Przetwarzanie tekstu ze zdjęcia na tekst edytowalny

Paweł Bielecki

Viacheslav Bylo

Volodymyr Tymkiv

Repozytorium z projektem: https://github.com/pawel2000pl/AO_Project/
W repozytorium znajduje się film z zaprezentowanym działaniem programu.

Założenia projektu

Celem projektu było utworzenie programu opartego na sztucznej sieci neuronowej zdolnej przetworzyć tekst drukowany ze zdjęcia na tekst edytowalny na komputerze. Program podczas analizy miał odfiltrowywać ze zdjęcia ewentualne szumy i nierówne oświetlenie obrazu. Całość została zaprogramowana przy pomocy pakietu Matlab (obróbka zdjęć), oraz w Pyton (sztuczna inteligencja).

Podział pracy

Paweł Bielecki - rozdzielanie tekstu ze zdjęcia na zdjęcia pojedynczych liter.
Viacheslav Bylo - napisanie oraz wytrenowanie modelu dla sieci neuronowej oraz napisanie serwera korzystającego z modelu.
Volodymyr Tymkiv - preprocessing danych dla modelu treningowego.

Działania programu

Rozdzielanie liter

Jak to działa

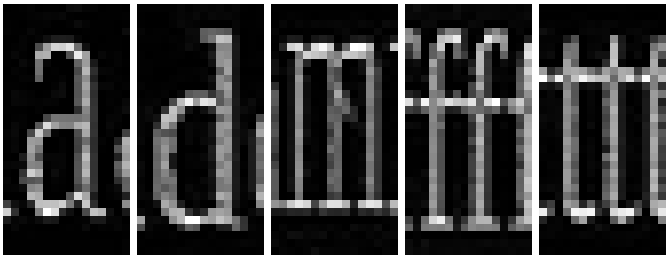
Pierwszym etapem działania programu zaraz po załadowaniu zdjęcia jest rozdzielenie całego zdjęcia w pojedyncze litery. Jako pierwszy krok następuje filtrowanie obrazu filtrem wyostrzającym. Kolejny krokiem jest binaryzacja z opcją *adaptive*, negatyw obrazu, oraz przefiltrowanie obrazu różnymi funkcjami mającymi na celu zniwelowanie dziur w obrazie i usunięcie szumów.
Tworzona jest kopia obrazu przefiltrowana przy pomocy funkcji *imdilate* z użyciem maski złożonej z zer i jedynek w środkowym wierszu, oraz przy pomocy . Rozmiar takiej maski to

15x15. Sprawia to, że linie tekstu stają się zamazane poziomo - stają się paskami. Paski te następnie są numerowane przy pomocy funkcji *bwlabel*. Każdy z nich z osobna jest używany jako kolejna maska do wyłuskania kolejnej linii z obrazu. Powstaje w ten sposób tablica zawierająca kolejne linie tekstu.
Następnym krokiem jest rozdzielanie liter z każdej z tych linii. Następuje to w podobny sposób, co rozdzielanie linii, jednak większość operacji jest pionowa, a nie pozioma.

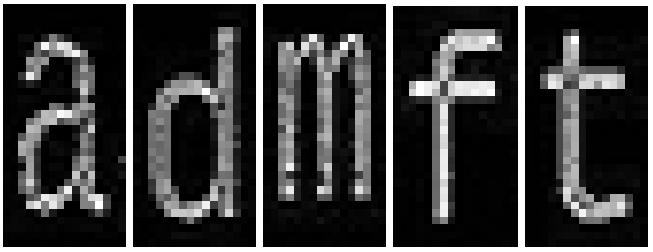
Działanie praktyce

Algorytm jest w stanie rozpoznać większość liter poprawnie. W zdjęciach testowych są szumy, oraz te zdjęcia nie są najwyższej jakości. Mimo to dla czcionki *Liberation Mono* litery są rozpoznawalne niemal idealnie. Niestety dla czcionek szeryfowych (*Liberation Serif*), które są miejscami cieńsze zdarza się łączyć sąsiednie litery szeryfami, lub jak w przypadku litery *t* złącza je kreskami poziomymi z powodu niewielkiej odległości między pikselami. Innym błędem jest rozdzielanie liter w połowie, tak jak zdarza się to w przypadku litery *m*.

Poniżej przykłady przy użyciu czcionki *Liberation Serif*, kolejno od lewej:
poprawnie rozpoznane litery: *a d m*, oraz niepoprawnie złączone litery: *f t.*



Poniżej przykłady przy użyciu czcionki *Liberation Mono*, kolejno od lewej:
poprawnie rozpoznane litery: *a d m f t.*



Teoretycznym rozwiązaniem tego problemu mogłoby być zwiększenie rozdzielczości zeskanowanego dokumentu.

