Wally Image Processing model - AutoML Vision vs. Handmade model
Projektdokumentation

von
Taras Sologub | Mat. Nr. 2153832
Tom Klanthe | Mat. Nr. 2285567

Unsere Ursprungsidee war es, den Haar Cascade Algorithmus von OpenCV zu verwenden, damit wir Gesichter auf einem Bild erkennen und extrahieren können. Dieser Algorithmus arbeitet perfekt mit Bildern, auf denen menschliche Gesichter vorhanden sind, aber nicht mit Cartoon Gesichtern. Wir hätten den Algorithmus für unsere Zwecke weiterentwickeln können, aber es würde uns viel Aufwand und Zeit kosten. Deswegen haben wir uns für folgende Implementierungen entschieden.

Tensorflow Lösung

Wir verwenden die Open-Source Machine Learning Bibliothek Tensorflow, mit dem vortrainierten Model “Faster RCNN Inception v2”. Mit Hilfe von OpenCV nehmen wir einen Live-Video über die Webcam auf und benutzen Tensorflow um zu versuchen Waldo zu finden.

Dataset Vorbereitung

Zielwerte für einfache maschinelle Lernprobleme sind in der Regel Skalare oder eine kategorische Zeichenkette. Die Trainingsdaten der Tensorflow Object Detection API verwendet eine Kombination aus beidem. Es besteht aus einem Satz von Bildern, die mit Beschriftungen der gewünschten Objekte und den Positionen, an denen sie in einem Bild erscheinen, versehen sind. Positionen werden mit zwei Punkten definiert, da (im 2D-Raum) zwei Punkte ausreichen, um einen Begrenzungsrahmen um ein Objekt zu zeichnen.

Um das Trainingsset zu erstellen, müssen wir also eine Reihe von Where's Waldo - Puzzle-Bildern mit den Stellen, an denen Waldo erscheint, erstellen. Wir haben zwei Waldo Bücher verwendet und diese komplett mit verschiedenen Auflösungen und unterschiedlichen Bildqualitäten abfotografiert. Es wurden ~100 Bilder gesammelt. 20% davon haben wir für die Evaluierung des Datasets genommen. Unser nächster Schritt ist es alle Bildern zu labeln. Mit Hilfe der “LabelImg” Anwendung haben wir die xml Dateien mit x und y Koordinaten von Waldo für jedes Bild erstellt. Mit Hilfe von `xml_to_csv.py` konvertieren wir alle xmls in eine csv Datei. Die Konvertierung braucht man für das Training und Evaluierung des Datasets.

filename	width	height	class	xmin	ymin	xmax	ymax
1.jpg	2048	1251	waldo	706	513	743	562

Die Training-csv und das Training-Dataset müssen in eine `.tfrecord` Datei gepackt werden. Dafür haben wir das Script `create_tf_record.py` benutzt. Genau so geschieht es mit der Evaluierungs-csv und dem Evaluierungs-Dataset. Jetzt haben wir zwei `.tfrecord` Dateien: `train.record` und `eval.record`.

Model Vorbereitung

Die Tensorflow Object Detection API bietet eine Reihe von vortrainierten Modellen mit unterschiedlichen Leistungen (in der Regel ein Kompromiss zwischen Geschwindigkeit und Genauigkeit), die auf mehreren öffentlichen Datensätzen trainiert wurden. Während das Modell von

Grund auf neu trainiert werden könnte, beginnend mit zufällig initialisierten Gewichtungen, würde dieser Prozess wahrscheinlich mehrere Wochen dauern. Stattdessen haben wir eine Methode namens Transferlernen verwendet: Ein Modell, welches normalerweise trainiert wird, um ein allgemeines Problem zu lösen, wird “re-trainiert”um unser Problem zu lösen.

Wir haben ein “RCNN Inception v2” Model verwendet, welcher auf ein COCO Dataset trainiert wird. Um es für unsere Zwecke weiter zu trainieren brauchen wir ein Pipeline Konfigurationsskript - `faster_rcnn_inception_v2_coco.config`. Die folgenden Parameter müssen dort angepasst werden:
- `fine_tune_checkpoint` - Pfad zu den Checkpoints on Google Cloud Bucket.
- `num_steps` - Anzahl von Iterationen, in unserem Fall 200 000
- `input_path` - für train_input_reader (Pfad zur train.record Datei)
- `input_path` - für eval_input_reader (Pfad zur eval.record Datei)
- `num_examples` - Größe des Evaluierungs Dataset
- `max_evals` - Anzahl von Iterationen für jede Evaluierung

Die letzte Datei, die konfiguriert werden muss, ist die `labels.txt` Map-Datei, welche die Labels aller unserer verschiedenen Objekte enthält. Da wir nur nach einem Objekttyp suchen, besteht unsere Datei nur aus einem Waldo Objekt.

Training
Für das Training haben wir Google Cloud verwenden. Die `.tfrecord` Dateien und andere training configs wurden in das Bucket hochgeladen, damit der Trainingsjob auf sie referenzieren kann. Die allgemeine Regel für den Abbruch des Trainings ist, wenn der Verlust auf unserem Evaluierungsset nicht mehr abnimmt oder generell sehr gering ist (in unserem Fall unter 0,01).
Nach ein paar Stunden und 70 000 Iterationen haben wir ein Model bekommen, welches Waldo auf den neuen Bildern identifizieren kann.

