UE 5 – Detektion mit Transfer Learning Methoden

Vorbereitung

Überblick

In dieser Übung widmen wir uns wieder der Detektionsaufgabe zu und nutzen dafür Deep Learning Verfahren. Das Ziel ist das Implementieren eines Detektionsmodells unter Einsatz vortrainierter Deep Learning basierter Detektionsmodelle (Transfer Learning). Da das Training solcher Modelle sehr rechenintensiv ist, könnt ihr euch die cloudbasierte Deep Learning Umgebung von Google mit freiem Zugang zu Servers mit GPUs (Google Colab) zunutze machen.

Aufgabe

Bevor ihr mit der Implementierung beginnt, solltet ihr folgende Fragen beantwortet haben:

- Was ist Transfer Learning (Siehe auch Vorlesungsinhalte)?
- Was sind grundlegende Elemente eines Detektionsmodells?

Vertiefte Informationen zum Thema findet ihr im Paper Deep Learning for Generic Object Detection: A Survey. Das Paper ist sehr umfangreich und bietet einen umfassenden Überblick über Deep Learning basierte Detektionsverfahren. Richtet eure Aufmerksamkeit insbesondere auf Kapitel 3, 4.2 und 5.

Das Schema in Abbildung 1 visualisiert die einzelnen Elemente eines Detektionsmodells. Anschließend folgt deren stichpunktartige Beschreibung.

Schritt 1: Merkmalsextraktion (Feature Extraction)

- Merkmalsextraktion wird üblicherweise mit Hilfe von ConvNets durchgeführt.
- Typische Merkmalsextraktoren sind VGG, ResNet und Inception.
- Der Unterschied zwischen der Merkmalsextraktion und der Klassifikation mit ConvNets liegt darin, dass die Fully-Connected Layer, die Klassifikationsaufgabe übernehmen, nicht vorhanden sind.
- Merkmalsextraktion ist die rechenintensivste Komponente eines Detektionsnetzes.

用ConvNets进行特征提取和分类的区别在于，承担分类任务的全连接层并不存在。

Schritt 2: Extraktion von Feature-Vektoren mit „*Prior*“ oder „*Anchor*“ Boxen

- *Anchor*-Boxen sind eine Reihe von vordefinierten Begrenzungsboxen mit einer bestimmten Höhe und Breite.
- Die Größe von *Anchor*-Boxen sollte basierend auf Objektgröße definiert werden.
- *Anchor*-Boxen dienen zur Erfassung vom Maßstab und vom Seitenverhältnis gegebener Objektklassen.
- *Anchor*-Boxen sind gleichmäßig verteilt über das ganze Bild.

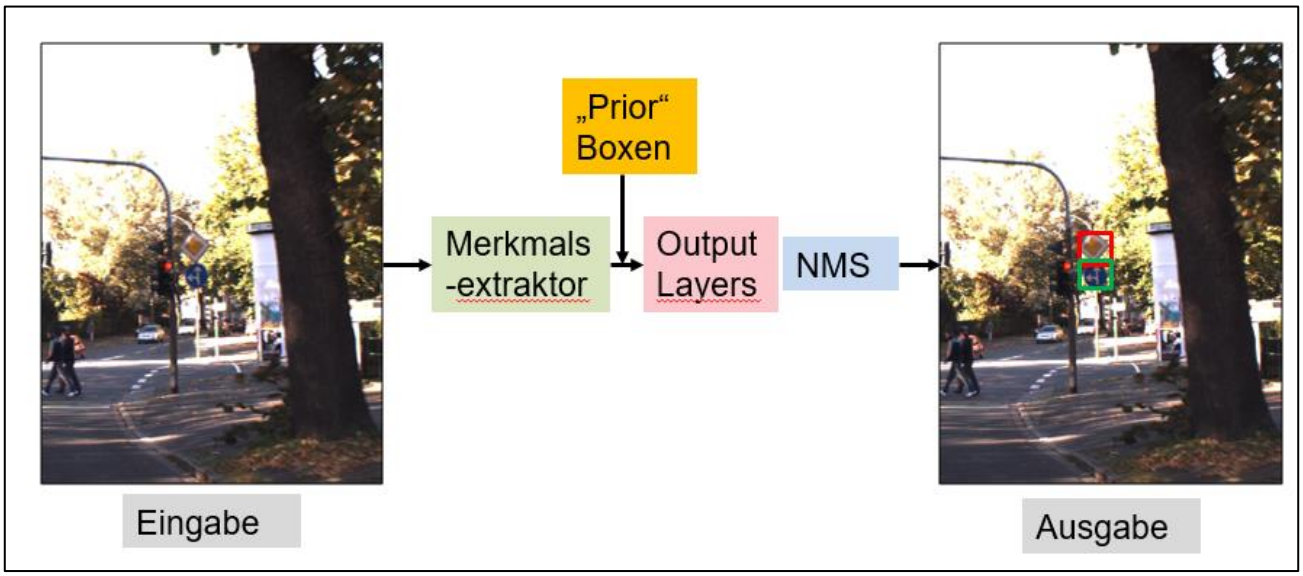

Abbildung 1: Schematische Darstellung eines Deep Learning basierten Detektionsmodells

- Durch die Vorgabe eventueller Position und Größe der Boxen im Bild beschleunigen / erleichtern *Anchor*-Boxen die Detektion von Objekten.
- Für jeden Pixel der Feature-Map wird ein Feature-Vektor fester Größe definiert (siehe Abbildung 2). Bei einer 3x3xd Faltung entsteht ein 1x1xd Feature-Vektor für jeden Anchor, der mit diesem Pixel assoziiert wird.

