite noch nicht kennen und mit Android oder iOS arbeiten, laden Sie die folgenden Beispielanwendungen herunter, um loszulegen.

Android Beispiel (https://github.com/tensorflow/examples/tree/master/lite/examples/object_detection/anc

iOS Beispiel (https://github.com/tensorflow/examples/tree/master/lite/examples/object_detection/ios)

Wenn Sie eine andere Plattform als Android oder iOS verwenden oder bereits mit den TensorFlow Lite-APIs (https://www.tensorflow.org/api_docs/python/tf/lite) vertraut sind, können Sie das Starterobjekterkennungsmodell (https://www.tensorflow.org/api_docs/python/tf/lite) und die zugehörigen Beschriftungen herunterladen.

Startermodell mit Metadaten herunterladen (https://tfhub.dev/tensorflow/lite-model/ssd_mobilenet_v1/1/

Weitere Informationen zu Metadaten und zugehörigen Feldern (z. B. `labels.txt`) finden Sie unter Lesen der Metadaten aus Modellen (https://www.tensorflow.org/lite/convert/metadata#read_the_metadata_from_models)

Wenn Sie ein benutzerdefiniertes Erkennungsmodell für Ihre eigene Aufgabe trainieren möchten, lesen Sie Modellanpassung (#model_customization) .

Für die folgenden Anwendungsfälle sollten Sie einen anderen Modelltyp verwenden:

- Vorhersage, welches einzelne Etikett das Bild am wahrscheinlichsten darstellt (siehe Bildklassifizierung (https://www.tensorflow.org/lite/models/image_classification/overview))
- Vorhersage der Zusammensetzung eines Bildes, z. B. Motiv gegenüber Hintergrund (siehe Segmentierung (https://www.tensorflow.org/lite/models/segmentation/overview))

Modellbeschreibung

In diesem Abschnitt wird die Signatur für Single-Shot-Detektormodelle (https://arxiv.org/abs/1512.02325) beschrieben, die von der TensorFlow-Objekterkennungs-API (htt orFlow Lite kon

Ein ehrerer Obj verden, die verschiedene Fruchtstücke enthalten, zusammen mit einem *Etikett* , das die Obstklasse angibt,

die sie darstellen (z. B. einen Apfel, eine Banane oder eine Erdbeere), und Daten, die angeben, wo jedes Objekt erscheint das Bild.

Wenn dem Modell anschließend ein Bild zur Verfügung gestellt wird, wird eine Liste der erkannten Objekte, die Position eines Begrenzungsrahmens, der jedes Objekt enthält, und eine Bewertung ausgegeben, die die Sicherheit angibt, dass die Erkennung korrekt war.

Eingabesignatur

Das Modell nimmt ein Bild als Eingabe.

Nehmen wir an, das erwartete Bild ist 300 x 300 Pixel groß und hat drei Kanäle (rot, blau und grün) pro Pixel. Dies sollte dem Modell als abgeflachter Puffer mit 270.000 Bytewerten (300 x 300 x 3) zugeführt werden. Wenn das Modell quantisiert wird (https://www.tensorflow.org/lite/performance/post_training_quantization) , sollte jeder Wert ein einzelnes Byte sein, das einen Wert zwischen 0 und 255 darstellt.

In unserem Beispiel-App-Code (https://github.com/tensorflow/examples/tree/master/lite/examples/object_detection/android) erfahren Sie, wie diese Vorverarbeitung unter Android durchgeführt wird.

Ausgabesignatur

Das Modell gibt vier Arrays aus, die den Indizes 0-4 zugeordnet sind. Die Arrays 0, 1 und 2 beschreiben `N` erkannte Objekte, wobei jedem Element ein Element in jedem Array entspricht.

Index	Name	Beschreibung
0	Standorte	Mehrdimensionales Array von [N] [4] Gleitkommawerten zwischen 0 und 1, wobei die inneren Arrays Begrenzungsrahmen in der Form [oben, links, unten, rechts] darstellen.
1	Klassen	Array von N Ganzzahlen (Ausgabe als Gleitkommawerte), die jeweils den Index einer Klassenbezeichnung aus der Beschriftungsdatei angeben
2		keit darstellen,
3		

is: Die Anzahl der Ergebnisse (10 im obigen Fall) ist ein Parametersatz beim Exportieren des Erkennungsmod TensorFlow Lite. Weitere Informationen finden Sie unter Modellanpassung (#model_customization) .

Stellen Sie sich zum Beispiel vor, ein Modell wurde trainiert, um Äpfel, Bananen und Erdbeeren zu erkennen. Wenn ein Bild bereitgestellt wird, wird eine festgelegte Anzahl von Erkennungsergebnissen ausgegeben - in diesem Beispiel 5.

