Nutzerhandbuch: Code Siegel NDH

Katharina Prasse

December 2021

1 Preprocessing

1.1 Code: increase_contrast.py

Dieser Code lädt die Input-Bilder und speichert sie als neue Datei mit erhöhtem Kontrast ab.

Kategorie	**Details**	**Kommentare**
Notwendige Nutzerinput	Befüllen Sie "image_folder" mit dem Pfad des Ordners, in dem die zu optimierenden Bilder abgespeichert sind. Befüllen Sie "out_path" mit dem Orderpfad des gewünschten Speicherorts der optimierten Bilder.	Der "out_path" muss schon existieren, er kann nicht durch den Code erzeugt werden.
Optionale Nutzerinputs	Die Variable "clip_limit" in der Funktion "save_imgs" kann verändert werden. Der Wert von "clip_limit" muss sich zwischen Null und Eins befinden. Je höher der eingesetzte Wert ist, desto stärker ist die Normalisierung.	Weitere Details können auf Wikipedia eingesehen werden (`https://en.wikipedia.org/wiki/Adaptive_histogram_equalization#CLAHE`.
Laufzeit	Der Algorithmus bearbeitet ein Bild nach dem anderen.	

1.2 Code: circular_hough_transform.py

Dieser Code lädt die Input-Bilder und schneidet die Siegel mit einer kreisförmigen Hough-Transformation aus (vgl. `https://de.wikipedia.org/wiki/Hough-Transformation#Hough-Transformation_f%C3%BCr_Kreise_und_generalisierte_Hough-Transformation`). Um die Qualität der Siegelerkennung zu steigern wird die Auflösung der Bilder stark reduziert bevor die Kanten des Bildes extrahiert werden. Nach der Hough-Transformationen werden die Siegelbilder in verschiedenen Varianten neu gespeichert: In hoher Auflösung, in niedriger Auflösung (299 x 299 Pixel), in niedriger Auflösung in graustufen, in niedriges Auflösung in graustufen mit erhöhtem Kontrast und ausschließlich die Kanten des Bildes in niedriger Auflösung. Dieser Code ist eine Erweiterung von "increase_contrast.py".

Kategorie	**Details**	**Kommentare**
Notwendige Nutzerinput	Befüllen Sie "image_folder" mit dem Pfad des Ordners, in dem die zu optimierenden Bilder abgespeichert sind. Befüllen Sie "out_path" mit dem Orderpfad des gewünschten Speicherorts der optimierten Bilder.	Der "out_path" muss schon existieren, er kann nicht durch den Code erzeugt werden.
Optionale Nutzerinputs	Die Variable "resize_factor" sollte abhängig von der Bildgröße gesetzt werden. Idealerweise umfassen alle Seiten des Photo nach der Reduzierung 60 Pixeln oder weniger. Die Extraktion der Bildkanten verwendet den Canny-Algorithmus (vgl. `https://de.wikipedia.org/wiki/Canny-Algorithmus`) gesteuert durch die Variablen "sigma", "low_threshold", "high_threshold". Die Hough-Transformation sucht in der Photographie nach Kreisen, deren Radii zwischen "min_radius" und "max_radius" liegen (nur ganze Zahlen werden berücksichtigt). Diese Werte sollen abhängig von der Siegelgröße angepasst werden. Die Anzahl der extrahierten Kreise wird gesetzt mit der Variable "total_num_peaks". Sie ist auf 5 gesetzt um Kreise aussortieren zu können, die offensichtlich kein Siegel enthalten, d.h. alle Kreise, die zu nah an einem Rand der Photographie sind. Der Abstand, der aussortiert wird, wird mit der Variable "threshold" bestimmt.	Die genannten Variablen sollte angepasst werden, wenn die Bildgröße, der Zustand der Siegel, oder die Art der Siegel (z.B. Größe, Form) von den verwendeten Siegeln abweichen. Falls in dem Siegelphoto keine anderen Kreise vorhanden sind, können die Variablen aus Effizienzgründen angepasst werden (optional).
Laufzeit	Der Algorithmus bearbeitet ein Bild nach dem anderen.	Für 8T Bilder betrug die Laufzeit 24 Stunden.