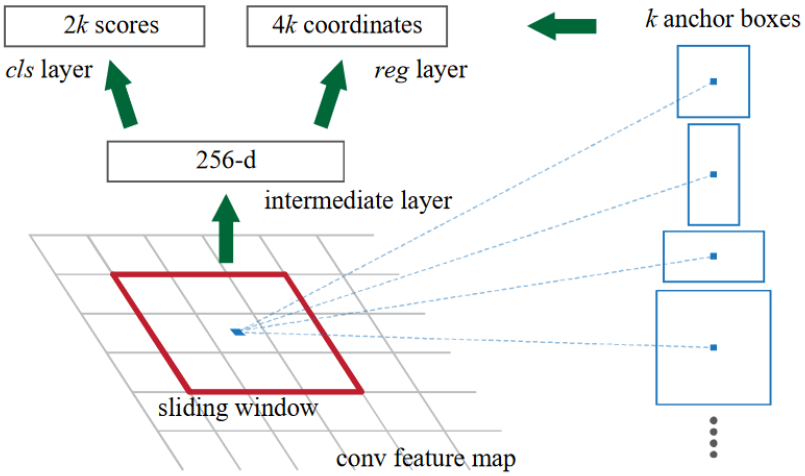

Abbildung 2: Anchor Boxen (https://arxiv.org/pdf/1506.01497.pdf)

Schritt 3: Auswertung der Feature-Vektoren in den Ausgabe-Layern des Detektionsmodells

- Die Ausgabe-Layer bestehen aus zwei Teilen, sog. „*heads*“: Regressions-Head und Klassifikations-Head.
- Der **Klassifikations-Head** ist ein Fully-Connected-Network mit einer Softmax-Aktivierungsfunktion beim letzten Layer. Dessen Ausgabe ist ein Vektor mit Wahrscheinlichkeitswerten für jede Klasse.
- Der **Regressions-Head** ist ein Fully-Connected-Network mit einer linearen Aktivierungsfunktion beim letzten Layer. Dessen Ausgabe ist ein Vektor mit den Koordinaten der Begrenzungsboxen.
- Der Regressions-Head produziert mehrere ***redundante*** Begrenzungsboxen.
  回归头产生了几个冗余的边界盒

Schritt 4: Eliminieren der redundanten Boxen

被认为是假阳性的冗余方框导致了低精度值

- Redundante Boxen, die als False Positives betrachtet werden, führen zu niedrigen Precision-Werten.
- NMS – **Non-Maximum-Suppresssion** wird zum Eliminieren von redundanten Begrenzungsboxen verwendet.
- Beim Training werden positiver (z.B.: IOU > 0,6) und negativer (IOU < 0,4) *Anchor*-Schwellwerte definiert. Die Boxen, deren IOU-Werte zwischen den definierten Schwelwerten liegen, werden verworfen. Negative *Anchor*-Boxen werden als Hintergrund klassifiziert. Positive *Anchor*-Boxen mit höchsten IOU-Werten werden beibehalten. 在训练过程中，定义了正（例如：IOU>0.6）和负（IOU<0.4）的锚点阈值。其IOU值位于定义的阈值之间的盒子被丢弃。
- Bei der Vorhersage werden keine *Ground-Truth*-Boxen vorgegeben. Die redundanten Boxen werden mit Hilfe der vorhergesagten Klassenwahrscheinlichkeiten eliminiert.

在预测中没有指定地面真实箱。使用预测的类别概率来消除多余的盒子。

Durchführung

Überblick

In der Vorbereitungsaufgabe habt ihr den Aufbau von Deep Learning basierten Detektionsmodellen und den Transfer Learning Ansatz kennengelernt. Das Ziel der praktischen Aufgabe ist die Implementierung eines Deep Learning basierten Detektionsmodells zur Detektion von Verkehrszeichen.

Dafür werdet ihr das TensorFlow Object Detection API (Applikation Programming Interface) verwenden.

Das Object Detection API befindet sich im **research**-Ordner des TensorFlow Repository (siehe Abbildung 3).