Klasse	Ergebnis	Ort
Apfel	0,92	[18, 21, 57, 63]
Banane	0,88	[100, 30, 180, 150]
Erdbeere	0,87	[7, 82, 89, 163]
Banane	0,23	[42, 66, 57, 83]
Apfel	0,11	[6, 42, 31, 58]

Vertrauenspunktzahl

Um diese Ergebnisse zu interpretieren, können wir die Punktzahl und den Ort für jedes erkannte Objekt betrachten. Die Punktzahl ist eine Zahl zwischen 0 und 1, die die Sicherheit angibt, dass das Objekt tatsächlich erkannt wurde. Je näher die Zahl an 1 liegt, desto sicherer ist das Modell.

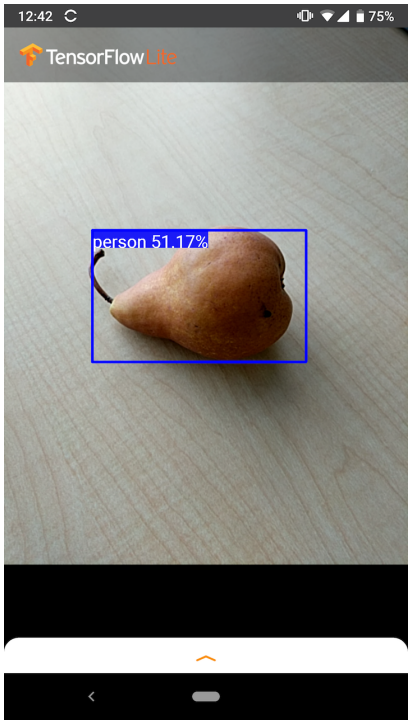
Abhängig von Ihrer Anwendung können Sie einen Grenzwert festlegen, unter dem Sie die Erkennungsergebnisse verwerfen. Für das aktuelle Beispiel ist ein vernünftiger Grenzwert eine Punktzahl von 0,5 (was einer 50% igen Wahrscheinlichkeit entspricht, dass die Erkennung gültig ist). In diesem Fall werden die letzten beiden Objekte im Array ignoriert, da diese Konfidenzwerte unter 0,5 liegen:

Klasse	Ergebnis	Ort
Apf		
Ban		
Erdb		

Klasse	Ergebnis	Ort
~~Banane~~	~~0,23~~	~~[42, 66, 57, 83]~~
~~Apfel~~	~~0,11~~	~~[6, 42, 31, 58]~~

Der von Ihnen verwendete Grenzwert sollte darauf basieren, ob Sie mit falsch positiven Ergebnissen (falsch identifizierte Objekte oder Bereiche des Bildes, die fälschlicherweise als Objekte identifiziert werden, wenn dies nicht der Fall ist) oder mit falsch negativen (echten Objekten) besser vertraut sind verpasst, weil ihr Selbstvertrauen gering war).

Im folgenden Bild wurde beispielsweise eine Birne (die kein Objekt ist, für dessen Erkennung das Modell trainiert wurde) fälschlicherweise als "Person" identifiziert. Dies ist ein Beispiel für ein falsches Positiv, das durch Auswahl eines geeigneten Grenzwerts ignoriert werden kann. In diesem Fall würde ein Grenzwert von 0,6 (oder 60%) das falsch positive Ergebnis bequem ausschließen.



Ort

Für n
Beg e
Sta

[oben, links, Unterseite, Recht]]

Der obere Wert gibt den Abstand der oberen Kante des Rechtecks vom oberen Bildrand in Pixel an. Der linke Wert repräsentiert den Abstand des linken Randes von der linken Seite des Eingabebildes. Die anderen Werte repräsentieren den unteren und rechten Rand auf ähnliche Weise.

is: Objekterkennungsmodelle akzeptieren Eingabebilder einer bestimmten Größe. Dies unterscheidet sich cheinlich von der Größe des von der Kamera Ihres Geräts aufgenommenen Rohbilds. Sie müssen Code schrei s Rohbild zuzuschneiden und an die Eingabegröße des Modells anzupassen (Beispiele hierfür finden Sie in ur elanwendungen (#get_started)). .

m Modell ausgegebenen Pixelwerte beziehen sich auf die Position im zugeschnittenen und skalierten Bild. Si n sie daher so skalieren, dass sie zum Rohbild passen, um sie korrekt zu interpretieren.

Leistungsbenchmarks

Leistungsbenchmarkzahlen für unser Startermodell (https://storage.googleapis.com/download.tensorflow.org/models/tflite/coco_ssd_mobilenet werden mit dem hier beschriebenen (https://www.tensorflow.org/lite/performance/benchmarks) Tool generiert.

Modellname	Modellgröße	Gerät	GPU	Zentralprozessor
COCO SSD MobileNet v1 (https://tfhub.dev/tensorflow/lite-model/ssd_mobilenet_v1/1/metadata/1?lite-format=tflite)	27 Mb	Pixel 3 (Android 10)	22ms	46ms *
		Pixel 4 (Android 10)	20ms	29ms *
				11ms **

* 4 Threads verwendet.