2 Feature_Generation&Extraction

2.1 Code: finetune_feat_extract.py

Dieser Code trainiert ein neuronales Netz anhand von annotierten Bildern. Die, in diesem Projekt vorhandene, geringe Anzahl an Trainingsbildern wird ausgeglichen durch eine Vielzahl an Transformationen (zufällige Rotation, Ausschneiden der Bildmitte in zufälliger Größe). Alle von Torchvision bereitgestellten, vortrainierten Modelle können verwendet werden. Die optinalen Nutzerinputs werden notwendig, wenn nicht die gleichen Einstellungen und das gleiche Training durchgeführt werden wie im Code beschreiben (z.B. Anzahl Epochen, Anzahl Trainingsklassen).

Kategorie	Details	Kommentare
Notwendige Nutzerinput	Befüllen Sie "data_dir" mit dem Pfad des Ordners, in dem die zu optimierenden Bilder abgespeichert sind. Befüllen Sie "out_path" mit dem Orderpfad des gewünschten Speicherorts der optimierten Bilder. Die Variable "outname" beschreibt das zu trainierende Modell eindeutig.	Der "out_path" muss schon existieren, er kann nicht durch den Code erzeugt werden.
Optionale Nutzerinputs	Die Variable "batch_size" sollte abhängig von der Verfügbarkeit von Trainingsdaten und RAM gesetzt werden. Falls verfügbar, sollte eine GPU verwendet werden, dies kann in der Variable "device" spezifiziert werden, andernfalls "cpu". Die Variable "num_workers" entspricht der Anzahl parallel in die RAM geladener Batches und sollte immer größer eins sein aus Effizienzgründen. Die Variable "model_name" bestimmt das verwendete Model, wobei ausgewählt werden kann, ob die vortrainierten oder zufällige Gewichte gelanden werden sollten "use_pretrained=T/F". Die "num_classes" Variable muss der Anzahl an Trainingsklassen entsprechen. Im Detail heißt das, die Anzahl Unterordner im "data_dir" die bestimmte Siegel enthalten. Pro Klasse sollten mindestens 50 Bilder verwendet werden und je mehr desto besser. Die Variabel "feat_extract" bestimmt welche Parameter der neuronalen Netzes traininert werden sollten. Bei "True" werden nur die Klassifikationslayers trainiert, bei "False" das gesamte Netz. "Num_epochs" definiert die Anzahl Epoche, die trainiert wird d.h. wie oft alle Trainingsbilder verwendet werden.	Die Variable "Num_epochs" sollte angepasst werden anhand der ausgegebenen Accurary (Genauigkeit) während des Trainings. Spätestens wenn die Accuracy konstant dem Wert eins entspricht, sollte das Training beendet werden. Die Variable "device" sollte an den jeweiligen Computer angepasst werden.
Laufzeit	Der Algorithmus bearbeitet die Photographien in Gruppen.	Für 200 Trainingsbilder dauert das Training ungefähr fünf bis zehn Minuten.

2.2 Code: passive_feat_extract_finetuned.py

Dieser Code extrahiert die Merkmale der Siegelbilder als Vorbereitung für die Clusteranalyse. Das verwendete Modell wird durch den Nutzer bereitgestellt z.B. das Modell dieses Projektarbeit bzw. ein selbsttrainiertes Modell.

Kategorie	**Details**	**Kommentare**
Notwendige Nutzerinput	Befüllen Sie "data_dir" mit dem Pfad des Ordners, in dem die zu optimierenden Bilder abgespeichert sind. Befüllen Sie "out_path" mit dem Orderpfad des gewünschten Speicherorts der optimierten Bilder. Die Variable "outname" sollte das trainierte Modell eindeutig beschreiben "Model_path" definiert den Speicherordner des Modelles. Unter "model_name" muss der Name des verwendeten Modells angegeben werden um die korrekte Modifikation des Classification headers sicherzustellen.	Der "out_path" muss schon existieren, er kann nicht durch den Code erzeugt werden.
Optionale Nutzerinputs	Die Variable "batch_size" sollte abhängig von RAM gesetzt werden, je höher desto besser. Falls verfügbar, sollte eine GPU verwendet werden, dies kann in der Variable "device" spezifiziert werden. Falls keine GPU vorhanden sein sollte, tragen Sie bitte "cpu" ein um das Training auf der CPU durchzuführen. Die Variable "num_workers" entspricht der Anzahl parallel in die RAM geladener Batches und sollte immer größer eins sein aus Effizienzgründen.	-
Laufzeit	Der Algorithmus bearbeitet ung. 8.000 Photographien in Gruppen von 200 Bildern innerhalb weniger Minuten.	-

2.3 Code: passive_feat_extract_pretrained.py

Dieser Code ist eine Alternative zu passive_feat_extract_finetuned.py, in der das vortrainierte Model direkt von torchvision geladen wird und nicht nachtrainiert wird. Dies kann auch als Maßstab verwendet werden, um die Effektivität der Trainings beurteilen zu können.