Innym efektem niepożądanym, jest rozpoznawanie wyrazów jako osobne linie tekstu. Dzieje się tak kiedy odległość między wyrazami na zdjęciu jest zbyt duża. Jednak wpływa to jedynie na rozłożenie tekstu, a nie na jego zawartość, więc nie stanowi dużego problemu. Rozwiązaniem jest zmniejszenie rozmiarów obrazu.

Identyfikacja liter

Ewaluacja danych

Dalej zajmowaliśmy się ewaluacją danych dla modelu treningowego.

Napisana funkcja load nadaje obrazom wartość ascii dla danej litery.

```
def load(filepath: str):
    filepath: Path = Path(filepath)

    name, _ = filepath.name.split('.', 1)
    code, _ = name.split('x')
    code = int(code)

    if code <= 26:
        code = int(code) + ord('A') - 1
    elif code >= 79:
        code = int(code) + ord('A') - 79
    elif code >= 27 and code <= 52:
        code = int(code) + ord('a') - 27
    elif code >= 53 and code <= 78:
        code = int(code) + ord('a') - 53

    return Image(cv2.imread(str(filepath)), code)
```

Napisanie modelu treningowego

Następną częścią było napisanie modelu dla wytrenowania sieci neuronowej, która miała przetwarzać otrzymane obrazy znaków w edytowalny tekst.

Trening tej sieci odbywał się na podstawie samodzielnie stworzonego zbioru danych, który przedstawiał sobą obrazy różnych znaków w różnych czcionkach, jak pokazano poniżej.

Dany model ma 3 warstwy konwolucyjne:

```
inpx = Input(shape=inpx)
layer1 = Conv2D(32, kernel_size=(3, 3), activation='relu')(inpx)
layer21 = Conv2D(64, (3, 3), activation='relu')(layer1)
layer22 = Conv2D(64, (3, 3), activation='relu')(layer21)
```

- warstwa 1 to warstwa Conv2d, która splata obraz za pomocą 32 filtrów każdy o rozmiarze (3*3).
- warstwa 21 jest ponownie warstwą Conv2D, która jest również używana do splatania obrazu i używa 64 filtrów każdy o rozmiarze (3*3).
- warstwa 22 jest identyczną do poprzedniej.

Dalej mamy:

```
layer3 = MaxPooling2D(pool_size=(3, 3))(layer22)
layer4 = Dropout(0.5)(layer3)
layer5 = Flatten()(layer4)
layer6 = Dense(250, activation='sigmoid')(layer5)
layer7 = Dense(y_train.shape[1], activation='softmax')(layer6)
```

- warstwa 3 to warstwa MaxPooling2D, która wybiera maksymalną wartość z macierzy o rozmiarze (3*3).
- warstwa 4 pokazuje Dropout z szybkością 0.5, to jest warstwa regularyzacji.
- warstwa 5 spłaszcza dane wyjściowe uzyskane z warstwy 4 i to spłaszczanie danych wyjściowych jest przekazywane do warstwy 6.
- warstwa 6 to ukryta warstwa sieci neuronowej zawierającej 250 neuronów.
- warstwa 7 to warstwa wyjściowa, która korzysta z funkcji softmax.

Następnie wywołujemy funkcje kompilacji i dopasowania:

```
model = Model([inpx], layer7)
model.compile(optimizer=keras.optimizers.Adam(),
              loss=keras.losses.categorical_crossentropy,
              metrics=['accuracy'])
```

- Najpierw utworzyliśmy obiekt modelu tak, jak pokazano w powyższych wierszach, gdzie [inpx] to dane wejściowe w modelu, a layer7 to dane wyjściowe modelu. Skompilowaliśmy model przy użyciu optymalizatora “Adam” oraz funkcji straty i wypisaliśmy dokładność.

```
try:
    model.fit(x_train, y_train, epochs=10, batch_size=500)
except KeyboardInterrupt:
    print('interrupted')
finally:
    score = model.evaluate(x_test, y_test, verbose=0)
    print('loss=', score[0])
    print('accuracy=', score[1])

    model.save(sys.argv[1])
    print('model saved')
```

- Na ostatnim wywołaniu model.fit wraz z parametrami takimi jak x_train (oznacza wektory obrazu), y_train (oznacza etykietę), liczba epok i wielkość partii. Używając funkcji fit x_train oraz zestaw danych y_train jest dostarczany do modelu w określonej wielkości partii.

Ten model nie klasyfikuje idealnie liter, gdyż jest wytrenowany na zbiorze który daje algorytm wycinania. Żeby pracował lepiej należałoby idealnie rozdzielić litery lub odfiltrować ręcznie zbiór do wytrenowania modelu.

