SEN P6: Transfer Learning

Vorbemerkungen: In diesem Praktikumsversuch soll mittels Transfer Learning ein auf den ImageNet-Datensatz vortrainiertes CNN auf eine neue Klassifikationsaufgabe umtrainiert werden. Bei der Klassifikationsaufgabe handelt es sich um die Klassifikation von Blumenarten auf Bildern. Als vortrainiertes Modell kommt das MobileNetV2 Modell[1] zum Einsatz, das sich durch eine verhältnismäßig geringe Anzahl an Parametern auszeichnet und sich daher auch für den Einsatz auf mobilen Endgeräten eignet.

Vorbereitung: Machen Sie sich mit dem Thema Tranfer Learning (für CNNs) vertraut. Als Ausgangspunkt für die Codeentwicklung können Sie sich am Jupyter Notebook Transfer_Learning_Demo.ipynb orientieren. Machen Sie sich außerdem mit dem Umgang mit Jupyter Notebooks vertraut. Hier kann das Dokument Jupyter_Notebook_Kurzanleitung.pdf als Hilfestellung leisten. Außerdem wird empfohlen, im Rahmen der Vorbereitung die Übungsaufgabe 13 zu machen, die sich der Entwicklung eines tiefen neuronalen Netzes widmet.

Anleitung

- Erstellen Sie ein neues Jupyter Notebook.
- Der zu verwendende Datensatz ist unter folgender URL zu finden:
  https://storage.googleapis.com/download.tensorflow.org/example_images/flower_photos.tgz
  Die Datei mit den Bildern ist per tar-Verfahren komprimiert und muss daher nach dem Download zunächst noch entpackt werden. Download und Entpacken kann mit folgender Codezeile bewerkstelligt werden:
  `PATH = tf.keras.utils.get_file('flower_photos', origin=_URL, untar=True)`
  wobei angenommen wird, dass in der Variable `_URL` die oben angegebene URL des Datenbestandes als String abgelegt ist.
- Finden Sie die heruntergeladenen und entpackten Daten im Dateisystem Ihres Rechners und schauen Sie sich ein paar der Bilder an.
- Laden Sie nun den gesamten Datenbestand von der Festplatte Ihres Rechners in das Jupyter Notebook. Laden Sie die Daten direkt in Batches der Größe 32 und sorgen Sie dafür, dass die Bilder auf die für von MobileNetV2 erwartete Größe skaliert werden.
- Ermitteln Sie die im Datensatz verwendeten Klassennamen.
- Machen Sie sich mit dem Datenbestand vertraut. Ermitteln Sie bspw. die Dimensionen (Shape) der Daten und lassen Sie ein paar Bilder und deren Klassen im Jupyter Notebook darstellen.