Kategorie	**Details**	**Kommentare**
Notwendige Nutzerinput	Befüllen Sie "data_dir" mit dem Pfad des Ordners, in dem die zu optimierenden Bilder abgespeichert sind. Befüllen Sie "out_path" mit dem Orderpfad des gewünschten Speicherorts der optimierten Bilder. Die Variable "outname" sollte das trainierte Modell eindeutig beschreiben	Der "out_path" muss schon existieren, er kann nicht durch den Code erzeugt werden.
Optionale Nutzerinputs	Die Variable "batch_size" sollte abhängig von RAM gesetzt werden, je höher desto besser. Falls verfügbar, sollte eine GPU verwendet werden, dies kann in der Variable "device" spezifiziert werden. Falls keine GPU vorhanden sein sollte, tragen Sie bitte "cpu" ein um das Training auf der CPU durchzuführen. Die Variable "num_workers" entspricht der Anzahl parallel in die RAM geladener Batches und sollte immer größer null sein aus Effizienzgründen. Die Variable "model_name" bestimmt welches Modell geladen werden soll, wobei die Variable "use_pretrained" angibt, ob die Geiwchte geladen oder zufällig gesetzt werden sollen.	-
Laufzeit	Der Algorithmus bearbeitet ung. 8.000 Photographien in Gruppen von 200 Bildern innerhalb einer Minute.	-

3 Clusteranalyse

3.1 Code: KMEANS

Dieser Code verwendet die extrahierten Merkmale und clustert sie in die angegebene Anzahl (K) Cluster. Optional kann die Dimension der Merkmale reduziert werden mit Hilfe einer Hauptkomponentenanalyse `https://en.wikipedia.org/wiki/Principal_component_analysis`.

Kategorie	**Details**	**Kommentare**
Notwendige Nutzerinput	Befüllen Sie "feature_path" mit dem Pfad des Ordners, in welchem die extrahierten Merkmale abgespeichert sind.	-
Optionale Nutzerinputs	Die Variable "num_clust" bestimmt die Anzahl an Clustern, d.h. Gruppen, in welche die Bilder gruppiert werden sollen. Die Variable "pca_num_components" bestimmt auf wie viele Dimensionen die Merkmale durch eine Hauptkomponentenanalyse reduziert werden.	In der jetztigen Projektarbeit haben Werte von K zwischen sieben und 15 sinnvolle Cluster produziert.
Hauptkomponentenanalyse	Mithilfe des Scree-Plots kann die optimale Anzahl an Komponenten ermittelt werden.	Vergleiche `https://en.wikipedia.org/wiki/Scree_plot`
Hintergrund	Der Algorithmus versucht die Photographien möglichst gleichmäßig auf die Cluster zu verteilen.	-

4 Clusteranalyse

4.1 Code: DBSCAN

Alternative zu KMEANS. Dieser Code verwendet die extrahierten Merkmale und clustert sie anhand eines Modelles das die Dichte der Merkmale zueinander misst. Optional kann die Dimension der Merkmale reduziert werden mit Hilfe von einer Hauptkomponentenanalyse `https://en.wikipedia.org/wiki/Principal_component_analysis`. Während dieser Projektarbeit konnten bessere Cluster durch den KMEANS ALgorithmus erzeugt werden.

Kategorie	**Details**	**Kommentare**
Notwendige Nutzerinput	Befüllen Sie "feature_path" mit dem Pfad des Ordners, in welchem die extrahierten Merkmale abgespeichert sind.	-
Optionale Nutzerinputs	Die Variablen "dbscan_eps" und "dbscan_ms" bestimmen die notwendige Dichte und die Mindestanzahl Datenpunkte, die notwendig sind, um dem gleichen Cluster zugeordnet zu werden. Die Variable "pca_num_components" bestimmt auf wie viele Dimensionen die Merkmale durch eine Hauptkomponentenanalyse reduziert werden.	In der jetztigen Projektarbeit wurde "dbscan_eps" auf Werte zwischen 0.027 und 0.5 gesetzt. Der Wert kann kleiner gewählt werden, je länger gefinetuned wurde. Für "dbscan_ms" wurden Werte zwischen 2 und 10 verwendet.
Hauptkomponentenanalyse	Mithilfe des Scree-Plots kann die optimale Anzahl an Komponenten ermittelt werden.	Vergleiche `https://en.wikipedia.org/wiki/Scree_plot`