Hinweis: da die neuen Modelle nicht mit TensorFlow 1.x implementiert werden, bietet sich an, von Anfang an TensorFlow 2.x zu verwenden: Object Detection API Tensorflow 2

Wichtige Informationen und Hinweise zur Benutzung dieses API sind in dem Unterordner **g3doc** hinterlegt. Die Sammlung der vortrainierten Modelle mit den Angaben zur Laufzeit, mAp-Werten und Ausgaben für die einzelnen Modellarchitekturen (detection_model_zoo.md) befindet sich ebenfalls in diesem Unterordner. Zum Training dieser Modelle wurden folgende Datensätze verwendet: der COCO-Datensatz, der KITTI -Datensatz, der Open-Images-

Datensatz, der AVA -v2.1-Datensatz, der iNaturalist-Species-Detection-Datensatz und der Snapshot Serengeti Datensatz.

```
tensorflow
├── models
│   ├── official
│   │   ├── ...
│   └── research
├── ...  ├── object_detection
│        │   ├── data
│        │   ├── models
│        │   ├── samples
│        │   ├── g3doc
│        │   │   ├── detection_model_zoo.md
│        │   │   ├── installation.md
│        │   │   ├── running_locally.md
│        │   │   ├── preparing_inputs.md
│        │   │   └── evaluation_protocols.md
│        │   ├── model_main.py
│        │   └── object_detection_tutorial.ipynb
│        │
│        ├── ...
├── ...
```

Abbildung 3: TensorFlow Object Detection API: Ordnerstruktur mit Auflistung von relevanten Ordnern und Dateien

Das **Object Detection API** hat folgende **Vorteile**:

它允许使用预先训练好的模型进行物体检测，以及构建和训练自己的模型。

- Es ermöglicht die Verwendung von bereits vortrainierten Modellen zur Objektdetektion sowie den Aufbau und das Training von eigenen Modellen.
- Als Detektoren werden Region-Proposal-Systeme wie Faster R-CNN und R-FCN, Context RCNN sowie Single Shot Detector (SSD) benutzt.

区域建议系统，如Faster R-CNN和R-FCN、Context RCNN和Single Shot Detector（SSD）被用作检测器。

- Als Merkmalsextraktoren kommen VGG-16, Resnet-101, Resnet-50, Inception v3, Inception v2, MobileNet zum Einsatz.
- Bedingt durch eine einheitliche Implementierung können Detektoren und Merkmalsextraktoren beliebig kombiniert werden. Dadurch kann die Performance der einzelnen Modelle aufgabenspezifisch gesteuert werden.

由于是统一实现，检测器和特征提取器可以根据需要进行组合。这意味着各个模型的性能可以在特定任务的基础上得到控制。

Aufgabe 1

Schaut euch das Object Detection API Demo, die sich im Unterordner colab_tutorials befindet. Führt dieses Notebook in Google Colab aus.

Aufgabe 2

In dieser Aufgabe solltet ihr die Daten für das Training vorbereiten. Die Daten sollen in TFRecord-Format konvertiert werden. TFRecord-Format ist ein standardisiertes TensorFlow Format zur Datenserialisierung. Unter

数据序列化是以二进制字符串的形式存储数据

Datenserialisierung wird das Speichern von Daten in Form von binären Zeichenfolgen verstanden. Zum Konvertieren der Daten ins TFRecord-Format ist eine spezifische Datenstruktur mit Informationen über den relativen Bildpfad, die Klasse und die x- und y-Koordinaten der Begrenzungsboxen erforderlich. Eine Anleitung zum Vorbereiten der Daten findet ihr in den Dateien using_your_own_dataset und preparing_inputs.

Bevor ihr TFRecords generiert, sollt ihr eine label_map.pbtxt-Datei anlegen und in dem **data**-Ordner:

$$tensorflow/models/research/object\_detection/data$$

speichern. Wie so eine pbtxt-Datei aufgebaut ist, könnt ihr hier schauen.

Aufgabe 3

In dieser Aufgabe sollt ihr ein vortrainiertes Modell auswählen und seine Konfigurationsdatei konfigurieren. Zur Auswahl eines passenden vortrainierten Modells können die Modelllaufzeit und mAp-Parameter der zur Verfügung gestellten vortrainierten Modelle aus **detection_model_zoo** herangezogen werden.

Die Konfigurationsdateien sind im configs-Ordner abgelegt:

$$tensorflow/models/research/object\_detection/samples/configs$$

Konfigurationsdateien enthalten fünf wesentliche Komponenten: *model, train_config, eval_config, train_input_config* und *eval_input_config*. Grundsätzlich könnt ihr alle Parameter konfigurieren und der Aufgabenstellung entsprechend anpassen. Für folgende Parameter sind Anpassungen erforderlich:

- num_classes
- num_steps
- input_path
- label_map_path 标签映射文件的路径
- num_examples
- fine_tune_checkpoint

Aufgabe 4

Nachdem die config-Datei konfiguriert wurde, sollt ihr die Trainingspipeline konfigurieren. Hinweise dafür findet ihr hier.

Zur Überwachung des Trainings könnt ihr das Visualisierungstool TensorBoard nutzen.

Training and evaluation:
https://github.com/tensorflow/models/blob/master/research/object_detection/g3doc/tf2_training_and_evaluation.md

Verwendet für diese Aufgabe die Daten der German Traffic Sign Detection Benchmark, die ihr bereits heruntergeladen habt.

Zusatzinformationen zu FLAGS https://medium.com/@zzzuzum/command-line-flags-python-and-tensorflow-85ab217dbd5