UNIVERSITÄT ZU LÜBECK - INSTITUT FÜR MEDIZINISCHE INFORMATIK

Report - Medical Information Retrieval

Jesse Kruse - 675710
Konrad von Kügelgen - 676609
Falco Lentzsch - 685454

December 6, 2019

1 Feature Extractor

In dieser Übung ging es zunächst darum, die OpenCV Bibliothek zu installieren und erste Erfahrungen damit zu machen. CV steht für *Computer Vision*. Es handelt sich dabei um eine Bibliothek um verschiedenste Operationen auf und mit Bildern durch zu führen. Als erstes haben wir daraufhin den Umgang mit Bildern als Objekten in Python gelernt - wie lädt man ein Bild und wie kann es im folgenden weiter verarbeitet werden. Unsere erste Aufgabe war es ein Bild unserer Wahl mit dem Befehl *cv2.imread()* zu laden, in einer Variable zu speichern, und dieses mittels *cv2.imshow()* anzuzeigen. Danach begann die eigentliche Aufgabe zu diesem ersten Thema. Dazu haben wir uns einen Test Datensatz von 1000 Bildern geladen, aus denen wir im weiteren Verlauf Features extrahieren wollen. Dieser Testdatensatz beinhaltete bereits klassifizierte Datenpunkte, was im späteren Verlauf noch wichtig ist. Zunächst interessierten uns allerdings nur die aus dem Bild auszulesenden Daten. Zur Verfügung stand uns wie bei allen Aufgaben ein grobes Programmgerüst mit Methoden, die wir zu implementierten. Dabei konnten wir am Ende frei wählen, welche Features wir zu unseren Bildern haben und abspeichern wollen. Diese werden im Folgenden kurz erklärt.
histogram: Dabei wird ein Bild genommen und aus den Grauwerten mit der Funktion *cv2.calcHist()* ein Histogramm erstellt. Die Daten die uns die Funktion liefert, sollten danach noch durch *flatten* in eine gemeinsame, ununterbrochene Liste geschrieben werden. Wir haben dies allerdings mit einer List Comprehension gelöst.
thumbnail_feautures: Hierbei wird ein Bild mit der Funktion *cv2.resize()* auf eine beliebige Dimension skaliert. Uns wurde hier 30x30 vorgeben. Auch hier sollten die Daten in eine Liste gebracht werden, bevor sie returned werden.
spatial feature: Auch bei diesem Feauture haben wir zunächst eine Veränderung der Dimension vorgenommen, hier 200x200. Danach haben wir eine Art Fenster der Größe 10x10 über das neue Bild geschoben und unser Bild so in Partitionen aufgeteilt. Für jeden dieser Bereiche haben dann den maximalen, minimalen und durchschnittlichen Grauwert bestimmt. Diese drei Werte haben wir in einer Liste gespeichert und in unsere Liste eingefügt *(via .extend)*, die das Feature für das ganze Bild wieder spiegelt. Unsere Rückgabe hier ist also eine Liste, bei der jeder Eintrag eine Liste ist, die für einen Bereich des Bildes Mittelwert, Maximum und Minimum beinhaltet.
partitionbased_histograms: Der Ablauf hier ist sehr ähnlich zu dem vorherigen Feature.
Zunächst wurde das Bild auf die Dimension 100x100 skaliert. Danach haben wir wie zuvor eine Art Fenster der Größe 4x4 über das geschoben. Für die einzelnen Bereiche wurden die Histogramm-Werte berechnet und diese in einer Liste kombiniert, wie bei dem zuerst genannten Feature. Zurück gegeben haben wir hier eine Liste von Listen, wo bei jede Teilliste das Histogramm zu einem Bildausschnitt beinhaltet.
All diese Feautures haben wir im folgenden für jedes Bild extrahiert und gespeichert.

2 Preprocessing

In dieser Übungseinheit ging es darum, die zuvor erstellte Klasse *Feature Extractor* zu nutzen, um für jedes Bild Features zu erstellen und diese zu speichern. Dazu haben wir 3 Methoden geschrieben.
get_image_paths: Diese Funktion bekommt einen Ordnerpfad übergeben, in dem sich die Bilder befinden, für die wir unsere Feautures extrahieren wollen und eine Liste der Dateien, die wir betrachten wollen. Für jede Datei Endung schauen wir uns nun alle Dateien im Ordnerpfad an und und fügen sie einer Liste hinzu. Diese Liste von Ordnerpfaden wird anschließend returned.
create_feature_list: Diese Funktion bekommt eine Liste von Bildpfaden. Für jedes Bild, was

sich hinter einem Bildpfad verbirgt, haben wir nun mit dem zuvor programmierten *Feature Extractor* eine Feature Liste erstellt und diese Liste einer weiteren Liste angefügt. Die Funktion liefert uns also eine Liste von Listen, wobei jede Teilliste den Pfad des Bildes und die Daten der extrahierten Features beinhaltet.
write_to_file: Diese Methode war nun dafür zuständig, die zuvor errechnetet und gewonnenen Features zu exportieren. Dazu sollten wir eine csv-Datei erstellen, die in jeder Zeile zunächst den Bildpfad enthielt und anschließend, jeweils mit einem Komma getrennt, die einzelnen Werte aus den zugehörigen Feature-Liste auflistete - das *Indexfile*. Zwar würde man beim anschauen der csv-Datei nicht mehr auf die Herkunft der Werte, ob nun *histogram* oder *thumbnail feature* schließen können, jedoch sind die Werte vergleichbar, da sie bei jedem einzelnen Bild in derselben Reihenfolge aufgelistet werden. Dazu testen wir zunächst ob die Anzahl der Bildpfade identisch mit der Anzahl an Sublisten in der feature list ist. Dann wird eine csv-Datei im *write*-Modus geöffnet und in jede Zeile nun der Bildpfad sowie die zugehörigen Werte geschrieben.