Połączenie skryptów w Matlab i Python

Do przetwarzania otrzymanych danych z algorytmu opisanego na początku, zostały napisane dwa w zasadzie oddzielne programy. Główny program sterujący jest napisany w Matlab. Po rozdzieleniu liter zapisuje on do pliku tymczasowego wyniki swojej pracy. Następnie uruchamia on skrypt w Python, który ładuje te pliki i kontynuuje pracę dekodując litery i wypisując je na konsoli. Po wszystkim plik tymczasowy jest usuwany.

Uruchamianie programu

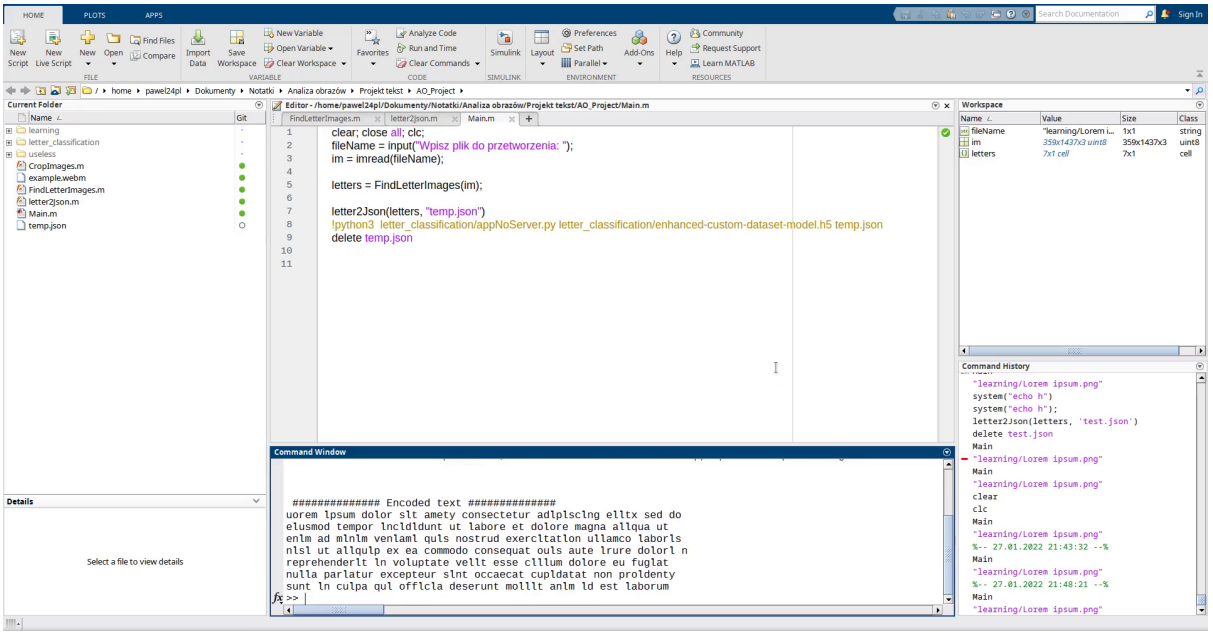
Przygotowanie środowiska

Przed pierwszym uruchomieniem należy zainstalować wszystkie zależności. Należą do nich:

- ☐ pełny pakiet Matlab
- ☐ python3
- ☐ pip3
- ☐ zależności pythona do instalacji przez pip3 zapisane w pliku *requirements.txt* (pip install -r requiremetns.txt)

Uruchomienie programu

Po zainstalowaniu wszystkich zależności należy uruchomić plik *Main.m* w Matlab. Po uruchomieniu poprosi on o podanie ścieżki do zdjęcia. Należy wtedy mu wpisać tę ścieżkę w cudzysłowie, np.: *"learning/Lorem ipsum.png"*. Po chwili na konsoli zostanie wypisane trochę tekstu związanego z bibliotekami *Python* po czym zacznie pojawiać się odkodowany tekst.



Widok Matlab po zakończeniu działania programu.

Wnioski

Pomimo że program nie nie miał szczególnie utrudnionego zadania - tekst był drukowany, nie było to pismo odręczne - nie działał on idealnie. Największe problemy były z rozdzieleniem liter. Przy niektórych czcionkach - szczególnie po wyrenderowaniu - było niemożliwe do rozróżnienia gdzie się zaczyna, a gdzie kończy litera. Ciężko było dopasować algorytm, który jednocześnie nie sklejałby dwóch liter *t* i nie rozdzielał nigdzie *m* w przypadku czcionek szeryfowych. Ostatecznie udało się stworzyć coś, co działa w większości przypadków poprawnie, jednak daleko mu do doskonałości czytania tekstu porównywalnej do człowieka.