** 2 auf dem iPhone verwendete Threads für das beste Leistungsergebnis.

Modellanpassung

Vorgefertigte Modelle

Mobiloptimierte Erkennungsmodelle mit einer Vielzahl von Latenz- und Präzisionseigenschaften finden Sie im Erkennungszoo (https://github.com/tensorflow/models/blob/master/research/object_detection/g3doc/tf1_detection_zoo.md#mobile-models). Jeder von ihnen folgt den in den folgenden Abschnitten beschriebenen Eingabe- und Ausgabesignaturen.

Die meisten Download-Zips enthalten eine `model.tflite` Datei. Wenn es keinen gibt, kann mit diesen Anweisungen (https://github.com/tensorflow/models/blob/master/research/object_detection/g3doc/running_on_mobile_tensorflowlite.md) ein TensorFlow Lite-Flatbuffer generiert werden. SSD-Modelle aus dem TF2 Object Detection Zoo (https://github.com/tensorflow/models/blob/master/research/object_detection/g3doc/tf2_detection_zoo.md) können mithilfe der Anweisungen hier (https://github.com/tensorflow/models/blob/master/research/object_detection/g3doc/running_on_mobile_tf2.md) auch in TensorFlow Lite konvertiert werden (https://github.com/tensorflow/models/blob/master/research/object_detection/g3doc/running_on_mobile_tf2.md). Es ist wichtig zu beachten, dass Erkennungsmodelle nicht direkt mit dem TensorFlow Lite Converter (https://www.tensorflow.org/lite/convert) konvertiert werden können, da sie einen Zwischenschritt zum Generieren eines mobilfreundlichen Quellmodells (https://www.tensorflow.org/lite/convert) erfordern. Die oben verlinkten Skripte führen diesen Schritt aus.

Sow
(htt
_ten
TF2
(htt
_tensorflowlite.md)

Exportskripte verfügen über Parameter, die eine größere Anzahl von Ausgabeobjekten oder eine langsamere, genauere Nachbearbeitung ermöglichen. Verwenden Sie `--help` für die Skripte, um eine vollständige Liste der unterstützten Argumente `--help` .

Derzeit wird die Inferenz auf dem Gerät nur mit SSD-Modellen optimiert. Eine bessere Unterstützung für andere Architekturen wie CenterNet und EfficientDet wird untersucht.

Wie wähle ich ein Modell zum Anpassen aus?

Jedes Modell verfügt über eine eigene Präzision (quantifiziert durch den mAP-Wert) und Latenzmerkmale. Sie sollten ein Modell auswählen, das für Ihren Anwendungsfall und die beabsichtigte Hardware am besten geeignet ist. Zum Beispiel sind die Edge-TPU-
(https://github.com/tensorflow/models/blob/master/research/object_detection/g3doc/tf1_detection_zoo.md#pixel4-edge-tpu-models)
Modelle ideal für Rückschlüsse auf Googles Edge-TPU auf Pixel 4.

Mit unserem Benchmark-Tool (https://www.tensorflow.org/lite/performance/measurement) können Sie Modelle bewerten und die effizienteste verfügbare Option auswählen.

Feinabstimmung von Modellen an benutzerdefinierten Daten

Die von uns bereitgestellten vorgefertigten Modelle sind darauf trainiert, 90 Objektklassen zu erkennen. Eine vollständige Liste der Klassen finden Sie in der Beschriftungsdatei in den Modellmetadaten
(https://tfhub.dev/tensorflow/lite-model/ssd_mobilenet_v1/1/metadata/1?lite-format=tflite) .

Sie können eine als Transferlernen bekannte Technik verwenden, um ein Modell neu zu trainieren, um Klassen zu erkennen, die nicht im ursprünglichen Satz enthalten sind. Sie können das Modell beispielsweise neu trainieren, um mehrere Gemüsesorten zu erkennen, obwohl die ursprünglichen Trainingsdaten nur ein Gemüse enthalten. Dazu benötigen Sie für jedes der neuen Labels, die Sie trainieren möchten, eine Reihe von Trainingsbildern. In unserem Colab
(htt rials/eager_few
_sh
zur
(htt rials/eager_few
_sh
ein Beispielen.

Informationen zur Feinabstimmung mit größeren Datensätzen finden Sie in diesen Handbüchern zum Trainieren Ihrer eigenen Modelle mit der TensorFlow-Objekterkennungs-API: TF1 (https://github.com/tensorflow/models/blob/master/research/object_detection/g3doc/tf1_training_and_evaluation.md), TF2 (https://github.com/tensorflow/models/blob/master/research/object_detection/g3doc/tf2_training_and_evaluation.md). Nach dem Training können sie mit den folgenden Anweisungen in ein TFLite-freundliches Format konvertiert werden: TF1 (https://github.com/tensorflow/models/blob/master/research/object_detection/g3doc/running_on_mobile_tensorflowlite.md), TF2 (https://github.com/tensorflow/models/blob/master/research/object_detection/g3doc/running_on_mobile_tensorflowlite.md)