3 Searcher

Mit der Arbeit an der *Searcher-* und *Query-Klasse* begann die eigentliche die Arbeit an der Information-Retrieval Phase. In der *Searcher-Klasse* wird dabei der Suchalgorithmus implementiert. Dazu wurde das zuvor generierte Indexfile aus dem Preprocessing geladen und anschließend die extrahierten Features mit den übergebenen query-features verglichen. Dabei wurde zunächst die Euklidische Distanz zwischen den beiden Merkmalsvektoren berechnet und diese in einer Liste gespeichert. Diese Liste wurde im Anschluss noch absteigend sortiert und als Ergebnis an die aufrufende Stelle returned. Neben der Euklidischen Distanz wurden im *Searcher* noch verschiedene andere Normen zur Abstandsberechnung implementiert. zum einen wäre da die Manhatten-Distanz zu nennen. Diese bildet nur die Summe der Beträge der Differenzen, ohne die Wurzel zu ziehen:

$$M(X,Y) = \sum_{i=1}^{n} |x_i - y_i|$$

Außerdem wurde noch die Minkowski-Metrik implementiert, welche die Verallgemeinerung der der Euklidischen- und der Manhatten-Distanz in Abhängigkeit von p darstellt:

$$D(X,Y) = (\sum_{i=1}^{n} |x_i - y_i|)^{1/p}$$

Neben diesen Metriken wurde auch noch die Cosinus-Distanz umgesetzt. Hierbei wird der Winkel zwischen zwei Vektoren vom Ursprung im Raum aus gemessen.

4 Query

Im Anschluss ging die Arbeit an der *Query-Klasse* los. Diese übernimmt verschiedene Funktionen. Zum einen nimmt sie von dem Benutzer ein Bild als Eingabe entgegen und extrahiert daraus mithilfe des *Feature-Extractors* die spezifischen Merkmale des Anfragebildes. Dann wird der *Searcher* aufgerufen(*run()-Methode*), um möglichst ähnliche Bilder aus der Datenbank zurückzugeben. Daneben wird in der Methode *check-code* zum ersten mal von den Labels des Datensatzes Gebrauch gemacht. Hier wird ermittelt, ob die erhaltenen Datensätze auch tatsächlich zu dem Angefragten passen. Dazu werden die Codes der zurückgegebenen mit dem des Querybildes verglichen. Außerdem bietet *Query* eine Visualisierung der ausgewählten

Bilder in der entsprechenden Reihenfolge, indem diese miteinander konkateniert werden und mithilfe von *cv* angezeigt. Optional war es noch möglich mithilfe der *check-code*-Methode die erhaltenen Bilder, je nachdem ob es sich um einen Treffer oder nicht handelt, mit einen grünen oder einem roten Rahmen zu versehen. In unserer Gruppe traten in diesem Abschnitt vermehrt Bugs und andere Schwierigkeiten auf. Daher tauschten wir hier nach und nach Teile durch die zur Verfügung gestellten Musterlösungen aus. Da wir in den folgenden Projekten größere Anteile am Code selbst entwerfen werden sind wir uns bewusst, dass das in Zukunft tunlichst vermieden werden sollte.

5 Evaluation

Im letzten Teil des ersten Szenarios wurde noch eine Klasse zur Evaluation des Programms erstellt. Hierbei werden verschiedene Werte errechnet um die Performance eines solchen Dataretreivalsystems zu messen. Zunächst die *precision_at_k*. Diese gibt an wie viele der *k* zurück gegebenen Elemente in einer Anfrage relevant waren. Auch hier kommt das mit der ceck_code()-Methode erstellte Dictionary *correct_prdiction_dictionary* zum Einsatz. Um eine fundiertere Aussage über das System treffen zu können benötigen wir im weiteren Verlauf die *average_precision*. Diese errechnet sich durch die Summe der Anteile der Relevanten gegenüber allen zurückgegeben Bildern geteilt durch alle Relevanten. Diese können wir nun über möglichst viele Queries hinweg errechnen und aufsummieren, um sie dann durch die Anzahl der vorgenommenen Queries zu Teilen. Dies liefert das *mean_average_precision*. Diese Maßzahl bietet zwar nur wenig Wert, um verschiedene Systeme zu vergleichen, jedoch lässt sich innerhalb des Systems eine Aussage darüber treffen, welche Features oder Featurekombinationen die besten Ergebnisse liefern.