Google Cloud AutoML Vision Lösung

Mit Google Cloud Vision ist es möglich eigene trainierte ML Modelle zu erstellen und Bilder gegen diese zu evaluieren.
Zuerst muss man einen Datensatz aus Bildern erstellen. Die Bilder werden in die Google Cloud hochgeladen und einem oder mehreren Labels zugewiesen. Die Labels müssen vom Benutzer selbst erstellt werden. Danach wird ein Model aus den Bilddaten generiert. Wenn das Model fertig trainiert wurde, kann man ein Bild über die Google Cloud API an den Service schicken. Dieses wird dann durch das Model analysiert und man bekommt einen Wert zwischen 0 und 1 zurück. Je höher dieser Wert, desto wahrscheinlicher ist es, dass das Objekt, mit dem man das Model trainiert hat, in diesem Bild vorhanden ist.

In Bezug auf dieses Projekt wollten wir in der Google Cloud ein eigenes Modell, welches Waldo Bilddaten beinhaltet, trainieren und auf die gefilmten Buchseiten anwenden.
Unsere erste Herangehensweise war es OpenCV zu nutzen um alle Gesichter in einem Waldo Gesamtbild zu finden und auszuschneiden. Diese Gesichter sollten dann über die API an die Google Cloud gesendet und analysiert werden. Das Problem bei dieser Methode war, dass die verschiedenen in OpenCV mitgelieferten Algorithmen für die Gesichtserkennung nicht mit den Gesichtern in dem Waldo Buch kompatibel waren. Bei unseren Tests mit verschiedenen Variationen

des Haar Cascade Classifier Algorithmus (für die Gesichtserkennung) wurde kein einziges Gesicht erkannt. Das liegt höchstwahrscheinlich daran, dass es sich in unserem Fall um Cartoon Gesichter handelt, welche nicht die Merkmale von normalen Gesichtern ausweisen.

Letztendlich haben wir uns für eine andere Methode entschieden, welche wir auch in unserem Python Script umgesetzt haben (`predict_wally.py` - zu finden im Ordner "autoML" des Repositories).
Das Programm besteht aus 2 Hauptteilen: Das Aufteilen des Gesamtbildes in kleinere Bilder und dem Senden der Bilddaten an den Google AutoML Service.
Das Aufteilen des Bildes in kleinere Bilder ist notwendig, damit uns der Service die Information zurückgeben kann, wie groß die Wahrscheinlichkeit ist, dass Waldo in einem Ausschnitt des Bildes vorhanden ist. Würde man das Gesamtbild an den Service schicken, bekäme man nur die Wahrscheinlichkeit zurück, ob Waldo vorhanden sei oder nicht. Waldo ist schließlich immer vorhanden. Außerdem wäre das Ergebnis wegen der Komplexität des Bildes sehr ungenau.

Das Aufteilen des Bildes findet mit Hilfe von OpenCV in der `crop_image()` Methode statt. Sie nimmt den Pfad des Bildes, und die Größe der Ausschnitte (die Bilder sind immer quadratisch) als Argument. Die zugeschnittenen Bilder werden dann im Ordner `cropped_images()` gespeichert.

Im zweiten Teil des Programms werden die Bilder per API an die Google Cloud gesendet und evaluiert. In Zeile 41 bis 49 wird der Zugriff auf den korrekten Google Service eingerichtet.
Die `get_prediction()` Methode nimmt ein Bild als Argument und wird für jedes zuvor ausgeschnittene Bild aufgerufen. Die `predict()` Methode in Zeile 64 bekommt als Argumente die Model ID unseres Models, einen Payload (ein Objekt mit dem ausgeschnittenen Bild) und eventuelle Parameter (in unserem Fall nur den Parameter `score_threshold`, welcher aber auf den Wert 0.0 gesetzt wurde um keine Werte bzw. Bilder zu filtern).
In der Google Cloud haben wir drei unterschiedliche Models erstellt, welche jeweils Bildausschnitte beinhalten, in denen Waldo zu sehen ist. Die Unterschiede bei diesen Models sind nur die Größe der Ausschnitte: 64 * 64, 128 * 128 und 256 * 256 Pixel.
Die Ergebnisse die uns der Service wiedergibt, werden in drei Arrays gespeichert, welche für die jeweilige Größe der in der Cloud vorhandenen Models stehen.
In Zeile 90 bis 99 wird das Bild, welches die höchste Wahrscheinlichkeit besitzt, Waldo zu beinhalten, ausgegeben. Wenn man das Programm startet, muss man als zweites Argument entweder 64, 128 oder 256 benutzen, da dies die Größe der Ausschnitte, sowie das zu benutzende Array bestimmt.

Da die Abfragen an den Google Service nur hintereinander erfolgen und die Geschwindigkeit dieser Abfragen stark von der Uploadgeschwindigkeit und der Menge der Bilder abhängt, kann ein Durchlauf des Programms mehrere Minuten dauern. Deswegen würde diese Methode mit einem Echtzeit Videobild nicht funktionieren.

Fazit
Unser Projekt ist zum Großteil erfolgreich gewesen, da wir es geschafft haben ein fertiges Programm zu entwickeln, welches Waldo über den Webcam Videofeed und auf den Webcam Fotos mit einer Genauigkeit von 85% findet. Ein kleiner Wermutstropfen ist jedoch, dass unsere Google Cloud AutoML Vison Implementierung nicht funktioniert hat. Dennoch haben wir viel über diese für uns neuen Technologien gelernt.