[1] https://arxiv.org/abs/1801.04381

- Splitten Sie nun die Daten in Trainings-, Validierungs- und Testdaten. Entnehmen Sie hierfür zunächst 20 % des Gesamtdatensatzes und machen Sie hieraus den Validierungsdatensatz. Entnehmen Sie aus den übrigen 80 % des Gesamtdatenbestandes 10 % als Testdatensatz. Der Rest soll als Trainingsdatensatz verwendet werden.
- Erstellen Sie nun einen Preprocessing-Layer zur Vorverarbeitung der Daten, indem Sie den `preprocess_input`-Layer des MobileNetV2 Modells verwenden.
- Instanziieren Sie nun das mit den ImageNet-Daten vortrainierte MobileNetV2 Modell. Achten Sie darauf, dass die Abmessungen (Shape) der Eingangsdaten korrekt (also auch mit Angabe der Anzahl der Bildkanäle) spezifiziert wird. Sorgen Sie außerdem dafür, dass als Basismodell nur der Feature Extractor von MobileNetV2 verwendet wird und die Classification Layer verworfen werden.
- Lassen Sie sich mittels der `summary()`-Methode einige Infos zu den einzelnen Layern des Basismodells ausgeben. Wieviele trainierbare Parameter hat das Basismodell? Vergleichen Sie die Parameterzahl mit dem Basismodell von ResNet-50 (siehe Transfer_Learning_Demo.ipynb).
- Ermitteln Sie, welche Dimension die Daten haben, die aus dem Feature Extractor kommen. Hierzu können Sie das Basismodell einfach auf einen Batch des Trainingsdatenbesandes anwenden und den Shape der Ausgangssignale ermitteln.
- Frieren Sie nun das komplette Basismodell ein, so dass dieses nicht trainierbar ist.
- Verifizieren Sie mittels `summary()`-Methode, dass nun alle Parameter „non-trainable“ sind.
- Erstellen Sie nun den Classifier Ihres CNN. Dieser soll aus einem GlobalAveragePooling2D Layer und einem Dense Layer mit geeigneter Aktivierungsfunktion bestehen. Beachten Sie bei der Erstellung des Dense Layers, dass es sich hier (anders als in Transfer_Learning_Demo.ipynb) nicht um ein binäres Klassifikationsproblem handelt.
- Erstellen Sie nun das Gesamtmodell, indem Sie mittels Keras Functional API einen Input Layer, den Preprocessing Layer, das (eingefrorene) Basismodell, den Pooling Layer, und den Dense Layer zusammenschalten. Sehen Sie zur Generalisierung zusätzlich noch einen Dropout Layer mit 30 % Dropout Rate zwischen Pooling und Dense Layer vor und achten Sie darauf, dass der Parameter `training=False` im Basismodell gesetzt wurde.
- Kompilieren Sie nun das Modell. Beachten Sie bei der Wahl des Loss darauf, dass es sich hier um ein Multiclass-Problem (und nicht um ein binäres Klassifikationsproblem) handelt.
- Trainieren Sie Ihr Modell nun über einige wenige (Vorschlag: 5) Epochen. Sie müssen nicht auf das Trainingsergebnis warten, sondern können in der Zwischenzeit schon mal mit der Codeentwicklung weitermachen.

Anmerkung: Wie Sie feststellen werden, ist das Modell nach so wenigen Epochen noch nicht vollständig konvergiert. Um die wertvolle Zeit im Praktikum aber nicht mit Trainingszeit zu verschwenden, wird hier empfohlen, das Modell zunächst nur über wenige Epochen zu trainieren und den Fokus darauf zu legen, den Code zu entwickeln. Ein umfangreicheres Training (über mehr Epochen) kann dann im Nachgang des Praktikums stattfinden.

- Lassen Sie die Lernkurve ausgeben und begutachten Sie diese.
- Führen Sie nun ein Fine Tuning des Modells ab Layer 135 durch. Wählen Sie hierfür die Lernrate deutlich kleiner (Vorschlag: Faktor 10) als im ersten Trainingschritt. Halten Sie die Anzahl der Traningsepochen erneut klein, um die Trainingszeit im Praktikum möglichst kurz zu halten.
- Schauen Sie sich auch die Lernkurve des Fine Tuning an.
- Untersuchen Sie die Performance Ihres Modells indem Sie Loss und Accuray auf den Testdaten ermitteln.
- Wenden Sie das Modell außerdem auf ein paar exemplarische Bilder an und überprüfen Sie die Ergebnisse.
  Hinweis: Da es sich um ein Multiclass-Problem handelt, liefert das Modell pro Bild einen Vektor mit so vielen Werten, wie es Klassen im Datenbestand gibt. Anders ausgedrückt: Für jedes Bild liefert das Modell eine Wahrscheinlichkeit für jede Klasse. Die tatsächliche Klassifikationsentscheidung (also die laut Modell wahrscheinlichste Klasse) kann aus den vom Modell ausgegebenen Wahrscheinlichkeitsvektoren (`prediction_probas`) mit der Argmax-Funktion (`np.argmax(prediction_probas, axis=1)`) ermittelt werden.